28th Annual European Symposium on Algorithms

ESA 2020, September 7–9, 2020, Pisa, Italy (Virtual Conference)

Edited by
Fabrizio Grandoni
Grzegorz Herman
Peter Sanders

LIPIcs – Vol. 173 – ESA 2020 www.dagstuhl.de/lipics

Editors

Fabrizio Grandoni
IDSIA, USI-SUPSI, Manno, Switzerland
fabrizio@idsia.ch

Grzegorz Herman
Jagiellonian University, Kraków, Poland
gherman@tcs.uj.edu.pl

Peter Sanders
Karlsruhe Institute of Technology, Germany
sanders@kit.edu

ACM Classification 2012
Applied computing → Transportation; Theory of computation → Facility location and clustering; Computing methodologies → Agent / discrete models; Computing methodologies → Algebraic algorithms; Computing methodologies → Combinatorial algorithms; Human-centered computing → Graph drawings; Mathematics of computing → Approximation algorithms; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Graph algorithms; Mathematics of computing → Graph coloring; Mathematics of computing → Graphs and surfaces; Mathematics of computing → Graph theory; Mathematics of computing → Hypergraphs; Mathematics of computing → Information theory; Mathematics of computing → Network flows; Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Random graphs; Mathematics of computing → Solvers; Mathematics of computing → Trees; Networks → Network algorithms; Theory of computation → Algorithmic game theory and mechanism design; Theory of computation → Approximation algorithms analysis; Theory of computation → Computational geometry; Theory of computation → Data structures design and analysis; Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms; Theory of computation → Distributed algorithms; Theory of computation → Dynamic graph algorithms; Theory of computation → Facility location and clustering; Theory of computation → Fixed parameter tractability; Theory of computation → Graph algorithms analysis; Theory of computation → Network flows; Theory of computation → Online algorithms; Theory of computation → Packing and covering problems; Theory of computation → Parallel algorithms; Theory of computation → Parallel computing models; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Pattern matching; Theory of computation → Problems, reductions and completeness; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Random network models; Theory of computation → Routing and network design problems; Theory of computation → Scheduling algorithms; Theory of computation → Shared memory algorithms; Theory of computation → Shortest paths; Theory of computation → Sorting and searching; Theory of computation → Stochastic approximation

ISBN 978-3-95977-162-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-162-7.

Publication date
August, 2020

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.ESA.2020.0
ISBN 978-3-95977-162-7 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the extended abstracts selected for presentation at ESA 2020, the 28th European Symposium on Algorithms. Due to the COVID-19 pandemia, the symposium was organized as a virtual meeting during September 7–9 by the University of Pisa, Italy as part of ALGO 2020. Performing the switch from physical to virtual meeting was accompanied by many discussions about how to handle this situation. However, eventually, the scientific quality of the program as well as the process of selecting contributions was surprisingly little affected. In particular, the number of submissions as well as the acceptance rate was similar to previous years. The PC-chairs gave an exceptional five-day deadline extension to mitigate potential difficulties of authors in handling the situation. This proved an effective means of achieving a high turnout of good submissions.

The scope of ESA includes original, high-quality, theoretical and applied research on algorithms and data structures. Since 2002, it has had two tracks: the Design and Analysis Track (Track A), intended for papers on the design and mathematical analysis of algorithms, and the Engineering and Applications Track (Track B), for submissions that also address real-world applications, engineering, and experimental analysis of algorithms. Information on past symposia, including locations and proceedings, is maintained at `http://esa-symposium.org`.

In response to the call for papers for ESA 2020, 313 papers were submitted, 262 for Track A and 51 for Track B (these are the counts after the removal of papers with invalid format and after withdrawals). Paper selection was based on originality, technical quality, exposition quality, and relevance. Each paper received at least three reviews. The program committees selected 83 papers for inclusion in the program, 70 from track A and 13 from track B, yielding an acceptance rate of about 1/4.

The European Association for Theoretical Computer Science (EATCS) sponsored a best paper award and a best student paper award. A submission was eligible for the best student paper award if all authors were doctoral, master, or bachelor students at the time of submission. The best student paper award for track A was given to Hanrui Zhang for the paper *Improved Prophet Inequalities for Combinatorial Welfare Maximization with (Approximately) Subadditive Agents*. The best paper award for track A was given to Moritz Venzin and Friedrich Eisenbrand for the paper *Approximate CVP_p in time* $2^{0.802n}$. The best paper award for track B was given to Georg Osang, Mael Rouxel-Labbé and Monique Teillaud for the paper *Generalizing CGAL Periodic Delaunay Triangulations*. No best student paper award has been given this year for track B.

We wish to thank all the authors who submitted papers for consideration, the invited speakers, the members of the program committees for their hard work, and all the external reviewers who assisted the program committees in the evaluation process. Special thanks go to the organizing committee, who helped us with the organization of the conference. All of these people did a great job in keeping ESA a hoard of excellence in a difficult situation.

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committees

Track A (Design and Analysis) Program Committee

- Amir Abboud, IBM Almaden Research Center
- Greg Bodwin, Georgia Tech
- Karl Bringmann, Saarland University and Max Planck Institute for Informatics
- Deeparnab Chakrabarty, Dartmouth College
- Daniel Dadush, Vrije Universiteit Amsterdam
- Michael Elkin, Ben-Gurion University of the Negev
- Leah Epstein, University of Haifa
- Manuela Fischer, ETH Zurich
- Fabrizio Frati, Roma Tre University
- Fabrizio Grandoni, IDSIA (PC Chair)
- Kasper Green Larsen, Aarhus University
- Jacob Holm, University of Copenhagen
- Michael Kapralov, Ecole Polytechnique Fédérale de Lausanne
- Petteri Kaski, Aalto University
- Telikepalli Kavitha, Tata Institute of Fundamental Research
- Tomasz Kociumaka, Bar-Ilan University
- Moshe Lewenstein, Bar-Ilan University
- Shi Li, University at Buffalo
- Yury Makarychev, Toyota Technological Institute at Chicago
- Krzysztof Onak, IBM T.J. Watson Research Center
- Seth Pettie, University of Michigan
- Marcin Pilipczuk, University of Warsaw
- Thomas Rothvoss, University of Washington
- Laura Sanità, Eindhoven University of Technology
- Saket Saurabh, The Institute of Mathematical Sciences
- Chris Schwiegelshohn, TU Dortmund
- Vera Traub, University of Bonn
- Carmine Ventre, King's College London
- David Wajc, Carnegie Mellon University
- Andreas Wiese, Universidad de Chile
- David P. Woodruff, Carnegie Mellon University
- Meirav Zehavi, Ben-Gurion University of the Negev

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Track B (Engineering and Applications) Program Committee

- Armin Biere, Johannes Kepler University Linz
- Maike Buchin, Ruhr Universität Bochum
- Markus Chimani, Osnabrück University
- Irene Finocchi, LUISS Guido Carli University Rome
- Travis Gagie, Dalhousie University Halifax
- Rolf Niedermeier, TU Berlin
- Kunihiko Sadakane, The University of Tokyo
- Peter Sanders, Karlsruhe Institute of Technology (PC Chair)
- Yihan Sun, Carnegie Mellon University
- Sivan Toledo, Tel-Aviv University
- Jesper Träff, Vienna University of Technology
- Renato Werneck, Amazon

List of External Reviewers

Anders Aamand
Andreas Abels
Mikkel Abrahamsen
Marek Adamczyk
Peyman Afshani
Akanksha Agrawal
Hugo Akitaya
Carlos Alegría
Josh Alman
Georgios Amanatidis
Daniel Anderson
Haris Angelidakis
Patrizio Angelini
Spyros Angelopoulos
Antonios Antoniadis
Lars Arge
Boris Aronov
Sepehr Assadi
Gennadiy Averkov
Pranjal Awasthi
Kyriakos Axiotis
Michael Axtmann
Yossi Azar
Arturs Backurs
Ainesh Bakshi
Eric Balkanski
Nikhil Bansal
Leonid Barenboim
Michael Barrus
Luca Becchetti
Soheil Behnezhad
Xiaohui Bei
Robert Benkoczi
Matthias Bentert
Kristóf Bérczi
Benjamin Bergougnoux
Giulia Bernardini
Aaron Bernstein
Aditya Bhaskara
Anup Bhattacharya
Sayan Bhattacharya
Sujoy Bhore
Therese Biedl
Armin Biere
Philip Bille
Georgios Birmpas
Thomas Bläsius
Guy Blelloch
Niclas Boehmer
Fritz Bökler
Manuel Borrazzo
Vladimir Braverman
Nick Brettell
Gerth Stølting Brodal
Brian Brubach
Niv Buchbinder
Valentin Buchhold
Kevin Buchin
Maike Buchin
Andrei Bulatov
Sergio Cabello
Tiziana Calamoneri
Pilar Cano
Yixin Cao
Nofar Carmeli
Diptarka Chakraborty
Sankardeep Chakraborty
Parinya Chalermsook
T-H. Hubert Chan
Timothy M. Chan
Hsien-Chih Chang
Panagiotis Charalampopoulos
Abhranil Chatterjee
Shiri Chechik
Ho-Lin Chen
Wei Chen
Victor Chepoi
Markus Chimani
Rajesh Chitnis
Eden Chlamtac
Janka Chlebikova
Philip Chodrow
Davin Choo
Keerti Choudhary
Raphael Clifford
Christian Coester
Ilan Cohen
Sarel Cohen
Vincent Cohen-Addad
Emilio Cruciani

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Giordano Da Lozzo
Konrad K. Dąbrowski
Ovidiu Daescu
Shantanu Das
Argyrios Deligkas
Holger Dell
Dariusz Dereniowski
Karen Devine
Laxman Dhulipala
Emilio Di Giacomo
Hu Ding
Michael Dinitz
Yann Disser
Anne Driemel
Andre Droschinsky
Ran Duan
Bartłomiej Dudek
Thomas Dybdahl Ahle
Eduard Eiben
Marek Elias
Alina Ene
David Eppstein
Jeff Erickson
Hossein Esfandiari
Yuri Faenza
Piotr Faliszewski
Dan Feldman
Moran Feldman
Zhili Feng
Henning Fernau
Diodato Ferraioli
Yuval Filmus
Arnold Filtser
Irene Finocchi
Johannes Fischer
Noah Fleming
Till Fluschnik
Efi Fogel
Fedor Fomin
Kyle Fox
Daniel Freund
Zachary Friggstad
Hu Fu
Takuro Fukunaga
Radoslav Fulek
Mitsuru Funakoshi
Stefan Funke
Federico Fusco
Travis Gagie
Jakub Gajarský
Arun Ganesh
Robert Ganian
Jie Gao
Naveen Garg
Bernd Gärtner
Cyril Gavoille
Pawel Gawrychowski
Burkay Genc
Michele Gentili
Arijit Ghosh
Prantar Ghosh
Archontia Giannopoulou
Daniel Gibney
Alexander Göke
Petr Golovach
Alexander Golovnev
Adrián Gómez Brandón
Gramoz Goranci
Mayank Goswami
Sander Gribling
Luca Grilli
Martin Grohe
Martin Gronemann
Allan Grønlund
Yan Gu
Luciano Gualà
Joachim Gudmundsson
Siddharth Gupta
Sushmita Gupta
Guru Guruganesh
Michel Habib
Torben Hagerup
Thekla Hamm
Sariel Har-Peled
David Harris
Avinatan Hassidim
Ishay Haviv
Meng He
Qizheng He
Klaus Heeger
Marc Hellmuth
Danny Hermelin
Tobias Heuer
Michael Horton
Chien-Chung Huang
Zhiyi Huang

Sophie Huiberts
Christoph Hunkenschröder
Thore Husfeldt
John Iacono
Sharat Ibrahimpur
Max Bernhard Ilsen
Shunsuke Inenaga
Gabor Ivanyos
Yoichi Iwata
Taisuke Izumi
Samin Jamalabadi
Bart M. P. Jansen
Klaus Jansen
Rajesh Jayaram
Łukasz Jeż
Dawei Jiang
Matthew Johnson
Gwenaël Joret
Praneeth Kacham
Naonori Kakimura
Sagar Kale
John Kallaugher
Naoyuki Kamiyama
Lior Kamma
Frank Kammer
Daniel Kane
Haim Kaplan
Adam Karczmarz
Charles Karney
Andreas Karrenbauer
Matthew Katz
Yasushi Kawase
Leon Kellerhals
Dominik Kempa
Eun Jung Kim
Evangelos Kipouridis
Sándor Kisfaludi-Bak
Aleks Kissinger
Kim-Manuel Klein
Peter Kling
Dušan Knop
Tomohiro Koana
Jochen Koenemann
Alexander Kononov
Christian Konrad
Tsvi Kopelowitz
Dominik Köppl
Evgenios Kornaropoulos
Guy Kortsarz
Dmitry Kosolobov
Martin Koutecky
Ioannis Koutis
Laszlo Kozma
Evangelos Kranakis
Ravishankar Krishnaswamy
Amer Krivosija
Dominik Krupke
Janardhan Kulkarni
Gunjan Kumar
Marvin Künnemann
O-Joung Kwon
Maria Kyropoulou
Arnaud Labourel
Jakub Łącki
Leon Ladewig
Bundit Laekhanukit
Victor Lagerkvist
John Lapinskas
Dolores Lara
Kim S. Larsen
Philip Lazos
Hung Le
Francois Le Gall
Euiwoong Lee
Johannes Lengler
Stefano Leucci
Asaf Levin
Roie Levin
Avivit Levy
Caleb Levy
Moshe Lewenstein
Jason Li
Jian Li
Yingyu Liang
William Lochet
Maarten Löffler
Brendan Lucier
Junjie Luo
Jayakrishnan Madathil
Sepideh Mahabadi
Arvind Mahankali
Konstantin Makarychev
Frederik Mallmann-Trenn
David Manlove
Fredrik Manne
Giovanni Manzini

Alberto Marchetti-Spaccamela
Mathieu Mari
Shaked Matar
Jannik Matuschke
Giancarlo Mauri
Samuel McCauley
Moti Medina
Arianne Meijer
Henk Meijer
Reshef Meir
George Mertzios
David L. Millman
Majid Mirzanezhad
Neeldhara Misra
Pranabendu Misra
Matthias Mnich
Hendrik Molter
Tobias Mömke
Rafaelle Mosca
Benjamin Moseley
Amer Mouawad
Noela Müller
Wolfgang Mulzer
Alexander Munteanu
Christopher Musco
Wojciech Nadara
Viswanath Nagarajan
Yuto Nakashima
Seffi Naor
Jesper Nederlof
Maryam Negahbani
Ofer Neiman
Yakov Nekrich
Alantha Newman
Huy Nguyen
André Nichterlein
Aleksandar Nikolov
Naomi Nishimura
Martin Nöllenburg
Navid Nouri
Alexander Nover
Krzysztof Nowicki
André Nusser
Timm Oertel
Karolina Okrasa
Neil Olver
Shmuel Onn
Joel Ouaknine
Andrea Pacifici
Rasmus Pagh
Dömötör Pálvölgyi
Fahad Panolan
Evanthia Papadopoulou
Irene Parada
Kunsoo Park
Nikos Parotsidis
Merav Parter
Alice Paul
Andrzej Pelc
Richard Peng
Martin Pergel
Jeff Phillips
Michał Pilipczuk
Solon Pissis
Adam Polak
Ely Porat
Julian Portmann
Nicola Prezza
Maximilian Probst Gutenberg
Kirk Pruhs
Manish Purohit
Yuval Rabani
Jakub Radoszewski
Akbar Rafiey
Saladi Rahul
Benjamin Raichel
Cyrus Rashtchian
Nidhi Rathi
R Ravi
Saurabh Ray
Bhaskar Ray Chaudhury
Ilya Razenshteyn
Igor Razgon
Rebecca Reiffenhäuser
Malte Renken
Thomas Robinson
Liam Roditty
Dennis Rohde
Lars Rohwedder
Massimiliano Rossi
Benjamin Rossman
Eva Rotenberg
Alan Roytman
Natan Rubin
Aviad Rubinstein
Paweł Rzążewski

Guy Saar
Kunihiko Sadakane
Mohammad Salavatipour
R.B. Sandeep
Peter Sanders
Piotr Sankowski
Ramprasad Saptharishi
Thatchaphol Saranurak
Srinivasa Rao Satti
Ignasi Sau
David Saulpic
Saket Saurabh
Nitin Saxena
Melanie Schmidt
Ulrike Schmidt-Kraepelin
Oded Schwartz
Uwe Schwiegelshohn
Andras Sebo
Saeed Seddighin
Paolo Serafino
Alkmini Sgouritsa
Roohani Sharma
Bruce Shepherd
Abhishek Shetty
Anastasios Sidiropoulos
Sebastian Siebertz
Ana Silva
Francesco Silvestri
Genevieve Simonet
Sahil Singla
Nodari Sitchinava
Benjamin Smith
Marcin Smulewicz
Dina Sokol
Shay Solomon
Frank Sommer
Rishi Sonthalia
Manuel Sorge
Krzysztof Sornat
José A. Soto
Sophie Spirkl
Ramanujan M. Sridharan
Piyush Srivastava
Frank Staals
Tatiana Starikovskaya
Barak Steindl
Teresa Anna Steiner
Noah Stephens-Davidowitz
Ben Strasser
Vijay Subramanya
He Sun
Kevin Sun
Yihan Sun
Subhash Suri
Svend Christian Svendsen
Zoya Svitkina
Alexander Svozil
Prafullkumar Tale
Jakub Tarnawski
Jan Arne Telle
Sharma V. Thankachan
Clayton Thomas
Mikkel Thorup
Daniel Ting
Sivan Toledo
Noam Touitou
Ohad Trabelsi
Jesper Träff
Nicolas Tremblay
Thorben Tröbst
Niklas Troost
Tom Tseng
Marc Uetz
Przemysław Uznański
Ali Vakilian
Greg Van Buskirk
Yann Vaxès
José Verschae
Aravindan Vijayaraghavan
Marc Vinyals
Ellen Vitercik
Magnus Wahlström
Erik Waingarten
Tomasz Walen
Haitao Wang
Hung-Lung Wang
Yipu Wang
Yiqiu Wang
Justin Ward
Karol Węgrzycki
Yuanhao Wei
Oren Weimann
Nicole Wein
Omri Weinstein
Renato Werneck
Chris Whidden

List of External Reviewers

Daniel Wiebking
Tilo Wiedera
Sebastian Wiederrecht
Mathijs Wintraecken
Sascha Witt
Michal Wlodarczyk
Prudence Wong
Sampson Wong
Marcin Wrochna
Christian Wulff-Nilsen
Yinzhan Xu
Sheng Yang
Jonathan Yaniv
Taisuke Yasuda
Shangdi Yu
Lydia Zakynthinou
Viktor Zamaraev
Or Zamir
Rico Zenklusen
Ruizhe Zhang
Zhao Zhang
Samson Zhou
Philipp Zschoche
Wiktor Zuba
Mark de Berg
Bart de Keijzer
Erik Jan van Leeuwen
André van Renssen
Rob van Stee
Marieke van der Wegen
Tom van der Zanden

Planar Bichromatic Bottleneck Spanning Trees

A. Karim Abu-Affash
Software Engineering Department, Shamoon College of Engineering, Beer-Sheva, Israel
abuaa1@sce.ac.il

Sujoy Bhore
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel
sujoy.bhore@gmail.com

Paz Carmi
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel
carmip@cs.bgu.ac.il

Joseph S. B. Mitchell
Department of Applied Mathematics and Statistics, Stony Brook University, NY, USA
joseph.mitchell@stonybrook.edu

Abstract

Given a set P of n red and blue points in the plane, a *planar bichromatic spanning tree* of P is a geometric spanning tree of P, such that each edge connects between a red and a blue point, and no two edges intersect. In the bottleneck planar bichromatic spanning tree problem, the goal is to find a planar bichromatic spanning tree T, such that the length of the longest edge in T is minimized. In this paper, we show that this problem is NP-hard for points in general position. Our main contribution is a polynomial-time $(8\sqrt{2})$-approximation algorithm, by showing that any bichromatic spanning tree of bottleneck λ can be converted to a planar bichromatic spanning tree of bottleneck at most $8\sqrt{2}\lambda$.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases Approximation Algorithms, Bottleneck Spanning Tree, NP-Hardness

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.1

Funding This work was partially supported by Grant 2016116 from the United States – Israel Binational Science Foundation. Work by J. Mitchell was partially supported by NSF (CCF-1526406, CCF-2007275).

1 Introduction

Let P be a bi-colored set of red and blue points in the plane and let $n = |P|$. A *bichromatic spanning tree* of P is a spanning tree of P whose edges are straight-line segments connecting between points of different colors. A spanning tree is *planar* if its edges do not cross each other. Borgelt et al. [15] studied the problem of computing a minimum-weight planar bichromatic spanning tree, and showed that the problem is NP-hard. Moreover, for points in general position, they gave an $O(\sqrt{n})$-approximation algorithm, and for points in convex position, they gave an exact cubic-time algorithm. Biniaz et al. [11] studied the problem of computing a maximum-weight planar bichromatic spanning tree and gave a (1/4)-approximation algorithm for the problem.

Algorithmic problems on bichromatic geometric input have appeared in many problems, including, e.g., trees [1, 12, 15], matchings [13, 18], and partitionings [19]. Often the bichromatic input is referred to as "red-blue" input, e.g. in red-blue intersection [4, 23], red-blue separation [9, 14, 17, 20], and red-blue connection problems [5, 10]. For a survey of many geometric problems on bichromatic (red-blue) points, see Kaneko and Kano [21].

Many of the structures studied in computational geometry are planar, including minimum spanning trees, minimum weight matchings, Delaunay/Voronoi diagrams, etc. Therefore, the planarity requirement is quite natural, and indeed many researchers have considered geometric problems dealing with crossing-free configurations in the plane; see, e.g. [2, 3, 6, 7, 8].

In this paper, we study the problem of computing a bottleneck planar bichromatic spanning tree of P, in which we seek a planar bichromatic spanning tree that minimizes the bottleneck, i.e., the length of the longest edge. To the best of our knowledge, this problem has not been considered before.

Our results. In Section 2, we prove that the problem of computing a bottleneck planar bichromatic spanning tree is NP-hard. Our proof is based on a reduction from the planar 3-SAT problem, and is influenced by the proof of Borgelt et al. [15]. As a corollary we obtain that the problem does not admit a PTAS. Next, in Section 3, we present a polynomial-time algorithm for computing a planar bichromatic spanning tree of bottleneck at most $8\sqrt{2}$ times the bottleneck OPT of an optimal such planar bichromatic spanning tree. We first compute a bottleneck bichromatic spanning tree having bottleneck λ that may have crossings (so λ is a lower bound on OPT). Then, we use the length λ to define a partition of the plane into convex cells, and to partition P into subsets according to these cells. Next, we construct planar bichromatic trees for each subset, and we connect these trees to obtain a planar bichromatic spanning tree of P. We show that this tree has a bottleneck at most $8\sqrt{2}\lambda$.

2 Hardness Proof

In this section, we prove the NP-hardness (Theorem 1) of computing a bottleneck planar bichromatic spanning tree; our proof also shows (Corollary 2, below) that there is no approximation factor less than $\sqrt{2}$, unless $P = NP$.

▶ **Theorem 1.** *Let P be a set of n red and blue points in the plane. Computing a bottleneck planar bichromatic spanning tree of P is NP-hard.*

Proof. We adapt the proof of Borgelt et al. [15], making modifications necessary to address the bottleneck version. For completeness, we explain the ingredients required for the proof.

The proof is based on a reduction from the planar 3-SAT problem. Given a 3-CNF formula F with n variables $X = \{x_1, x_2, \ldots, x_n\}$ and m clauses $Y = \{C_1, C_2, \ldots, C_m\}$, let $G_F = (V, E)$ be the graph of F, i.e., $V = X \cup Y$ and $E = \{(x_i, C_j) : x_i \text{ or } \widetilde{x_i} \text{ appears in } C_j\}$. If G_F is planar, then F is called a planar 3-CNF formula. The planar 3-SAT problem is to determine whether a given planar 3-CNF formula F is satisfiable; the problem is NP-complete [22].

Let F be a planar 3-SAT formula. We construct, in polynomial time, a set P of red and blue points in the plane, such that F is satisfiable if and only if there exists a planar bichromatic spanning tree of P of bottleneck 1. Consider the graph G_F. It is well known that G_F can be embedded in the plane in polynomial time, using polynomial area, i.e., with the $n + m$ nodes of G_F placed at grid points of a regular square grid of size $O(n + m)$-by-$O(n + m)$ [16].

The construction is based on chains of pairs of red and blue points. We call the pairs in the chain *sites*. The distance between the two points comprising each site is less than 1, and the distance between two points of different colors in consecutive (along the chain) sites is exactly 1; see Figure 1(a). Now, for every two consecutive sites, there are two possible edges to connect them using edges of length at most 1: we either connect (with an edge

of length exactly 1) the blue point of the first site with the red point of the second site, or the other way around. Moreover, the chain is constructed in such a way that if we connect the blue point in the leftmost site to the red point in the next site, this forces the choice of connections in one direction along the chain; see Figure 1(b).

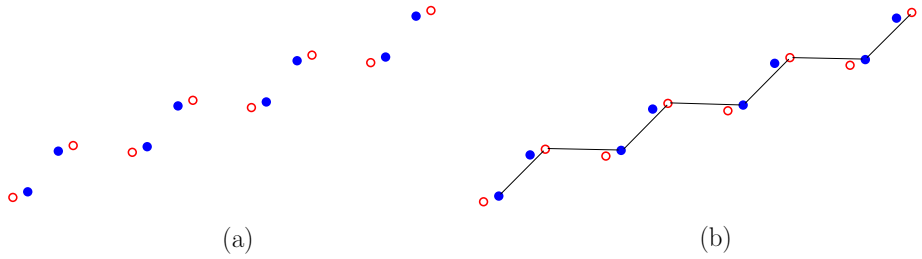


Figure 1 A chain of red-blue sites.

Variables. Each variable $x_i \in X$ is represented by a circular chain of $O(m)$ sites, a special red point r_i, and a red-blue path; see Figure 2. The addition of the red-blue path to the variable gadget of [15] is required, since without it the special red point r_i can be connected to both sides of the chain without increasing the bottleneck, which is not the case in the minimum weight version. The red-blue path forces r_i to be connected exactly to one side.

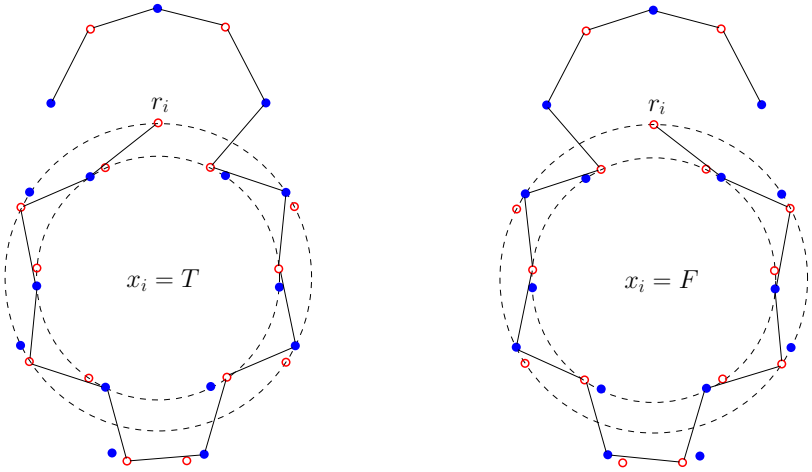


Figure 2 The trees corresponding to the true and the false assignments of x_i.

The sites on the circular chain are located on two circles, an inner circle and an outer circle. We locate the points in such a way that the (bichromatic) distances are 1 between consecutive sites on the circular chain, the distance is 1 between r_i and the blue points of its neighboring sites in the chain, and the distance is 1 between the endpoints of the red-blue path and the corresponding points of the chain. Moreover, the points are located such that there are only two possible optimal trees (i.e., planar bichromatic spanning trees of bottleneck 1) of the points, depending on the connection of r_i to the chain. In each of the two trees, r_i is connected to exactly one site of the chain. We associate (arbitrarily) one possibility with the assignment $x_i = T$, and the other with the assignment $x_i = F$; see Figure 2. Thus, the value of x_i will determine the tree of these points, and vice virsa. Moreover, if $x_i = T$ (resp., $x_i = F$), then the red points on the right (resp., left) of the inner circle are free to be connected to points outside the gadget, and the red points on the left (resp., right) of the inner circle cannot be connected to points outside the gadget without crossing, and vice versa.

Clauses. Each clause C_j is represented by a single red point r_j and three chains that are connected to the respective variables of C_j; see Figure 3(a). The distance between r_j and each chain is 1. In any optimal tree, r_j will be connected to at least one of the three chains. However, it cannot be connected to any chain if all the chains are connected to variables that are in the wrong state; see Figure 3(b).

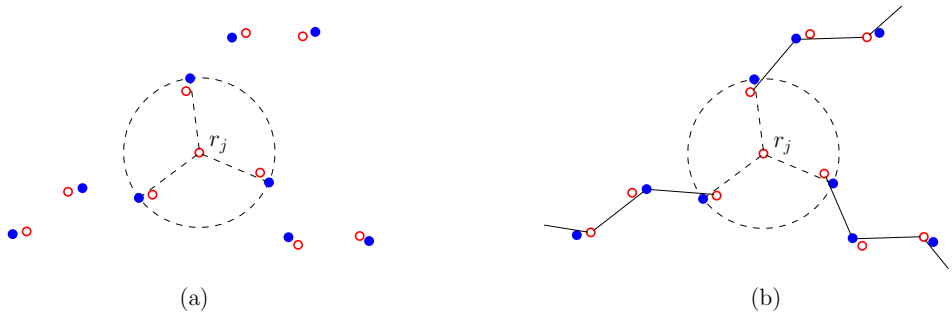


Figure 3 The gadget corresponding to the clause C_j.

We connect between the variables and the clauses such that, in any optimal tree, one of the three chains of the clause has to be connected to a red point on the inner circle of the corresponding variable. Assume that x_i appears unnegated in a clause C_j and negated in a clause C_k, i.e., $C_j = (x_i \vee \cdot \vee \cdot)$ and $C_k = (\widetilde{x_i} \vee \cdot \vee \cdot)$. We connect the chain of C_j that is respective to x_i to a site on the right of the inner circle of the gadget x_i, and we connect the chain of C_k that is respective to x_i to a site on the left of the inner circle of the gadget x_i; see Figure 4. This connection ensures that, if x_i is assigned T, then the red point on the right of the inner circle of x_i is free to be connected to the chain of C_j, and this connection can produce a path through the chain that ends at r_j. On the other hand, if x_i is assigned T, then the red point on the left of the inner circle of x_i cannot be connected to the chain of C_k, which does not allow a connection between the chain and r_k. The same argument holds when x_i is assigned F.

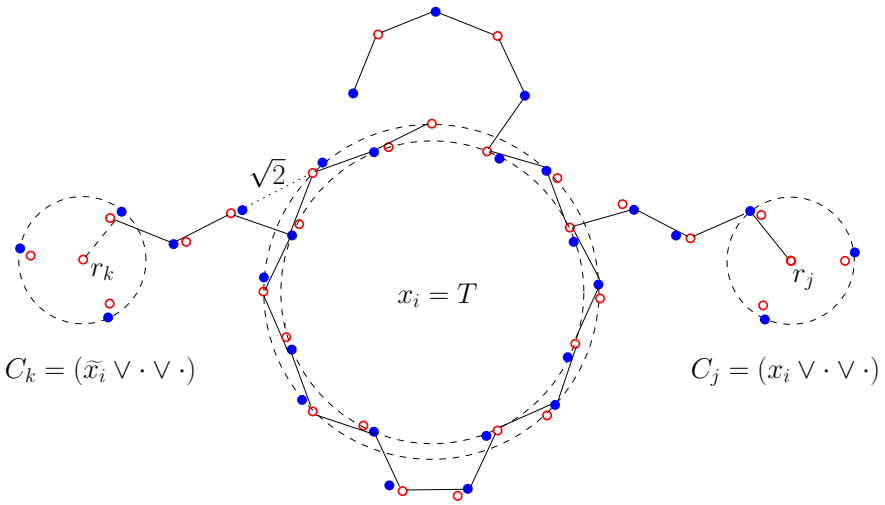


Figure 4 The connection between the variable x_i and the clauses C_j and C_k.

Finally we need to connect all variables to each other by some fixed part of the tree, because the whole construction needs to be a tree and not a forest. These connections can easily be made using red-blue chains having distance at most 1 between every two consecutive points in the path. Also, we need to make sure that the distance between different parts of the construction is large enough to avoid shortcuts.

Overall, the reduction is performed in polynomial (in n, m) time, since the planar graph G_F is drawn on a regular $O(n+m)$-by-$O(n+m)$ grid, and each of our variable gadgets is of size $O(m)$, resulting in overall polynomial area and a polynomial-size construction that is computed in polynomial (in n, m) time. ◀

Notice that in the reduction we proved that if the 3-SAT formula is not satisfiable, then any planar bichromatic spanning tree of P has an edge of length greater than 1. Actually, we can push the length of this edge to be closer to $\sqrt{2}$. That is, we can draw the connection between each clause and its corresponding variables, such that the distance between each chain of the clause and the corresponding site on the inner circle of the variable is 1, and the distance between each chain of the clause and the sites on the outer circle of the variable is at least $\sqrt{2}-\varepsilon$, for any $0 < \varepsilon < \sqrt{2}-1$; see Figure 4. This implies that the bottleneck planar bichromatic spanning tree problem cannot be approximated within a factor less than $\sqrt{2}$, unless $P = NP$.

▶ **Corollary 2.** *The bottleneck planar bichromatic spanning tree problem cannot be approximated within a factor less than $\sqrt{2}$, unless $P = NP$. In particular, there is no PTAS (unless $P = NP$).*

3 Approximation Algorithm

Let P be a set of red and blue points in the plane and let $n = |P|$. Let T be a bichromatic spanning tree of P of minimum bottleneck (T may have crossings and can be computed in $O(n \log n)$ time [12]). Let λ denote the bottleneck of T, i.e., the length of the longest edge in T. Notice that λ is the lower bound for any bichromatic spanning tree of P, in particular for any planar bottleneck bichromatic spanning tree of P. In this section, we show how to compute a planar bichromatic spanning tree of P, such that its bottleneck is at most $8\sqrt{2}\lambda$.

Our algorithm partitions the plane into disjoint cells satisfying the following properties:

1. Each cell is convex and contains points of both colors.
2. In each cell, the distance between any two points is at most $5\sqrt{2}\lambda$.
3. The cells are connected, i.e., if we consider the graph with the cells as its vertices and there is an edge between two cells if they are adjacent (sharing a common boundary), then this graph is connected.
4. We can construct a planar bichromatic spanning tree of the points in each cell and we can connect them without crossings.

Assume, w.l.o.g., that $\lambda = 1$. Consider an axis-parallel grid, with each (square) cell of edge length 3 and all points of P in the interior (not on the boundary) of these cells; see Figure 5. We say that a cell $C_{i,j}$ is bichromatic if it contains points of both colors and we say that $C_{i,j}$ is monochromatic (red or blue) if all of the points in $C_{i,j}$ have the same color, otherwise, we say that $C_{i,j}$ is an empty cell.

Our algorithm consists of two stages. In Stage 1, we modify the grid cells to satisfy properties (1)-(3), and, in Stage 2, we construct a planar bichromatic spanning tree of the points in each cell and connect between these trees to obtain a planar bichromatic spanning tree of P.

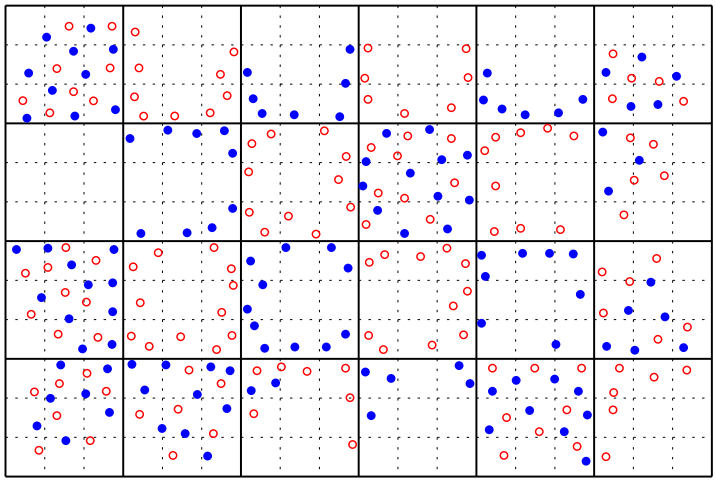


Figure 5 The grid partitioning the points of P.

Stage 1

In this stage, we consider the monochromatic cells and we partition and merge portions of them in order to obtain a subdivision in which all cells are convex and bichromatic. Let $C_{i,j}$ be a 3×3 cell of the grid. Since $C_{i,j}$ is a 3×3 cell, $C_{i,j}$ is the union of 9 unit sub-cells, labelled $C^k_{i,j}$, for $k = 1, 2, 3, \ldots, 9$, as shown in Figure 6(a). Notice that, since $C_{i,j}$ is a monochromatic cell, the points of $C_{i,j}$ are of distance at most 1 from the boundary of $C_{i,j}$, and therefore, $C^5_{i,j}$ is empty of points of P. The region $C_{i,j} \setminus C^5_{i,j}$ is the union of four trapezoids $\mathcal{T}^l_{i,j}$, $\mathcal{T}^r_{i,j}$, $\mathcal{T}^t_{i,j}$, and $\mathcal{T}^b_{i,j}$, such that $\mathcal{T}^l_{i,j}$ (resp., $\mathcal{T}^r_{i,j}$, $\mathcal{T}^t_{i,j}$, and $\mathcal{T}^b_{i,j}$) is the trapezoid obtained by connecting the left (resp., right, top, and bottom) corners of $C_{i,j}$ by diagonals to the left (resp., right, top, and bottom) corners of $C^5_{i,j}$; see Figure 6(b).

(a) (b)

Figure 6 (a) The 9 unit sub-cells of cell $C_{i,j}$. (b) The trapezoids $\mathcal{T}^l_{i,j}$, $\mathcal{T}^r_{i,j}$, $\mathcal{T}^t_{i,j}$, and $\mathcal{T}^b_{i,j}$.

Stage 1.1

In this stage, we introduce a directed graph G in which the vertices are the monochromatic cells and the edges are defined as follows. Let $C_{i,j}$ be a monochromatic cell, and let $\mathcal{N}(C_{i,j}) = \{C_{i,j-1}, C_{i,j+1}, C_{i-1,j}, C_{i+1,j}\}$ be the set of cells that share a grid edge with $C_{i,j}$. Let $C \in \mathcal{N}(C_{i,j})$ be a monochromatic cell and assume, w.l.o.g., that $C = C_{i,j+1}$. There is a directed edge from $C_{i,j}$ to $C_{i,j+1}$ if and only if $C_{i,j}$ and $C_{i,j+1}$ are of different colors and the trapezoid $\mathcal{T}^r_{i,j}$ is not empty of points of P; see Figure 7.

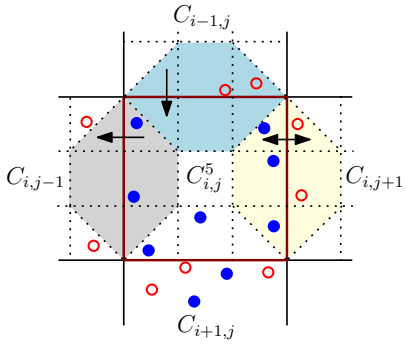


Figure 7 Directed edges between monochromatic cell $C_{i,j}$ and its monochromatic neighbors $(C_{i,j-1}, C_{i-1,j}, C_{i,j+1})$.

Stage 1.2

In this stage, we modify the grid cells by partitioning and merging some of the monochromatic cells with their neighbors, guided by the directed edges introduced in Stage 1.1. Before describing how to modify the grid cells, we describe the following *cell partition procedure* that we will apply in this stage to the empty and some of the monochromatic cells.

Cell partition procedure. For a monochromatic cell $C_{i,j}$, partition $C_{i,j} \setminus C^5_{i,j}$ into trapezoids $\mathcal{T}^l_{i,j}$, $\mathcal{T}^r_{i,j}$, $\mathcal{T}^t_{i,j}$, and $\mathcal{T}^b_{i,j}$, and merge them with the cells $C_{i,j-1}$, $C_{i,j+1}$, $C_{i-1,j}$, and $C_{i+1,j}$, respectively; see Figure 8(b).

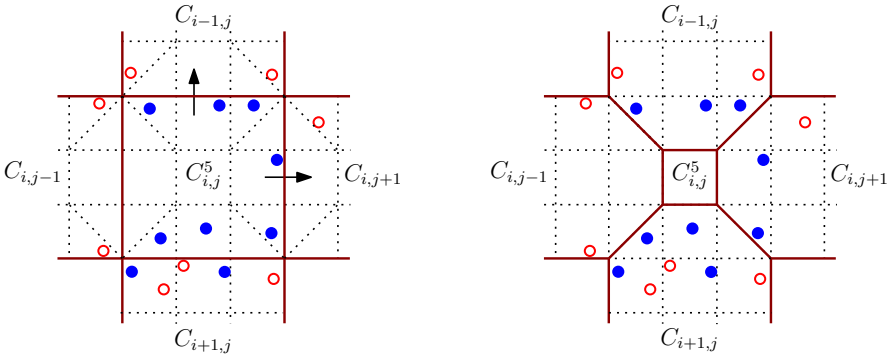


Figure 8 (a) $d_{in}(C_{i,j}) = 0$ and $d_{out}(C_{i,j}) > 0$. (b) Partitioning and merging $C_{i,j}$ with its neighbors.

Let $d_{in}(C_{i,j})$ (resp., $d_{out}(C_{i,j})$) denote the in-degree (resp., the out-degree) of the vertex corresponding to the monochromatic cell $C_{i,j}$ in the graph G. We apply the following three steps on the empty and monochromatic cells.

Step 1. We apply this step as long as there exists a cell $C_{i,j}$ with $d_{in}(C_{i,j}) = 0$ and $d_{out}(C_{i,j}) > 0$. For each such cell, we apply the cell partition procedure on $C_{i,j}$ and remove the out-going edges from $C_{i,j}$ and from its neighbors $C_{i,j-1}$, $C_{i,j+1}$, $C_{i-1,j}$, and $C_{i+1,j}$; see Figure 8.

Step 2. We apply this step on the monochromatic cells $C_{i,j}$ with $d_{in}(C_{i,j}) > 0$. Consider the grid as an arbitrary white-black chessboard. For each white cell with $d_{in}(C_{i,j}) > 0$, we apply the cell partition procedure on $C_{i,j}$.

Step 3. We apply this step on the empty and the monochromatic cells $C_{i,j}$ with $d_{in}(C_{i,j}) = 0$ and $d_{out}(C_{i,j}) = 0$ (that are not considered in the previous steps). For each such cell, we apply the cell partition procedure on $C_{i,j}$.

We call a cell $C_{i,j}$ a *partitioned cell* if $C_{i,j}$ has been partitioned (using the cell partition procedure), and we call it an *extended cell* otherwise. If we have two adjacent partitioned cells $C_{i,j}$ and $C_{i,j+1}$, then we call the merged area of the two trapezoids $\mathcal{T}^r_{i,j}$ and $\mathcal{T}^l_{i,j+1}$ a *lune*; see Figure 9.

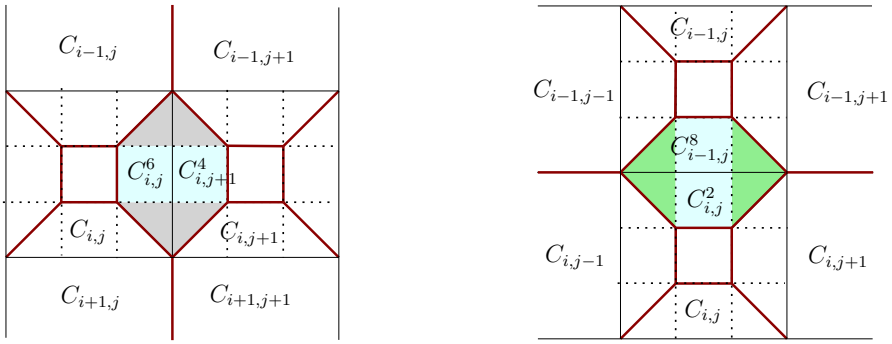


Figure 9 (a) vertical and (b) horizontal lunes.

At the end of this stage, we have three types of non-empty convex cells: original 3×3 cells, extended cells, and lunes. Clearly, each original cell is bichromatic, otherwise, it would have been partitioned or extended in Steps 1–3. Observe that each extended cell $C_{i,j}$ is bichromatic, since $d_{in}(C_{i,j}) > 0$. Observe also that each non-empty lune L is monochromatic. To see this, assume, w.l.o.g., that L was obtained by partitioning the cells $C_{i,j}$ and $C_{i,j+1}$ and merging the trapezoids $\mathcal{T}^r_{i,j}$ and $\mathcal{T}^l_{i,j+1}$. Thus, L cannot be bichromatic, since otherwise, there would be a directed edge from $C_{i,j}$ to $C_{i,j+1}$ and vice versa, which means that one of the cells $C_{i,j}$ and $C_{i,j+1}$ (the black one in the chessboard) is extended in Step 2.

Stage 1.3

In this stage, we get rid of the lunes, by partitioning each lune into sub-pieces and merging the sub-pieces with adjacent extended cells as follows. Let L_1 be a vertical lune obtained by merging two adjacent trapezoids $\mathcal{T}^r_{i,j}$ and $\mathcal{T}^l_{i,j+1}$; see Figure 9(a). (A horizontal lune will be treated analogously.) As observed above, L_1 is monochromatic (or empty) which means that the subregion $C^6_{i,j} \cup C^4_{i,j+1} \subseteq L_1$ is empty of points of P.

We consider the four triangles obtained by removing $C^6_{i,j}$ from $\mathcal{T}^r_{i,j}$ and removing $C^4_{i,j+1}$ from $\mathcal{T}^l_{i,j+1}$, and we merge them with the cells $C_{i-1,j}$, $C_{i-1,j+1}$, $C_{i+1,j}$, and $C_{i+1,j+1}$ according to the following cases. We describe how to merge the top triangles with $C_{i-1,j}$ and $C_{i-1,j+1}$. (Merging the bottom triangles with $C_{i+1,j}$, and $C_{i+1,j+1}$ is done analogously.) Let v_t be the top vertex of L_1; see Figure 10.

- If both $C_{i-1,j}$ and $C_{i-1,j+1}$ are extended cells, then we merge the top-left triangle with $C_{i-1,j}$ and the top-right triangle with $C_{i-1,j+1}$; see Figure 10(a).
- If $C_{i-1,j}$ is a partitioned cell and $C_{i-1,j+1}$ is an extended cell, then we have another horizontal lune L_2 between $C_{i-1,j}$ and $C_{i,j}$; see Figure 10(b). Notice that the union of the top-left triangle of L_1 and the right-bottom triangle of L_2 is exactly the sub-cell $C^3_{i,j}$.

Notice also that L_2 is monochromatic and has the same color as L_1. Thus, the points of P in $C^3_{i,j}$ are of distance 1 from v_t. In this case, we merge the top-right triangle of L_1 and the right-top triangle of L_2 with $C_{i-1,j+1}$. Moreover, we merge the region of $C^3_{i,j}$ intersecting the disk of radius 1 centered at v_t with $C_{i-1,j+1}$; see Figure 10(b).

- If $C_{i-1,j}$ is an extended cell and $C_{i-1,j+1}$ is a partitioned cell, then this case is symmetric to the previous case; see Figure 10(c).
- If both $C_{i-1,j}$ and $C_{i-1,j+1}$ are partitioned cells, then we have four lunes incident to v_t; see Figure 10(d). Since all of the lunes are monochromatic and have the same color, the triangles of these lunes that are incident to v_t are empty of points of P and, therefore, we remove these triangles from the division.

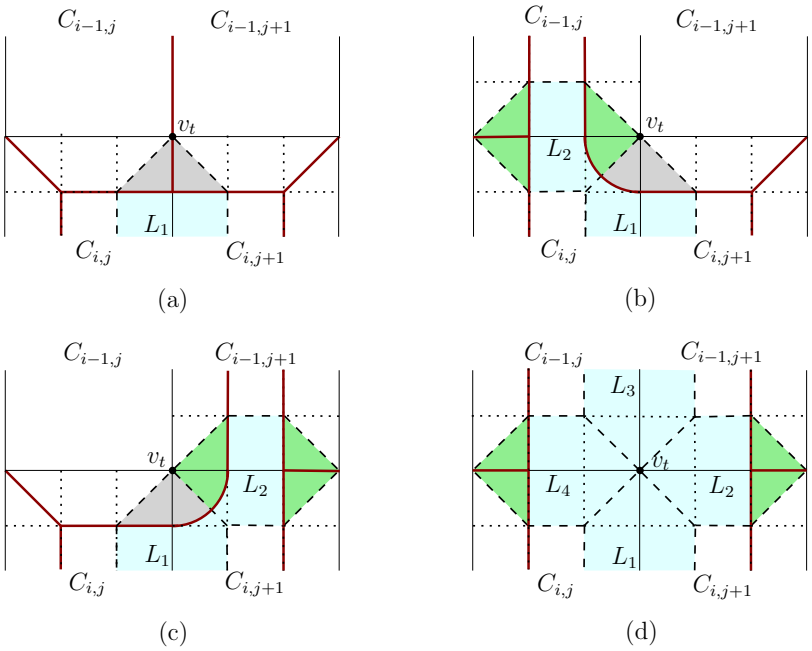


Figure 10 Merging the top triangles of L_1 with the cells $C_{i-1,j}$ and $C_{i-1,j+1}$. The gray and the green regions are part of vertical and horizontal lunes, respectively, and the light blue regions are empty of points of P.

Moreover, in each partitioned cell $C_{i,j}$ such that $i = 1$, $i = n$, $j = 1$, or $j = n$, we treat the trapezoids adjacent to the boundary of the grid as half-lunes and we merge them with their adjacent extended cells as in the lunes case.

Notice that, at the end of this stage, we have two types of non-empty cells: original 3×3 cells and extended cells, and both types are convex and bichromatic cells; see Figure 11. From now on, we refer to both types of these cells as extended cells and denote them by $\hat{C}$. That is, $\hat{C}_{i,j}$ is either an original 3×3 cell $C_{i,j}$ or an extended cell obtained by merging $C_{i,j}$ with trapezoids from its neighbors.

Stage 2

In this stage, we construct a planar bichromatic spanning tree in each (extended) cell and connect them to each other to obtain, overall, a planar bichromatic spanning tree of P. For each cell $\hat{C}_{i,j}$, we denote by $\hat{P}_{i,j}$ the set of points of P lying in $\hat{C}_{i,j}$. If $C_{i,j}$ has been partitioned, then we set $\hat{P}_{i,j} = \emptyset$.

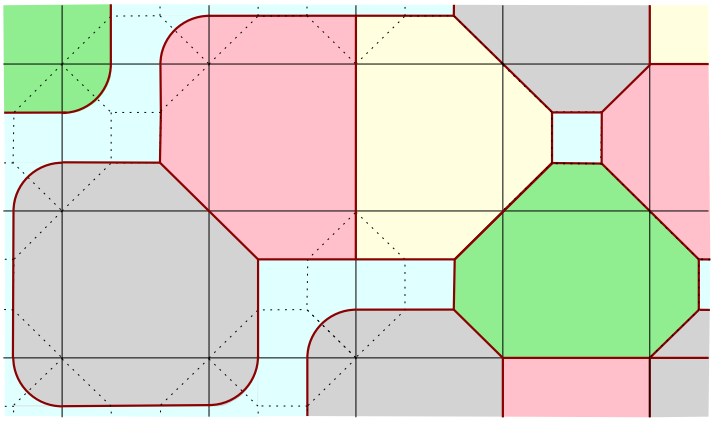


Figure 11 A subdivision obtained at the end of Stage 1. The light blue regions are empty of points of P.

Stage 2.1

In each cell $\hat{C}_{i,j}$, we construct a planar bichromatic spanning tree $T_{i,j}$ of $\hat{P}_{i,j}$ as follows. We select an arbitrary red point $s \in \hat{P}_{i,j}$ as a center of the tree and connect it to each blue point in the cell to produce a star. We extend the edges of the star to partition the cell into convex cones, possibly except one cone; see Figure 12. If we have a non-convex cone, then we divide it into two convex cones by adding its bisector, as shown in Figure 12(right). Then, we connect all the red points in each cone to one of the blue points on the lines bounding the cone.

Figure 12 Constructing a planar bichromatic spanning tree in a cell.

▶ **Lemma 3.** *Let $T_{i,j}$ be a tree constructed in Stage 2.1 in cell $\hat{C}_{i,j}$. Any (red or blue) point p in the plane can be connected to a point of opposite color of $T_{i,j}$ without crossing the edges of $T_{i,j}$.*

Proof. Let s be the center of $T_{i,j}$ and recall that its color is red. Consider the cones produced by the rays between s and the blue points of $T_{i,j}$. Let C be the cone containing p and let a and b be the two blue points defining C. By the way we constructed $T_{i,j}$, all the points in C are red and connected to exactly one of the points a and b, assume, w.l.o.g., a. We distinguish between two cases with respect to the color of p.

Case 1: p is a blue point. If the edge (s, p) does not cross the edges of $T_{i,j}$, then we connect p to s. Otherwise, we connect p to the endpoint of the first edge (from p) crossing (s, p); see Figure 13(a).

Case 2: p is a red point. In this case, we connect p to a; see Figure 13(b). ◀

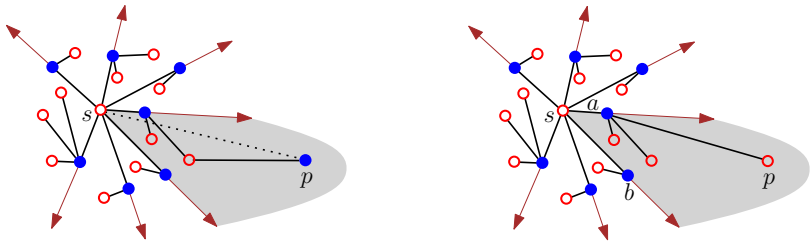


Figure 13 (a) p and s are of different colors. (b) p and s are of the same color.

Stage 2.2

In this stage, we connect between the trees that are constructed in Stage 2.1 to obtain a planar bichromatic spanning tree of P. Let $\hat{C}_{i,j}$ and $\hat{C}_{k,l}$ be two (extended) cells. We say that $\hat{C}_{k,l}$ is a *side* adjacent (or s-adjacent for short) cell of $\hat{C}_{i,j}$, if one of the following holds:

- $k = i + 1$ and $l = j$, or
- $k = i$ and $l = j + 1$,

and we say that $\hat{C}_{k,l}$ is a *diagonal* adjacent (or d-adjacent for short) cell of $\hat{C}_{i,j}$, if one of the following holds:

- $k = i - 1$, $l = j + 1$, and $C_{i-1,j}$ and $C_{i,j+1}$ have been partitioned, or
- $k = i - 1$, $l = j - 1$, and $C_{i,j-1}$ and $C_{i-1,j}$ have been partitioned.

We construct a bichromatic spanning tree T' of P by traversing the cells starting from an arbitrary (non-empty) cell (in breadth first search (BFS) manner). That is, we first initiate a tree T' by an arbitrary tree $T_{i,j}$ that is constructed in a cell $\hat{C}_{i,j}$. Then, we connect T' to all the trees constructed in the cells adjacent to $\hat{C}_{i,j}$, and proceed from these trees. More precisely, in each step, we consider a tree $T_{i,j}$, which is already connected to T', and we connect T' to all of the trees constructed in the cells adjacent to $\hat{C}_{i,j}$ via $T_{i,j}$ (if they are not connected yet to T'). In the following, we describe how to connect T' to all the trees constructed in the cells adjacent to $\hat{C}_{i,j}$.

Let $\hat{C}$ be a cell adjacent to $\hat{C}_{i,j}$, such that the tree T_C constructed in $\hat{C}$ is not connected yet to T'. Let v_{tr}, v_{tl}, v_{br}, and v_{bl} be the top-right, top-left, bottom-right, and bottom-left vertices of the grid incident to $C_{i,j}$, respectively; see Figure 14. We distinguish between two cases.

Case 1: $\hat{C}$ is a d-adjacent cell of $\hat{C}_{i,j}$. Assume, w.l.o.g., that $\hat{C} = \hat{C}_{i-1,j+1}$. Then, the boundaries of $\hat{C}_{i,j}$ and $\hat{C}_{i-1,j+1}$ share a common (diagonal) edge $\overline{ab}$; see Figure 14. Moreover, the convex hull of $\hat{C}_{i,j} \cup \hat{C}_{i-1,j+1}$ does not contain any point of $P \setminus (\hat{P}_{i,j} \cup \hat{P}_{i-1,j+1})$ (this can be seen clearly in Figure 11). Let $p \in \hat{P}_{i,j}$ be the closest point to the line passing through $\overline{ab}$, such that no edge of T' crosses the triangle Δpab. By Claim 4, such a point p exists. Then, any edge connecting p to any point of $T_{i-1,j+1}$ does not cross any non-empty cell except $\hat{C}_{i,j}$ and $\hat{C}_{i-1,j+1}$. Therefore, by Lemma 3, we can connect p to $T_{i-1,j+1}$ without crossing any other edge of T'.

Case 2: $\hat{C}$ is an s-adjacent cell of $\hat{C}_{i,j}$. Assume, w.l.o.g., that $\hat{C} = \hat{C}_{i,j+1}$. Let p be the rightmost point in $\hat{P}_{i,j}$, such that no edge of T' crosses the triangle $\Delta p v_{br} v_{tr}$; see Figure 15. By Claim 4, such a point p exists. Let H be the convex hull of $\hat{P}_{i,j+1} \cup \{p\}$. We consider two sub-cases.

Case 2.1: $H \cap (P \setminus (\hat{P}_{i,j} \cup \hat{P}_{i,j+1})) = \emptyset$ (i.e., H does not contain any point of P that is not in $\hat{P}_{i,j} \cup \hat{P}_{i,j+1}$); see Figure 15. Therefore, by Lemma 3, we can connect $T_{i,j+1}$ to $T_{i,j}$ via p, without crossing any other edge of T'.

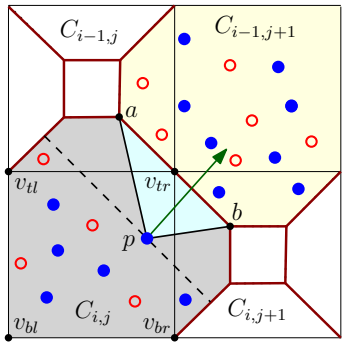


Figure 14 Illustration of Case 1. We connect $T_{i-1,j+1}$ to $T_{i,j}$ via p.

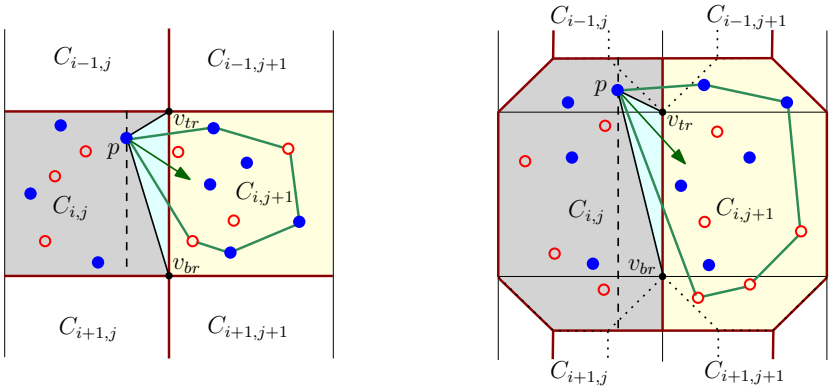


Figure 15 The convex hull H of $\hat{P}_{i,j+1} \cup \{p\}$ (consisting of green segments) does not contain any point of P that is not in $\hat{P}_{i,j} \cup \hat{P}_{i,j+1}$. We connect $T_{i,j+1}$ to $T_{i,j}$ via p.

Case 2.2: $H \cap (P \setminus (\hat{P}_{i,j} \cup \hat{P}_{i,j+1})) \neq \emptyset$ (i.e., H contains a point of P that is not in $\hat{P}_{i,j} \cup \hat{P}_{i,j+1}$). In this case, H contains a point in $\hat{P}_{i-1,j} \cup \hat{P}_{i-1,j+1}$ or in $\hat{P}_{i+1,j} \cup \hat{P}_{i+1,j+1}$. Assume, w.l.o.g., that H contains a point in $\hat{P}_{i-1,j} \cup \hat{P}_{i-1,j+1}$; see Figure 16. Notice that exactly one of the sets $\hat{P}_{i-1,j}$ or $\hat{P}_{i-1,j+1}$ is an empty set, since, in this case, exactly one of the cells $C_{i-1,j}$ or $C_{i-1,j+1}$ has been partitioned. We further distinguish between two cases.

1. $H \cap \hat{P}_{i-1,j} \neq \emptyset$; see Figure 16(a). In this case we first connect $T_{i,j+1}$ to $T_{i-1,j}$ as follows. Let $q \in \hat{P}_{i-1,j}$ be the closest point to the line passing through the boundary edge between $\hat{C}_{i-1,j}$ and $\hat{C}_{i,j+1}$. Then, the convex hull of $\hat{P}_{i,j+1} \cup \{q\}$ does not contain any point of P that is not in $\hat{P}_{i,j+1} \cup \{q\}$. Therefore, by Lemma 3, we can connect $T_{i,j+1}$ to $T_{i-1,j}$ via q, without crossing any other edge of T'.
   Moreover, if $T_{i-1,j}$ is not connected yet to T', then we apply Case 2 on $\hat{C}_{i-1,j}$ to connect $T_{i-1,j}$ to $T_{i,j}$.
2. $H \cap \hat{P}_{i-1,j+1} \neq \emptyset$. If $T_{i-1,j+1}$ is not connected yet to T', then we first connect $T_{i-1,j+1}$ to $T_{i,j}$ as follows. Let $q \in \hat{P}_{i,j}$ be the closest point to the line passing through the boundary edge between $\hat{C}_{i,j}$ and $\hat{C}_{i-1,j+1}$; see Figure 16(b). Then, the convex hull of $\hat{P}_{i-1,j+1} \cup \{q\}$ does not contain any point of P that is not in $\hat{P}_{i-1,j+1} \cup \{q\}$. Therefore, by Lemma 3, we can connect $T_{i-1,j+1}$ to $T_{i,j}$ via q, without crossing any other edge of T'.
   Moreover, we connect $T_{i,j+1}$ to $T_{i-1,j+1}$ as follows. Let a be the bottom-right corner of $C_{i-1,j+1}$. Thus, $\Delta p v_{tr} a \cap \hat{P}_{i-1,j+1} \neq \emptyset$ and $\Delta p v_{tr} a \cap (\hat{P}_{i,j} \setminus \{p\}) = \emptyset$. Let z be the bottommost point in $\Delta p v_{tr} a \cap \hat{P}_{i-1,j+1}$, such that no edge of T' crosses the triangle

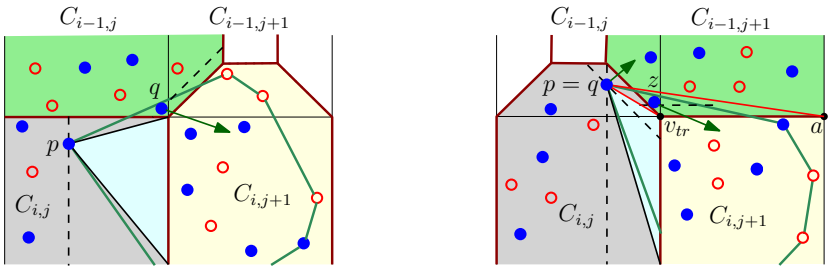


Figure 16 (a) H contains points from $\hat{P}_{i-1,j}$. We connect $T_{i,j+1}$ to $T_{i-1,j}$ via q. (b) H contains a point z from $\hat{P}_{i-1,j+1}$. We connect $T_{i-1,j+1}$ to $T_{i,j}$ via q and we connect $T_{i,j+1}$ to $T_{i-1,j+1}$ via z.

$\Delta z v_{tr} a$; see Figure 16(b). By Claim 4, such a point z exists. Then, the convex hull of $\hat{P}_{i,j+1} \cup \{z\}$ does not contain any point of P that is not in $\hat{P}_{i,j+1} \cup \{z\}$. Therefore, by Lemma 3, we can connect $T_{i,j+1}$ to $T_{i-1,j+1}$ via z, without crossing any other edge of T'. (Notice that the two edges added in this case do not cross each other.)

Correctness Proof

Recall that T is a bichromatic spanning tree of P of minimum bottleneck λ. In this section, we prove that T' is a planar bichromatic spanning tree of P of bottleneck at most $8\sqrt{2}\lambda$. Notice that every point $p \in P$ is contained in a bichromatic cell $\hat{C}_{i,j}$ and hence, it is connected to $T_{i,j}$, the tree constructed in Stage 2.1 in $\hat{C}_{i,j}$. Therefore, to show that T' is a bichromatic spanning tree of P, it is sufficient to show that each tree $T_{i,j}$ is connected to T'.

▷ Claim 4. Let T' be the tree constructed at some step during Stage 2.2 and assume that T' is planar. Let $T_{i,j}$ be a tree constructed in $\hat{C}_{i,j}$ and assume that $T_{i,j}$ is already connected to T'. Let $\hat{C}$ be an adjacent cell of $\hat{C}_{i,j}$ that shares an edge $\overline{ab}$ with $\hat{C}_{i,j}$ and let T_C be the tree constructed in $\hat{C}$, and assume that T_C is not connected yet to T'. Then, there exists a point p in $\hat{C}_{i,j}$, such that no edge of T' crosses the triangle Δpab.

Proof. Assume, w.l.o.g., that $\hat{C} = \hat{C}_{i,j+1}$, $a = v_{tr}$, and $b = v_{br}$; see Figure 17. The following procedure shows the existence of such a point p. We sweep leftwards in $\hat{C}_{i,j}$ with a vertical line l, starting from $\overline{v_{tr}v_{br}}$ until we meet a point, or an edge of T'. If we first meet a point, then this point satisfies the claim. Otherwise, we first meet an edge (p', q') of T'; see Figure 17. This could only be when exactly one of the endpoints p' or q' is outside $\hat{C}_{i,j}$. Let C_l and C_r be the two sub-cells obtained by partitioning $\hat{C}_{i,j}$ with the line that goes through the points p' and q'. Let C_r be the sub-cell containing v_{tr} and v_{br}. We keep sweeping leftwards only inside C_r. As before, if we first meet a point, then this point satisfies the claim. Otherwise, we meet an edge (p'', q'') of T' before we meet a point. Then, one of the endpoints p'' or q'' is outside $\hat{C}_{i,j}$. Let C_{rl} and C_{rr} be the two sub-cells obtained by partitioning C_r with the line that goes through the points p'' and q''. Let C_{rr} be the sub-cell containing v_{tr} and v_{br}. We keep sweeping leftwards only inside C_{rr}, until we meet a point, and this point satisfies the claim. Notice that, in the last sweep we meet a point before we meet an edge of T'. This follows from the planarity of T'. ◁

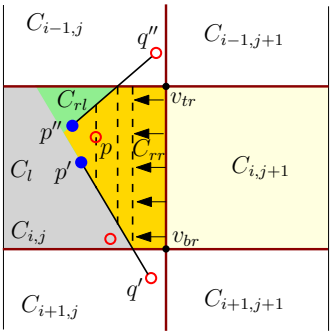


Figure 17 Illustration of the proof of Claim 4.

▶ **Lemma 5.** *Let $T_{i,j}$ be a tree constructed in $\hat{C}_{i,j}$ in Stage 2.1 and assume that $T_{i,j}$ is already connected to T'. Then, at the end of Stage 2.2, all the trees that are constructed in the cells adjacent to $\hat{C}_{i,j}$ are connected to T' as well.*

Proof. Let $\hat{C}$ be an adjacent cell of $\hat{C}_{i,j}$. We distinguish between two cases.

Case 1: $\hat{C}$ is a d-adjacent cell of $\hat{C}_{i,j}$. Assume, w.l.o.g., that $\hat{C} = \hat{C}_{i-1,j+1}$. Then, in Stage 2.2, Case 1, we connect between $T_{i,j}$ and $T_{i-1,j+1}$.

Case 2: $\hat{C}$ is an s-adjacent cell of $\hat{C}_{i,j}$. Assume, w.l.o.g., that $\hat{C} = \hat{C}_{i,j+1}$. As described in Stage 2.2, we select a point $p \in \hat{C}_{i,j}$ and compute the convex hull H of $\{p\} \cup \hat{P}_{i,j+1}$. Then, we consider two cases. In Case 2.1, when H does not contain any point of $P \setminus (\hat{P}_{i,j} \cup \hat{P}_{i,j+1})$, we connect $T_{i,j}$ directly to $T_{i,j+1}$ (via p). And, in Case 2.2, when H contains a point of $P \setminus (\hat{P}_{i,j} \cup \hat{P}_{i,j+1})$, we connect $T_{i,j+1}$ to $T_{i,j}$ via the tree $T_{i-1,j+1}$, in case that H contains a point of $\hat{P}_{i-1,j+1}$ (or via the tree $T_{i+1,j+1}$, in case that H contains a point of $\hat{P}_{i+1,j+1}$). In the case that H contains a point of $\hat{P}_{i-1,j}$ (symmetrically, H contains a point of $\hat{P}_{i+1,j}$), we connect $T_{i,j+1}$ to $T_{i,j}$ via the tree $T_{i-1,j}$. If $T_{i-1,j}$ is already connected to $T_{i,j}$, then we are done. Otherwise, since $\hat{C}_{i-1,j}$ is an s-adjacent cell of $\hat{C}_{i,j}$, we will try to connect $T_{i-1,j}$ to $T_{i,j}$ in the next iteration in Stage 2.2 (by applying Case 2 once again). In the next iteration, either we connect $T_{i-1,j}$ to $T_{i,j}$ in one of the cases described above or we end up by connecting $T_{i,j-1}$ to $T_{i-1,j}$. In the latter case, if $T_{i,j-1}$ is already connected to $T_{i,j}$, then we are done. otherwise, $T_{i+1,j}$ is already connected to $T_{i,j}$. In this case, we connect $T_{i,j-1}$ to $T_{i,j}$ either directly or via $T_{i+1,j}$, and we are done. ◀

▶ **Lemma 6.** *Let p and q be two points of P, such that p and q are of different colors, $|pq| \leq \lambda$ and p belongs to T'. Then, q also belongs to T'.*

Proof. Since $|pq| \leq \lambda$, either p and q are in the same cell or they are in adjacent cells. If they are in the same cell $\hat{C}_{i,j}$, then, after Stage 2.1, they are connected in $T_{i,j}$, and the lemma holds. Otherwise, assume, w.l.o.g., that $p \in \hat{C}_{i,j}$ and $q \in \hat{C}$, where $\hat{C}$ is adjacent to $\hat{C}_{i,j}$. Then, after Stage 2.1, p belongs to $T_{i,j}$ and q belongs to T_C, the tree constructed in $\hat{C}$. Since $T_{i,j}$ is part of T' and $\hat{C}$ is adjacent to $\hat{C}_{i,j}$, by Lemma 5, $T_{i,j}$ is connected to T_C and therefore q belongs to T'. ◀

▶ **Lemma 7.** *Let $T_{i,j}$ be a tree constructed in $\hat{C}_{i,j}$. Then, $T_{i,j}$ is connected to T'.*

Proof. Assume by contradiction that $T_{i,j}$ is not connected to T'. Let a be a point from T' and let b be a point from $T_{i,j}$. Since T is a bottleneck bichromatic spanning tree of P, there is a path Π between a and b in T. Let p be the last point (from a) on Π that belongs to T', i.e., no point of T' appears on the sub-path of Π between p and b. Since b does not belong

to T', such a point p exists. Let q be the point between p and b on Π that is connected to p. By the selection of p, q does not belong to T'. Since the bottleneck of T is λ, we have $|pq| \leq \lambda$. Therefore, by Lemma 6, p and q are connected in T', which contradicts that q does not belong to T'. ◀

▶ **Lemma 8.** *T' is planar.*

Proof. Each $T_{i,j}$ is planar. We start with $T' = T_{i,j}$, where $T_{i,j}$ is an arbitrary tree constructed in $\hat{C}_{i,j}$, and in each step, we extend T' by connecting it to the trees corresponding to the cells adjacent to the current cell. We connect T' to a "new" tree $T_{i,j}$ by picking a point p in T', such that the convex hull H of $\{p\} \cup \hat{P}_{i,j}$ is empty of any other points and no edge of T' crosses H. In Claim 4, we showed that such a point p always exists. Thus, connecting p to any point of $T_{i,j}$ will not cross any other edge of T' nor of any other tree. Moreover, in Lemma 3, we show that we can always connect p to $T_{i,j}$ without crossing any of the edges of $T_{i,j}$. Therefore, connecting T' to $T_{i,j}$ does not produce any crossing. ◀

▶ **Lemma 9.** *The bottleneck of T' is at most $8\sqrt{2}\lambda$.*

Proof. Consider Figure 11. After Stage 1, each extended cell is contained in a square of size $5\lambda \times 5\lambda$, and hence the bottleneck of each tree constructed in Stage 2.1 is at most $5\sqrt{2}\lambda$. Moreover, every two d-adjacent cells are contained in a square of size $8\lambda \times 8\lambda$ and every two s-adjacent cells are contained in a square of size $8\lambda \times 5\lambda$. Thus, each edge added in Stage 2.2 is of length at most $8\sqrt{2}\lambda$. Therefore, each edge in T' is of length at most $8\sqrt{2}\lambda$. ◀

The algorithm consists of two main stages, and each one of them can be implemented in polynomial time. Therefore, the total running time of the algorithm is polynomial. The following theorem summarizes the result of this section.

▶ **Theorem 10.** *Let P be a set of n red and blue points in the plane. One can compute in polynomial time a planar bichromatic spanning tree of P of bottleneck at most $8\sqrt{2}$ times the bottleneck of an optimal bichromatic spanning tree of P.*

References

1. M. Abellanas, J. Garcia-Lopez, G. Hernández-Peñalver, M. Noy, and P. A. Ramos. Bipartite embeddings of trees in the plane. *Discr. Appl. Math.*, 93(2-3):141–148, 1999.
2. A. K. Abu-Affash, A. Biniaz, P. Carmi, A. Maheshwari, and M. Smid. Approximating the bottleneck plane perfect matching of a point set. *Comput. Geom.*, 48(9):718–731, 2015.
3. A. K. Abu-Affash, P. Carmi, M. J. Katz, and Y. Trabelsi. Bottleneck non-crossing matching in the plane. *Comput. Geom.*, 47(3):447–457, 2014.
4. P. K. Agarwal. Partitioning arrangements of lines II: Applications. *Discr. Comput. Geom.*, 5(1):533–573, 1990.
5. P. K. Agarwal, H. Edelsbrunner, and O. Schwarzkopf. Euclidean minimum spanning trees and bichromatic closest pairs. *Discr. Comput. Geom.*, 6(1):407–422, 1991.
6. O. Aichholzer, S. Cabello, R. Fabila-Monroy, D. Flores-Peñaloza, T. Hackl, C. Huemer, F. Hurtado, and D. R. Wood. Edge-removal and non-crossing configurations in geometric graphs. In *EuroCG*, pages 119–122, 2008.
7. N. Alon, S. Rajagopalan, and S. Suri. Long non-crossing configurations in the plane. In *SoCG*, pages 257–263, 1993.
8. G. Aloupis, J. Cardinal, S. Collette, E. D. Demaine, M. L. Demaine, M. Dulieu, R. Fabila-Monroy, V. Hart, F. Hurtado, S. Langerman, M. Saumell, C. Seara, and P. Taslakian. Matching points with things. In *LATIN, volume 6034 of LNCS*, pages 456–467, 2010.

9 S. Arora and K. L. Chang. Approximation schemes for degree-restricted MST and red–blue separation problems. *Algorithmica*, 40(3):189–210, 2004.

10 M. J. Atallah and D. Z. Chen. On connecting red and blue rectilinear polygonal obstacles with nonintersecting monotone rectilinear paths. *Int. J. Comput. Geom. Appl.*, 11(4):373–400, 2001.

11 A. Biniaz, P. Bose, K. Crosbie, J.-L. De Carufel, D. Eppstein, A. Maheshwari, and M. H. M. Smid. Maximum plane trees in multipartite geometric graphs. *Algorithmica*, 81(4):1512–1534, 2019.

12 A. Biniaz, P. Bose, D. Eppstein, A. Maheshwari, P. Morin, and M. H. M. Smid. Spanning trees in multipartite geometric graphs. *Algorithmica*, 80(11):3177–3191, 2018.

13 A. Biniaz, A. Maheshwari, and M. Smid. Bottleneck bichromatic plane matching of points. In *CCCG*, 2014.

14 J.-D. Boissonnat, J. Czyzowicz, O. Devillers, J. Urrutia, and M. Yvinec. Computing largest circles separating two sets of segments. *Int. J. Comput. Geom. Appl.*, 10(1):41–53, 2000.

15 M. G. Borgelt, M. Van Kreveld, M. Löffler, J. Luo, D. Merrick, R. I. Silveira, and M. Vahedi. Planar bichromatic minimum spanning trees. *J. Discrete Algorithms*, 7(4):469–478, 2009.

16 H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990. doi:10.1007/BF02122694.

17 E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, H. Meijer, M. H. Overmars, and S. Whitesides. Separating point sets in polygonal environments. *Int. J. Comput. Geom. Appl.*, 15(4):403–419, 2005.

18 A. Dumitrescu and R. Kaye. Matching colored points in the plane: some new results. *Comput. Geom.*, 19(1):69–85, 2001.

19 A. Dumitrescu and J. Pach. Partitioning colored point sets into monochromatic parts. *Int. J. Comput. Geom. Appl.*, 12(05):401–412, 2002.

20 H. Everett, J.-M. Robert, and M. J. van Kreveld. An optimal algorithm for the (<= k)-levels, with applications to separation and transversal problems. *Int. J. Comput. Geom. Appl.*, 6(3):247–261, 1996.

21 A. Kaneko and M. Kano. Discrete geometry on red and blue points in the plane – a survey. *Discr. Comput. Geom.*, 25:551–570, 2003.

22 D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

23 H. G. Mairson and J. Stolfi. Reporting and counting intersections between two sets of line segments. In *Theoretical Foundations of Computer Graphics and CAD*, volume 40 of *NATO ASI Series*, pages 307–325, 1988.

Parallel Batch-Dynamic Trees via Change Propagation

Umut A. Acar
Carnegie Mellon University, Pittsburgh, PA, USA
umut@cs.cmu.edu

Daniel Anderson
Carnegie Mellon University, Pittsburgh, PA, USA
dlanders@cs.cmu.edu

Guy E. Blelloch
Carnegie Mellon University, Pittsburgh, PA, USA
guyb@cs.cmu.edu

Laxman Dhulipala
Carnegie Mellon University, Pittsburgh, PA, USA
ldhulipa@cs.cmu.edu

Sam Westrick
Carnegie Mellon University, Pittsburgh, PA, USA
swestric@cs.cmu.edu

Abstract

The dynamic trees problem is to maintain a forest subject to edge insertions and deletions while facilitating queries such as connectivity, path weights, and subtree weights. Dynamic trees are a fundamental building block of a large number of graph algorithms. Although traditionally studied in the single-update setting, dynamic algorithms capable of supporting batches of updates are increasingly relevant today due to the emergence of rapidly evolving dynamic datasets. Since processing updates on a single processor is often unrealistic for large batches of updates, designing parallel batch-dynamic algorithms that achieve provably low span is important for many applications.

In this work, we design the first work-efficient parallel batch-dynamic algorithm for dynamic trees that is capable of supporting both path queries and subtree queries, as well as a variety of nonlocal queries. Previous work-efficient dynamic trees of Tseng et al. were only capable of handling subtree queries [*ALENEX'19*, (2019), pp. 92–106]. To achieve this, we propose a framework for algorithmically dynamizing static round-synchronous algorithms to obtain parallel batch-dynamic algorithms. In our framework, the algorithm designer can apply the technique to any suitably defined static algorithm. We then obtain theoretical guarantees for algorithms in our framework by defining the notion of a computation distance between two executions of the underlying algorithm.

Our dynamic trees algorithm is obtained by applying our dynamization framework to the parallel tree contraction algorithm of Miller and Reif [*FOCS'85*, (1985), pp. 478–489], and then performing a novel analysis of the computation distance of this algorithm under batch updates. We show that k updates can be performed in $O(k \log(1 + n/k))$ work in expectation, which matches the algorithm of Tseng et al. while providing support for a substantially larger number of queries and applications.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Shared memory algorithms

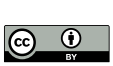
Keywords and phrases Dynamic trees, Graph algorithms, Parallel algorithms, Dynamic algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.2

Related Version A full version is available at [2], `https://arxiv.org/abs/2002.05129`.

Funding This work was supported by NSF grants CCF-1901381, CCF-1910030 and CCF-1919223.

Acknowledgements The authors would like to thank Ticha Sethapakdi for helping with the figures.



1 Introduction

The dynamic trees problem, first posed by Sleator and Tarjan [27] is to maintain a forest of trees subject to the insertion and deletion of edges, also known as *links* and *cuts*. Dynamic trees are used as a building block in a multitude of applications, including maximum flows [27], dynamic connectivity and minimum spanning trees [11], and minimum cuts [19], making them a fruitful line of work with a rich history. There are a number of established sequential dynamic tree algorithms, including link-cut trees [27], top trees [28], Euler-tour trees [15], and rake-compress trees [5], all of which achieve $O(\log(n))$ time per operation.

Since they already perform such little work, there is often little to gain by processing single updates in parallel, hence parallel applications often process *batches* of updates. We are therefore concerned with the design of parallel *batch-dynamic* algorithms. Parallel batch-dynamic algorithms have been developed for several graph problems including incremental connectivity [26], Euler-Tour trees [29], and for fully dynamic connectivity [1]. Parallel batch-dynamic algorithms have also been recently studied in the MPC model [16, 10].

By applying batches it is often possible to obtain significant parallelism while preserving work efficiency. However, designing and implementing dynamic algorithms is difficult, and arguably even more so in the parallel setting.

The goals of this paper are twofold. First and foremost, we are interested in designing a parallel batch-dynamic algorithm for dynamic trees that supports a wide range of applications. On another level, based on the observation that parallel dynamic algorithms are usually quite complex and difficult to design, we are also interested in easing the design process of parallel batch-dynamic algorithms as a whole. To this end, we propose a framework for algorithmically dynamizing static parallel algorithms to obtain efficient parallel batch-dynamic algorithms. We then define a cost model that captures the *computation distance* between two executions of the static algorithm which allows us to bound the runtime of dynamic updates. There are several benefits of using algorithmic dynamization, some more theoretical some practical:

1. Proving correctness of a batch dynamized algorithm relies simply on the correctness of the parallel algorithm, which presumably has already been proven.
2. It is easy to implement different classes of updates. For example, for dynamic trees, in addition to links and cuts, it is very easy to update edge weights or vertex weights for supporting queries such as path length, subtree sums, or weighted diameter. One need only change the values of the weights and propagate.
3. Due to the simplicity of our approach, we believe it is likely to make it easier to program parallel batch-dynamic algorithms, and also result in practical implementations.

Using our algorithmic dynamization framework, we obtain a parallel batch-dynamic algorithm for rake-compress trees that generalizes the sequential data structure work efficiently without loss of generality. Specifically, our main contribution is the following theorem.

▶ **Theorem 1.** *The following operations can be supported on a bounded-degree dynamic tree of size n using the CRCW* PRAM*:*

- *Batch insertions and deletions of k edges in $O(k \log(1 + n/k))$ work in expectation and $O(\log(n) \log^*(n))$ span w.h.p.*
- *Batch connectivity, subtree sum, and path sum queries for batches of size k in $O(k \log(1 + n/k))$ work in expectation and $O(\log(n))$ span w.h.p.*
- *Independent parallel connectivity, subtree sum, path sum, diameter, lowest common ancestor, center, and median queries in $O(\log n)$ time per query w.h.p.*

Arbitrary-degree trees can be handled by transforming them into bounded degree trees using known techniques. We compare the capabilities of parallel rake-compress trees with other dynamic tree algorithms in Table 1. In summary, they support more operations than existing

Table 1 The known capabilities of various dynamic tree algorithms. Nonlocal queries are operations such as computing centers and medians. Our work extends rake-compress trees [5], which were previously only sequential, to also support parallel operations.

	Parallel Operations		Queries Supported		
	Updates	Queries	Path	Subtree	Nonlocal
Link-cut trees [27]			✓		
(Parallel) Euler-tour trees [15, 29]	✓	✓		✓	
Top trees [28]		✓	✓	✓	✓
Rake-compress trees [5]		✓	✓	✓	✓
Parallel rake-compress trees (this paper)	✓	✓	✓	✓	✓

parallel data structures, and support the same broad set of operations as existing non-parallel data structures. Theorem 1 is obtained by dynamizing the parallel tree contraction algorithm of Miller and Reif [20] and performing a novel analysis of the computation distance.

Standalone algorithms for dynamic parallel tree contraction have previously been proposed, but are inefficient and not fully general. In particular, Reif and Tate [25] give an algorithm for parallel dynamic tree contraction that can process a batch of k leaf insertions or deletions in $O(k \log(n))$ work. Unlike our algorithm, theirs is not work efficient, as it performs $\Omega(n \log(n))$ work for batches of size $\Omega(n)$, and it can only modify the tree at the leaves.

Lastly, as some evidence of the applicability of algorithmic dynamization, in the full version of this paper [2], we demonstrate two other applications of the technique. Specifically, we consider map-reduce based computations, and dynamic sequences with splitting and joining. To summarize, the main contributions of this paper are:

1. An algorithmic framework for dynamizing round-synchronous parallel algorithms, and a cost model for analyzing the performance of algorithms resulting from the framework
2. An analysis of the computation distance of Miller and Reif's tree contraction algorithm under batch edge insertions and deletions, which shows that it can be efficiently dynamized
3. The first work-efficient parallel algorithm for batch-dynamic trees that supports subtree queries, path queries, and nonlocal queries such as centers and medians.

Technical overview

A *round-synchronous* algorithm consists of a sequence of rounds, where a round executes in parallel across a set of processes, and each process runs a sequencial *round computation* reading and writing from shared memory and doing local computation. The round synchronous model is similar to Valiant's well-known Bulk Synchronous Parallel (BSP) model [30], except that communication is done via shared memory. Algorithmic dynamization works by running the round-synchronous algorithm while tracking all write-read dependences – i.e., a dependence from a write in one round to a read in a later round. Then, whenever a batch of changes are made to the input, *change propagation* propagates the changes through the original computation, only rerunning round computations if the values they read have changed. We note that depending on the algorithm, changes to the input could drastically change the underlying computation, introducing new dependencies, or invalidating old ones. Part of the novelty of this paper is bounding the work and span of this update process.

The idea of change propagation has been applied in the sequential setting and used to generate efficient dynamic algorithms [3, 4]. The general idea of parallel change propagation has also been used in various practical systems [9, 14, 7, 23, 24] but none of them have been analyzed theoretically.

To capture the cost of running the change propagation algorithm for a particular parallel algorithm and class of updates we define a *computational distance* between two computations, which corresponds to the total work of the round computations that differ in the two computations. The *input configuration* for a computation consists of the input I, stored in shared memory, and an initial set of processes P. We show the following bounds, where the *work* is the sum of the time of all round computations, and *span* is the sum over rounds of the maximum time of any round computation in that round.

▶ **Theorem 2.** *Given a round-synchronous algorithm A that with input configuration (I, P) does W work in R rounds and S span, then, on the CRCW* PRAM*,*

1. *the initial run of the algorithm with tracking takes $O(W)$ work in expectation and $O(S + R \log W)$ time w.h.p.,*
2. *running change propagation from input configuration (I, P) to configuration (I', P') takes $O(W_\Delta + R')$ work in expectation and $O(S' + R' \log W')$ time w.h.p., where W_Δ is the computation distance between the two configurations, and S',R',W' are the maximum span, rounds and work for the two configurations.*

We show that the work can be reduced to $O(W_\Delta)$, and that the $\log W$ and $\log W'$ terms can be reduced to $\log^* W$ when the round-synchronous algorithms have certain restrictions that are satisfied by all of our example algorithms, including our main result on dynamic trees. We also present similar results in other parallel models of computation.

With our dynamization framework and cost model, we develop an algorithm for dynamic trees that support a broad set of queries including subtree sums, path queries, lowest common ancestors, diameter, center, and median queries. This significantly improves over previous work on batch-dynamic Euler tour trees [29], which only support subtree sums.

Our dynamic trees algorithm is a parallel version of the sequential rake-compress tree (RC tree) data structure. Previous work showed that in the sequential setting, one can generate an RC tree (or forest) as a byproduct of Miller and Reif's tree contraction process, which supports the wide collection of queries mentioned above, all in logarithmic time, w.h.p. [5]. Our approach generalizes this sequential algorithm to allow for batches of edge insertions or deletions, work efficiently in parallel. The challenge is in analyzing the computation distance incurred by batch updates in the parallel batch-dynamic setting. In Section 4 we do just that, and obtain the following result:

▶ **Theorem 3.** *In the round synchronous model, Miller and Reif's tree contraction algorithm does $O(n)$ work in expectation and has $O(\log n)$ rounds and span w.h.p. Furthermore, given forests T with n vertices, and T' with k modifications to the edge list of T, the computation distance of the algorithm on the two inputs is $O(k \log(1 + n/k))$ in expectation.*

The bounds can then be plugged into Theorem 2 to show that a set of k edges can be inserted or deleted in a batch in $O(k \log(1 + n/k))$ work in expectation and $O(\log^2 n)$ span w.h.p. We show that the span can be improved to $O(\log n \log^* n)$ w.h.p. on the CRCW PRAM model. The last step in obtaining our dynamic trees framework is to plug the dynamized tree contraction algorithm into the RC trees framework [5] (see Section 5).

2 Preliminaries

2.1 Parallel Models

The parallel random access machine (PRAM) model is a classic parallel model with p processors that work in lock-step, connected by a parallel shared-memory [17]. In this paper we primarily consider the Concurrent-Read Concurrent-Write model (CRCW PRAM), where

memory locations are allowed to be concurrently read and concurrently written to. If multiple writers write to the same location concurrently, we assume that an arbitrary writer wins. We analyze algorithms on the CRCW PRAM in terms of their *work* and *span*. The span (or parallel time) of an algorithm is the minimum running time achievable when arbitrarily many processors are available. The work is the product of the span and the number of processors.

The threaded random access machine (TRAM) is closely related to the PRAM, but more closely models current machines and programming paradigms [8]. In the binary forking TRAM (binary forking model for short), a process can *fork* another process to run in parallel, and can *join* to wait for all forked calls to complete. In the binary forking model, the *work* of an algorithm is the total number of instructions it performs, and the *span* is the longest chain of sequentially dependent instructions.

2.2 Parallel Primitives

The following parallel procedures are used throughout the paper. *Scan* takes as input an array A of length n, an associative binary operator $\oplus$, and an identity element $\perp$ such that $\perp \oplus x = x$ for any x, and returns the array $(\perp, \perp \oplus A[0], \perp \oplus A[0] \oplus A[1], \ldots, \perp \oplus_{i=0}^{n-2} A[i])$ as well as the overall sum, $\perp \oplus_{i=0}^{n-1} A[i]$. Scan takes $O(n)$ work and $O(\log n)$ span (assuming $\oplus$ takes $O(1)$ work) [17] on the CRCW PRAM, and in the binary forking model.

Filter takes an array A and a predicate f and returns a new array containing $a \in A$ for which $f(a)$ is true, in the same order as in A. Filter can be done in $O(n)$ work and $O(\log n)$ span on the CRCW PRAM (assuming f takes $O(1)$ work) [17], and in the binary forking model. The *Approximate Compaction* problem is similar to a Filter. It takes an array A and a predicate f and returns a new array containing $a \in A$ for which $f(a)$ is true where some of the entries in the returned array can have a null value. The total size of the returned array is at most a constant factor larger than the number of non-null elements. Gil et al. [12] describe a parallel approximate compaction algorithm that uses linear space and achieves $O(n)$ work and $O(\log^*(n))$ span w.h.p. on the CRCW PRAM.

A *semisort* takes an input array of elements, where each element has an associated key and reorders the elements so that elements with equal keys are contiguous. The purpose is to collect equal keys together, rather than sort them. Semisorting a sequence of length n can be performed in $O(n)$ expected work and $O(\log n)$ depth w.h.p. on the CRCW PRAM [13] and in the binary forking model [8], assuming access to a uniformly random hash function mapping keys to integers in the range $[1, n^{O(1)}]$.

3 Dynamization Framework

3.1 Round-synchronous algorithms

In this framework, we consider dynamizing algorithms that are *round synchronous*. The round synchronous framework encompasses a range of classic BSP [30] and PRAM algorithms. A round-synchronous algorithm consists of M *processes*, with process IDs bounded by $O(M)$. The algorithm performs sequential rounds in which each active process executes, in parallel, a *round computation*. At the end of a round, any processes can decide to *retire*, in which case they will no longer execute in any future round. The algorithm terminates once there are no remaining active processes – i.e., they have all retired. Given a fixed input, round-synchronous algorithms must perform deterministically. Note that this does not preclude us from implementing randomized algorithms (indeed, our dynamic trees algorithm is randomized), it just requires that we provide the source of randomness as an input to

the algorithm, so that its behavior is identical if re-executed. An algorithm in the round synchronous framework is defined in terms of a procedure ComputeRound(r, p), which performs the computation of process p in round r. The initial run of a round-synchronous algorithm must specify the set P of initial process IDs.

Memory model

Processes in a round-synchronous algorithm may read and write to local memory that is not persisted across rounds. They also have access to a *shared memory*. The input to a round-synchronous algorithm is the initial contents of the shared memory. Round computations can read and write to shared memory with the condition that writes do not become visible until the end of the round. Reads can only access shared locations that have been written to, and shared locations can only be written to once, hence concurrent writes are not permitted. The contents of the shared memory at termination is considered to be the algorithm's output. Change propagation is driven by tracking all reads and writes to shared memory.

Pseudocode

We describe round-synchronous algorithms using the following primitives:

1. The **read** instruction reads the given shared memory locations and returns their values,
2. The **write** instruction writes the given value to the given shared memory location.
3. Processes may retire by invoking the **retire process** instruction.

Measures

The following measures will help us to analyse the efficiency of round-synchronous algorithms. For convenience, we define the *input configuration* of a round-synchronous algorithm as the pair (I, P), where I is the input to the algorithm (i.e. the initial state of shared memory) and P is the set of initial process IDs.

▶ **Definition 4** (Initial work, Round complexity, and Span)**.** *The* initial work *of a round-synchronous algorithm on some input configuration (I, P) is the sum of the work performed by all of the computations of each processes over all rounds when given that input. Its* round complexity *is the number of rounds that it performs, and its* span *is the sum of the maximum costs per round of the computations performed by each process.*

3.2 Change propagation

Given a round-synchronous algorithm, a *dynamic update* consists of a change to the input configuration, i.e. changing the contents of shared memory, and/or adding or deleting processes. The initial run and change propagation algorithms maintain the following data:

1. $R_{r,p}$, the memory locations read by process p in round r
2. $W_{r,p}$, the memory locations written by process p in round r
3. S_m, the set of round, process pairs that read memory location m
4. $X_{r,p}$, which is **true** if process p retired in round r

Algorithm 1 depicts the procedure for executing the initial run of a round-synchronous algorithm before making any dynamic updates.

To help formalize change propagation, we define the notion of an *affected computation*. The task of change propagation is to identify the affected computations and rerun them.

Algorithm 1 Initial run.

```
1:  procedure RUN(P)
2:    local r ← 0
3:    while P ≠ ∅ do
4:      for each process p ∈ P do in parallel
5:        COMPUTEROUND(r, p)
6:        R_{r,p} ← {memory locations read by p in round r}
7:        W_{r,p} ← {memory locations written to by p in round r}
8:        X_{r,p} ← (true if p retired in round r else false)
9:      for each  m ∈ ∪_{p∈P} R_{r,p} do in parallel
10:       S_m ←  S_m ∪ {(r, p) | m ∈ R_{r,p} ∧  p ∈ P}
11:     P ← P \ {p ∈ P : X_{r,p} = true}
12:     r ← r + 1
```

▶ **Definition 5** (Affected computation)**.** *Given a round-synchronous algorithm A and two input configurations (I, P) and (I', P'), the* affected computations *are the round and process pairs (r, p) such that either:*

1. *process p runs in round r on one input configuration but not the other*
2. *process p runs in round r on both input configurations, but reads a variable from shared memory that has a different value in one configuration than the other*

The change propagation algorithm is depicted in Algorithm 2. It works by maintaining the affected computations as three disjoint sets, P, the set of processes that read a memory location that was rewritten, L, processes that outlived their previous self, i.e. that retired the last time they ran, but did not retire when re-executed, and D, processes that retired earlier than their previous self. First, at each round, the algorithm determines the set of computations that should become affected because of shared memory locations that were rewritten in the previous round (Lines 12–14). These are used to determine P, the set of affected computations to rerun this round (Line 15). To ensure correctness, the algorithm must then reset the reads that were performed by the computations that are no longer alive, or that will be reran, since the set of locations that they read may differ from last time (Lines 18–19). Lines 22–26 perform the re-execution of all processes that read a changed memory location, or that lived longer (did not retire) than in the previous configuration. The algorithm then subscribes the reads of these computations to the memory locations that they read (Lines 28–29). Finally, on Lines 32–36, the algorithm updates the set of changed memory locations (U), the set of computations that lived longer than their previous self (L) and the set of computations that retired earlier then their previous self (D).

3.3 Correctness

In this section, we sketch a proof of correctness of the change propagation algorithm (Algorithm 2). Intuitively, correctness is assured because of the write-once condition on global shared memory, which ensures that computations can not have their output overwritten, and hence do not need to be re-executed unless data that they depend on is modified.

▶ **Lemma 6.** *Given a dynamic update, re-executing only the affected computations for each round will result in the same output as re-executing all computations on the new input.*

Proof. Since by definition they read the same values, computations that are not affected, if re-executed, would produce the same output as they did the first time. Since all shared memory locations can only be written to once, values written by processes that are not

Algorithm 2 Change propagation.

```
1: // U = sequence of memory locations that have been modified
2: // P+ = sequence of new process IDs to create
3: // P− = sequence of process IDs to remove
4: procedure PROPAGATE(U, P+, P−)
5:   local D ← P−                                  // Processes that died earlier than before
6:   local L ← P+                                  // Processes that lived longer than before
7:   local A ← ∅                                   // Affected computations at each round
8:   local r ← 0
9:   while U ≠ ∅ ∨ D ≠ ∅ ∨ L ≠ ∅ ∨ ∃r' ≥ r : (A_r' ≠ ∅) do
10:    // Determine the computations that become affected
11:    // due to the newly updated memory locations U
12:    local A' ← ∪_{m∈U} S_m
13:    for each r' ∈ ∪_{(r',p)∈A'} {r'} do in parallel
14:      A_r' ← A_r' ∪ {p | (r', p) ∈ A'}
15:    local P ← A_r \ D                           // Processes to rerun
16:    // Forget the prior reads of all processes that are
17:    // now dead or will be rerun on this round
18:    for each m ∈ ∪_{p∈P∪D} R_{r,p} do in parallel
19:      S_m ← S_m \ {(r, p) | m ∈ R_{r,p} ∧ p ∈ P ∪ D}
20:    local X^prev = {p ↦ X_{r,p} | p ∈ P}
21:    // (Re)run all changed or newly live processes
22:    for each process p in P ∪ L do in parallel
23:      COMPUTEROUND(r, p)
24:      R_{r,p} ← {memory locations read by p in round r}
25:      W_{r,p} ← {memory locations written to by p in round r}
26:      X_{r,p} ← (true if p retired in round r else false)
27:    // Remember the reads performed by processes on this round
28:    for each m ∈ ∪_{p∈P∪L} R_{r,p} do in parallel
29:      S_m ← S_m ∪ {(r, p) | m ∈ R_{r,p} ∧ p ∈ P ∪ L}
30:    // Update the sets of changed memory locations,
31:    // newly live processes, and newly dead processes
32:    U ← ∪_{p∈(P∪L)} W_{r,p}
33:    L' ← {p ∈ P | X_p^prev = true ∧ X_{r,p} = false}
34:    L ← L ∪ L' \ {p ∈ L | X_{r,p} = true}
35:    D' ← {p ∈ P | X_p^prev = false ∧ X_{r,p} = true}
36:    D ← D ∪ D' \ {p ∈ D | X_p^prev = true}
37:    r ← r + 1
```

re-executed can not have been overwritten, and hence it is safe to not re-execute them, as their output is preserved. Therefore re-executing only the affected computations will produce the same output as re-executing all computations. ◀

▶ **Theorem 7** (Consistency). *Given a dynamic update, change propagation correctly updates the output of the algorithm.*

Proof sketch. Follows from Lemma 6 and the fact that all reads and writes to global shared memory are tracked in Algorithm 2, and since global shared memory is the only method by which processes communicate, all affected computations are identified. ◀

3.4 Cost analysis

To analyze the work of change propagation, we need to formalize a notion of *computation distance*. Intuitively, the computation distance between two computations is the work performed by one and not the other. We then show that change propagation can efficiently re-execute the affected computations in work proportional to the computation distance.

▶ **Definition 8** (Computation distance)**.** *Given a round-synchronous algorithm A and two input configurations, the* computation distance W_Δ *between them is the sum of the work performed by all of the affected computations with respect to both input configurations.*

▶ **Theorem 9.** *Given a round-synchronous algorithm A with input configuration (I, P) that does W work in R rounds and S span, then*

1. *the initial run of the algorithm with tracking takes $O(W)$ work in expectation and $O(S + R \cdot \log(W))$ span w.h.p.,*
2. *running change propagation on a dynamic update to the input configuration (I', P') takes $O(W_\Delta + R')$ work in expectation and $O(S' + R' \log(W'))$ span w.h.p., where S', R', W' are the maximum span, rounds, and work of the algorithm on the two input configurations,*

These bounds hold on the CRCW PRAM *and in the binary forking* TRAM *model.*

Proof. We begin by analyzing the initial run. By definition, all executions of the round computations, ComputeRound, take $O(W)$ work and $O(S)$ span in total, with at most an additional $O(\log(M)) = O(\log(W))$ span to perform the parallel for loop. We will show that all additional work can be charged to the round computations, and that at most an additional $O(\log(W))$ span overhead is incurred.

We observe that $R_{r,p}, W_{r,p}$ and $X_{r,p}$ are at most the size of the work performed by the corresponding computations, hence the cost of Lines 6 – 8 can be charged to the computation. The reader sets S_m can be implemented as dynamic arrays with lazy deletion (this will be discussed during change propagation). To append new elements to S_m (Line 10), we can use a semisort performing linear work in expectation to first bucket the shared memory locations in $\cup_{p \in P} R_{r,p}$, whose work can be charged to the corresponding computations that performed the reads. This adds an additional $O(\log(W))$ span w.h.p. since the number of reads is no more than W in total. Finally, removing retired computations from P (Line 11) requires a compaction operation. Since compaction takes linear work, it can be charged to the execution of the corresponding processes. The span of compaction is at most $O(\log(W))$.

Summing up, we showed that all additional work can be charged to the round computations, and the algorithm incurs at most $O(\log(W))$ additional span per round w.h.p. Hence the cost of the initial run is $O(W)$ work in expectation and $O(S + R \cdot \log(W))$ span w.h.p.

We now analyze the change propagation procedure (Algorithm 2). The core of the work is the re-execution of the affected readers on Line 23, which, by definition takes $O(W_\Delta)$ work, and $O(S')$ span, with at most $O(\log(W'))$ additional span to perform the parallel for loop. Since some rounds may have no affected computations, the algorithm could perform up to $O(R')$ additional work to process these rounds. We will show that all additional work can be charged to the affected computations, incurring at most an additional $O(\log(W'))$ span.

Lines 12 – 14 bucket the newly affected computations by round. This can be achieved with an expected linear work semisort and by maintaining the A_r sets as dynamic arrays. The work is chargeable to the affected computations and the span is at most $O(\log(W'))$ w.h.p. Computing the current set of affected computations (Line 15) requires a filter/compaction operation, whose work is charged to the affected computations and span is at most $O(\log(W'))$.

Updating the reader sets S_m (Line 19) can be done as follows. We maintain S_m as dynamic arrays with lazy deletion, meaning that we delete by marking the corresponding slot as empty. When more than half of the slots have been marked empty, we perform compaction,

whose work is charged to the updates and whose span is at most $O(\log(W'))$. In order to perform deletions in constant time, we augment the set $R_{r,p}$ so that it remembers, for each entry m, the location of (r,p) in S_m. Therefore these updates take constant amortized work each (using a dynamic array), charged to the corresponding affected computations, and at most $O(\log(W'))$ span if a resize/compaction is triggered.

X^{prev} can be implemented as an array of size $|P|$, with work charged to the affected computations in P. As in the initial run, the cost of updating $R_{r,p}, W_{r,p}$ and $X_{r,p}$ can also be charged to the work performed by the affected computations.

Updating the reader sets S_m (Line 29) is a matter of appending to dynamic arrays, and, as mentioned earlier, remembering for each $m \in R_{r,p}$, the location of (r,p) in S_m. The work can be charged to the affected computations, and the span is at most $O(\log(W'))$.

Collecting the updated locations U (Line 32) can similarly be charged to the affected computations, and incurs no more than $O(\log(W'))$ span. On Lines 33 – 36, the sets L' and D' are computed by a compaction over P, whose work is charged to the affected computations in P. Updating L and D correspondingly requires a compaction operation, whose work is charged to the affected computations in L and D respectively. Each of these compactions costs $O(\log(W'))$ span.

We can finally conclude that all additional work performed by change propagation can be charged to the affected computations, and hence to the computation distance W_Δ, while incurring at most $O(\log(W'))$ additional span per round w.h.p. Therefore the total work performed by change propagation is $O(W_\Delta + R')$ in expectation and the span is $O(S' + R' \cdot \log(W'))$ w.h.p. ◀

We now show that for a special class of round-synchronous algorithms, the span overhead can be reduced. Our dynamic trees algorithm falls into this special case.

▶ **Definition 10.** *A* restricted round-synchronous *algorithm is a round-synchronous algorithm such that each round computation performs only a constant number of reads and writes, and each shared memory location is read only by a constant number of computations, and only in the round directly after it was written.*

▶ **Theorem 11.** *Given a restricted round-synchronous algorithm A with input configuration (I,P) that does W work in R rounds and S span, then*

1. *the initial run of the algorithm with tracking takes $O(W)$ work and $O(S + R\log^*(W))$ span w.h.p. on the CRCW* PRAM *and $O(S + R\log(W))$ span in the binary forking model,*
2. *change propagation on a dynamic update to the input configuration (I', P') takes $O(W_\Delta)$ work (in expectation on the CRCW* PRAM*), and $O(S' + R'\log^*(W'))$ span w.h.p. on the CRCW* PRAM *and $O(S' + R'\log(W'))$ span in the binary forking model, where S', R', W' are the maximum span, rounds, and work of the algorithm on the two input configurations.*

Proof sketch. Rather than recreate the entirety of the proof of Theorem 9, we simply sketch the differences. In essence, we obtain the result by removing the uses of scans, and semisorts, which were the main cause of the $O(\log(W'))$ span overhead and the randomized work. Instead, we rely only on (possibly approximate) compaction, which is only randomized on the CRCW PRAM. We also lose the R' term in the work since computations can only read from locations written in the previous round, and hence the set of rounds on which there exists an affected computation must be contiguous.

The main technique that we will make use of is the sparse array plus compaction technique. In situations where we wish to collect a set of items from each executed process, we would, in the unrestricted model, require a scan, which costs $O(\log(W'))$ span on the CRCW PRAM.

If each executed process, however, only produces a constant number of these items, we can allocate an array that is a constant size larger than the number of processes, and each process can write its set of items to a designated offset. We can then perform (possibly approximate) compaction on this array to obtain the desired set, with at most a constant factor additional blank entries. This takes $O(\log^*(W'))$ span w.h.p. on the CRCW PRAM, and $O(\log(W'))$ span in the binary forking model.

Maintaining S_m in the initial run and during change propagation is the first bottleneck, originally requiring a semisort. Since each computation performs a constant number of writes, we can collect the writes using the sparse array plus compaction technique. Since, in the restricted model, each modifiable will only be read by a constant number of readers, we can update S_m in constant time.

To compute the affected computations A_r also originally required a semisort, but in the restricted model, since all reads happen on the round directly after the write, no semisort is needed, since they will all have the same value of r. Collecting the affected computations from the written modifiables can also be achieved using the sparse array and compaction technique, using the fact that each computation wrote to a constant number of modifiables, and each modifiable is subsequently read by a constant number of computations. Additionally, A_r will be empty at the beginning of round r, so computing P requires only a compaction.

Lastly, collecting the updated locations U can also be performed using the sparse array and compaction technique. In summary, we can replace all originally $O(\log(W'))$ span operations with (approximate) compaction in the restricted setting, and hence we obtain the given span bounds since this takes $O(\log^*(W'))$ span w.h.p. on the CRCW PRAM, and $O(\log(W'))$ span in the binary forking model. ◀

▶ Remark 12 (Space usage). We do not formally specify an implementation of the memory model, but one simple way to achieve good space bounds is to use hashtables to implement global shared memory. Each write to a particular global shared memory location maps to an entry in the hashtable. When a round computation is invalidated during a dynamic update, its writes can be purged from the hashtable to free up space, preventing unbounded space blow up. Since the algorithm must also track the reads of each global shared memory location, using this implementation, the space usage is proportional to the number of shared memory reads and writes. In the restricted round-synchronous model, the number of reads must be proportional to the number of writes, and hence the space usage is proportional to the number of writes.

4 Dynamizing Tree Contraction

In this section, we show how to obtain a dynamic tree contraction algorithm by applying our dynamization technique to the static tree contraction algorithm of Miller and Reif [20]. In Section 5, we will show how to use this to obtain a parallel batch-dynamic trees framework.

Tree contraction

Tree contraction is the process of shrinking a tree down to a single vertex by repeatedly performing local contractions. Each local contraction deletes a vertex and merges its adjacent edges if it had degree two. Tree contraction has a number of useful applications, studied extensively in [21, 22, 5]. It can be used to perform various computations by associating data with edges and vertices and defining how data is accumulated during local contractions.

Various versions of tree contraction have been proposed depending on the specifics of local contractions. We consider an undirected variant of the randomized version proposed by Miller and Reif [20], which makes use of two operations: *rake* and *compress*. The former

removes all nodes of degree one from the tree, except in the case of a pair of adjacent degree one vertices, in which case only one of them is removed by tiebreaking on the vertex IDs. The latter operation, compress, removes an independent set of vertices of degree two that are not adjacent to any vertex of degree one. Compressions are randomized with coin flips to break symmetry. Miller and Reif showed that it takes $O(\log n)$ rounds w.h.p. to fully contract a tree of n vertices in this manner.

Input forests

The algorithms described here operate on undirected forests $F = (V, E)$, where V is a set of vertices, and E is a set of undirected edges. If $(u, v) \in E$, we say that u and v are *adjacent*, or that they are *neighbors*. A vertex with no neighbors is said to be *isolated*, and a vertex with one neighbor is called a *leaf*.

We assume that the forests given as input have bounded degree. That is, there exists some constant t such that each vertex has at most t neighbors. We will explain how to handle arbitrary-degree trees momentarily.

The static algorithm

The static tree contraction algorithm (Algorithm 3) works in rounds, each of which takes a forest from the previous round as input and produces a new forest for the next round. On each round, some vertices may be *deleted*, in which case they are removed from the forest and are not present in all remaining rounds. Let $F^i = (V^i, E^i)$ be the forest after i rounds of contraction, and thus $F^0 = F$ is the input forest. We say that a vertex v is *alive* at round i if $v \in V^i$, and is *dead* at round i if $v \notin V^i$. If $v \in V^i$ but $v \notin V^{i+1}$ then v was deleted in round i. There are three ways for a vertex to be deleted: it either *finalizes* (Line 32), *rakes* (Line 21), or *compresses* (Line 26). Finalization removes isolated vertices. Rake removes all leaves from the tree, with one special exception. If two leaves are adjacent, then to break symmetry and ensure that only one of them rakes, the one with the lower identifier rakes into the other (Line 8). Finally, compression removes an independent set of degree two vertices that are not adjacent to any degree one vertices, as in Miller and Reif's algorithm. The choice of which vertices are deleted in each round is made locally for each vertex based upon its own degree, the degrees of its neighbors, and coin flips for itself and its neighbors (Line 13). For coin flips, we assume a function $\textsc{Heads}(i, v)$ which indicates whether or not vertex v flipped heaps on round i. It is important that $\textsc{Heads}(i, v)$ is a function of both the vertex and the round number, as coin flips must be repeatable for change propagation to be correct.

The algorithm produces a *contraction data structure* which serves as a record of the contraction process. The contraction data structure is a tuple, (A, D), where $A[i][u]$ is a list of pairs containing the vertices adjacent to u in round i, and the positions of u in the adjacency lists of the adjacent vertices. $D[u]$ stores the round on which vertex u contracted. The algorithm also records $\text{leaf}[i][u]$, which is true if vertex u is a leaf at round i. An implementation of the tree contraction algorithm in our framework is shown in Algorithm 3.

Updates

We consider update operations that implement the interface of a batch-dynamic tree data structure. This requires supporting batches of links and cuts. A *link (insertion)* connects two trees in the forest by a newly inserted edge. A *cut (deletion)* deletes an edge from the forest, separating a single tree into two trees. We formally specify the interface for batch-dynamic trees and give a sample implementation of their operations in terms of the tree contraction data structure in the full version of this paper [2].

Algorithm 3 Tree contraction algorithm.

```
1: procedure ComputeRound(i, u)
2:   local ((v_1, p_1), ..., (v_t, p_t)), ℓ ← read(A[i][u], leaf[i][u])
3:   if v_i = ⊥ ∀i then                                  // A vertex with no neighbors finalizes
4:     DoFinalize(i, u)
5:   else if ℓ then                                      // A leaf vertex rakes if its neighbor is
6:     local (v, p) ← (v_i, p_i) such that v_i ≠ ⊥       // not a leaf, or if it has the lower ID
7:     local ℓ' ← read(leaf[i][v])
8:     if ¬ℓ' ∨ u < v then DoRake(i, u, (v, p))
9:     else DoAlive(i, u, ((v_1, p_1), ..., (v_t, p_t)))
10:  else                                                // If the vertex has exactly two
11:    if ∃(v, p), (v', p') : {v_1, ..., v_t} \ {⊥} = {v, v'} then   //  neighbors, it will compress
12:      local ℓ', ℓ'' ← read(leaf[i][v], leaf[i][v'])   // if neither of them are
13:      local c ← Heads(i, u) ∧ ¬Heads(i, v) ∧ ¬Heads(i, v')   // leaves and it flips heads
14:      if (¬ℓ' ∧ ¬ℓ'' ∧ c) then                        // and they both flip tails
15:        DoCompress(i, u, (v, p), (v', p'))
16:      else
17:        DoAlive(i, u, ((v_1, p_1), ..., (v_t, p_t)))
18:    else
19:      DoAlive(i, u, ((v_1, p_1), ..., (v_t, p_t)))
20:
21: procedure DoRake(i, u, (v, p))                       // When a vertex rakes, it replaces itself with
22:   write(A[i + 1][v][p], ⊥)                           // null (⊥) in its neighbor's adjacency list in
23:   write(D[u], i)                                     // in the next round
24:   retire process
25:
26: procedure DoCompress(i, u, (v, p), (v', p'))         // When a vertex compresses, it replaces itself
27:   write(A[i + 1][v][p], (v', p'))                    // with its opposite neighbors in each neighbor's
28:   write(A[i + 1][v'][p'], (v, p))                    // adjacency list in the next round
29:   write(D[u], i)
30:   retire process
31:
32: procedure DoFinalize(i, u)
33:   write(D[u], i)
34:   retire process
35:
36: procedure DoAlive(i, u, ((v_1, p_1), ..., (v_t, p_t)))   // If a vertex remains alive, it writes itself into
37:   local nonleaves ← 0                                // its neighbors' adjacency lists in the next
38:   for j ← 1 to t do                                  // round. It must also determine whether it
39:     if v_j ≠ ⊥ then                                  // it will be a leaf in the next round
40:       write(A[i + 1][v_j][p_j], (u, j))
41:       nonleaves += 1 - read(leaf[i][v_j])
42:     else
43:       write(A[i + 1][u][j], ⊥)
44:   write(leaf[i + 1][u], nonleaves = 1)
```

Handling trees of arbitrary degree

To handle trees of arbitrary degree, we can split each vertex into a path of vertices, one for each of its neighbors. This technique is standard and has been described in [18], for example. This results in a tree of degree 3, with at most $O(n + m)$ vertices and $O(m)$ edges for an initial tree of n vertices and m edges. For edge-weighted trees, the additional edges can be

given a suitable identity weight to preserve query values. It is simple to maintain such a transformation dynamically. For a batch insertion, a work-efficient semisort can be used to group each new neighbor by their endpoints, and then for each vertex, an appropriate number of new vertices can be added to the path. Batch deletion can be handled similarly.

4.1 Analysis

We now analyse the initial work, round, complexity, span, and computation distance of the tree contraction algorithm. This section is dedicated to proving the following theorem.

▶ **Theorem 13.** *Given a forest of n vertices, the initial work of tree contraction is $O(n)$ in expectation, the round complexity and the span is $O(\log(n))$ w.h.p. and the computation distance induced by updating k edges is $O(k \log(1 + n/k))$ in expectation.*

Let $F = (V, E)$ be the set of initial vertices and edges of the input tree, and denote by $F^i = (V^i, E^i)$, the set of remaining (alive) vertices and edges at round i. We use the term *at round i* to denote the beginning of round i, and *in round i* to denote an event that occurs during round i. For some vertex v at round i, we denote the set of its adjacent vertices by $A^i(v)$, and its degree with $\delta^i(v) = \left|A^i(v)\right|$. A vertex is *isolated* at round i if $\delta^i(v) = 0$. When multiple forests are in play, it will be necessary to disambiguate which is in focus. For this, we will use subscripts: for example, $\delta^i_F(v)$ is the degree of v in the forest F^i, and E^i_F is the set of edges in the forest F^i.

4.1.1 Analysis of construction

We first show that the static tree contraction algorithm is efficient. This argument is similar to Miller and Reif's argument in Theorem 2.1 of [21].

▶ **Lemma 14.** *For any forest (V, E), there exists $\beta \in (0, 1)$ such that $\mathbf{E}\left[\left|V^i\right|\right] \leq \beta^i |V|$, where V^i is the set of vertices remaining after i rounds of contraction.*

Proof. We begin by considering trees, and then extend the argument to forests. Given a tree (V, E), consider the set V' of vertices after one round of contraction. We would like to show there exists $\beta \in (0, 1)$ such that $\mathbf{E}[|V'|] \leq \beta |V|$. If $|V| = 1$, then this is trivial since the vertex finalizes (it is deleted with probability 1). For $|V| \geq 2$, Consider the following sets, which partition the vertex set:

$H = \{v : \delta(v) \geq 3\}$
$L = \{v : \delta(v) = 1\}$
$C = \{v : \delta(v) = 2 \wedge \forall u \in A(v), u \notin L\}$
$C' = \{v : \delta(v) = 2\} \setminus C$

Note that at least half of the vertices in L must be deleted, since all leaves are deleted, except those that are adjacent to another leaf, in which case exactly one of the two is deleted. Also, in expectation, $1/8$ of the vertices in C are deleted. Vertices in H and C' necessarily do not get deleted. Now, observe that $|C'| \leq |L|$, since each vertex in C' is adjacent to a distinct leaf. Finally, we also have $|H| < |L|$, which follows from standard arguments about compact trees. Therefore in expectation,

$$\frac{1}{2}|L| + \frac{1}{8}|C| \geq \frac{1}{4}|L| + \frac{1}{8}|H| + \frac{1}{8}|C'| + \frac{1}{8}|C| \geq \frac{1}{8}|V|$$

vertices are deleted, and hence

$$\mathbf{E}[|V'|] \leq \frac{7}{8}|V|.$$

Equivalently, for $\beta = \frac{7}{8}$, for every i, we have $\mathbf{E}\left[\left|V^{i+1}\right| \mid V_i\right] \leq \beta \left|V^i\right|$, where V^i is the set of vertices after i rounds of contraction. Therefore $\mathbf{E}\left[\left|V^{i+1}\right|\right] \leq \beta\, \mathbf{E}\left[\left|V^i\right|\right]$. Expanding this recurrence, we have $\mathbf{E}\left[\left|V^i\right|\right] \leq \beta^i |V|$. To extend the proof to forests, simply partition the forest into its constituent trees and apply the same argument to each tree individually. Due to linearity of expectation, summing over all trees yields the desired bounds. ◀

▶ **Lemma 15.** *On a forest of n vertices, after $O(\log n)$ rounds of contraction, there are no vertices remaining w.h.p.*

Proof. For any $c > 0$, consider round $r = (c+1) \cdot \log_{1/\beta}(n)$. By Lemma 14 and Markov's inequality, we have

$$\mathbf{P}\left[|V^r| \geq 1\right] \leq \beta^r n = n^{-c}.$$ ◀

Proof of initial work, rounds, and span in Theorem 13

Proof. At each round, the construction algorithm performs $O\left(\left|V^i\right|\right)$ work, and so the total work is $O\left(\sum_i \mathbf{E}\left[\left|V^i\right|\right]\right)$ in expectation. By Lemma 14, this is $O(|V|) = O(n)$. The round complexity and the span follow from Lemma 15. ◀

4.1.2 Analysis of dynamic updates

Intuitively, tree contraction is efficiently dynamizable due to the observation that, when a vertex locally makes a choice about whether or not to delete, it only needs to know who its neighbors are, and whether or not its neighbors are leaves. This motivates the definition of the *configuration* of a vertex v at round i, denoted $\kappa_F^i(v)$, defined as

$$\kappa_F^i(v) = \begin{cases} (\{(u, \ell_F^i(u)) : u \in A_F^i(v)\}), & \text{if } v \in V_F^i \\ \text{dead}, & \text{if } v \notin V_F^i, \end{cases}$$

where $\ell_F^i(u)$ indicates whether $\delta_F^i(u) = 1$ (the *leaf status* of u). Consider some input forest $F = (V, E)$, and let $F' = (V, (E \setminus E^-) \cup E^+)$ be the newly desired input after a batch cut with edges E^- and/or a batch-link with edges E^+. We say that a vertex v is *affected* at round i if $\kappa_F^i(v) \neq \kappa_{F'}^i(v)$.

▶ **Lemma 16.** *The execution in the tree contraction algorithm of process p at round r is an affected computation if and only if p is an affected vertex at round r.*

Proof. The code for ComputeRound for tree contraction reads only the neighbors, and corresponding leaf statuses, which are precisely the values encoded by the configuration. Hence if vertex p is alive in both forests the computation p is affected if and only if vertex p is affected. If instead p is dead in one forest but not the other, vertex p is affected, and the process p will have retired in one computation but not the other, and hence it will be an affected computation. Otherwise, if vertex p is dead in both forests, then the process p will have retired in both computations, and hence be unaffected. ◀

This means that we can bound the computation distance by bounding the number of affected vertices. First, we show that vertices that are not affected at round i have nice properties.

▶ **Lemma 17.** *If v is unaffected at round i, then either v is dead at round i in both F and F', or v is adjacent to the same set of vertices in both.*

Proof. Follows directly from $\kappa_F^i(v) = \kappa_{F'}^i(v)$. ◀

▶ **Lemma 18.** *If v is unaffected at round i, then v is deleted in round i of F if and only if v is also deleted in round i of F', and in the same manner (finalize, rake, or compress).*

Proof. Suppose that v is unaffected at round i. Then by definition it has the same neighbors at round i in both F and F'. The contraction process depends only on the neighbors of the vertex, and hence proceeds identically in both cases. ◀

If a vertex v is not affected at round i but is affected at round $i+1$, then we say that v *becomes affected in round* i. A vertex can become affected in many ways.

▶ **Lemma 19.** *If v becomes affected in round i, then at least one of the following holds:*
1. *v has an affected neighbor u at round i which was deleted in either F^i or $(F')^i$.*
2. *v has an affected neighbor u at round $i+1$ where $\ell_F^{i+1}(u) \neq \ell_{F'}^{i+1}(u)$.*

Proof. First, note that since v becomes affected, we know v does not get deleted, and furthermore that v has at least one child at round i. If v were to be deleted, then by Lemma 18 it would do so in both forests, leading it to being dead in both forests at the next round and therefore unaffected. If v were to have no children, then v would rake, but we just argued that v cannot be deleted.

Suppose that the only neighbors of v which are deleted in round i are unaffected at round i. Then v's set of children in round $i+1$ is the same in both forests. If all of these are unaffected at round $i+1$, then their leaf statuses are also the same in both forests at round $i+1$, and hence v is unaffected, which is a contradiction. Thus case 2 of the lemma must hold. In any other scenario, case 1 of the lemma holds. ◀

▶ **Lemma 20.** *If v is not deleted in either forest in round i and $\ell_F^{i+1}(v) \neq \ell_{F'}^{i+1}(v)$, then v is affected at round i.*

Proof. Suppose v is not affected at round i. If none of v's neighbors are deleted in this round in either forest, then $\ell_F^{i+1}(v) = \ell_{F'}^{i+1}(v)$, a contradiction. Otherwise, if the only neighbors that are deleted do so via a compression, since compression preserves the degree of its endpoints, we will also have $\ell_F^{i+1}(v) = \ell_{F'}^{i+1}(v)$ and thus a contradiction. So, we consider the case of one of v's children raking. However, since v is unaffected, we know $\ell_F^i(u) = \ell_{F'}^i(u)$ for each child u of v. Thus if one of them rakes in round i in one forest, it will also do so in the other, and we will have $\ell_F^{i+1}(v) = \ell_{F'}^{i+1}(v)$. Therefore v must be affected at round i. ◀

Lemmas 19 and 20 give us tools to bound the number of affected vertices for a consecutive round of contraction: each affected vertex that is deleted affects its neighbors, and each affected vertex whose leaf status is different in the two forests at the next round affects its neighbor. This strategy actually overestimates which vertices are affected, since case 1 of Lemma 19 does not necessarily imply that v is affected at the next round. We wish to show that the number of affected vertices at each round is not large. Intuitively, we will show that the number of affected vertices grows only arithmetically in each round, while shrinking geometrically, which implies that their total number can never grow too large. Let A^i denote the set of affected vertices at round i. We begin by bounding the size of $|A^0|$.

▶ **Lemma 21.** *For a batch update of size k, we have $|A^0| \leq 3k$.*

Proof. The computation for a given vertex u at most reads its neighbors, and if it has a single neighbor, its neighbor's leaf status. Therefore, the addition/deletion of a single edge affects at most 3 vertices at round 0. Hence $|A^0| \leq 3k$. ◀

We say that an affected vertex u *spreads to* v in round i, if v was unaffected at round i and v becomes affected in round i in either of the following ways:

1. v is a neighbor of u at round i and u is deleted in round i in either F or F', or
2. v is a neighbor of u at round $i+1$ and the leaf status of u changes in round i, i.e., $\ell_F^{i+1}(v) \neq \ell_{F'}^{i+1}(v)$.

Let $s = |A^0|$. For each of F and F', we now inductively construct s disjoint sets for each round i, labeled $A_1^i, A_2^i, \ldots A_s^i$. These sets will form a partition of A^i. First, arbitrarily partition A^0 into s singleton sets, and let $A_1^0, \ldots, A_s^0$ be these singleton sets. In other words, each affected vertex in A^0 is assigned a unique number $1 \leq j \leq s$, and is then placed in A_j^0.

Given sets $A_1^i, \ldots, A_s^i$, we construct sets $A_1^{i+1}, \ldots, A_s^{i+1}$ as follows. Consider some $v \in A^{i+1} \setminus A^i$. By Lemmas 19 and 20, there must exist at least one $u \in A^i$ such that u spreads to v. Since there could be many of these, let $S^i(v)$ be the set of vertices which spread to v in round i. Define

$$j^i(v) = \begin{cases} j, & \text{if } v \in A_j^i \\ \min_{u \in S^i(v)} \left(j \text{ where } u \in A_j^i\right), & \text{otherwise} \end{cases}$$

In other words, $j^i(v)$ is v's set identifier if v is affected at round i, or otherwise the minimum set identifier j such that a vertex from A_j^i spread to v in round i. We can then produce the following for each $1 \leq j \leq k$:

$$A_j^{i+1} = \{v \in A^{i+1} \mid j^i(v) = j\}$$

Informally, each affected vertex from round i which stays affected also stays in the same place, and each newly affected vertex picks a set to join based on which vertices spread to it.

We say that a vertex v is a *frontier* at round i if v is affected at round i and at least one of its neighbors in either F or F' is unaffected at round i. It is easy to show that any frontier at any round is alive in both forests and has the same set of unaffected neighbors in both at that round, and thus, the set of frontier vertices at any round is the same in both forests. It is also easy to show that if a vertex v spreads to some other vertex in round i, then v is a frontier at round i. We show next that the number of frontier vertices within each A_j^i is bounded.

▶ **Lemma 22.** *For any i, j, each of the following statements hold:*

1. *The subforests induced by A_j^i in each of F^i and $(F')^i$ are trees.*
2. *A_j^i contains at most 2 frontier vertices.*
3. *$|A_j^{i+1} \setminus A_j^i| \leq 2$.*

Proof. Statement 1 follows from rake and compress preserving connectedness, and the fact that if u spreads to v then u and v are neighbors in both forests either at round i or round $i+1$. We prove statement 2 by induction on i, and conclude statement 3 in the process. At round 0, each A_j^0 contains at most 1 frontier. We now consider some A_j^i. Suppose there is a single frontier vertex v in A_j^i. If v compresses in one of the forests, then v will not be a frontier in A_j^{i+1}, but it will spread to at most two newly affected vertices which may be frontiers at round $i+1$. Thus the number of frontiers in A_j^{i+1} will be at most 2, and $|A_j^{i+1} \setminus A_j^i| \leq 2$.

If v rakes in one of the forests, then v must also rake in the other forest (if not, then v could not be a frontier, since its neighbor would be affected). It spreads to one newly affected vertex (its neighbor) which may be a frontier at round $i+1$. Thus the number of frontiers in A_j^{i+1} will be at most 1, and $|A_j^{i+1} \setminus A_j^i| \leq 1$.

Now suppose there are two frontiers u and v in A_j^i. Due to statement 1 of the Lemma, each of these must have at least one affected neighbor at round i. Thus if either is deleted, it will cease to be a frontier and may add at most one newly affected vertex to A_j^{i+1}, and this newly affected vertex might be a frontier at round $i+1$. The same can be said if either u or v spreads to a neighbor due to a leaf status change. Thus the number of frontiers either remains the same or decreases, and there are at most 2 newly affected vertices. Hence statements 2 and 3 of the Lemma hold. ◀

Now define $A_{F,j}^i = A_j^i \cap V_F^i$, that is, the set of vertices from A_j^i which are alive in F at round i. We define $A_{F',j}^i$ similarly for forest F'.

▶ **Lemma 23.** *For every i, j, we have*

$$\mathbf{E}\left[\left|A_{F,j}^i\right|\right] \leq \frac{6}{1-\beta},$$

and similarly for $A_{F',j}^i$.

Proof. Let $F_{A,j}^i$ denote the subforest induced by $A_{F,j}^i$ in F^i. By Lemma 22, this subforest is a tree, and has at most 2 frontier vertices. By Lemma 14, if we applied one round of contraction to $F_{A,j}^i$, the expected number of vertices remaining would be at most $\beta \cdot \mathbf{E}[|A_{F,j}^i|]$. However, some of the vertices that are deleted in $F_{A,j}^i$ may not be deleted in F^i. Specifically, any vertex in $A_{F,j}^i$ which is a frontier or is the neighbor that spread to a frontier might not be deleted. There are at most two frontier vertices and two associated neighbors. By Lemma 22, two newly affected vertices might also be added. We also have $|A_{F,j}^0| = 1$. Therefore we conclude the following, which similarly holds for forest F':

$$\mathbf{E}\left[\left|A_{F,j}^{i+1}\right|\right] \leq \beta\, \mathbf{E}\left[\left|A_{F,j}^i\right|\right] + 6 \leq 6\sum_{r=0}^{\infty} \beta^r = \frac{6}{1-\beta}.$$ ◀

▶ **Lemma 24.** *For a batch update of size k, we have for every i,*

$$\mathbf{E}\left[\left|A^i\right|\right] \leq \frac{36}{1-\beta}k.$$

Proof. Follows from Lemmas 21 and 23, and the fact that

$$\left|A^i\right| \leq \sum_{j=1}^{s} \left(\left|A_{F,j}^i\right| + \left|A_{F',j}^i\right|\right).$$ ◀

Proof of computation distance in Theorem 13

Proof. Let F be the given forest and F' be the desired forest. Since each process of tree contraction does constant work each round, Lemma 16 implies that the algorithm does $O\left(\left|A^i\right|\right)$ work at each round i, so $W_\Delta = \sum_i \left|A^i\right|$.

Since at least one vertex is either raked or finalized each round, we know that there are at most n rounds. Consider round $r = \log_{1/\beta}(1 + n/k)$, using the β given in Lemma 14. We now split the rounds into two groups: those that come before r and those that come after.

For $i < r$, we bound $\mathbf{E}\left[\left|A^i\right|\right]$ according to Lemma 24, yielding

$$\sum_{i<r} \mathbf{E}\left[\left|A^i\right|\right] = O(rk) = O\left(k \log\left(1 + \frac{n}{k}\right)\right)$$

work. Now consider $r \leq i < n$. For any i we know $\left|A^i\right| \leq \left|V_F^i\right| + \left|V_{F'}^i\right|$, because each affected vertex must be alive in at least one of the two forests at that round. We can then apply the bound given in Lemma 14, and so

$$\begin{aligned}\sum_{r \leq i < n} \mathbf{E}\left[\left|A^i\right|\right] &\leq \sum_{r \leq i < n} \left(\mathbf{E}\left[\left|V_F^i\right|\right] + \mathbf{E}\left[\left|V_{F'}^i\right|\right]\right) \\ &\leq \sum_{r \leq i < n} \left(\beta^i n + \beta^i n\right) \\ &= O(n\beta^r) \\ &= O\left(\frac{nk}{n+k}\right) \\ &= O(k),\end{aligned}$$

and thus

$$\mathbf{E}\left[W_\Delta\right] = O\left(k \log\left(1 + \frac{n}{k}\right)\right) + O(k) = O\left(k \log\left(1 + \frac{n}{k}\right)\right). \quad \blacktriangleleft$$

5 Parallel Rake-compress Trees

Dynamic trees typically provide support for dynamic connectivity queries. Most dynamic tree data structures also support some form of augmented value query. For example, Link-cut trees [27] support root-to-vertex path queries, and Euler-tour trees [15] support subtree sum queries. Top trees [28, 6] support both path and subtree queries, as well as nonlocal queries such as centers and medians, but no parallelization of them is known. The only existing parallel batch-dynamic tree data structure is that of Tseng et al. [29], which is based on Euler-tour trees, and hence only handles subtree queries.

Rake-compress trees [5] (RC trees) are another sequential dynamic trees data structure, based on tree contraction, and have also been shown to be capable of handling both path and subtree queries, as well as nonlocal queries, all in $O(\log(n))$ time. In this section, we will explain how our parallel batch-dynamic algorithm for tree contraction can be used to derive a parallel batch-dynamic version of RC trees, leading to the first work-efficient algorithm for batch-dynamic trees that can handle this wide range of queries. We use a slightly different set of definitions than the original presentation of RC trees in [5], which correct some subtle corner cases and simplify the exposition, although the resulting data structure is equivalent. All of the query algorithms for sequential RC trees therefore work on our parallel version.

Contraction and clusters

RC trees are based on the idea that the tree contraction process can be interpreted as a recursive clustering of the original tree. Formally, a *cluster* is a connected subset of vertices and edges of the original tree. Note, importantly, that a cluster may contain an edge without containing both of its endpoints. The *boundary* vertices of a cluster C are the vertices $v \notin C$ that are adjacent to an edge $e \in C$. The *degree* of a cluster is the number of boundary vertices of that cluster. The vertices and edges of the original tree form the base clusters. Clusters are merged using the following simple rule: Whenever a vertex v is deleted, all of the clusters that have v as a boundary vertex are merged with the base cluster containing v. This implies that all clusters formed will have degree at most two. A cluster of degree zero is called a *nullary* cluster, a cluster of degree one a *unary* cluster, and a cluster of degree two a

binary cluster. All non-base clusters have a unique *representative vertex*, which corresponds to the vertex that was deleted to form it. The full version of this paper [2] provides additional details and some diagrams that explain what each kind of cluster looks like.

5.1 Building and maintaining RC trees

Given a tree and an execution of the tree contraction algorithm, the RC tree consists of *nodes* which correspond to the clusters formed by the contraction process. The children of a node are the nodes corresponding to the clusters that merged together to form it. An example tree, a clustering, and the corresponding RC tree are depicted in Figure 1. Note that in the case of a disconnected forest, the RC tree will have multiple roots.

We will sketch here how to maintain an RC tree subject to batch-dynamic updates in parallel using our algorithm for parallel batch-dynamic tree contraction. This requires just two simple augmentations to the tree contraction algorithm. Recall that tree contraction (Algorithm 3) maintains an adjacency list for each vertex at each round. Whenever a neighbor u of a vertex v rakes into v, the process u writes a null value into the corresponding position in v's adjacency list. This process can be augmented to also write, in addition to the null value, the identity of the vertex that just raked. Second, when storing the data for a neighboring edge in a vertex's adjacency list, we additionally write the name of the representative vertex if that edge corresponds to a compression, or null if the edge is an edge of the original tree. The RC tree can then be inferred using this augmented data as follows.

1. Given any cluster C with representative v, its unary children can be determined by looking at the vertices that raked into v. The children are precisely the unary clusters represented by these vertices. For the final cluster, these are its only children.
2. Given a binary or unary cluster C with representative v, its binary children can be determined by inspecting v's adjacency list at the moment it was deleted. The binary clusters corresponding to the edges adjacent to v at its time of death are the binary children of the cluster C.

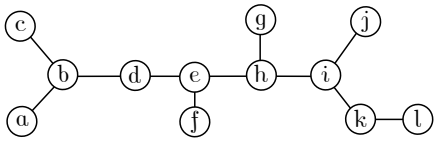
It then suffices to observe that this information about the clusters can be recorded during the contraction process. By employing change propagation, the RC tree can therefore be maintained subject to batch-dynamic updates. Since each cluster consists of a constant amount of information, this can be done in the same work and span bounds as the tree contraction algorithm. We therefore have the following result.

▶ **Theorem 25.** *We can maintain a rake-compress tree of a tree on n vertices subject to batch insertions and batch deletions of size k in $O(k \log(1 + n/k))$ work in expectation and $O(\log^2(n))$ span per update w.h.p. The span can be improved to $O(\log(n) \log^*(n))$ w.h.p. on the CRCW* PRAM*.*

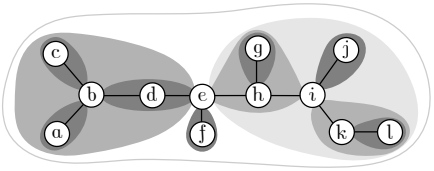
5.2 Applications

Most kinds of queries assume that the vertices and/or edges of the input tree are annotated with data, such as weights or labels. In order to support queries, each cluster is annotated with some additional information. The algorithm must then specify how to combine the data from multiple constituent clusters whenever a set of clusters merge. These annotations are generated during the tree contraction algorithm, and are therefore available for querying immediately after performing an update.

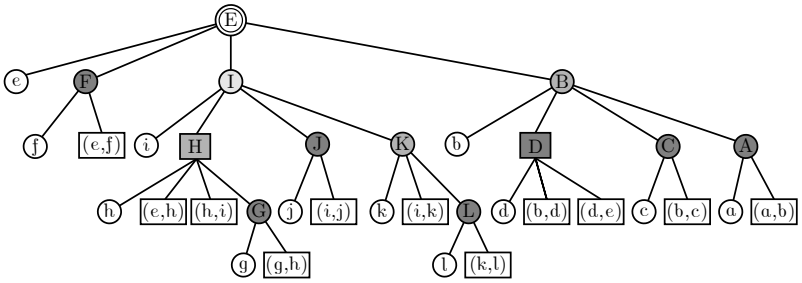
Once the clusters are annotated with the necessary data, the queries themselves typically perform a bottom-up or top-down traversal of the RC tree, or possibly in the case of more complicated queries, a combination of both. A variety of queries is described in [5].



(a) A tree.



(b) A recursive clustering of the tree produced by tree contraction. Clusters produced in earlier rounds are depicted in a darker color.



(c) The corresponding RC tree. (Non-base) unary clusters are shown as circles, binary clusters as rectangles, and the finalize (nullary) cluster at the root with two concentric circles. The base clusters (the leaves) are labeled in lowercase, and the composite clusters are labeled with the uppercase label of their representative vertex.

Figure 1 A tree, a clustering, and the corresponding RC tree.

Batch queries

We can also implement *batch queries*, in which we answer k queries simultaneously in $O(k \log(1 + n/k))$ work in expectation and $O(\log(n))$ span w.h.p. This improves upon the work bound of $O(k \log(n))$ obtained by simply running independent queries in parallel. The idea is to detect when multiple traversals would intersect, and to eliminate redundant work that they would perform. An example in which this technique is applicable is finding a representative vertex of a connected component. When traversing upwards, if multiple query paths intersect, then only one proceeds up the tree and subsequently brings the answer back down for the other ones. The following theorem is the main tool that we can use for analyzing batch queries. The proof is similar to that of the computation distance in Theorem 13, and can be found in the full version of this paper [2].

▶ **Theorem 26.** *Given a tree on n vertices and a corresponding RC tree, k root-to-leaf paths in the RC tree touch $O(k \log(1 + n/k))$ distinct RC tree nodes in expectation.*

In the full version of this paper [2], we will show that batch connectivity, subtree sum, and path sum queries given batches of size k can be answered in $O(k \log(1 + n/k))$ work in expectation and $O(\log(n))$ span w.h.p.

6 Conclusion

In this paper we showed that we can obtain work-efficient parallel batch-dynamic algorithms by applying an algorithmic dynamization technique to corresponding static algorithms. Using this technique, we obtained the first work-efficient parallel algorithm for batch-dynamic trees that supports more than just subtree queries. Our framework also demonstrates the

broad benefits of algorithmic dynamization; much of the complexity of designing parallel batch-dynamic algorithms by hand is removed, since the static algorithms are usually simpler than their dynamic counterparts. We note that although the round synchronous model captures a very broad class of algorithms, the breadth of algorithms suitable for dynamization is less clear. To be suitable for dynamization, an algorithm additionally needs to have small computational distance between small input changes. As some evidence of broad applicability, however, the practical systems mentioned in the technical overview of the introduction have been applied broadly and successfully – again without any theoretical justification, yet.

References

1 Umut A Acar, Daniel Anderson, Guy E Blelloch, and Laxman Dhulipala. Parallel batch-dynamic graph connectivity. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2019.

2 Umut A Acar, Daniel Anderson, Guy E Blelloch, Laxman Dhulipala, and Sam Westrick. Parallel batch-dynamic trees via change propagation. *arXiv preprint*, 2020. `arXiv:2002.05129`.

3 Umut A Acar, Guy E Blelloch, and Robert Harper. Adaptive functional programming. In *ACM Symposium on Principles of Programming Languages (POPL)*, 2002.

4 Umut A Acar, Guy E Blelloch, Robert Harper, Jorge L Vittes, and Shan Leung Maverick Woo. Dynamizing static algorithms, with applications to dynamic trees and history independence. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.

5 Umut A Acar, Guy E Blelloch, and Jorge L Vittes. An experimental analysis of change propagation in dynamic trees. In *Algorithm Engineering and Experiments (ALENEX)*, 2005.

6 Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Transactions on Algorithms (TALG)*, 1(2):243–264, 2005.

7 Pramod Bhatotia, Alexander Wieder, Rodrigo Rodrigues, Umut A. Acar, and Rafael Pasquini. Incoop: MapReduce for incremental computations. In *ACM Symposium on Cloud Computing (SoCC)*, 2011.

8 Guy E Blelloch, Jeremy T Fineman, Yan Gu, and Yihan Sun. Optimal parallel algorithms in the binary-forking model. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2020.

9 Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. Mapreduce online. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

10 Laxman Dhulipala, David Durfee, Janardhan Kulkarni, Richard Peng, Saurabh Sawlani, and Xiaorui Sun. Parallel batch-dynamic graphs: Algorithms and lower bounds. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

11 Greg N Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985.

12 Joseph Gil, Yossi Matias, and Uzi Vishkin. Towards a theory of nearly constant time parallel algorithms. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1991.

13 Yan Gu, Julian Shun, Yihan Sun, and Guy E. Blelloch. A top-down parallel semisort. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2015.

14 Pradeep Kumar Gunda, Lenin Ravindranath, Chandramohan A. Thekkath, Yuan Yu, and Li Zhuang. Nectar: Automatic management of data and computation in data centers. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

15 Monika R. Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.

16 Giuseppe F Italiano, Silvio Lattanzi, Vahab S Mirrokni, and Nikos Parotsidis. Dynamic algorithms for the massively parallel computation model. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2019.

17 Joseph JáJá. *An Introduction to Parallel Algorithms*, volume 17. Addison-Wesley Reading, 1992.

18 Donald B Johnson and Panagiotis Metaxas. Optimal algorithms for the vertex updating problem of a minimum spanning tree. In *International Parallel Processing Symposium (IPPS)*, 1992.

19 David R Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, 2000.

20 Gary L. Miller and John H. Reif. Parallel tree contraction and its application. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, October 1985.

21 Gary L. Miller and John H. Reif. Parallel tree contraction part 1: Fundamentals. In *Randomness and Computation*, pages 47–72. JAI Press, Greenwich, Connecticut, 1989. Vol. 5.

22 Gary L. Miller and John H. Reif. Parallel tree contraction part 2: Further applications. *SIAM Journal on Computing*, 20(6):1128–1147, 1991.

23 Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: A timely dataflow system. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2013.

24 Daniel Peng and Frank Dabek. Large-scale incremental processing using distributed transactions and notifications. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

25 John H Reif and Stephen R Tate. Dynamic parallel tree contraction. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 1994.

26 Natcha Simsiri, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Work-efficient parallel union-find with applications to incremental graph connectivity. In *European Conference on Parallel Processing (Euro-Par)*, 2016.

27 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

28 Robert E Tarjan and Renato F Werneck. Self-adjusting top trees. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.

29 Thomas Tseng, Laxman Dhulipala, and Guy Blelloch. Batch-parallel Euler tour trees. In *Algorithm Engineering and Experiments (ALENEX)*, 2019.

30 Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33:103–111, 1990.

Reconstructing Biological and Digital Phylogenetic Trees in Parallel

Ramtin Afshar
University of California-Irvine, CA, USA
afsharr@uci.edu

Michael T. Goodrich
University of California-Irvine, CA, USA
goodrich@uci.edu

Pedro Matias
University of California-Irvine, CA, USA
pmatias@uci.edu

Martha C. Osegueda
University of California-Irvine, CA, USA
mosegued@uci.edu

Abstract

In this paper, we study the parallel query complexity of reconstructing biological and digital phylogenetic trees from simple queries involving their nodes. This is motivated from computational biology, data protection, and computer security settings, which can be abstracted in terms of two parties, a ***responder***, Alice, who must correctly answer queries of a given type regarding a degree-d tree, T, and a ***querier***, Bob, who issues batches of queries, with each query in a batch being independent of the others, so as to eventually infer the structure of T. We show that a querier can efficiently reconstruct an n-node degree-d tree, T, with a logarithmic number of rounds and quasilinear number of queries, with high probability, for various types of queries, including ***relative-distance queries*** and ***path queries***. Our results are all asymptotically optimal and improve the asymptotic (sequential) query complexity for one of the problems we study. Moreover, through an experimental analysis using both real-world and synthetic data, we provide empirical evidence that our algorithms provide significant parallel speedups while also improving the total query complexities for the problems we study.

2012 ACM Subject Classification Theory of computation → Parallel computing models

Keywords and phrases Tree Reconstruction, Parallel Algorithms, Privacy, Phylogenetic Trees, Data Structures, Hierarchical Clustering

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.3

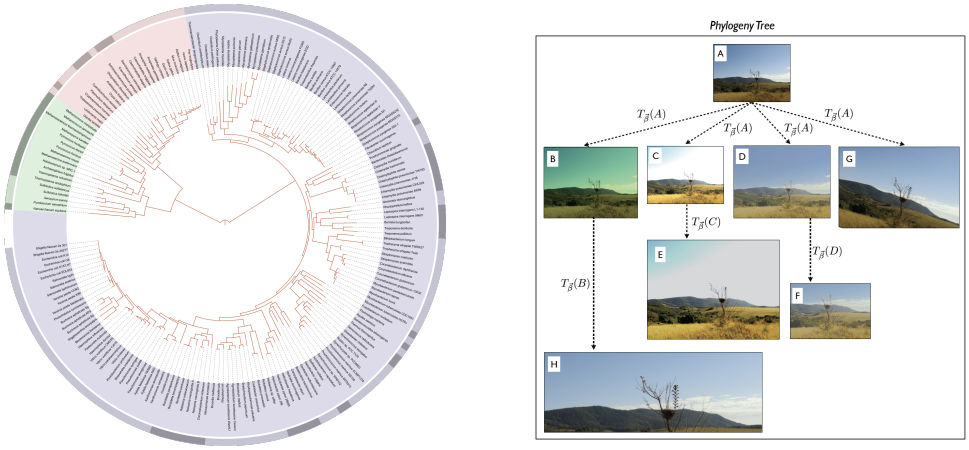
Related Version The full version of the paper is available at arXiv [2], `https://arxiv.org/abs/2006.15259`.

Supplementary Material The complete source code for our experiments, including the implementation of our algorithms and the algorithms we compared against, is available at `https://github.com/UC-Irvine-Theory/ParallelTreeReconstruction`.

Funding This article reports on work supported by NSF grant 1815073.

1 Introduction

Phylogenetic trees represent evolutionary relationships among a group of objects. For instance, each node in a biological phylogenetic tree represents a biological entity, such as a species, bacteria, or virus, and the branching represents how the entities are believed to have evolved from common ancestors [29, 8, 34]. (See Figure 1a.) In a digital phylogenetic tree,



(a) (b)

Figure 1 Two phylogenetic trees. (a) A biological phylogenetic tree of life, showing relationships between species whose genomes had been sequenced as of 2006; public domain image by Ivica Letunic, retraced by Mariana Ruiz Villarreal. (b) A digital phylogenetic tree of images, from Dias et al. [14].

on the other hand, each node represents a data object, such as a computer virus [22, 35], a source-code file [27], a text file or document [32, 40], or a multimedia object (such as an image or video) [6, 16, 15, 14] and the branching represents how these objects are believed to have evolved through edits or data compression/corruption. (See Figure 1b.)

In this paper, we are interested in studying efficient methods for reconstructing phylogenetic trees from queries regarding their structure, noting that there are differences in the types of queries one may perform on the two types of phylogenetic trees. In particular, with some exceptions,[1] in a biological phylogenetic tree we can only perform queries involving the leaves of the tree, since these typically represent living biological entities and internal nodes represent ancestors that are likely to be extinct. In digital phylogenetic trees, on the other hand, we can perform queries involving any of the nodes in the tree, including internal nodes, since these represent digital artifacts, which are often archived. The former type of phylogenetic tree has also received attention in the context of *hierarchical clustering*, where the goal is to provide a hierarchical grouping structure of items according to their similarity [18]. To support reconstruction of both biological and digital phylogenetic trees, therefore, we study both types of querying regimes in this paper.

More specifically, with respect to biological phylogenetic trees, we focus on ***relative-distance*** queries, where one is given three leaf nodes (corresponding to species), x, y, and z, and the response is a determination of which pair, (x, y), (x, z), or (y, z), is a closest pair, hence, has the most-recent common ancestor [29]. With respect to digital phylogenetic trees, we instead focus on ***path queries***, where one is given two nodes, v and w, in the tree and the response is "true" if and only if v is an ancestor of w.

The motivation for reconstructing phylogenetic trees comes from a desire to better understand the evolution of the objects represented in a given phylogenetic tree. For example, understanding how biological species evolved is useful for understanding and

[1] One notable exception to this restriction of only being able to ask queries involving leaves in a biological phylogenetic tree is for phylogenetic trees of biological viruses, for which genetic sequencing may be known for all instances; hence, ancestor-descendant path queries might also be appropriate for reconstructing some biological phylogenetic trees.

categorizing the fossil record and understanding when species are close relatives [29, 18]. Similarly, understanding how digital objects have been edited and transformed can be useful for data protection, computer security, privacy, copyright disputes, and plariarism detection [22, 35, 27, 32, 40, 6, 16, 15, 14]. For instance, understanding the evolutionary process of a computer virus can provide insights into its ancestry, characteristics of the attacker, and where future attacks might come from and what they might look like [35].

The efficiency of a tree reconstruction algorithm can be characterized in terms of its ***query-complexity*** measure, $Q(n)$, which is the total number of queries of a certain type needed to reconstruct a given tree. This parameter comes from machine-learning and complexity theory, e.g., see [1, 9, 17, 42], where it is also known as "decision-tree complexity," e.g., see [48, 5]. Previous work on tree reconstruction has focused on sequential methods, where queries are issued and answered one at a time. For example, in pioneering work for this research area, Kannan et al. [29] show that an n-node biological phylogenetic tree can be reconstructed sequentially from $O(n \log n)$ three-node relative-distance queries.

Indeed, their reconstruction algorithms are inherently sequential and involve incrementally inserting leaf nodes into the phylogenetic tree reconstructed for the previously-inserted nodes.

In many tree reconstruction applications, queries are expensive [29, 18, 22, 35, 27, 32, 40, 6, 16, 15, 14], but can be issued in batches. For example, there is nothing preventing the biological experiments [29] that are represented in three-node relative-distance queries from being issued in parallel. Thus, in order to speed up tree reconstruction, in this paper we are interested in parallel tree reconstruction. To this end, we also use a ***round-complexity*** parameter, $R(n)$, which measures the number of rounds of queries needed to reconstruct a tree such that the queries issued in any round comprise a batch of independent queries. That is, no query issued in a given round can depend on the outcome of another query issued in that round, although both can depend on answers to queries issued in previous rounds. Roughly speaking, $R(n)$ corresponds to the span of a parallel reconstruction algorithm and $Q(n)$ corresponds to its work. In this paper, we are interested in studying complexities for $R(n)$ and $Q(n)$ with respect to biological and digital phylogenetic trees with fixed maximum degree, d.

1.1 Related Work

The general problem of reconstructing graphs from distance queries was studied by Kannan et al. [30], who provide a randomized algorithm for reconstructing a graph of n vertices using $\tilde{O}(n^{3/2})$ distance queries.[2]

Previous parallel work has focused on inferring phylogenetic trees through Bayesian estimation [4]. However, we are not aware of previous parallel work using a similar query models to ours. With respect to previous work on sequential tree reconstruction, Culberson and Rudnicki [13] provide the first sub-quadratic algorithms for reconstructing a weighted undirected tree with n vertices and bounded degree d from additive queries, where each query returns the sum of the weights of the edges of the path between a given pair of vertices. Reyzin and Srivastava [38] show that the Culberson-Rudnicki algorithm uses $O(n^{3/2} \cdot \sqrt{d})$ queries.

Waterman et al. [47] introduce the problem of reconstructing biological phylogenetic trees, using additive queries, which are more powerful than relative-distance queries. Hein [24] shows that this problem has a solution that uses $O(dn \log_d n)$ additive queries, when the

[2] The $\tilde{O}(\cdot)$ notation hides poly-logarithmic factors.

tree has maximum degree d, which is asymptotically optimal [31]. Kannan et al. [29] show that an n-node binary phylogenetic tree can be reconstructed from $O(n \log n)$ three-node relative-distance queries. Their method appears inherently sequential, however, as it is based on an incremental approach that mimics insertion-sort. Similarly, Emamjomeh-Zadeh and Kempe [18] also give a sequential method using relative-distance queries that has a query complexity of $O(n \log n)$. Their algorithm, however, was designed for a different context, namely, hierarchical clustering.

Additionally, there exists some work (e.g. [28, 7, 25]) in an alternative perspective of the problem reconstructing phylogenetic trees, in which the goal is to find the best tree explaining the similarity and the relationship between a given fixed (or dynamic) set of data sequences (e.g. of species), using Maximum Parsimony [19, 21, 39] or Maximum Likelihood [20, 10]. This contrasts with our approach of recovering the "ground truth" tree known only to an oracle, which is consistent with its answers about the tree.

With respect to digital phylogenetic tree reconstruction, there are a number of sequential algorithms with $O(n^2)$ query complexities, including the use of what we are calling path queries, where the queries are also individually expensive, e.g., see [22, 35, 27, 32, 40, 6, 16, 15, 14]. Jagadish and Sen [26] consider reconstructing undirected unweighted degree-d trees, giving a deterministic algorithm that requires $O(dn^{1.5} \log n)$ separator queries, which answer if a vertex lies on the path between two vertices. They also give a randomized algorithm using an expected $O(d^2 n \log^2 n)$ number of separator queries, and they give an $\Omega(dn)$ lower bound for any deterministic algorithm. Wang and Honorio [46] consider the problem of reconstructing bounded-degree rooted trees, giving a randomized algorithm that uses expected $O(dn \log^2 n)$ path queries. They also prove that any randomized algorithm requires $\Omega(n \log n)$ path queries.

Our Contributions. In this paper, we study the parallel phylogenetic tree reconstruction problem with respect to the two different types of queries mentioned above:

- We show that an n-node rooted biological (binary) phylogenetic tree can be reconstructed from three-node ***relative-distance queries*** with $R(n)$ that is $O(\log n)$ and $Q(n)$ that is $O(n \log n)$, with high probability (w.h.p.)[3]. Both bounds are asymptotically optimal.
- We show that an n-node fixed-degree digital phylogenetic tree can be reconstructed from ***path queries***, which ask whether a given node, u, is an ancestor of a given node, w, with $R(n)$ that is $O(\log n)$ and $Q(n)$ that is $O(n \log n)$, w.h.p. We also provide an $\Omega(dn + n \log n)$ lower bound for any randomized or deterministic algorithm suggesting that our algorithm is optimal in terms of query complexity and round complexity. Further, this asymptotically-optimal $Q(n)$ bound actually improves the sequential complexity for this problem, as the previous best bound for $Q(n)$, due to Wang and Honorio [46], had a $Q(n)$ bound of $O(n \log^2 n)$ for reconstructing fixed-degree rooted trees using path queries. Of course, our method also applies to biological phylogenetic trees that support path queries.

A preliminary announcement of some of this paper's results, restricted to binary trees, was presented in [3]. Most of our algorithms are quite simple, although their analyses are at times nontrivial. Moreover, given the many applications of biological and digital phylogenetic tree reconstruction, we feel that our algorithms have real-world applications. Thus, we

[3] We say that an event occurs with high probability if it occurs with probability at least $1 - 1/n^c$, for some constant $c \geq 1$.

have done an extensive experimental analysis of our algorithms, using both real-world and synthetic data for biological and digital phylogenetic trees. Our experimental results provide empirical evidence that our methods achieve significant parallel speedups while also providing improved query complexities in practice.

2 Preliminaries

In graph theory, an ***arborescence*** is a directed graph, T, with a distinguished vertex, r, called the ***root***, such that, for any vertex v in T that is not the root, there is exactly one path from r to v, e.g., see Tutte [43]. That is, an arborescence is a graph-theoretic way of describing a rooted tree, so that all the edges are going away from the root. In this paper, when we refer to a "rooted tree" it should be understood formally to be an arborescence.

We represent a rooted tree as $T = (V, E, r)$, with a vertex set V, edge set E, and root $r \in V$. The ***degree*** of a vertex in such a tree is the sum of its in-degree and out-degree, and the degree of a tree, T, is the maximum degree of all vertices in T. So, an arborescence representing a binary tree would have degree 3. Because of the motivating applications, e.g., from computational biology, we assume in this paper that the trees we want to reconstruct have maximum degree that is bounded by a fixed constant, d.

Let us review a few terms regarding rooted trees.

▶ **Definition 1** (ancestry). *Given a rooted tree, $T = (V, E, r)$, we say u is **parent** of v (and v is a **child** of u) if there exists a directed edge (u, v) in E. The **ancestor** relation is the transitive closure of the parent relation, and the **descendant** relation is the transitive closure of the child relation. We denote the number of descendants of vertex s by $D(s)$. A node without any children is called a **leaf**. Given two leaf nodes, u and v in T, their **lowest common ancestor**, $lca(u, v)$, is the node, w in T, that is an ancestor of both u and v and has no child that is also an ancestor of u and v.*

We next define the types of queries we consider in this paper for reconstructing a rooted tree, $T = (V, E, r)$.

▶ **Definition 2.** *A **relative-distance query** for T is a function, closer, which takes three leaf nodes, u, v, and w in T, as input and returns the pair of nodes from the set, $\{u, v, w\}$, that has the lower lowest common ancestor. That is, $closer(u, v, w) = (u, v)$ if $lca(u, v)$ is a descendant of $lca(u, w) = lca(v, w)$. Likewise, we also have that $closer(u, v, w) = (u, w)$ if $lca(u, w)$ is a descendant of $lca(u, v) = lca(v, w)$, and $closer(u, v, w) = (v, w)$ if $lca(v, w)$ is a descendant of $lca(u, v) = lca(u, w)$.*

Definition 2 assumes T is a binary tree (of degree 3). Note that in this paper we restrict relative-distance queries to leaves, since these represent, e.g., current species in the application of reconstructing biological phylogenetic trees.

▶ **Definition 3.** *A **path query** for T is a function, path, that takes two nodes, u and v in T, as input and returns 1 if there is a (directed) path from vertex u to v, and otherwise returns 0. Also, for $u \in V$ and $W \subseteq V$, we define $count(u, W) = \sum_{v \in W} path(u, v)$, which is the number of descendants of u in W.*

We next study some preliminaries involving the structure of degree-d rooted trees that will prove useful for our parallel algorithms.

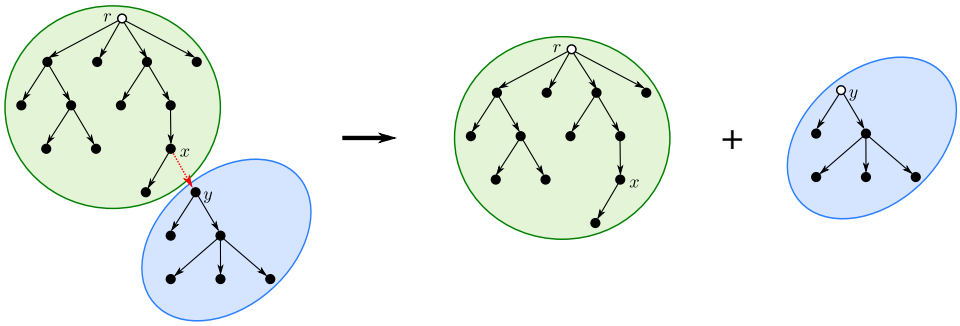


Figure 2 Illustration of a divide-conquer approach for trees. The edge (x, y) is an even-edge-separator. Note that the root of T'' is r, while y becomes root of T'.

▶ **Definition 4.** *Let $T = (V, E, r)$ be a degree-d rooted tree. We say that an edge $e = (x, y) \in E$ is an **even-edge-separator** if removing e from T partitions it into two rooted trees, $T' = (V', E', y)$ and $T'' = (V'', E'', r)$, such that $\frac{|V|}{d} \leq |V'| \leq \frac{|V|(d-1)}{d}$ and $\frac{|V|}{d} \leq |V''| \leq \frac{|V|(d-1)}{d}$. (See Figure 2.)*

▶ **Lemma 5.** *Every rooted tree of degree-d has an even-edge-separator.*

Proof. This follows from a result by Valiant [45, Lemma 2]. ◀

As we will see, this fact is useful for designing simple parallel divide-and-conquer algorithms. Namely, if we can find an even-edge-separator, then we can cut the tree in two by removing that edge and recurse on the two remaining subtrees in parallel (see Figure 2).

3 Reconstructing Biological Phylogenetic Trees in Parallel

Relative-distance queries model an experimental approach to constructing a biological phylogenetic tree, e.g., where DNA sequences are compared to determine which samples are the most similar [29]. In simple terms, pairs of DNA sequences that are closer to one another than to a third sequence are assumed to be from two species with a common ancestor that is more recent than the common ancestor of all three. In this section, for the sake of tree reconstruction, we assume the ***responder*** has knowledge of the absolute structure of a rooted binary phylogenetic tree; hence, each response to a $closer(u, v, w)$ query is assumed accurate with respect to an unknown rooted binary tree, T. As in the pioneering work of Kannan et al. [29], we assume the distance comparisons are accurate and consistent. The novel dimension here is that we consider parallel algorithms for phylogenetic tree reconstruction.

As mentioned above, we consider relative-distance queries to occur between leaves of a rooted binary tree, T. That is, in our query model, the ***querier*** has no knowledge of the internal nodes of T and can only perform queries using leaves. Because T is a binary phylogenetic tree, we may assume it is a ***proper*** binary tree, where each internal node in T has exactly two children.

3.1 Algorithm

At a high level, our parallel reconstruction algorithm (detailed in Algorithm 1) uses a randomized divide-and-conquer approach, similar to Figure 2. In our case, however, the division process is random three-way split through a vertex separator, rather than an edge-separator-based binary split. Initially, all leaves belong to a single partition, L. Then two

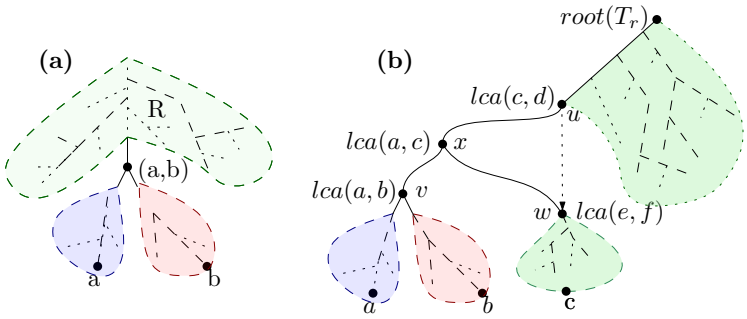


Figure 3 **(a)** The subgroups leaves are split into. **(b)** The linking step attaching T_a, T_b and T_r.

Algorithm 1 Reconstruct a binary tree of a set of leaves, L.

Function reconstruct-phylogenetic(L)**:**
  if $|L| \leq 3$ **then return** the tree formed by querying L
  Pick two leaves, $a, b \in L$, uniformly at random
  for ***each*** $c \in L$ ***s.t.*** $c \neq a, b$ **do in parallel**
    Perform query $closer(a, b, c)$
  Split the leaves in L into R, A, and B based on results
  parallel do
    $T_a \leftarrow$ reconstruct-phylogenetic$(A \cup \{a\})$
    $T_b \leftarrow$ reconstruct-phylogenetic$(B \cup \{b\})$
    $T_r \leftarrow$ reconstruct-phylogenetic(R)
  Let v be a new node, labeled "$lca(a, b)$"
  Set v's left child to $root(T_a)$ and the right to $root(T_b)$
  if $R = \emptyset$ **then return** tree, T_v, rooted at v
  else return link(v, T_r)

leaves, a and b, are chosen uniformly at random from L and each remaining leaf, c, is queried in parallel against them using relative-distance queries. Notice that the lowest common ancestor of a and b splits the tree into three parts. Given a and b, the other leaves are split into three subsets (R, A, and B) according to their query result (as shown in Figure 3(a)):

- A: leaves close to a, i.e., for which $closer(a, b, c) = (a, c)$
- B: leaves close to b, i.e., for which $closer(a, b, c) = (b, c)$
- R: remaining leaves, i.e., for which $closer(a, b, c) = (a, b)$

We then recursively construct the trees in parallel: T_a, for $A \cup \{a\}$; T_b, for $B \cup \{b\}$; and T_r, for R. The remaining challenge, of course, is to merge these trees to reconstruct the complete tree, T. The subtree of T formed by subset $A \cup B$ is rooted at an internal node, $v = lca(a, b)$; hence, we can create a new node, v, label it "$lca(a, b)$" and let T_a and T_b be v's children. If $R = \emptyset$, then we are done. Otherwise, we need to determine the parent of v in T; that is, we need to link v into T_r using function link(v, T_r).

To identify the parent of v, in T, let us assume inductively that each internal node $u \in T_r$ has a label "$lca(c, d)$", since we have already recursively labeled each internal node in T. Recall that v is labeled with "$lca(a, b)$". The crucial observation is to note if there exists an edge $(u \to w)$ in T_r, such that u is labeled "$lca(c, d)$" and $closer(a, c, d) = (a, z)$ for $z \in \{c, d\}$, and w is either leaf z or an ancestor of z labeled "$lca(e, f)$" with $closer(a, e, f) = (e, f)$, (See Figure 3(b)), then edge $(u \to w)$ must be where the parent of v belongs in T, and if

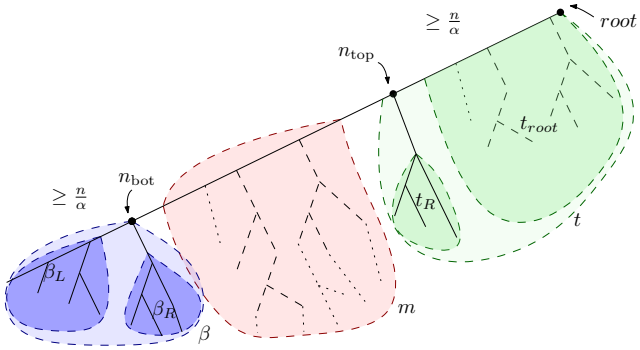


Figure 4 A left-heavy tree drawing displaying node n_{top}, node n_{bot} and the relevant partitions.

there is no such edge, the parent of v is the root of T and the sibling of v is the root of T_r. We can determine the edge $(u \to w)$ in a single parallel round by performing the query, $closer(a, c, d)$, for each each internal node $u \in T$ (where the label of u is "$lca(c, d)$"). The full pseudo-code of function $\mathsf{link}(v, T_r)$ is provided in the full version of the paper [2]. It is also worth noting that if the oracle can identify cases where all three leaves share a single lca, simple modifications to Algorithm 1 would enable it to handle trees of higher degree.

3.2 Analysis

The correctness of our algorithm follows from the way relative-distance queries always return a label for the lowest common ancestor for the two closest leaves among the three input nodes. Furthermore, executing the three recursive calls can be done in parallel, because $A \cup \{a\}$, $B \cup \{b\}$, and R form a partition of the set of leaves, L, and at every stage we only perform relative-distance queries relevant to the respective partition.

▶ **Theorem 6.** *Given a set, L, of n leaves in a proper binary tree, T, such as a biological phylogenetic tree, we can reconstruct T using relative-distance queries with a round complexity, $R(n)$, that is $O(\log n)$ and a query complexity, $Q(n)$, that is $O(n \log n)$, with high probability.*

Proof. Because the recursive calls we perform in each call to the reconstruct algorithm are done in parallel on a partition of the leaf nodes in L, we perform $\Theta(n)$ work per round. Thus, showing that the number of rounds, $R(n)$, is $O(\log n)$ w.h.p. also implies that $Q(n)$ is $O(n \log n)$ w.h.p. To prove this, we show that each round in Algorithm 1 has a constant probability of decreasing the problem size by at least a constant factor for each of its recursive calls. For analysis purposes, we consider the left-heavy representation of each tree, in which the tree rooted at the left child of any node is always at least as big as the tree rooted at its right child. (See Figure 4.) Using this view, we can characterize when a partition determines a "good split" and provide bounds on the sizes of the partitions, as follows.

▶ **Lemma 7.** *With probability of at least $\frac{\alpha-5}{2\alpha^2}$, a round of Algorithm 1 decreases the problem size of any recursive call by at least a factor of $\frac{\alpha-1}{\alpha}$, for a constant $\alpha > 5$, thus experiencing a **good split**.*

Proof. Define the ***spine*** to be all nodes on the path from the root to its left-most leaf in a left-heavy drawing. Let n_{bot} be the bottom-most node on the spine that has at least n/α of the nodes in its sub-tree. Conversely, let n_{top} be the parent of the top-most spine node that has at most $(1 - 1/\alpha) \cdot n$ descendants. (See Figure 4.) Consider the three resulting trees obtained from separating at the incoming edge to n_{bot} and the outgoing edge from n_{top} to

its left child. As shown in Figure 4, let t be the resulting tree retaining the root, β the tree rooted at n_{bot} and m the tree between n_{top} and n_{bot}. Within β let β_L and β_R be the trees rooted at the left and right child of n_{bot}. Similarly, for t, let t_R be the tree rooted at the right child of n_{top}. Finally, let t_{root} be the remaining tree when cut at n_{top}.

Consider the size of tree β, $|\beta|$, since this is the first tree rooted in the spine with over n/α nodes, then β_L must have had strictly under n/α nodes. Since the trees are in left-heavy order, β_R can have at most as many nodes as β_L so $\frac{n}{\alpha} \leq |\beta| < \frac{2n}{\alpha}$. Furthermore, we know that $|\beta|+|m|+|t_R|+|t_{root}|+1 = n$. Due to the left-heavy order, $|t_R| \leq |\beta|+|m|$. By definition of n_{top}, it's necessary that $|t_{root}| < \frac{n}{\alpha}$, thus $2(|\beta|+|m|)+1 > n-\frac{n}{\alpha}$ and $|\beta|+|m| > \left(\frac{\alpha-1}{2\alpha}\right)n-\frac{1}{2}$. Using the previous inequality and $|t| = n-|\beta|-|m|$, we find $|t| < \left(\frac{\alpha+1}{2\alpha}\right)n+\frac{1}{2}$. Also, $|m| = n-|t|-|\beta|$, so $|m| > n-\frac{5n+\alpha n}{2\alpha}$.

$$\frac{n}{\alpha} \leq |\beta| < \frac{2n}{\alpha}, \qquad \frac{n}{\alpha} \leq |t| < \left(\frac{\alpha+1}{2\alpha}\right)n+\frac{1}{2}, \qquad \left(\frac{\alpha-5}{2\alpha}\right)n < |m| \leq \left(\frac{\alpha-2}{\alpha}\right)n \quad (1)$$

Picking a leaf from β and another from m guarantees that $\beta \subseteq (A \cup \{a\})$ and $t \subseteq R$. Thus, using Equation (1), each of the three sub-problem sizes, $|A \cup \{a\}|$, $|B \cup \{b\}|$, and $|R|$, will be at most $\left(\frac{\alpha-1}{\alpha}\right)n$, when $\alpha > 5$. $Pr[\text{good split}] \geq Pr[\text{leaf in } \beta] \cdot Pr[\text{leaf in } m]$. Asymptotically, $Pr[\text{leaf in } \beta] \approx Pr[\text{node in } \beta]$, thus $Pr[\text{good split}] > \frac{\alpha-5}{2\alpha^2}$, which established the lemma. ◀

Returning to the proof of Theorem 6, let $p = \frac{\alpha-5}{2\alpha^2}$. From Lemma 7, we expect it will take $1/p$ rounds to obtain a good split. Every good split will reduce the problem-size by at least a constant factor, $\frac{\alpha-1}{\alpha}$. Thus, we are guaranteed to have just a single node left after we get $N_{\text{SPLITS}} = \log_{\frac{\alpha}{\alpha-1}}(n)$ good splits. Consider the geometric random variable, X_i, describing the number of rounds required to obtain the i-th good split, then $X = X_1 + \ldots + X_{N_{\text{SPLITS}}}$ describes the total number of rounds required by the algorithm. By linearity of expectation, $E[X] = \frac{2\alpha^2}{(\alpha-5)} \cdot N_{\text{SPLITS}} = \frac{2\alpha^2}{(\alpha-5)} \cdot \log_{\frac{\alpha}{\alpha-1}}(n)$. Therefore, since $\alpha > 5$ is a constant, this already implies an expected $O(\log n)$ rounds for Algorithm 1. Moreover, by a Chernoff bound for the sum of independent geometric random variables (see [23, 33]), $Pr\left[X > C \cdot E[X]\right] \leq e^{\frac{-(C-1)\cdot p \cdot N_{\text{SPLITS}}}{5}}$ for any constant $C \geq 3$ and constant $\alpha > 5$. Thus, the probability that we take over $C \cdot E[X]$ rounds is $O(1/n^{C-1})$. Therefore, by a union bound across the n leaves, our algorithm completes in $O(\log n)$ rounds w.h.p. ◀

▶ **Corollary 8.** *Algorithm 1 is optimal when asking $\theta(n)$ queries per round.*

The query complexity of Algorithm 1 matches an $\Omega(n \log n)$ lower bound for $Q(n)$, due to Kannan et al. [29]. Besides, we need $\Omega(\log n)$ rounds if we have $\theta(n)$ processors; hence, the round complexity of Algorithm 1 is also optimal.

4 Reconstructing Phylogenetic Trees from Path Queries

Let $T = (V, E, r)$ be a rooted (biological or digital) phylogenetic tree with fixed degree, d. In this section, we show how a querier can reconstruct T by issuing $Q(n) \in O(n \log n)$ path queries in $R(n) \in O(\log n)$ rounds, w.h.p., where $n = |V|$. We provide a lower bound to prove that our algorithm is optimal in terms of query complexity and round complexity. At the outset, the only thing we assume the querier knows is n and V, that is, the vertex set for T, and that the names of the nodes in V are unique, i.e., we may assume, w.l.o.g., that $V = \{1, 2, \ldots, n\}$. The querier doesn't know E or r – learning these is his goal.

4.1 Algorithms

We start by learning r, which we show can be done via any maximum-finding algorithm in Valiant's parallel model [44], which only counts parallel steps involving comparisons. The challenge, of course, is that the ancestor relationship in T is, in general, not a total order, as required by a maximum-finding algorithm. This does not actually pose a problem, however.

▶ **Lemma 9.** *Let A be a parallel maximum-finding algorithm in Valiant's model, with $O(f(n))$ span and $O(g(n))$ work. We can use A to find the root, r in a rooted tree $T = (V, E, r)$, using $R(n) \in O(f(n))$ rounds and $Q(n) \in O(g(n) + n)$ total queries.*

Proof. We pick an arbitrary vertex $v \in T$. In the first round, we perform queries $path(u, v)$ in parallel for every other vertex $u \in V$ to find S, the ancestor set for v. If $S = \emptyset$, then v is the root. Otherwise, we know all the vertices in a path from root to the parent of v, albeit unsorted. Still, note that for S the ancestor relation is a total order; hence, we can simulate A with path queries to resolve the comparisons made by A. We have just a single round and $O(n)$ queries more than what it takes for A to find the maximum. Thus, we can find the root in $O(f(n))$ rounds and $O(g(n) + n)$ queries. ◀

Thus, by well-known maximum-finding algorithms, e.g., see [11, 41, 44]:

▶ **Corollary 10.** *We can find r of a rooted tree $T = (V, E, r)$ deterministically in $O(\log \log n)$ rounds and $O(n)$ queries.*

Determining the rest of the structure of T is more challenging, however. At a high level, our approach to solving this challenge is to use a separator-based divide-and-conquer strategy, that is, find a "near" edge-separator in T, divide T using this edge, and recurse on the two remaining subtrees in parallel. The difficulty, of course, is that the querier has no knowledge of the edges of T; hence, the very first step, finding a "near" edge-separator, is a bottleneck computation. Fortunately, as we show in Lemma 11, if v is a randomly-chosen vertex, then, with probability depending on d, the path from root r to v includes an edge-separator.

▶ **Lemma 11.** *Let $T = (V, E, r)$ be a rooted tree of degree d and let v be a vertex chosen uniformly at random from V. Then, with probability at least $\frac{1}{d}$, an even-edge-separator is one of the edges on the path from r to v.*

Proof. By Lemma 5, T has an even-edge-separator. Let $e = (x, y)$ be an even-edge-separator for $T = (V, E, r)$ and let $T' = (V', E', y)$ be the subtree rooted at y when we remove e. Then, every path from r to each $v \in V'$ must contain e. By Definition 4, T' has at least $|V|/d$ vertices. Therefore, if we choose v uniformly at random from V, then with probability $\frac{|V'|}{|V|} \geq \frac{1}{d}$, the path from r to v contains e. ◀

▶ **Definition 12** (splitting-edge)**.** *In a degree-d rooted tree, an edge $(parent(s), s)$ is a splitting-edge if $\frac{|V|}{d+2} \leq D(s) \leq \frac{|V|(d+1)}{d+2}$, where $D(s)$ is the number of descendants of s.*

Note that a degree-d rooted tree T always has a splitting-edge, as every even-edge-separator is also a splitting-edge and by Lemma 5, it always has an even-edge-separator – a fact we use in our tree-reconstruction algorithm, which we describe next. This recursive algorithm (given in pseudo-code in Algorithm 2), assumes the existence of a randomized method, find-splitting-edge, which returns a splitting-edge in T, with probability $\Omega(1/d)$, and otherwise returns *Null*. Our reconstruction algorithm is therefore a randomized recursive algorithm that takes as input a set of vertices, V, with a (known) root vertex $r \in V$, and

returns the edge set, E, for V. At a high level, our algorithm is to repeatedly call the method, find-splitting-edge, until it returns a splitting-edge, at which point we divide the set of vertices using this edge and recurse on the two resulting subtrees.

Algorithm 2 Reconstruct a rooted tree with path queries.

```
1  Function reconstruct-rooted-tree(V, r):
2      E ← ∅
3      if |V| ≤ g then // g is a chosen constant
4          return edges found by a quadratic brute-force algorithm
5      while true do
6          Pick a vertex v ∈ V uniformly at random
7          for z ∈ V do in parallel
8              Perform query path(z, v)
9          Let Y be the vertex set of the path from r to v
10         splitting-edge ← find-splitting-edge(v, Y, V)
11         if splitting-edge ≠ Null then
12             (u, w) ← splitting-edge
13             E ← E ∪ {(u, w)}
14             for z ∈ V do in parallel
15                 Perform query path(w, z)
16             split V into V1, V2 at (u, w) using query results
17             parallel do
18                 E ← E ∪ reconstruct-rooted-tree(V1, w)
19                 E ← E ∪ reconstruct-rooted-tree(V2, r)
20             return E
```

In more detail, during each iteration of a repeating **while** loop, we choose a vertex $v \in V$ uniformly at random. Then, we find the vertices on the path from r to v and store them in a set, Y, using the fact that a vertex, z, is on the path from r to v if and only if $path(z, v) = 1$. Then, we attempt to find a splitting-edge using the function find-splitting-edge (shown in pseudo-code in Algorithm 3). If find-splitting-edge is unsuccessful, we give up on vertex v, and restart the **while** loop with a new choice for v. Otherwise, find-splitting-edge succeeded and we cut the tree at the returned splitting-edge, (u, w). All vertices, $z \in V$, where $path(w, z) = 1$ belong to the subtree rooted at w, thus belonging to V_1, whereas the remaining vertices belong to V_2 and the partition containing both u and rooted at r. Thus, after cutting the tree we recursively reconstruct-rooted-tree on V_1 and V_2.

The main idea for our efficient tree reconstruction algorithm lies in our find-splitting-edge method (see Algorithm 3), which we describe next. This method takes as input the vertex v, the vertex set Y, (comprising the vertices on the path from r to v), and the vertex set V. As we show, with probability depending on d, the output of this method is a splitting-edge; otherwise, the output is *Null*. Our algorithm performs a type of "noisy" search in Y to either locate a likely splitting-edge or return *Null* as an indication of failure.

Our find-splitting-edge algorithm consists of two phases. We enter **Phase 1** if the size of path Y is too big, i.e., $|Y| > |V|/K = \frac{|V|}{C_2 \log |V|}$, where C_2 is a predetermined constant and $K = C_2 \log |V|$. The purpose of this phase is either to pass a shorter path including an even-edge-separator to the second phase or to find a splitting-edge in this iteration. The search on the set Y is noisy, because it involves random sampling. In particular, we take a random sample S of size $m = C_1\sqrt{|V|}$ from path Y (where C_1 is a predetermined

Algorithm 3 Finding a splitting-edge from vertex set, Y, on the path from vertex v to the root r.

Function find-splitting-edge(v, Y, V)**:**
  splitting-edge $\leftarrow$ *Null*
  $m = C_1\sqrt{|V|}$, $K = C_2 \log |V|$
  Phase 1:
    if $|Y| > |V|/K$ **then**
      $S \leftarrow$ subset of m random elements from Y
      $S \leftarrow S \cup \{v, r\}$
      for *each* $s \in S$ **do in parallel**
        $X_s \leftarrow$ subset of K random elements from V
        Perform queries to find $count(s, X_s)$
      if $\forall s \in S : count(s, X_s) < \frac{K}{d+1}$ **then return** *Null*
      if $\forall s \in S : count(s, X_s) > \frac{Kd}{d+1}$ **then return** *Null*
      if $\exists s \in S : \frac{K}{d+1} \leq count(s, X_s) \leq \frac{Kd}{d+1}$ **then**
        return verify-splitting-edge(s, V)
      for *each* $\{a, b\} \in S$ **do in parallel**
        perform query $path(a, b)$
      Find w, z such that they are two consecutive nodes in the sorted order of S such that $count(w, X_w) > \frac{Kd}{d+1}$ and $count(z, X_z) < \frac{K}{d+1}$
      $Y \leftarrow$ nodes from Y in the path from w to z
    Phase 2:
      if $|Y| > |V|/K$ **then return** *Null*
      for *each* $s \in Y$ **do in parallel**
        $X_s \leftarrow$ subset of K random elements from V
        Perform queries to find $count(s, X_s)$
      if $\exists s \in Y$ s.t. $\frac{K}{d+1} \leq count(s, X_s) \leq \frac{Kd}{d+1}$ **then**
        return verify-splitting-edge(s, V)
  return *Null*

constant). We include r and v, the two endpoints of the path Y, to S. Then, we estimate the number of descendants of s, $D(s)$, for each $s \in S$. To estimate this number for each $s \in S$, we take a random sample X_s of K elements from V and we perform queries to find $count(s, X_s)$. Here, we use $m \cdot K \in O(\sqrt{|V|} \log |V|)$ queries in a single round. Then, if all the estimates were less than $K/(d+1)$, we return *Null* as an indication of failure (we guess that all the nodes on the path Y have too few descendants to be a separator). Similarly, if all the estimates were greater than $\frac{Kd}{d+1}$, we return *Null* (we guess that all the nodes on the path Y have too many descendants to be a separator). If there exists a node s such that $\frac{K}{d+1} \leq count(s, X_s) \leq \frac{Kd}{d+1}$, we check if s is a splitting-edge by counting its descendants using a function, verify-splitting-edge. This function takes vertex s and the full vertex set V to return edge (find-parent$(s, V), s$) if $\frac{|V|}{d+2} \leq count(s, V) \leq |V| \cdot \frac{d+1}{d+2}$ and return *Null* otherwise.

If neither of these three cases happens, we perform queries to sort elements of S using a trivial quadratic work parallel sort which takes $O(m^2) \in O(|V|)$ queries in a single round. We know that two consecutive nodes w and z exist on the sorted order of S, where $count(w, X_w) > \frac{Kd}{d+1}$ and $count(z, X_z) < \frac{K}{d+1}$. We find all the nodes on Y starting at w and ending at z, and use this as our new path Y.

Function find-parent(s, V).

find Y, ancestor set of s from V using $|V|$ queries in parallel
$m = C_1\sqrt{|V|}$
for $i \leftarrow 1$ ***to*** 2 & $|Y| > m$ **do**
    $S \leftarrow$ random subset of Y with m elements
    sort S as $x_1 < ... < x_m$ using $O(m^2)$ queries in parallel
    find $Y' = \{u \in Y \mid u \leq x_1\}$ using $O(|Y|)$ queries in parallel, replace Y with Y'
if $|Y| \leq m$ **then**
    sort Y using $O(m^2)$ queries in parallel
    return (minimum of this path)
return *Null*

In **Phase 2**, we expect a path of size under $|Y|/K$, we will later prove this is true with high probability. Otherwise, we just return *Null*. In this phase, we estimate the number of descendants much like we did in the previous phase, except the only difference is that we estimate the number of descendants for all the nodes on our new path Y. If there exists a node $s \in Y$ such that $\frac{K}{d+1} \leq count(s, X_s) \leq \frac{Kd}{d+1}$, we verify if it is a splitting-edge, as described earlier.

Finally, let us describe how we find the parent of a node s in V. We first find, Y, the set of ancestors of v in V in parallel using $|V|$ queries. Let $x \succ y$ describe the total order of nodes in path Y, where for any $x, y \in Y : x \succ y$ if and only if $path(x, y) = 1$. The parent of s is the lowest vertex on the path. Then, the key idea is that if $|Y| \in O(\sqrt{|V|})$, we can sort them using $O(|V|)$ queries. If the path is greater than this amount, we instead use S, a sample of size $O(\sqrt{|V|})$ from the path. Next, we sort the sample to obtain $x_1 < \ldots < x_m$ for S and then find all of the nodes in Y which are less than the smallest sample x_1. Finally, we replace Y with these descendants of x_1 and repeat the whole procedure again. We later prove that with high probability after two iterations of this sampling, the size of the path is $O(\sqrt{|V|})$, allowing us to sort all nodes in Y to return the minimum (see Function find-parent).

4.2 Analysis

The correctness of the algorithm follows from the fact that our method first finds the root, r, of T and then finds the parent of each other node, v in T.

▶ **Theorem 13.** *Given a set, V, of nodes of a rooted tree, T, such as a biological or digital phylogenetic tree, with degree bounded by a fixed constant, d, we can construct T using path queries with round complexity, $R(n)$, that is $O(\log n)$ and query complexity, $Q(n)$, that is $O(n \log n)$, with high probability.*

Our proof of Theorem 13 begins with Lemma 14.

▶ **Lemma 14.** *In a rooted tree, $T = (V, E, r)$, let Y be a (directed) path, where $|Y| > m = C_1\sqrt{|V|}$. If we take a sample, S, of m elements from Y, then with probability $1 - \frac{1}{|V|}$, every two consecutive nodes of S in the sorted order of S are within distance $O\left(\frac{|Y| \log |V|}{\sqrt{|V|}}\right)$ from each other in Y.*

Proof. Note that some nodes of Y may be picked more than once as we pick S in parallel. Divide the path Y into $\frac{\sqrt{|V|}}{\log |V|}$ equal size sections (the difference between the size of any two sections is at most 1). For each $1 \leq i \leq \frac{\sqrt{|V|}}{\log |V|}$, let A_i be the subset of S lying in the i^{th}

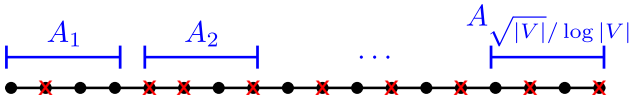


Figure 5 Illustration of how scattered sample S is on path Y. The i^{th} blue interval represents the i^{th} section of Y, the black dots correspond to the nodes on the path Y, and red crossed marks represent elements of S.

section of Y. (See Figure 5.) It is clear that each node $s \in S$ ends up in section i with probability $\frac{\log|V|}{\sqrt{|V|}}$, and therefore, for each $1 \le i \le \frac{\sqrt{|V|}}{\log|V|}$, $E[|A_i|] = C_1 \log|V|$. Thus, using standard a Chernoff bound, $\Pr[|A_i| = 0] < \frac{1}{|V|^2}$ for any constant $C_1 > 6\ln 10$. Using a union bound, all the sections are non-empty with probability at least $1 - \frac{1}{|V|}$. Hence, the distance between any two consecutive nodes of S from each other in Y is at most $\frac{2|Y|\log|V|}{\sqrt{|V|}}$. ◀

Lemma 14 allows us to analyze the find-parent method, as follows.

▶ **Lemma 15.** *The find-parent(s, V) method outputs the parent of s with probability at least $1 - 2/|V|$, with $Q(n) \in O(n)$ and $R(n) \in O(1)$.*

Proof. The find-parent method succeeds if, after the **for** loop, the size of the set of remaining ancestors of s, Y, is $|Y| \le m$, so it is enough to show that this occurs with probability at least $1 - 2/|V|$. By Lemma 14, the size of Y at the end of the first iteration is $|Y| \in O(m \log|V|)$, with probability at least $1 - 1/|V|$. Similarly, a second iteration, if required, further reduces the size of Y into $|Y| \in o(m)$, with probability at least $1 - 1/|V|$. Thus, by a union bound, the probability of success is at least $1 - 2/|V|$.

The query complexity can be broken down as follows, where $m \in O(\sqrt{|V|})$:

1. $O(|V|)$ queries in 1 round to determine the ancestor set, Y, of s.
2. $O(m^2) + O(|Y|) \in O(|V|)$ queries in 2 rounds for each of the (at most) 2 iterations performed in find-parent, whose purpose is to discard non-parent ancestors of s in Y.
3. $O(m^2) \in O(|V|)$ to find, in 1 round, the minimum among the remaining ancestors of Y (at most m w.h.p.). If $|Y| > m$, then no further queries are issued.

In total, the above amounts to $Q(n) \in O(n)$ and $R(n) \in O(1)$. ◀

We next analyze the find-splitting-edge method.

▶ **Lemma 16.** *Any call to find-splitting-edge returns true with probability $\frac{1}{2d}$; hence Algorithm 2 calls find-splitting-edge $O(d)$ times in expectation.*

Proof. By Lemma 11, we know that if we pick a vertex v, uniformly at random, then with probability $\frac{1}{d}$, an even-edge-separator lies on the path from r to v. We show that if there is such an even-edge-separator (Definition 4) on that path, find-splitting-edge(v, Y, V) returns a splitting-edge (Definition 12) with probability at least $\frac{1}{2}$, and otherwise returns *Null*. Using an intricate Chernoff-bound analysis (see full version for details [2]), we can prove that there exists a constant $C_2 > 0$, as used in line 3 of Algorithm 3 such that the following probability bounds always hold:

$$\begin{cases} \Pr\left(count(s, X_s) \ge \frac{K}{d+1}\right) \ge 1 - \frac{1}{|V|^2} & \text{if } count(s, V) \ge \frac{|V|}{d}, \\ \Pr\left(count(s, X_s) \le K\frac{d}{d+1}\right) \ge 1 - \frac{1}{|V|^2} & \text{if } count(s, V) \le |V|\frac{d-1}{d}, \end{cases} \tag{2}$$

$$\begin{cases} \Pr\left(count(s, X_s) < \frac{K}{d+1}\right) \geq 1 - \frac{1}{|V|^2} & \text{if } count(s, V) < \frac{|V|}{d+2}, \\ \Pr\left(count(s, X_s) > K\frac{d}{d+1}\right) \geq 1 - \frac{1}{|V|^2} & \text{if } count(s, V) > |V|\frac{d+1}{d+2} \end{cases} \tag{3}$$

It is clear that we either return a splitting-edge or *Null* when passing through verify-splitting-edge. We break the probability of returning *Null* according to the phases. We call a vertex v **ineligible** if $count(v, V) < \frac{|V|}{d+2}$ or $count(v, V) > \frac{|V|(d+1)}{d+2}$ ($(parent(v), v)$ is not a splitting-edge). On the other hand, we call vertex v **candidate** if after estimating the number of its descendants: $\frac{K}{d+1} \leq count(v, X) \leq \frac{Kd}{d+1}$. Let (a, b) be an even-edge-separator on path Y.

Phase 1:

- lines 10,11: By Definition 4, $\frac{|V|}{d} \leq count(b, V) \leq \frac{|V|(d-1)}{d}$. We add $\{r, v\}$ to S in line 6 of the algorithm (the two endpoints of path Y). Notice that $\frac{|V|}{d} \leq count(b, V) \leq count(r, V)$ and that $count(v, V) \leq count(b, V) \leq \frac{|V|(d-1)}{d}$. So, by Equation (2), with probability at least $1 - \frac{2}{|V|^2}$, $count(r, X_r) \geq \frac{K}{d+1}$ and $count(v, X_v) \leq \frac{Kd}{d+1}$, and consequently, we don't return *Null* in lines 10,11 of the algorithm.
- line 12: Equation (3) shows that an ineligible node is not a candidate with probability $1 - \frac{1}{|V|^2}$. Thus, by a union bound, none of our candidates is ineligible in line 12 with probability at least $1 - \frac{|S|}{|V|^2}$. Moreover, if there exists a candidate s in S, the algorithm outputs a splitting-edge $(parent(s), s)$ with probability at least $1 - \frac{2}{|V|}$, by Lemma 15.
- lines 15,16: Let us partition S into S_l and S_r (see Figure 6), as follows:

  $$S_l = \{s \in S \mid count(s, V) > count(b, V)\}, \qquad S_r = \{s \in S \mid count(s, V) < count(b, V)\}$$

  Then, by definition of b:

  $$\forall s \in S_l : \quad count(s, V) > |V|/d, \qquad \forall s \in S_r : \quad count(s, V) < |V|(d-1)/d.$$

  and thus, by Equation (2) and a union bound, we have with probability at least $1 - \frac{|S|}{|V|^2}$:

  $$\forall s \in S_l : \quad count(s, X_s) \geq K/(d+1), \qquad \forall s \in S_r : \quad count(s, X_s) \leq Kd/(d+1).$$

  Finally, since S does not contain any candidate nodes (otherwise we would have picked them in line 12), the above inequalities imply that:

  $$\forall s \in S_l : \quad count(s, X_s) > Kd/(d+1), \qquad \forall s \in S_r : \quad count(s, X_s) < K/(d+1).$$

  Therefore, $w \in S_l$ and $z \in S_r$, which implies that the subpath from w to z in Y must include vertex b. This means that with probability $1 - O(\frac{1}{|V|})$, we either find a splitting-edge in this phase or pass b to the next phase.

Now, consider **Phase 2:**

- line 17: Here, if $|Y| > |V|/K$, then we have passed through Phase 1. Using Lemma 14, we know that since S was a sample of size $C_1\sqrt{|V|}$ from Y, with probability $1 - \frac{1}{|V|}$ the distance between any two consecutive nodes of S in Y was $O(|Y|\frac{\log|V|}{\sqrt{|V|}}) \in O(\sqrt{|V|}\log|V|)$. Thus, the size of path Y after passing through line 15 is at most $O(\sqrt{|V|}\log|V|)$. Thus, the probability of returning *Null* in line 17 is at most $\frac{2}{|V|}$.

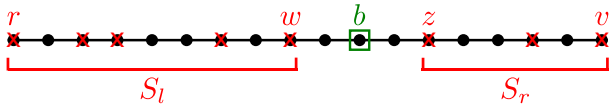


Figure 6 Illustration of the path reduction in Phase 1 of find-splitting-edge. At the end of this phase, the path Y is trimmed down into the subpath consisting of the nodes between w and z, which contains b w.h.p.

- line 21: By Equation (3), with probability at least $1 - \frac{|V|}{|V|^2}$, no ineligible node is between candidate set. Besides, for a candidate node s, the algorithm outputs a splitting-edge $(parent(s), s)$ with probability at least $1 - \frac{2}{|V|}$, by Lemma 15.
- line 22: The probability that we return *Null* here is equal to the probability that our candidate set in line 21 is empty. By Equation (2), with probability at least $1 - \frac{2}{|V|^2}$, b is between candidates at line 21 and candidate set is non-empty. Thus, the total probability of failing to return a splitting-edge in this phase is at most $O(\frac{1}{|V|})$.

Therefore, for $|V|$ greater than the chosen constant g, the probability of returning *Null* in the existence of an even-edge-separator is at most $O(\frac{1}{|V|}) \leq 1/2$. Thus, the probability of returning a splitting-edge in any call to find-splitting-edge is at least $\frac{1}{2d}$. ◀

▶ **Lemma 17.** *The subroutine find-splitting-edge(v, Y, V) has query complexity, $Q(n)$, that is $O(|V|)$, and round complexity, $R(n)$, that is $O(1)$.*

Proof. The queries done by find-splitting-edge(v, Y, V), in the worst case, can be broken down as follows, where $m = O(\sqrt{|V|})$:

Phase 1: A total of $O(|V|)$ queries in $O(1)$ rounds, consisting of:
- $O(mK) \in O(\sqrt{|V|}\log|V|)$ queries in one round for estimating the number of descendants for the m samples.
- $O(m^2) \in O(|V|)$ queries in one round for sorting the m samples.
- $O(|Y|) \in O(|V|)$ queries in a round to find the subpath of Y that is the input for Phase 2.
- $O(|V|)$ queries in $O(1)$ rounds for determining the parent of s (see Lemma 15).

Phase 2: If we enter this phase, it spends $O(|V|)$ queries in $O(1)$ rounds:
- $|Y| \cdot K \in O(|V|)$ queries in one round to evaluate count(s, X_s) for each $s \in Y$.
- $O(|V|)$ queries in one round to find the number of descendants of node s.
- $O(|V|)$ queries in $O(1)$ rounds to determine the parent of s (see Lemma 15).

Overall, the above break down amounts to $Q(n) \in O(n)$ and $R(n) \in O(1)$. ◀

Now, recall Theorem 13: *Given a set, V, of nodes of a rooted tree, T, such as a biological or digital phylogenetic tree, with degree bounded by a fixed constant, d, we can construct T using path queries with round complexity, $R(n)$, that is $O(\log n)$ and query complexity, $Q(n)$, that is $O(n \log n)$, with high probability.*

Proof. The expected query complexity $Q(n)$ of Algorithm 2 is dominated by the two recursive calls $\left(Q\left(\frac{n}{d+2}\right)\right.$ and $\left.Q\left(\frac{n(d+1)}{d+2}\right)\right)$ and the calls to find-splitting-edge. By Lemma 16, we call find-splitting-edge an expected $O(d)$ times, incurring a cost of $O(dn) \in O(n)$ path queries in $O(d) \in O(1)$ rounds (see Lemma 17). Thus, $Q(n)$ and $R(n)$ are:

$$Q(n) = Q\left(\frac{n}{d+2}\right) + Q\left(\frac{n(d+1)}{d+2}\right) + O(n),$$

$$R(n) = \max\left(R\left(\frac{n}{d+2}\right), R\left(\frac{n(d+1)}{d+2}\right)\right) + O(1)$$

which shows it needs $Q(n) \in O(n \log n)$ and $R(n) \in O(\log n)$ in expectation. To prove the high probability results, note that the main algorithm is a divide-and-conquer algorithm with two recursive calls per call; hence, it can be modeled with a recursion tree that is a binary tree, B, with height $h = O(\log_{\frac{d+2}{d+1}} n) = O(\log n)$. For any root-to-leaf path in B, the time taken can modeled as a sum of independent random variables, $X = X_1 + X_2 + \cdots + X_h$, where each X_i is the number of calls to find-splitting-edge (each of which uses $O(|V|)$ queries in $O(1)$ rounds) required before it returns true, which is a geometric random variable with parameter $p = \frac{1}{2d}$. Thus, by a Chernoff bound for sums of independent geometric random variables (e.g., see [23, 33]), the probability that X is more than $O(d \log_{\frac{d+2}{d+1}} n)$ is at most $1/n^{c+1}$, for any given constant $c \geq 1$. The theorem follows, then, by a union bound for the n root-to-leaf paths in B. ◀

4.3 Lower Bound

We establish the following simple lower bound, which extends and corrects lower-bound proofs of Wang and Honorio [46].

▶ **Theorem 18.** *Reconstructing an n-node, degree-d tree requires $\Omega(dn + n \log n)$ path queries. This lower bound holds for the worst case of a deterministic algorithm and for the expected value of a randomized algorithm.*

Proof. Consider an n-node, degree-d tree, T, as shown in Figure 7, which consists of a root, r, with d children, each of which is the root of a chain, T_i, of at least one node rooted at a child of r. Since a querier, Bob, can determine the root, r, in $O(n)$ queries anyway, let us assume for the sake of a lower bound that r is known; hence, no additional information is gained by path queries involving the root. Let us denote the vertices in chain T_i as V_i. In order to reconstruct T, Bob must determine the nodes in each V_i and must also determine their order in T_i. For a given path query, $path(u, v)$, say this query is ***internal*** if $u, v \in V_i$, for some $i \in [1, d]$, and this query is ***external*** otherwise. Note that even if Bob knows the full structure of T except for a given node, v, he must perform at least $d - 1$ external queries in the worst case, for a deterministic algorithm, or $\Omega(d)$ external queries in expectation, for a randomized algorithm, in order to determine the chain, T_i, to which v belongs. Furthermore, the result of an (internal or external) query, $path(u, v)$, provides no additional information for a vertex w distinct from u and v regarding the set, V_i, to which w belongs. Thus, Bob must perform $\Omega(d)$ external queries for each vertex $v \neq r$, i.e., he must perform $\Omega(dn)$ external queries in total. Moreover, note that the results of external queries involving a vertex, v,

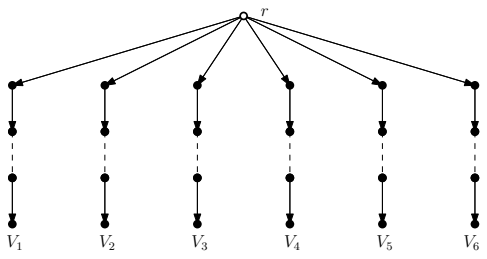


Figure 7 Illustration of the $\Omega(dn + n \log n)$ lower bound for path queries in directed rooted trees (shown for $d = 6$).

provide no information regarding the location of v in its chain, T_i. Even if Bob knows all the vertices that comprise each V_i, he must determine the ordering of these vertices in the chain, T_i, in order to reconstruct T. That is, Bob must ***sort*** the vertices in V_i using a comparison-based algorithm, where each comparison is an internal query involving two vertices, $u, v \in V_i$. By well-known sorting lower bounds (which also hold in expectation for randomized algorithms), e.g., see [23, 12], determining the order of the vertices in each T_i requires $\Omega(|V_i| \log |V_i|)$, as one of the chain can be as great as $n - d$ vertices, then he needs $\Omega(n \log n)$ internal queries. ◀

▶ **Corollary 19.** *Algorithm 2 is optimal for bounded-degree trees when asking $\theta(n)$ queries per round.*

The query complexity of Algorithm 2 matches the lower bound provided by Theorem 18 when d is constant. Besides, we need $\Omega(d + \log n)$ rounds if we have $\theta(n)$ processors; hence, the round complexity of Algorithm 2 is also optimal.

5 Experiments

Given that both of our algorithms are randomized and perform optimally with high probability, we carried out experiments to analyze the practicality of our algorithms and compare their performance with the best known algorithms for reconstructing rooted trees. [4]

5.1 Reconstructing Biological Phylogenetic Trees

In order to assess the practical performance of Algorithm 1, we performed experiments using synthetic phylogenetic trees and real-world biological phylogenetic trees from the phylogenetic library TreeBase [36], which is a database of biological phylogenetic trees, comprising over 100,000 distinct taxa in total.

[4] The complete source code for our experiments, including the implementation of our algorithms and the algorithms we compared against, is available at github.com/UC-Irvine-Theory/ParallelTreeReconstruction.

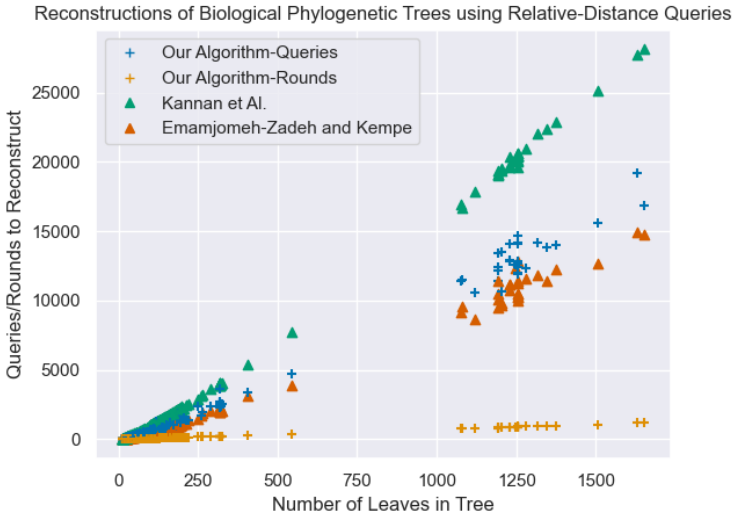


Figure 8 A scatter plot showing the number of queries and rounds for each of the three tree reconstruction algorithms for real trees from TreeBase. Since our algorithm is parallel, we include round complexity to serve as a comparison for the sequential complexity.

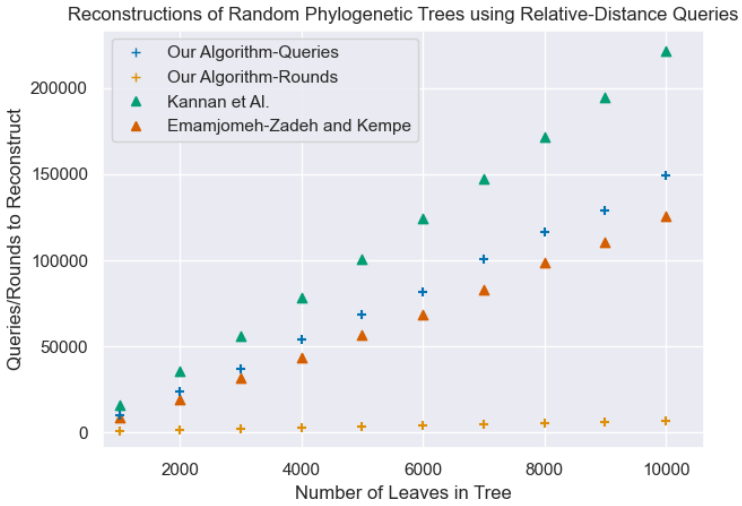


Figure 9 A plot showing the average number of queries and rounds for each of the three tree reconstruction algorithms. Each data point represents the average for 10 randomly generated trees.

We implemented an oracle interface, instantiated it with the relevant trees, and implemented our algorithm along with two other phylogenetic tree reconstruction algorithms that use relative-distance queries. The first is by Emamjomeh-Zadeh and Kempe [18], which is a randomized sequential divide-and-conquer algorithm. The second is by Kannan et al. [29], where they use a sequential deterministic procedure reminiscent of insertion-sort. All three algorithms achieve the optimal asymptotic query complexity of $\Theta(n \log n)$ in expectation.

Real Data. We instantiated our oracles with 1,220 biological phylogenetic trees from the TreeBase collection and used them to run all three algorithms. The results, shown in Figure 8, suggest that our algorithm outperforms the algorithm by Kannan et al., both in terms of its round complexity and query complexity. However our algorithm almost matches Emamjomeh-Zadeh and Kempe's in terms of total queries and we believe the small difference is a direct result of the cost incurred while parallelizing the link step of Algorithm 1. It remains clear that Algorithm 1 outperforms the two other algorithms when considering the parallel speed-up.

Synthetic Data. We also tested this algorithm using synthetic data and found similar results, detailed in Figure 9. We detail the method used to generate these random tree instances in Section 5.2, however, given our algorithm's strict focus on biological phylogenetic trees, we use only full binary trees, where each internal node has exactly two children.

5.2 Reconstructing Phylogenetic Trees from Path Queries

To assess the practical performance of our method for reconstructing (biological and digital) phylogenetic trees from path queries, we performed experiments using both synthetic and real data to compare our algorithm with the algorithm by Wang and Honorio [46], which is the best known reconstruction algorithm for phylogenetic trees from path queries. Our experimental results provide evidence that Algorithm 2 provides significant parallel speedup, while simultaneously improving the total number of queries.

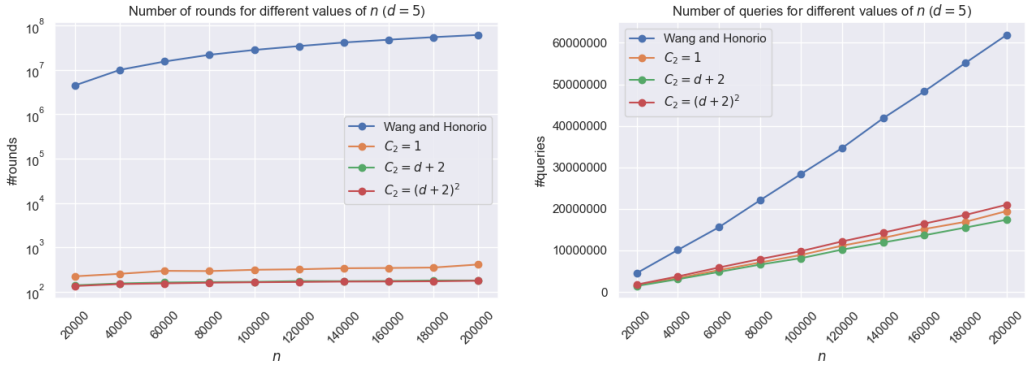


Figure 10 Comparing Our Algorithm's number of rounds (left) and total queries (right) with Wang and Honorio's [46], for fixed $d = 5$ and varying n.

Synthetic Data. In order to generate random instances of trees with maximum degree, d, we synthesized a data set of random degree-d trees of n nodes for different values of n and d. To generate a random tree, T, for a given n and d, we first generated a random Prüfer sequence [37] of a labeled tree, which defines a unique sequence associated with that tree, and converted it to its associated tree. In particular, each n-node tree $T = (V, E)$ has a unique code sequence $s_1, s_2, \ldots, s_{n-2}$, where $s_i \in V$ for all $1 \leq i \leq n-2$ and every node $v_i \in V$ of degree d_i appears exactly $d_i - 1$ times in this sequence. Therefore, in order to generate random degree-d rooted trees we generate a random Prüfer sequence while imposing conditions that: (i) all vertices appear at most $d-1$ times in the code and (ii) there is at least one node such that it appears exactly $d-1$ times. Converting each such sequence to its associated tree gave us a random degree-d tree instance that we used in our experiments.

Since our parallel reconstruction algorithm using path queries is parameterized by a constant, C_2, we ran our algorithm using different values for C_2. The constant C_2 controls sample size from V used to estimate the number of descendants of a node. Furthermore, to reduce noise from randomization, each data-point will be averaged for 3 runs on 10 randomly generated trees. In Figure 10, we compare our algorithm's rounds and total number of queries with the one by Wang and Honorio [46], for fixed degree trees $d = 5$ and varying tree-sizes. These results provide empirical evidence that our algorithm provides a noticeable speedup in parallel round complexity while also outperforming the algorithm by Wang and Honorio [46] in total number of queries.

In Figure 11, we compare Algorithm 2 with the one by Wang and Honorio [46] for fixed size and varying values of d. Again, this supports our theoretical findings that our algorithm achieves both a significant parallel speedup and a simultaneous improvement in the number of total queries.

In Figure 12, we study the behavior of Algorithm 2 under different values of C_2, so as to experimentally find the best value for C_2. While our high probability analysis requires $C_2 \approx (d+2)^4$, Figure 12 suggests that we do not need that high probability reassurance in practice, and we can use smaller sample to reduce the total number of queries.

Real Data. Our experiments on real-world biological phylogenetic trees also confirm the superiority of our algorithm in terms of performance as compared to the one by Wang and Honorio [46]. Similar to our experiments using relative-distance queries, we used a dataset comprised of trees from the phylogenetic library TreeBase [36]. Figure 13 summarizes our experimental results, where each data point corresponds to an average performance of 3 runs

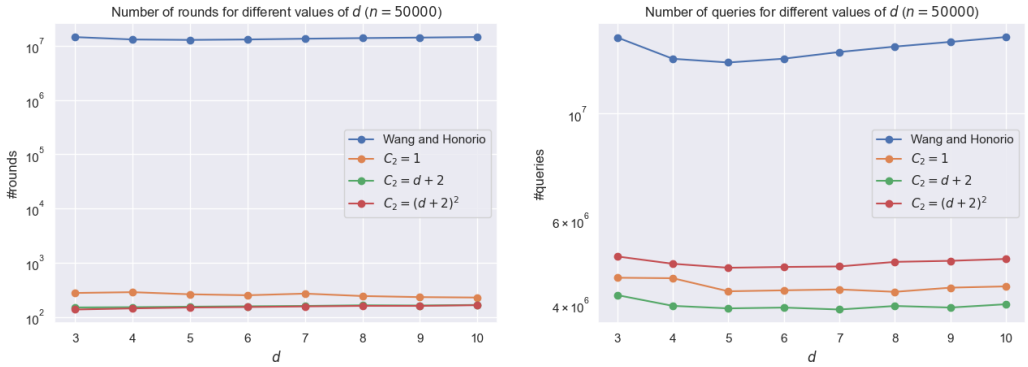


Figure 11 Comparing our algorithm's number of rounds (left) and total queries (right) with Wang and Honorio's [46], for $n = 50000$ and varying values for d.

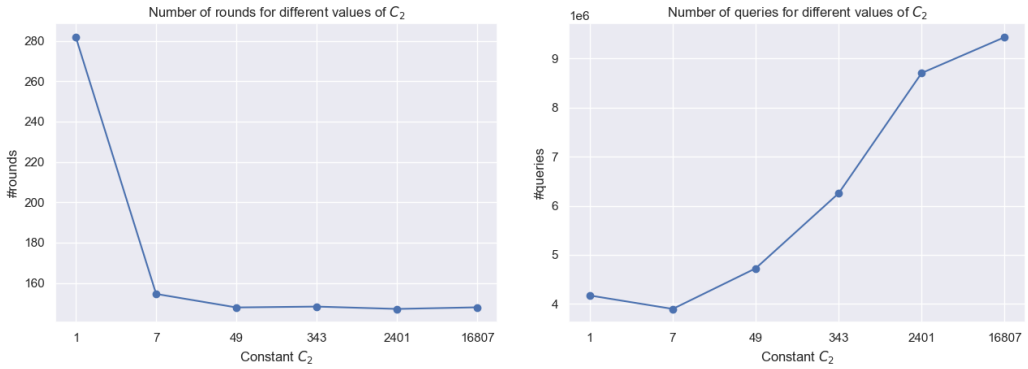


Figure 12 Change in the total number of rounds (left) and total number of queries (right) when running our algorithm for varying values of C_2 ($n = 50000$, $d = 5$).

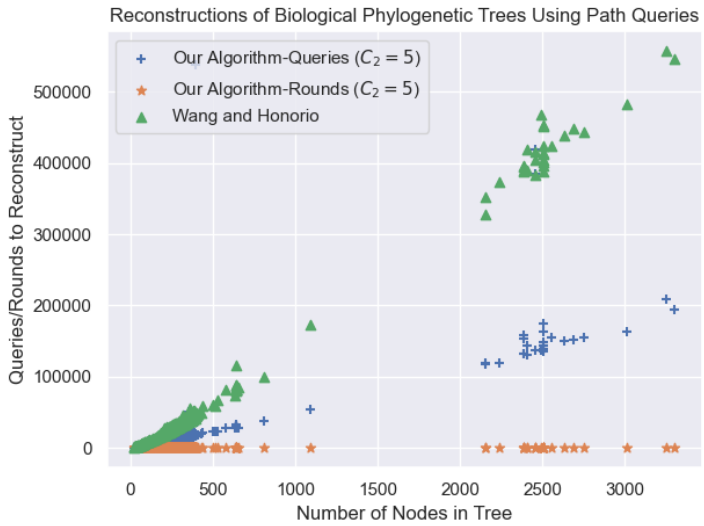


Figure 13 A scatter plot comparing the number of queries and rounds of our algorithm and with the one by Wang and Honorio [46] for real-world trees from TreeBase [36]. Since our algorithm is parallel, we include round complexity to serve as a comparison for the sequential complexity.

on the same tree. Our algorithm is superior in both queries and rounds for all the values of C_2 we tried: $C_2 \in \{1, d+2, (d+2)^2\}$. The best performance corresponds to $C_2 = d+2 = 5$, which is the one illustrated in Figure 13.

References

1 Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. The query complexity of finding a hidden permutation. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2013. doi:10.1007/978-3-642-40273-9_1.

2 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osagueda. Reconstructing biological and digital phylogenetic trees in parallel. *ArXiv*, 2020. arXiv:2006.15259.

3 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Reconstructing binary trees in parallel dimension (brief announcement). In *The 32nd ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2020, PA, USA, July 14-17, 2020*, to appear.

4 Gautam Altekar, Sandhya Dwarkadas, John P. Huelsenbeck, and Fredrik Ronquist. Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference. *Bioinform.*, 20(3):407–415, 2004. doi:10.1093/bioinformatics/btg427.

5 Anna Bernasconi, Carsten Damm, and Igor E. Shparlinski. Circuit and decision tree complexity of some number theoretic problems. *Inf. Comput.*, 168(2):113–124, 2001. doi:10.1006/inco.2000.3017.

6 Paolo Bestagini, Marco Tagliasacchi, and Stefano Tubaro. Image phylogeny tree reconstruction based on region selection. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 2059–2063. IEEE, 2016. doi:10.1109/ICASSP.2016.7472039.

7 Anupam Bhattacharjee, Kazi Zakia Sultana, and Zalia Shams. Dynamic and parallel approaches to optimal evolutionary tree construction. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering, CCECE 2006, May 7-10, 2006, Ottawa Congress Centre, Ottawa, Canada*, pages 119–122. IEEE, 2006. doi:10.1109/CCECE.2006.277582.

8 Gerth Stølting Brodal, Rolf Fagerberg, Christian N. S. Pedersen, and Anna Östlin. The complexity of constructing evolutionary trees using experiments. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2001. doi:10.1007/3-540-48224-5_12.

9 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. *Artif. Intell.*, 174(9-10):551–569, 2010. doi:10.1016/j.artint.2010.02.003.

10 Benny Chor and Tamir Tuller. Maximum likelihood of evolutionary trees: hardness and approximation. In *Proceedings Thirteenth International Conference on Intelligent Systems for Molecular Biology 2005, Detroit, MI, USA, 25-29 June 2005*, pages 97–106, 2005. doi:10.1093/bioinformatics/bti1027.

11 Richard Cole and Uzi Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 206–219. ACM, 1986. doi:10.1145/12130.12151.

12 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

13 Joseph C. Culberson and Piotr Rudnicki. A fast algorithm for constructing trees from distance matrices. *Inf. Process. Lett.*, 30(4):215–220, 1989. doi:10.1016/0020-0190(89)90216-0.

14 Zanoni Dias, Siome Goldenstein, and Anderson Rocha. Exploring heuristic and optimum branching algorithms for image phylogeny. *J. Vis. Commun. Image Represent.*, 24(7):1124–1134, 2013. doi:10.1016/j.jvcir.2013.07.011.

15 Zanoni Dias, Siome Goldenstein, and Anderson Rocha. Large-scale image phylogeny: Tracing image ancestral relationships. *IEEE Multim.*, 20(3):58–70, 2013. doi:10.1109/MMUL.2013.17.

16 Zanoni Dias, Anderson Rocha, and Siome Goldenstein. Image phylogeny by minimal spanning trees. *IEEE Trans. Information Forensics and Security*, 7(2):774–788, 2012. doi:10.1109/TIFS.2011.2169959.

17 Shahar Dobzinski and Jan Vondrák. From query complexity to computational complexity. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1107–1116. ACM, 2012. doi:10.1145/2213977.2214076.

18 Ehsan Emamjomeh-Zadeh and David Kempe. Adaptive hierarchical clustering using ordinal queries. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 415–429. SIAM, 2018. doi:10.1137/1.9781611975031.28.

19 James S. Farris. Methods for Computing Wagner Trees. *Systematic Biology*, 19(1):83–92, March 1970. doi:10.1093/sysbio/19.1.83.

20 Joseph Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of molecular evolution*, 17(6):368–376, 1981.

21 Walter M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology*, 20(4):406–416, 1971. URL: http://www.jstor.org/stable/2412116.

22 Leslie Ann Goldberg, Paul W. Goldberg, Cynthia A. Phillips, and Gregory B. Sorkin. Constructing computer virus phylogenies. *J. Algorithms*, 26(1):188–208, 1998. doi:10.1006/jagm.1997.0897.

23 M. T. Goodrich and R. Tamassia. *Algorithm Design and Applications*. Wiley, New York, NY, 2011.

24 Jotun J Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of mathematical biology*, 51(5):597–603, 1989.

25 John P Huelsenbeck. Performance of phylogenetic methods in simulation. *Systematic biology*, 44(1):17–48, 1995.

26 M. Jagadish and Anindya Sen. Learning a bounded-degree tree using separator queries. In Sanjay Jain, Rémi Munos, Frank Stephan, and Thomas Zeugmann, editors, *Algorithmic Learning Theory - 24th International Conference, ALT 2013, Singapore, October 6-9, 2013. Proceedings*, volume 8139 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2013. doi:10.1007/978-3-642-40935-6_14.

27 Jeong-Hoon Ji, Su-Hyun Park, Gyun Woo, and Hwan-Gue Cho. Generating pylogenetic tree of homogeneous source code in a plagiarism detection system. *International Journal of Control, Automation, and Systems*, 6(6):809–817, 2008.

28 Neil C Jones, Pavel A Pevzner, and Pavel Pevzner. *An introduction to bioinformatics algorithms*. MIT press, 2004.

29 Sampath Kannan, Eugene L. Lawler, and Tandy J. Warnow. Determining the evolutionary tree using experiments. *J. Algorithms*, 21(1):26–50, 1996. doi:10.1006/jagm.1996.0035.

30 Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. *ACM Trans. Algorithms*, 14(4):40:1–40:30, 2018. doi:10.1145/3199606.

31 Valerie King, Li Zhang, and Yunhong Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 444–453. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644179.

32 Guilherme D Marmerola, Marina A Oikawa, Zanoni Dias, Siome Goldenstein, and Anderson Rocha. On the reconstruction of text phylogeny trees: evaluation and analysis of textual relationships. *PloS one*, 11(12):e0167822, 2016.

33 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

34 Mark Pagel. Inferring the historical patterns of biological evolution. *Nature*, 401(6756):877–884, 1999.

35 Avi Pfeffer, Catherine Call, John Chamberlain, Lee Kellogg, Jacob Ouellette, Terry Patten, Greg Zacharias, Arun Lakhotia, Suresh Golconda, John Bay, Robert Hall, and Daniel Scofield. Malware analysis and attribution using genetic information. In *7th International Conference on Malicious and Unwanted Software, MALWARE 2012, Fajardo, PR, USA, October 16-18, 2012*, pages 39–45. IEEE Computer Society, 2012. doi:10.1109/MALWARE.2012.6461006.

36 WH Piel, L Chan, MJ Dominus, J Ruan, RA Vos, and V Tannen. Treebase v. 2: A database of phylogenetic knowledge. e-biosphere, 2009.

37 Heinz Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys*, 27(1918):742–744, 1918.

38 Lev Reyzin and Nikhil Srivastava. On the longest path algorithm for reconstructing trees from distance matrices. *Inf. Process. Lett.*, 101(3):98–100, 2007. doi:10.1016/j.ipl.2006.08.013.

39 F. James Rohlf. J. felsenstein, inferring phylogenies, sinauer assoc., 2004, pp. xx + 664. *J. Classif.*, 22(1):139–142, 2005. doi:10.1007/s00357-005-0009-4.

40 Bingyu Shen, Christopher W. Forstall, Anderson de Rezende Rocha, and Walter J. Scheirer. Practical text phylogeny for real-world settings. *IEEE Access*, 6:41002–41012, 2018. doi:10.1109/ACCESS.2018.2856865.

41 Yossi Shiloach and Uzi Vishkin. Finding the maximum, merging and sorting in a parallel computation model. In Wolfgang Händler, editor, *CONPAR 81: Conference on Analysing Problem Classes and Programming for Parallel Computing, Nürnberg, Germany, June 10-12, 1981, Proceedings*, volume 111 of *Lecture Notes in Computer Science*, pages 314–327. Springer, 1981. doi:10.1007/BFb0105127.

42 Gábor Tardos. Query complexity, or why is it difficult to seperate NP ^{a} cap co np^{a} from p^{a} by random oracles a? *Combinatorica*, 9(4):385–392, 1989. doi:10.1007/BF02125350.

43 W.T. Tutte. *Graph Theory*. Cambridge Mathematical Library. Cambridge University Press, 2001. URL: https://books.google.com/books?id=uTGhooU37h4C.

44 Leslie G. Valiant. Parallelism in comparison problems. *SIAM J. Comput.*, 4(3):348–355, 1975. doi:10.1137/0204030.

45 Leslie G. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Computers*, 30(2):135–140, 1981. doi:10.1109/TC.1981.6312176.

46 Zhaosen Wang and Jean Honorio. Reconstructing a bounded-degree directed tree using path queries. In *57th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2019, Monticello, IL, USA, September 24-27, 2019*, pages 506–513. IEEE, 2019. doi:10.1109/ALLERTON.2019.8919924.

47 Michael S Waterman, Temple F Smith, Mona Singh, and William A Beyer. Additive evolutionary trees. *Journal of theoretical Biology*, 64(2):199–213, 1977.

48 Andrew Chi-Chih Yao. Decision tree complexity and betti numbers. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 615–624. ACM, 1994. doi:10.1145/195058.195414.

Kruskal-Based Approximation Algorithm for the Multi-Level Steiner Tree Problem

Reyan Ahmed
University of Arizona, Tucson, AZ, USA
abureyanahmed@email.arizona.edu

Faryad Darabi Sahneh
University of Arizona, Tucson, AZ, USA
faryad@email.arizona.edu

Keaton Hamm
University of Texas at Arlington, Arlington, TX, USA
keaton.hamm@uta.edu

Stephen Kobourov
University of Arizona, Tucson, AZ, USA
kobourov@cs.arizona.edu

Richard Spence
University of Arizona, Tucson, AZ, USA
rcspence@email.arizona.edu

Abstract

We study the multi-level Steiner tree problem: a generalization of the Steiner tree problem in graphs where terminals T require varying priority, level, or quality of service. In this problem, we seek to find a minimum cost tree containing edges of varying rates such that any two terminals u, v with priorities $P(u)$, $P(v)$ are connected using edges of rate $\min\{P(u), P(v)\}$ or better. The case where edge costs are proportional to their rate is approximable to within a constant factor of the optimal solution. For the more general case of non-proportional costs, this problem is hard to approximate with ratio $c \log \log n$, where n is the number of vertices in the graph. A simple greedy algorithm by Charikar et al., however, provides a $\min\{2(\ln |T| + 1), \ell\rho\}$-approximation in this setting, where ρ is an approximation ratio for a heuristic solver for the Steiner tree problem and ℓ is the number of priorities or levels (Byrka et al. give a Steiner tree algorithm with $\rho \approx 1.39$, for example).

In this paper, we describe a natural generalization to the multi-level case of the classical (single-level) Steiner tree approximation algorithm based on Kruskal's minimum spanning tree algorithm. We prove that this algorithm achieves an approximation ratio at least as good as Charikar et al., and experimentally performs better with respect to the optimum solution. We develop an integer linear programming formulation to compute an exact solution for the multi-level Steiner tree problem with non-proportional edge costs and use it to evaluate the performance of our algorithm on both random graphs and multi-level instances derived from SteinLib.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases multi-level, Steiner tree, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.4

Supplementary Material All algorithms, implementations, the ILP solver, experimental data and analysis are available on Github at `https://github.com/abureyanahmed/Kruskal_based_approximation`.

Funding The research for this paper was partially supported by NSF grants CCF-1740858, CCF-1712119, and DMS-1839274.

1 Introduction

We study the following generalization of the Steiner tree problem where terminals have priorities, levels, or quality of service (QoS) requirements. Variants of this problem are known in the literature under different names including multi-level network design (MLND), quality-of-service multicast tree (QoSMT) [4], quality-of-service Steiner tree [10, 18], and Priority Steiner Tree [6]. Motivated by multi-level graph visualization, we refer to this problem as the multi-level Steiner tree problem.

▶ **Definition 1** (Multi-level Steiner tree (MLST)). *Let $G = (V, E)$ be a connected graph, and $T \subseteq V$ be a subset of terminals. Each terminal $t \in T$ has a priority $P(t) \in \{1, 2, \ldots, \ell\}$. A multi-level Steiner tree (MLST) is a tree G' with edge rates $y(e) \in \{1, 2, \ldots, \ell\}$ such that for any two terminals u, $v \in T$, the u–v path in G' uses edges of rate greater than or equal to* $\min\{P(u), P(v)\}$.

We use 1 for the lowest priority and ℓ for the highest, and assume without loss of generality that there exists $v \in V$ such that $P(v) = \ell$. If $\ell = 1$, then Definition 1 reduces to the definition of Steiner tree.

The cost of an MLST G' is defined as the sum of the edge costs in G' at their respective rates. Specifically, for $1 \leq i \leq \ell$, we denote by $c_i(e)$ the cost of including edge e with rate i, in which the cost of an MLST is $\sum_{e \in E(G')} c_{y(e)}(e)$. Naturally, an edge with a higher rate should be more costly, so we assume that $c_1(e) \leq c_2(e) \leq \ldots \leq c_\ell(e)$ for all $e \in E$. The MLST problem is to compute an MLST with minimum cost.

We note that equivalent formulations [4, 6] include a root (or source) vertex $r \in V$ in which the problem is to compute a tree rooted at r such that the path from r to every terminal $t \in T$ uses edges of rate at least as good as $P(t)$. One can observe that Definition 1 is equivalent to this formulation as we can fix the root to be any terminal $r \in T$ such that $P(r) = \ell$. In an optimized MLST, the path from the root to any terminal uses non-increasing edge rates. Note that this becomes relevant for the discussion of the exact value of the approximation given by our algorithm and the state-of-the-art algorithm [4]. We use the phrase "multi-level" since a tree G' with a root having top priority and edge rates $y(\cdot)$ induces a sequence of ℓ nested Steiner trees, where the tree induced by $\{e \in E : y(e) \geq i\}$ is a Steiner tree over terminals $T_i = \{t \in T : P(t) \geq i\}$ for $1 \leq i \leq \ell$.

We distinguish the special case with proportional costs, where the cost of an edge is equal to its rate multiplied by some "base cost" (e.g., $c_1(e)$). This is similar to the rate model in [4] as well as the setup in [10].

▶ **Definition 2.** *An instance of the MLST problem has proportional costs if $c_i(e) = ic_1(e)$ for all $e \in E$ and for all $i \in \{1, 2, \ldots, \ell\}$. Otherwise, the instance has non-proportional costs.*

For $u, v \in T$, we define $\sigma(u, v)$ to equal the cost of a minimum cost u–v path in G using edges of rate $\min\{P(u), P(v)\}$. In other words, $\sigma(u, v)$ represents the minimum possible cost of connecting u and v using edges of the appropriate rate. Note that σ is symmetric, but does not satisfy the triangle inequality, and is not a metric. Lastly, we denote by H_k the k^{th} harmonic number given by $H_k = 1 + \frac{1}{2} + \ldots + \frac{1}{k}$.

1.1 Related work

The Steiner tree (ST) problem admits a simple $2\left(1 - \frac{1}{|T|}\right)$-approximation (see Section 2.1). Currently, the best known approximation ratio is $\rho = \ln 4 + \varepsilon \approx 1.39$ by Byrka et al. [3]. It is NP-hard to approximate the ST problem with ratio better than $\frac{96}{95} \approx 1.01$ [5].

In [1], simple top-down and bottom-up approaches are considered for the MLST problem with proportional costs. In the top-down approach, a Steiner tree is computed over terminals $\{v \in T : P(v) = \ell\}$. For $i = \ell-1, \ldots, 1$, the Steiner tree over terminals $\{v \in T : P(v) = i+1\}$ is contracted into a single vertex, and a Steiner tree is computed over terminals with $P(v) = i$. In the bottom-up approach, a Steiner tree is computed over all terminals, which induces a feasible solution by setting the rate of all edges to ℓ. These approaches are $(\frac{\ell+1}{2})\rho$- and $\ell\rho$-approximations, respectively [1] (moreover, these bounds are tight). It is worth noting that the bottom-up approach can perform arbitrarily poorly in the non-proportional setting.

If edge costs are proportional, Charikar et al. [4] give a simple 4ρ-approximation algorithm (which we later denote by C_1) by rounding the vertex priorities up to the nearest power of 2, then computing a ρ–approximate Steiner tree for the terminals at each rounded-up priority. They then give an $e\rho \approx 4.213$-approximation for the same problem (using $\rho \approx 1.55$ [12]). Karpinski et al. [10] tighten the analysis from [4] to show that this problem admits a 3.802-approximation with an unbounded number of priorities. Ahmed et al. [1] generalize the above techniques by considering a composite heuristic which computes Steiner trees over a subset of the priorities, and show that this achieves a $2.351\rho \approx 3.268$-approximation for $\ell \leq 100$. They provide experimental comparisons of the simple top-down, bottom-up, 4ρ-approximation of Charikar et al. [4], and a generalized composite algorithm. The experiments in [1] show that the bottom-up approach typically provides the worst performance while the composite algorithm typically performs the best, and these results match the theoretical guarantees.

For non-proportional costs, which is the more general setting, Charikar et al. [4] give a $\min\{2(\ln|T|+1), \ell\rho\}$-approximation for QoSMT, consisting of taking the better solution returned by two sub-algorithms (which we denote by C_{2a} and C_{2b}, Section 2.2). On the other hand, Chuzhoy et al. [6] show that PST cannot be approximated with ratio better than $\Omega(\log\log n)$ in polynomial time unless $\text{NP} \subseteq \text{DTIME}(n^{O(\log\log\log n)})$. However, the problem setup for PST [6] is slightly more specific; each edge has a single cost c_e and a Quality of Service (priority) given as input, and a solution consists of a tree such that the path from the root to each terminal t uses edges of QoS at least as good as $P(t)$.

1.2 Our contributions

In this paper, we propose approximation algorithms for the MLST problem based on Kruskal's and Prim's algorithms for computing a minimum spanning tree (MST). We show that the Kruskal-based algorithm is a $2\ln|T|$-approximation even for non-proportional costs, matching the state-of-the-art algorithms. An interesting feature of this algorithm is that for the single level case, it reduces to the standard Kruskal approximation to the Steiner tree problem, which is not the case of other state-of-the-art algorithms for MLST. We also show that, somewhat surprisingly, a natural approach based on Prim's algorithm can perform rather poorly. We then describe an integer linear program (ILP) to compute exact solutions to the MLST problem given non-proportional edge costs and evaluate the approximation ratios of the proposed approximation algorithms experimentally. Specifically, we provide an experimental comparison between the algorithm of Charikar et al. [4] and our Kruskal-based algorithm, in which the latter performs better with respect to the optimum a majority of the time in both proportional and non-proportional settings. Experiments are performed on random graphs from various generators as well as instances of the MLST problem derived from the SteinLib library [11] of hard ST instances. Finally, we describe a class of graphs for which the Kruskal-based algorithm always performs significantly better than that by Charikar et al. [4].

2 Preliminaries

In this section, we review some existing approximation algorithms that are pivotal for the subsequent developments in this paper.

2.1 Kruskal- and Prim-based approximations for the ST problem

A well-known $2\left(1-\frac{1}{|T|}\right)$-approximation algorithm for the ST problem first constructs the metric closure graph $\tilde{G}$ over T: the complete graph $K_{|T|}$ where each vertex corresponds to a terminal in T, and each edge has weight equal to the length of the shortest path between corresponding terminals. An MST over $\tilde{G}$ induces $|T|-1$ shortest paths in G; combining all induced paths and removing cycles yields a feasible Steiner tree whose cost is at most $2\left(1-\frac{1}{|T|}\right)$ times the optimum.

For computing an MST over $\tilde{G}$, one can use any known MST algorithm (e.g., Kruskal's, Prim's, or Borůvka's algorithm). However, one can directly construct a Steiner tree from scratch based on these MST algorithms without the need to construct $\tilde{G}$; Poggi de Aragão and Werneck provide details for such implementations [7] (see also [13, 17]).

Specifically, the Prim-based approximation algorithm for the ST problem due to Takahashi and Matsuyama [13] grows a tree rooted at a fixed terminal. In each iteration, the closest terminal not yet connected to the tree is connected through its shortest path. The process continues for $|T|-1$ iterations until all terminals are spanned. The resulting Steiner tree achieves the $2\left(1-\frac{1}{|T|}\right)$ approximation guarantee [13]. The Kruskal-based algorithm for the ST problem due to Wang [14] maintains a forest initially containing $|T|$ singleton trees. In each iteration, the closest pair of trees is connected via a shortest path between them. The process continues for $|T|-1$ iterations until the resulting forest is a tree. Widmayer showed that this algorithm achieves the $2\left(1-\frac{1}{|T|}\right)$ bound [16].

2.2 Review of the QoSMT algorithm of Charikar et al.

Charikar et al. [4] give a $\min\{2(\ln|T|+1), \ell\rho\}$-approximation for QoSMT which we denote by C_2, consisting of taking the better of the solutions returned by two sub-algorithms (denoted $\mathrm{C}_{2\mathrm{a}}$ and $\mathrm{C}_{2\mathrm{b}}$). For this section, we focus primarily on the $2(\ln|T|+1)$-approximation, Algorithm $\mathrm{C}_{2\mathrm{a}}$. The $\ell\rho$-approximation, Algorithm $\mathrm{C}_{2\mathrm{b}}$, simply computes a ρ-approximate Steiner tree over the terminals of each priority separately, then merges the ℓ computed trees and prunes cycles to output a tree; this leads to a better approximation ratio if $\ell \ll |T|$.

The first sub-algorithm ($\mathrm{C}_{2\mathrm{a}}$) sorts the terminals T by decreasing priority $P(\cdot)$, starting with a root node r (here, we may treat the root as any terminal with priority ℓ). Then, for $i=1,\ldots,|T|$, the i^{th} terminal t_i is connected to the existing tree spanning the previous $i-1$ terminals using the minimum cost path with edges of rate at least $P(t_i)$, where the cost of this path is defined as the connection cost of t_i.

The authors show that for $1 \le m \le |T|$, the m^{th} most expensive connection cost is at most $\frac{2\mathrm{OPT}}{m}$, which implies that the total cost is at most $2\mathrm{OPT}\left(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{|T|}\right) \le 2(\ln|T|+1)\mathrm{OPT}$. While not explicitly mentioned in [4], this approximation ratio is roughly tight (see Figure 1). Algorithm $\mathrm{C}_{2\mathrm{a}}$ can be implemented by running Dijkstra's algorithm from t_i until a vertex already in the tree is encountered. The running time of $\mathrm{C}_{2\mathrm{a}}$ is roughly $|T|$ times the running time of Dijkstra's algorithm, or $O(nm+n^2\log n)$ [4].

3 Kruskal-based MLST algorithms

We propose Algorithm KruskalMLST for the MLST problem. The main distinction compared to Algorithm C_{2a} is that the subsequent algorithm connects the "closest" pairs of terminals first, rather than connecting terminals in order of priority. Algorithm KruskalMLST proceeds as follows: initializing $S = T$, while $|S| \geq 1$, find terminals $u, v \in S$ with $P(u) \geq P(v)$ which minimize the cost of connecting them. If $\mathcal{P}$ is the u–v path chosen, then the rate of each edge in $\mathcal{P}$ is upgraded to $P(v)$ (if its rate is less). Remove v from S. We will say that v is connected at the current iteration. At this point, we do not need to worry about v anymore, and u (the node it is connected to) essentially becomes responsible for v for the rest of the algorithm. When $|S| = 1$, if there are no cycles, then the resulting tree is a feasible MLST rooted at some vertex r with $P(r) = \ell$. Otherwise, we can prune one edge from each cycle with the lowest rate to produce a tree. We note that KruskalMLST takes $|T| - 1$ iterations while C_{2a} takes $|T|$ iterations; this follows as the setting for MLST does not specify a root vertex while QoSMT does. As such, there is a small constant difference in the approximation ratios, which is not significant.

When finding $u, v \in S$ which minimize $\sigma(u, v)$, Algorithm KruskalMLST takes into account edges which have already been included at lower rates. In other words, line 6 seeks a pair of vertices (u, v) which minimizes the cost of "upgrading" the rates of some edges so that u and v are connected via a path of rate $\min\{P(u), P(v)\}$. We denote this cost by $\sigma'(u, v)$, and observe that $\sigma'(u, v) \leq \sigma(u, v)$.

Algorithm KruskalMLST(graph G, priorities P, costs c).

1: Initialize $y(e) = 0$ for $e \in E$
2: $c'_i(e) = c_i(e)$ for $i \in [\ell], e \in E$
3: $S = T$
4: **while** $|S| > 1$ **do**
5: Compute $\sigma'(\cdot, \cdot)$ for all $(\cdot, \cdot) \in S \times S$
6: Find $u, v \in S$ with $P(u) \geq P(v)$ which minimizes $\sigma'(u, v)$
7: $\mathcal{P}$ = path chosen of cost $\sigma'(u, v)$
8: $y(e) = \max\{y(e), P(v)\}$ for $e \in \mathcal{P}$
9: $c'_i(e) = \max\{0, c_i(e) - c_{y(e)}(e)\}$ for $e \in \mathcal{P}$ and $i \in \{1, \ldots, \ell\}$
10: $S = S \setminus \{v\}$
11: **end while**
12: **return** y

▶ **Theorem 3.** *Algorithm* KruskalMLST *is a* $2 \ln |T|$*-approximation to the MLST problem.*

Proof. Define the connection cost of v to be $\sigma'(u, v)$ (line 6), and note that the cost of the returned solution is the sum of the connection costs over all terminals $T \setminus \{r\}$. Now let t_1, t_2, …, $t_{|T|-1}$ be the terminals in sorted order by which they were connected, and let OPT denote the cost of a minimum cost MLST for the instance. We have the following lemma.

▶ **Lemma 4.** *For* $2 \leq m \leq |T|$*, consider the iteration of Algorithm* KruskalMLST *when* $|S| = m$*. Let* t_i *be the terminal connected during this iteration (where* $i = |T| + 1 - m$*). Then the connection cost of* t_i *is at most* $\frac{2\text{OPT}}{m}$*.*

Proof. Note that immediately before t_i is connected, we have $S = \{t_i, t_{i+1}, \ldots, t_{|T|-1}, r\}$ of size m. Consider the optimum solution $\mathcal{T}^*$ for the instance, and let $\mathcal{T}'$ be the minimal subtree of $\mathcal{T}^*$ containing all terminals in S. The total cost of the edges in $\mathcal{T}'$ is at most OPT. Perform a depth-first traversal starting from any terminal in $\mathcal{T}'$ and returning to that terminal. Since every edge in $\mathcal{T}'$ is traversed twice, the cost of the traversal is at most 2OPT.

Consider pairs of consecutive terminals t_j, t_k visited for the first time along the traversal. The path connecting t_j and t_k in $\mathcal{T}'$ necessarily uses edges of rate at least $\min\{P(t_j), P(t_k)\}$. Then, the cost of the edges along this path is at least $\sigma(t_j, t_k)$. There are m pairs of consecutive terminals along the traversal (including the pair containing the first and last terminals visited), and the sum of the costs of these m paths is at most 2OPT. Hence, some pair t_j, t_k of terminals is connected by a path of cost $\leq \frac{2\text{OPT}}{m}$ in the optimum solution, implying that for this pair t_j, t_k, we have $\sigma'(t_j, t_k) \leq \sigma(t_j, t_k) \leq \frac{2\text{OPT}}{m}$. Since KruskalMLST selects the pair which minimizes $\sigma'(\cdot, \cdot)$, the connection cost of t_i is at most $\frac{2\text{OPT}}{m}$. ◀

Lemma 4 immediately implies Theorem 3. Indeed, summing from $m = 2$ to $m = |T|$, the total cost is at most $2\text{OPT}\left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{|T|}\right) = 2\text{OPT}(H_{|T|} - 1) \leq 2\ln|T|\text{OPT}$. ◀

An interesting note is that Algorithm KruskalMLST reduces to the Kruskal-based algorithm [14] for computing a Steiner tree, when there are no priorities on the terminals (i.e., the single level case when $\ell = 1$). As mentioned earlier, this is a $2(1 - \frac{1}{|T|})$-approximation, whereas algorithm $\text{C}_{2\text{a}}$ is still a $2\ln|T|$ one, and this is an advantage of the proposed algorithm.

A simple variant of our algorithm, GreedyMLST, yields the same theoretical approximation ratios and is easier to implement. The difference is that GreedyMLST does not update the costs σ at each iteration of the while loop.

Algorithm GreedyMLST(graph G, priorities P, costs c).

```
Initialize y(e) = 0 for e ∈ E
S = T
while |S| > 1 do
    Find u, v ∈ S with P(u) ≥ P(v) which minimizes σ(u, v)
    P = path chosen of cost σ(u, v)
    y(e) = max(y(e), P(v)) for e ∈ P
    S = S \ {v}
end while
return y
```

▶ **Theorem 5.** *Algorithm* GreedyMLST *is a* $2\ln|T|$*-approximation to the MLST problem.*

The proof follows the same argument as that for Theorem 3; indeed the use of σ' implies that KruskalMLST should perform better than GreedyMLST, but is more costly to run.

3.1 Tightness

The approximation ratio for Algorithms $\text{C}_{2\text{a}}$ [4] and GreedyMLST is tight up to a constant, even if $\ell = 1$ or if $|E| = O(|V|)$. As a tightness example, we use a graph construction $(G_i)_{i\geq 0}$ given by Imase and Waxman [9] for the inapproximability of the dynamic Steiner tree problem. Let G_0 contain two vertices v_0, v_1 with an edge of cost 1 connecting them. We say that v_0 and v_1 are depth zero vertices. For $i \geq 1$, graph G_i is obtained by replacing each edge uv in G_{i-1} with two depth i vertices w_1, w_2, and adding edges uw_1, w_1v, uw_2, and w_2v.

Let $G = G_k$ for sufficiently large k, let $\ell = 1$ (i.e., the Steiner tree problem), and let each edge of G_i have a cost of $\frac{1}{2^i}$, so that the cost of any shortest v_0-v_1 path is 1. Let the terminals T be the vertices of some v_0-v_1 path (Figure 1, left), so that OPT $= 1$. Note that any u-v path contains 2^k edges, so $|T| = 2^k + 1$. Algorithm $\mathrm{C}_{2\mathrm{a}}$ first sorts the terminals by priority; since all terminals in G_k have the same priority, we consider a worst possible ordering where T is ordered in increasing depth, with v_0 the root. In this case, it is possible that Algorithm $\mathrm{C}_{2\mathrm{a}}$ connects v_1 to v_0 via a shortest path which does not include other terminals, then connects subsequent terminals via shortest paths which include no other terminal, as shown in Figure 1. Conversely in the worst case, Algorithm GreedyMLST may connect depth k, $k-1$, $k-2$, ... terminals in order while avoiding previously-used paths, as Algorithm GreedyMLST does not consider existing edges. In both cases, the cost of the returned solution is

$$\text{Cost} = \frac{1}{2}k + 1 = \frac{1}{2}\log_2(|T| - 1) + 1 \geq \frac{1}{2}\left(\log_2 |T| + 1\right) \text{OPT} \approx \left(0.72 \ln |T| + \frac{1}{2}\right) \text{OPT}.$$

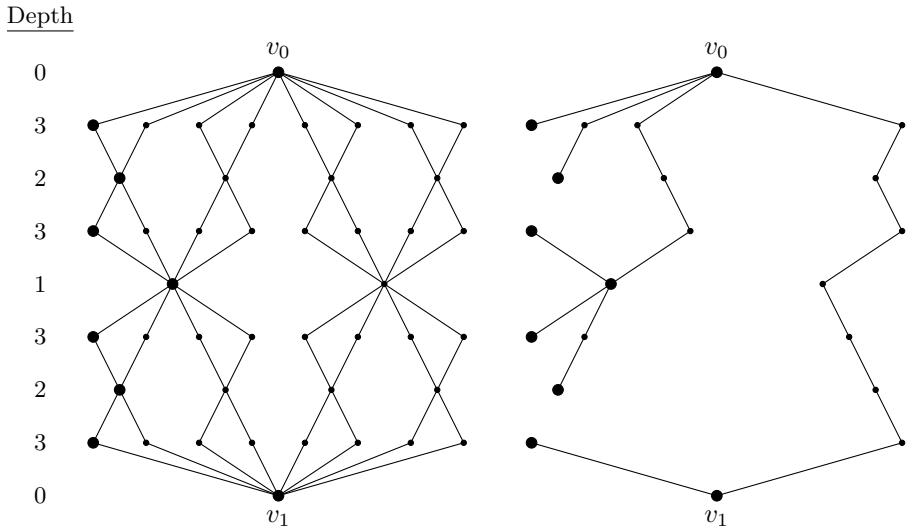


Figure 1 Left: Example instance where $G = G_3$ using the construction by Imase and Waxman [9], $\ell = 1$, with terminals bolded. All edges have cost $\frac{1}{8}$ so that OPT $= 1$. Right: Example solution $\mathcal{T}$ which could be returned by Algorithms $\mathrm{C}_{2\mathrm{a}}$ and GreedyMLST, with cost $\frac{20}{8}$. Note that in hindsight, G may be sparsified so that $|E| = O(|V|)$, by letting $E = E(\mathcal{T}) \cup E(\mathcal{T}^*)$, then contracting each simple path between two terminals to a single edge with cost equal to the length of the path.

3.2 Running Time

The running time of Algorithm GreedyMLST is similar to that of Algorithm $\mathrm{C}_{2\mathrm{a}}$, namely $|T|$ times the running time of Dijkstra's algorithm. This can be implemented as follows: before line 4, for each terminal $t \in T$, run Dijkstra's algorithm from t using edge weights $c_{P(t)}(\cdot)$, and only keep track of distances from t to terminals with priority $\geq P(t)$. Thus, each terminal $t \in T$ keeps a dictionary of distances from t to a subset of T. Then at each iteration (line 6), find the minimum distance among at most $|T|$ distances. The running time of KruskalMLST is $|T|^2$ times that of Dijkstra's algorithm due to the update step (line 9 of Algorithm KruskalMLST).

4 Prim-based MLST algorithm

A natural approach based on Prim's algorithm is as follows. Choose a root terminal r with $P(r) = \ell$ and remove r from T. Then, find a terminal $v \in T$ whose connection cost is minimum, where the connection cost is defined to be the cost of installing or upgrading edges from r to v using rate $P(v)$ (namely, using edge costs $c_{P(v)}(\cdot)$). Remove v from T, and decrement costs. Repeat this process of connecting the existing MLST to the closest terminal until T is empty. Interestingly, unlike Algorithm GreedyMLST, this approach can return a solution $|T|$ times the optimum, which is rather poor. We remark that Algorithm $\mathrm{C}_{2\mathrm{a}}$ [4] is similar to the Prim-based algorithm, where terminals are connected in order of priority rather than connecting the closest terminals first.

As an example, suppose G is a cycle containing $|V| = \ell + 1$ vertices v_1, v_2, v_3, …, v_ℓ, r in that order (Figure 2, left). Let $P(v_i) = i$, and let $P(r) = \ell$. Let $c_i(rv_\ell) = 1$ (edge rv_ℓ has cost 1 regardless of rate), and let $c_i(rv_1) = i(1-\varepsilon)$. Let all other edges have cost zero (or perhaps a small $\varepsilon' \ll \varepsilon$), regardless of rate. Then the Prim-based algorithm greedily connects v_1, v_2, …, v_ℓ in that order, incurring a cost of $1-\varepsilon$ at each iteration. Hence the cost returned is $\ell(1-\varepsilon) \approx |T|$, while $\mathrm{OPT} = 1$.

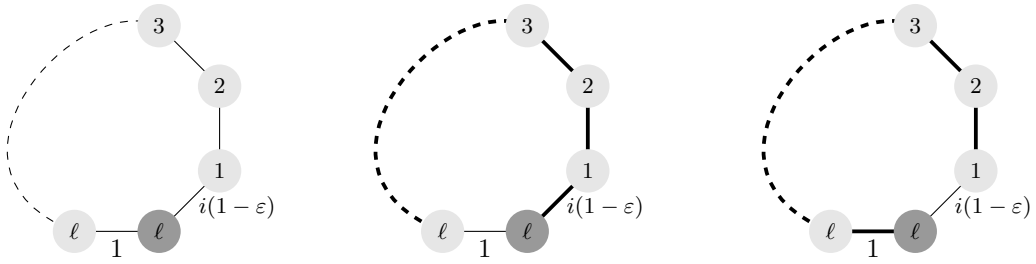


Figure 2 Left: Simple example demonstrating that a Prim-based algorithm can perform poorly. The priorities $P(\cdot)$ and edge costs $c_i(\cdot)$ are shown, and the root r is bolded. Center: Solution found by the Prim-based algorithm with cost $\ell(1-\varepsilon)$. Right: Optimum solution with cost $\mathrm{OPT} = 1$.

5 Integer linear programming (ILP) formulation

In [1], ILP formulations were given for the MLST problem with proportional costs. We extend these and give an ILP formulation for non-proportional costs. First, direct the graph G by replacing each edge $e = uv$ with two directed edges (u, v) and (v, u). Let $x^i_{uv} = 1$ if (u, v) appears in the solution with rate greater than or equal to i, and 0 otherwise. Let $c'_i(u, v)$ denote the incremental cost of edge (u, v) with rate i, defined as $c_i(e) - c_{i-1}(e)$ where $e = uv$ and $c_0(e) = 0$. Fix a root $r \in T$ with $P(r) = \ell$. For $i = 1, \ldots, \ell$, let $T_i = \{t \in T : P(t) \geq i\}$ denote the set of terminals requiring priority at least i. For every edge $e = (u, v)$ we define two flow variables f^i_{uv} and f^i_{vu}.

$$\text{Minimize} \quad \sum_{i=1}^{\ell} \sum_{(u,v)\in E} c'_i(u,v) x^i_{uv} \text{ subject to} \tag{1}$$

$$\sum_{(v,w)\in E} f^i_{vw} - \sum_{(u,v)\in E} f^i_{uv} = \begin{cases} |T_i| - 1 & \text{if } v = r \\ -1 & \text{if } v \in T_i \setminus \{r\} \\ 0 & \text{else} \end{cases} \qquad \forall\, v \in V; 1 \leq i \leq \ell \tag{2}$$

$$x_{uv}^i \leq x_{uv}^{i-1} \qquad \forall\,(u,v) \in E; 2 \leq i \leq \ell \tag{3}$$

$$0 \leq f_{uv}^i \leq (|T_i| - 1) \cdot x_{uv}^i \qquad \forall\,(u,v) \in E; 1 \leq i \leq \ell \tag{4}$$

$$x_{uv}^i \in \{0,1\} \qquad \forall\,(u,v) \in E; 1 \leq i \leq \ell \tag{5}$$

In the optimal solution, the edges of rate greater than or equal to i form a Steiner tree over T_i, so the flow constraint ensures that this property holds. The second constraint ensures that if an edge is selected at rate i or greater, then it must be selected at lower rates. The third constraint ensures that the indicator variable is set equal to one if and only if the corresponding edge is in a tree. The last constraint ensures that the x_{uv}^i variables are 0–1.

▶ **Theorem 6.** *The optimal solution for the ILP induces an MLST with cost* OPT.

The proof is deferred to Appendix A. Additionally, it can be seen from the formulation that the number of variables is $O(\ell|E|)$ and the number of constraints is $O(\ell(|E| + |V|))$.

6 Experiments

We run two primary kinds of experiments: first, we compare the various MLST approximation algorithms discussed here on random graphs from different generators; second, to provide comparison with the Steiner tree literature, we perform experiments on instances generated using the SteinLib library [11]. In both cases, we consider natural questions about how the number of priorities, number of vertices, and decay rate of terminals with respect to priorities affect the running times and (experimental) approximation ratios (cost of returned solution divided by OPT) of the algorithms explored here. We also record how often the algorithms proposed here provide better approximation ratios than pre-existing algorithms. Moreover, we illustrate a class of graphs for which Algorithm KruskalMLST always performs better than Algorithm $\mathrm{C}_{2\mathrm{a}}$.

6.1 Experiment Parameters

We run experiments first to test runtime vs. parameters discussed above, and then to test the experimental approximation ratio vs. the parameters. Each set of experiments has several parameters: the graph generator (random generators or SteinLib instances), the maximum number of priorities ℓ, $|V|$, how the size of the terminal sets T_i (terminals requiring priority at least i) decrease as i decreases, and proportional vs. non-proportional edge costs.

In what follows, we use the Erdős–Rényi (ER) [8], Watts–Strogatz (WS) [15], and Barabási–Albert (BA) [2] models or SteinLib instances [11] to generate the input graph (more on how SteinLib instances are given priorities later). The number of vertices of the graphs generated by different models varies from 10 to 100. We consider number of priorities $\ell \in \{2, \ldots, 7\}$, and adopt two methods for selecting terminal sets (equivalently priorities): linear and exponential. A terminal set T_ℓ with lowest priority of size $n(1 - \frac{1}{\ell+1})$ in the linear case and $\frac{n}{2}$ in the exponential case is chosen uniformly at random. For each subsequent priority, $\frac{1}{\ell+1}$ terminals are deleted at random in the linear case, whereas half the remaining terminals are deleted in the exponential case. Priorities and terminal sets are related via $T_i = \{t \in T : P(t) \geq i\}$. For the proportional edge weight case, we choose $c_1(e)$ uniformly at random from $\{1, \ldots, 10\}$ for each edge independently and set $c_i(e) = ic_1(e)$ for $i = 1, \ldots, \ell$.

For the non-proportional setting, we select the incremental edge costs $c_1(e)$, $c_2(e) - c_1(e)$, $c_3(e) - c_2(e)$, ..., $c_\ell(e) - c_{\ell-1}(e)$ uniformly at random from $\{1, 2, 3, \ldots, 10\}$ for each edge independently.

In the case that the input graph comes from SteinLib, it has a prescribed terminal set (since SteinLib graphs are instances of ST problem for a single priority). For these inputs, priorities are generated in two ways: filtered terminals and augmented terminals. To generate filtered terminals we divide the set of original terminals from the SteinLib into ℓ sets (with $\ell \in \{2, \ldots, 6\}$). We assign the first set as the topmost priority terminals. We assign the second set to the next priority and so on. For the augmented case, we start with the initial terminals from the SteinLib instance and add additional terminals uniformly at random from the remaining vertices. We assign 5 vertices as top priority terminals, double the number of terminals in the next priority, and so on until the maximum number of terminals is reached (we assign $\ell \in \{2, 3, 4\}$ priorities). Augmentation makes sense given that some of the original SteinLib instances have very few terminals. We have generated our datasets from two subsets of SteinLib: I080 and I160; we generate both types of terminals (filtered and augmented) for each of these datasets. The reason we run our experiment on the two SteinLib datasets is that the sizes of the graphs are relatively smaller. Our exact algorithm based on the ILP formulation has an exponential running time, and will not be able to terminate in a reasonable time for the datasets that contain large instances.

An experimental instance of the MLST problem here is thus characterized by five parameters: graph generator, number of vertices $|V|$, number of priorities ℓ, terminal selection method TSM $\in$ {Linear,Exponential}, and proportionality of the edge weights TE $\in$ {Prop,Non-prop}. As there is randomness involved, we generated five instances for every choice of parameters (e.g., ER, $|V| = 70$, $\ell = 4$, Linear, Non-prop).

For the following experiments, we implement the KruskalMLST and C_1 algorithms in the proportional case, and the KruskalMLST and C_{2a} algorithms in the non-proportional case. We note here that Algorithm GreedyMLST achieves much poorer results with respect to OPT than KruskalMLST in practice despite having similar theoretical guarantees. The reason for the poor performance is that the algorithm over–counts the edge costs when it is considered multiple times. On the other hand, KruskalMLST updates the cost of the edges so that for a particular edge and rate, it pays only once. Hence, in our experiment we only use KruskalMLST. To compute the approximation ratios, we use the ILP described in Section 5 using CPLEX 12.6.2 as an ILP solver.

6.2 Results

As one would expect, runtime for both the ILP and all approximation algorithms increased as $|V|$ or ℓ increased. Runtime was typically higher for linear terminal selection than for exponential. See Figures 12–14 in the Appendix for detailed plots. We do note that the running times of the approximation algorithms are significantly faster than the running time of the ILP; the latter takes a couple of minutes for whereas the approximation algorithms take only a couple of seconds for the same instances generated in our experiments.

There was no discernible trend in plots of Ratio (defined as cost/OPT) vs. $|V|$, ℓ, or the terminal selection method (linear or exponential). In all cases, for all graph generators, both the KruskalMLST and C_1 (or C_{2a} in the non-proportional case) exhibited similar statistical behavior independent of the given parameter (see Figures 5–9 in the Appendix for detailed plots). For a brief illustration, we show the behavior for Erdős–Rényi graphs with $p = (1 + \varepsilon)\frac{\ln n}{n}$ in Figure 3, and include the performance of the Composite Algorithm of [1] (CMP) as it gives the best *a priori* approximation ratio guarantee. In the non-proportional

case, Charikar et al. [4] have used another algorithm C_{2b} beside C_{2a} and returned the best solution. In the experiment, we only compare with C_{2a} since C_{2b} runs an iteration for each priority to get the final solution and in this paper, we are primarily interested in techniques that run in a single iteration similar to the spanning tree algorithms.

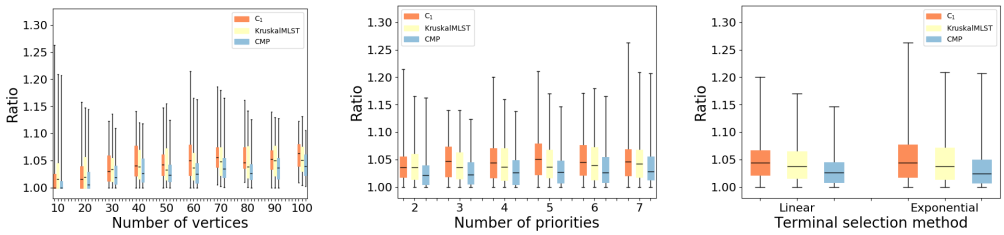


Figure 3 Performance of C_1 [4], KruskalMLST, and CMP [1] on Erdős–Rényi graphs w.r.t. $|V|$, ℓ, and terminal selection method with proportional edge weights.

From Figure 3, we see that on average KruskalMLST outperforms C_{2a}. However, it is instructive to compare the instance-wise performance of the different algorithms. Tables 1 and 2 show comparisons of the statistical performance of the the two approximation algorithms for various graph generators in the proportional and non-proportional case, respectively. For each graph generator, there are a total of 1140 instances consisting of 5 graphs for each set of parameters ($|V|$, ℓ, etc.).

Table 1 Statistics of Algorithms C_1 [4] and KruskalMLST (abbreviated K) with proportional edge cost. Best Approx. reports the percentage of instances (out of 1140) that each algorithm achieved strictly better experimental approximation ratio. Best performance in each category is bolded. The statistics correspond to the experimental approximation ratio.

Graph Generator	ER		WS		BA		SteinLib	
Algorithm	C_1	K	C_1	K	C_1	K	C_1	K
Equal to OPT	73	**133**	391	**679**	94	**202**	4	**8**
Mean	1.048	**1.044**	1.016	**1.012**	1.028	**1.021**	1.2355	**1.1918**
Median	1.044	**1.037**	1.006	**1.0**	1.019	**1.016**	1.2072	**1.1707**
Min	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
Max	1.263	**1.202**	1.31	**1.18**	1.212	**1.126**	1.7488	**1.6404**
Best Approx.	40.53%	**54.29%**	24.92%	**50.78%**	30.62%	**69.38%**	31.50	**59.12%**

Table 2 Statistics of Algorithms C_{2a} [4] and KruskalMLST (abbreviated K) with non-proportional edge cost. Best Approx. reports the percentage of instances (out of 1140) that each algorithm achieved strictly better experimental approximation ratio. Best performance in each category is bolded.

Graph Generator	ER		WS		BA	
Algorithm	C_{2a}	K	C_{2a}	K	C_{2a}	K
Equal to OPT	16	**26**	16	**30**	10	**26**
Mean	1.123	**1.109**	1.099	**1.081**	1.121	**1.097**
Median	1.109	**1.099**	1.087	**1.067**	1.096	**1.08**
Min	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
Max	1.667	**1.54**	1.863	**1.601**	1.941	**1.667**
Best Approx.	37.20%	**61.22%**	34.83%	**63.85%**	30.62%	**68.24%**

We see from these tables that KruskalMLST consistently outperforms the algorithms of [4] in each of the statistical categories, and also achieves better instance-wise results a majority of the time, although this behavior depends somewhat on the graph generator. A full suite of figures is given in the Appendix to further illustrate the performance of each algorithm for the various generators. The trends are essentially the same and are as follows. KruskalMLST outperforms C_{2a} on a majority of instances, but has marginally longer runtime (though the difference is not appreciable); the number of priorities has little effect on runtime or experimental approximation ratio; the number of vertices increases the runtime for some generators, but has little effect on the experimental approximation ratio; experimental approximation ratios are typically better on average for exponentially decreasing terminal sets (which makes sense given that $|T|$ is smaller and the approximation guarantees are $O(\ln |T|)$). Finally, we note that the Composite algorithm of [1] can achieve better approximation in the proportional edge cost setting, but is not known to work for the non-proportional setting; additionally Composite suffers from exponential growth in runtime with respect to ℓ, which is a feature not exhibited by KruskalMLST.

6.3 Graphs for which KruskalMLST always outperforms C_{2a}

Here we generate a special class of graphs for which the Kruskal-based algorithm always provides near-optimal solutions, but Algorithm C_{2a} performs poorly. This class of graphs consists of cycles with randomly added edges. Begin with a cycle $v_1, v_2, \cdots, v_n, v_1$ and set the weight of edge v_1v_n be $w - \epsilon$ where length of the path $v_1, v_2, \cdots, v_n$ is w. We select v_1 and v_n as higher-priority terminals, and the remaining vertices as lower-priority terminals. An algorithm that works in a top-down manner will take the edge v_1v_n for higher priority and pay significantly more than the optimal solution [1]. Doing this to every edge (v_i, v_{i+1}) results an MLST instance where a top-down approach performs arbitrarily poorly. On these graphs, the algorithm provided in Charikar et al. [4] for proportional instances of MLST performs noticeably worse than our Kruskal-based approach (see Figure 4). We generated 500 graphs of this type (augmented with some additional edges at random). The script to generate these graphs are available on Github at `https://github.com/abureyanahmed/Kruskal_based_approximation`.

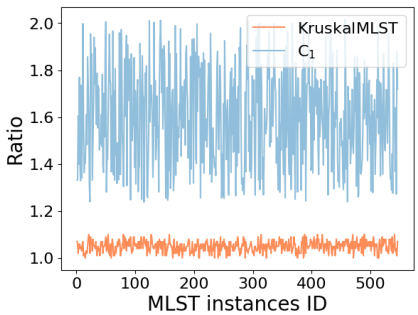


Figure 4 A class of graphs for which the Algorithm KruskalMLST significantly outperforms Algorithm C_{2a} [4]. The x–axis is the instance number and carries no meaning of time; the y–axis is the approximation ratio.

7 Conclusion

We proposed two algorithms for the MLST problem based on Kruskal's and Prim's algorithms. We showed that the Kruskal-based algorithm is a logarithmic approximation, matching the best approximation guarantee of Charikar et al. [4], while the Prim-based algorithm can perform arbitrarily poorly. We formulated an ILP for the general MLST problem and provided an experimental comparison between the algorithm provided by Charikar et al. [4], Ahmed et al. [1], and the Kruskal-based algorithm, KruskalMLST. We demonstrated that KruskalMLST compares favorably to other algorithms in terms of experimental approximation ratio for both the proportional and non-proportional edge costs while incurring a minor cost in run time. Finally, we generated a special class of graphs for which KruskalMLST always performs significantly better than that by Charikar et al. [4]. A natural question is whether the analysis of any of these algorithms GreedyMLST, KruskalMLST, or C_{2a} can be tightened, improving the approximability gap between $O(\log \log n)$ and $O(\log n)$ for the MLST problem with non-proportional edge costs.

References

1 Abu Reyan Ahmed, Patrizio Angelini, Faryad Darabi Sahneh, Alon Efrat, David Glickenstein, Martin Gronemann, Niklas Heinsohn, Stephen Kobourov, Richard Spence, Joseph Watkins, and Alexander Wolff. Multi-level Steiner trees. In *17th International Symposium on Experimental Algorithms, (SEA)*, pages 15:1–15:14, 2018. doi:10.4230/LIPIcs.SEA.2018.15.

2 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

3 Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013. doi:10.1145/2432622.2432628.

4 M. Charikar, J. Naor, and B. Schieber. Resource optimization in QoS multicast routing of real-time multimedia. *IEEE/ACM Transactions on Networking*, 12(2):340–348, April 2004. doi:10.1109/TNET.2004.826288.

5 Miroslav Chlebík and Janka Chlebíková. The Steiner tree problem on graphs: Inapproximability results. *Theoret. Comput. Sci.*, 406(3):207–214, 2008. doi:10.1016/j.tcs.2008.06.046.

6 Julia Chuzhoy, Anupam Gupta, Joseph (Seffi) Naor, and Amitabh Sinha. On the approximability of some network design problems. *ACM Trans. Algorithms*, 4(2):23:1–23:17, 2008. doi:10.1145/1361192.1361200.

7 Marcus Poggi de Aragão and Renato F Werneck. On the implementation of mst-based heuristics for the Steiner problem in graphs. In *Workshop on algorithm engineering and experimentation*, pages 1–15. Springer, 2002.

8 Paul Erdös and Alfréd Rényi. On random graphs, i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

9 Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991. doi:10.1137/0404033.

10 Marek Karpinski, Ion I. Mandoiu, Alexander Olshevsky, and Alexander Zelikovsky. Improved approximation algorithms for the quality of service multicast tree problem. *Algorithmica*, 42(2):109–120, 2005. doi:10.1007/s00453-004-1133-y.

11 T. Koch, A. Martin, and S. Voß. SteinLib: An updated library on Steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2000. URL: http://elib.zib.de/steinlib.

12 Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005. doi:10.1137/S0895480101393155.

13 H. Takahashi and A. Matsuyama. An approximate solution for Steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.

14 SM Wang. A multiple source algorithm for suboptimum Steiner trees in graphs. In *Proc. International Workshop on Graphtheoretic Concepts in Computer Science (H. Noltemeier, ed.), Trauner, W urzburg*, pages 387–396, 1985.

15 Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998.

16 Peter Widmayer. On approximation algorithms for Steiner's problem in graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 17–28. Springer, 1986.

17 Y.F. Wu, P. Widmayer, and C.K. Wong. A faster approximation algorithm for the Steiner problem in graphs. *Acta Inform.*, 23(2):223–229, 1986.

18 Guoliang Xue, Guo-Hui Lin, and Ding-Zhu Du. Grade of service Steiner minimum trees in the Euclidean plane. *Algorithmica*, 31(4):479–500, 2001. doi:10.1007/s00453-001-0050-6.

A Proof of Theorem 6

Proof. We first show that the flow variables take only integer values from zero to $|T_i| - 1$ although it is not specifically mentioned in the formulation. Note that for every priority the ILP generates a connected component in order to fulfill the conditions of the second equation. The algorithm will compute a tree for every priority, otherwise, there is a cycle at a tree of a particular priority and removing an edge from the cycle minimizes the objective. According to the second equation, the flow variable corresponding to an incoming edge connected to a terminal that is not root is equal to one if the edge is in the tree. Since the difference between the incoming and outgoing flow is $|T_i| - 1$ for the root and zero for any intermediate node, every flow variable must be equal to an integer. Also if we do not have integer flows (for example the incoming flow is one and there are two outgoing flows with values 1/2), then because of the conditions in second equation cycles will be generated. Because of this property, the fourth equation ensures that x^i_{uv} is equal to one iff the corresponding flow variable has a value greater than or equal to one. In other words, an indicator variable is equal to one iff the corresponding edge is in the tree. Note that, the formulation has only one assumption on the edge weights: the cost of an edge for a particular rate is greater than or equal to the weight of the edge having lower rates. Hence, the formulation computes the optimal solution for (non-)proportional instances. ◀

B Additional Experimental Results

In this section, we provide some details of the experiments discussed in Section 6.

B.1 Graph Generator Parameters

Given a number of vertices, n, and probability p, the model $\mathrm{ER}(n, p)$ assigns an edge to any given pair of vertices with probability p. An instance of $\mathrm{ER}(n, p)$ with $p = (1 + \varepsilon)\frac{\ln n}{n}$ is connected with high probability for $\varepsilon > 0$ [8]. For our experiments we use $n \in \{10, 15, 20, \cdots, 100\}$, and $\varepsilon = 1$.

The Watts–Strogatz model [15] is used to generate graphs that have the small-world property and high clustering coefficient. The model, denoted by $\mathrm{WS}(n, K, \beta)$, initially creates a ring lattice of constant degree K, and then rewires each edge with probability $0 \leq \beta \leq 1$ while avoiding self-loops or duplicate edges. In our experiments, the values of K and β are set to 6 and 0.2 respectively.

The Barabási–Albert model generates networks with power-law degree distribution, i.e., few vertices become hubs with extremely large degree [2]. The model is denoted by $\mathrm{BA}(m_0, m)$, and uses a preferential attachment mechanism to generate a growing scale-free network. The model starts with a graph on m_0 vertices. Then, each new vertex connects to $m \leq m_0$ existing nodes with probability proportional to its instantaneous degree. This model is a network growth model. In our experiments, we let the network grow until the desired network size n is attained. We vary m_0 from 10 to 100 in our experiments, and set $m = 5$.

B.2 Computing Environment

For computing the optimum solution, we implemented the ILP described in Section 5 using CPLEX 12.6.2 as an ILP solver. The model of the HPC system we used for our experiment is Lenovo NeXtScale nx360 M5. It is a distributed system; the models of the processors in this HPC are Xeon Haswell E5-2695 Dual 14-core and Xeon Broadwell E5-2695 Dual 14-core. The speed of a processor is 2.3 GHz. There are 400 nodes each having 28 cores. Each node has 192 GB memory. The operating system is CentOS 6.10.

B.3 Experimental Setup

We have considered proportional and non-proportional instances separately. The Kruskal-based algorithm is the same in both settings, but the algorithms of [4] admit 2 variants: C_1 for proportional edge costs which is a 4ρ–approximation, and C_{2a} for non-proportional edge costs which is a $2(\ln|T|+1)$–approximation. In figures below, Ratio stands for the approximation ratio given by the cost of the solution returned by the approximation algorithm divided by the optimum cost OPT returned by the ILP.

All box plots shown below show the minimum, interquartile range (IQR) and maximum, aggregated over all instances using the parameter being compared.

B.4 Approximation Ratio vs. Parameters – Proportional edge costs

First, we take a look at how the approximation ratio of the approximation algorithms is affected by the parameters chosen. Figures 3, 5, and 6 illustrate the change in approximation for different parameters ($|V|$, ℓ, and the terminal selection method) in the case of proportional edge costs. For comparison to [1], we include the performance of the Composite algorithm (CMP) described therein.

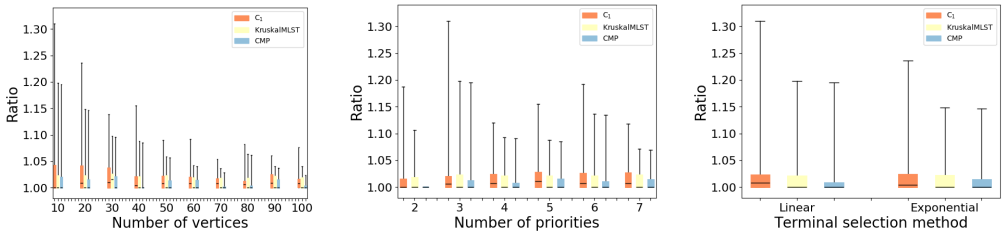


Figure 5 Performance of C_1 [4], KruskalMLST, CMP [1] on Watts–Strogatz graphs w.r.t. $|V|$, ℓ, and terminal selection method with proportional edge weights.

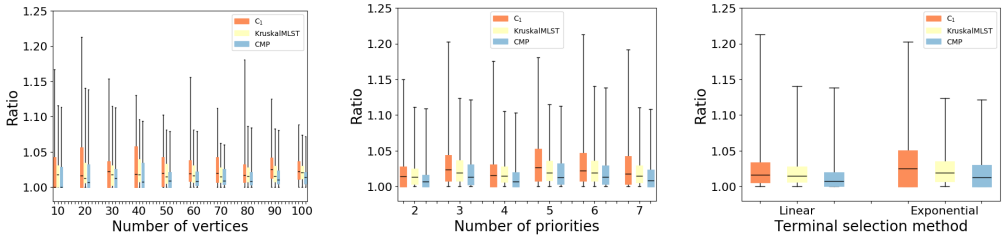


Figure 6 Performance of C_1 [4], KruskalMLST, and CMP [1] on Barabási–Albert graphs w.r.t. $|V|$, ℓ, and terminal selection method with proportional edge weights.

We see that for Erdős–Rényi graphs, the number of vertices marginally increases the approximation ratio over time, while for the other generators this does not appear to be the case. Overall, no discernible trend occurs for the number of priorities regardless of the generator. Interestingly, for randomly generated graphs, there appears to be no relation to the rate of decrease of terminal sets (i.e., linear vs. exponential) with the statistics of the approximation ratios.

B.5 Approximation Ratio vs. Parameters – Non-Proportional Edge Costs

Here we consider the case non-proportional edge cost, in which we compare Algorithms C_{2a} and KruskalMLST. The Composite algorithm of [1] was not designed for non-proportional edge costs and so is not included here. Figures 7–9 show the approximation ratios vs. parameters for each of the random graph generators discussed above.

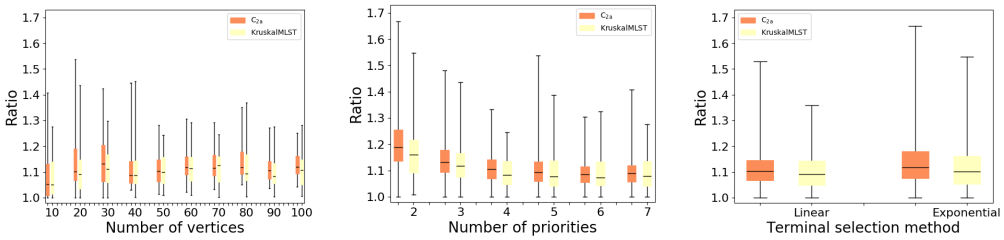


Figure 7 Performance of C_{2a} [4] and KruskalMLST w.r.t. $|V|$, ℓ, and terminal selection method with non-proportional edge weights on Erdős–Rényi graphs.

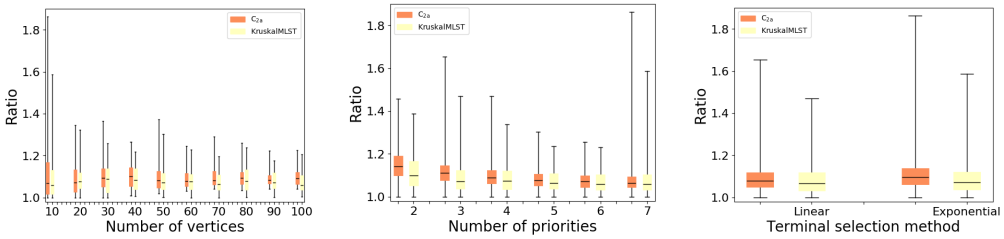


Figure 8 Performance of C_{2a} [4] and KruskalMLST w.r.t. $|V|$, ℓ, and terminal selection method with non-proportional edge weights on Watts–Strogatz graphs.

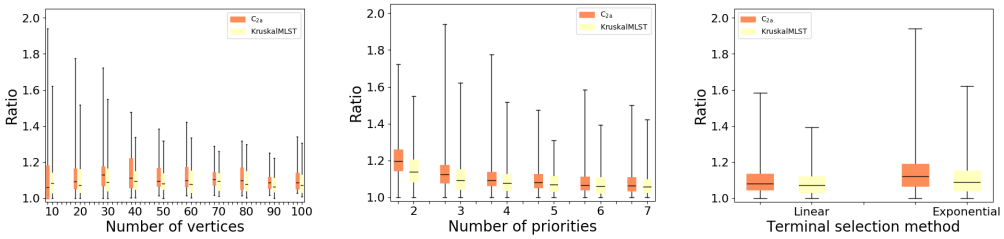


Figure 9 Performance of C_{2a} [4] and KruskalMLST w.r.t. $|V|$, ℓ and terminal selection method with non-proportional edge weights on Barabási–Albert graphs.

In the non-proportional case, it is interesting that the approximation ratio appears to be little affected by any of the parameters, and even appears to decrease with respect to the number of priorities. It is unclear if this trend would continue for large number of priorities, but it is an interesting one nonetheless. Of additional note is that KruskalMLST typically has less variance in its approximation ratio than the algorithms of Charikar et al. [4] in both the proportional and non-proportional case.

B.6 Approximation Ratio vs. Parameters – SteinLib Instances

For the experiments on the SteinLib graphs [11], we first extended two datasets (I080 and I160) to have priorities via filtering or augmenting as described in Section 6. We provide the plots showing the Performance of C_1 [4], KruskalMLST, and CMP [1] on I080 and I160 graphs w.r.t. ℓ with filtered priorities in Figure 10, and for augmented priorities in Figure 11.

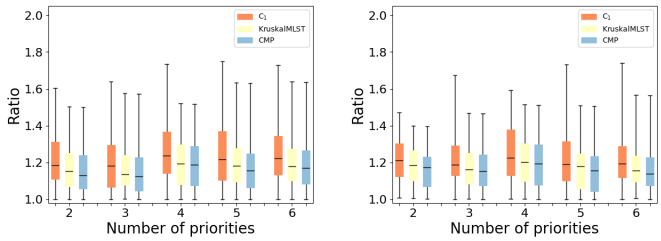


Figure 10 Performance of C_1 [4], KruskalMLST, and CMP [1] on I080 and I160 graphs w.r.t. ℓ with filtered priorities.

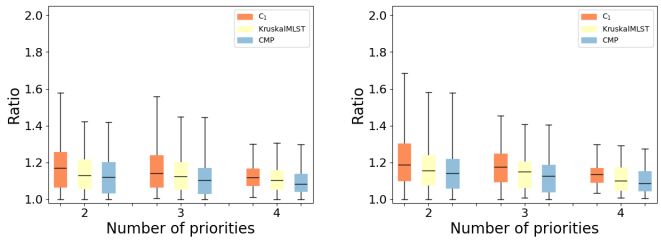


Figure 11 Performance of C_1 [4], KruskalMLST, and CMP [1] on I080 and I160 graphs w.r.t. ℓ with augmented priorities.

B.7 Runtime vs. Parameters – Proportional Edge Costs

Now we take a look at the affect of the parameters mentioned above on the average runtimes of the approximation algorithms in the case of proportional edge costs. Figures 12–14 show the runtime of the algorithms C_{2a}, KruskalMLST, and GreedyMLST versus $|V|$, ℓ, and the terminal selection method.

As is to be expected, on all generators, the average runtime increases as $|V|$ increases, as does the variance in the runtime. Interestingly, average runtime does not appear to be much affected by the number of priorities, although the variance in runtime does substantially increase with ℓ. Runtime is lower for exponentially decreasing terminals, which makes sense given that in this case, the overall size of the terminal sets is smaller than in the linearly decreasing case.

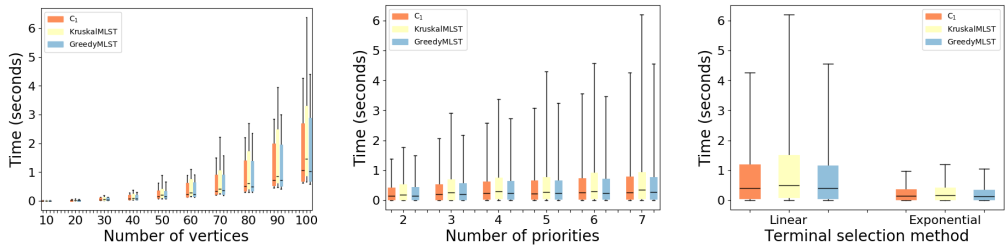


Figure 12 Experimental running times for computing approximation algorithm solutions w.r.t. $|V|$, ℓ, and terminal selection method with proportional edge weights on Erdős–Rényi graphs.

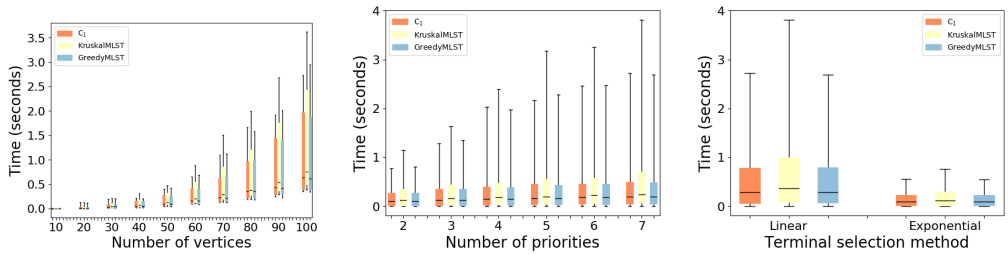


Figure 13 Experimental running times for computing approximation algorithm solutions w.r.t. $|V|$, ℓ, and terminal selection method with proportional edge weights on Watts–Strogatz graphs.

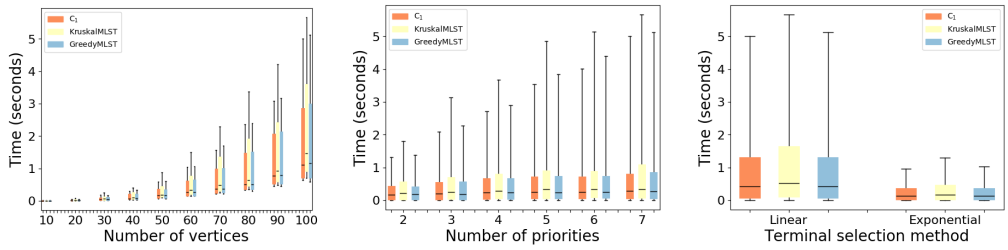


Figure 14 Experimental running times for computing approximation algorithm solutions w.r.t. $|V|$, ℓ, and terminal selection method with proportional edge weights on Barabási–Albert graphs.

B.8 Runtime vs. Parameters – Non-Proportional Edge Costs

Now we take a look at the affect of the parameters mentioned above on the average runtimes of the approximation algorithm in the non-proportional case. Figures 15–17 show the runtime of the algorithms C_{2a}, KruskalMLST, and GreedyMLST versus $|V|$, ℓ, and the terminal selection method.

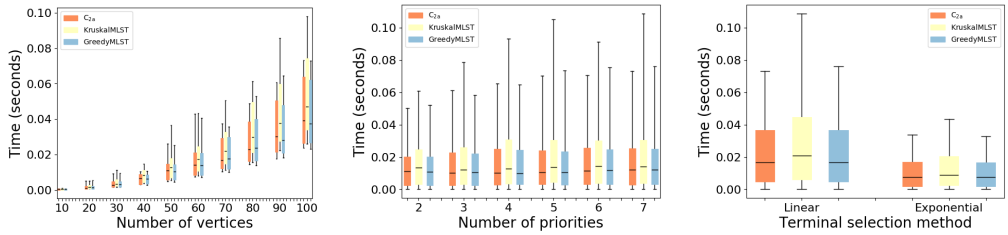


Figure 15 Experimental running times for computing approximation algorithm solutions w.r.t. $|V|$, ℓ, and terminal selection method with non-proportional edge weights on Erdős–Rényi graphs.

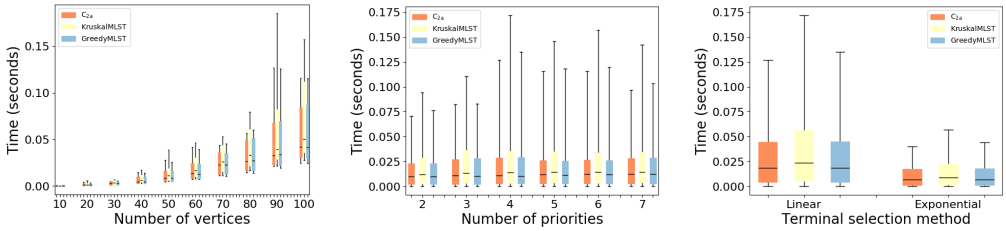


Figure 16 Experimental running times for computing approximation algorithm solutions w.r.t. $|V|$, ℓ, and terminal selection method with non-proportional edge weights on Watts–Strogatz graphs.

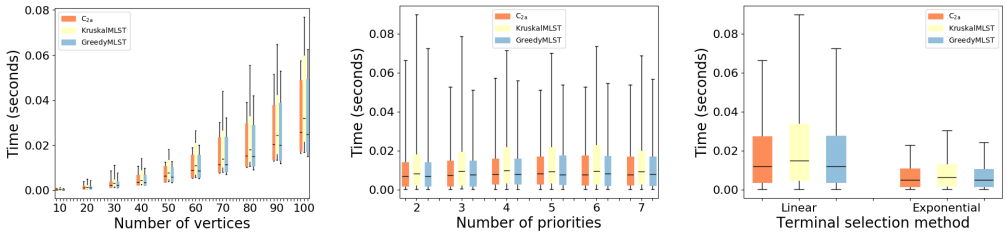


Figure 17 Experimental running times for computing approximation algorithm solutions w.r.t. $|V|$, ℓ, and terminal selection method with non-proportional edge weights on Barabási–Albert graphs.

The trends are essentially the same as in the case of proportional edge costs; however, we note that the overall runtimes are almost two orders of magnitude smaller on average in the non-proportional trials run here.

C ILP Solver

Without doubt, the most time consuming part of the experiments above was calculating the exact solutions of all MLST instances. For illustration, we show the runtime trends for the ILP solver with respect to $|V|$, ℓ, and the terminal selection method for proportional edge costs in Figures 18–20 and for non-proportional edge costs in Figures 21–23 for all of the random graph generators.

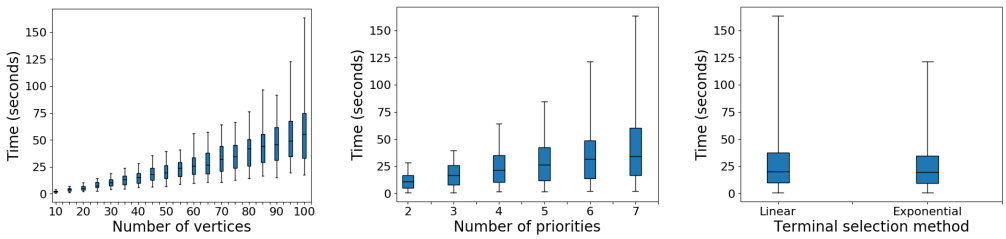


Figure 18 Experimental running times for computing exact solutions w.r.t. $|V|$, ℓ, and terminal selection method with proportional edge weights on Erdős–Rényi graphs.

As expected, the running time of the ILP gets worse as $|V|$ and ℓ increase. The running time of the ILP is worse for the linear terminal selection method, again likely because of the overall larger terminal set T. Note that the running time of the approximation algorithms are significantly faster than the running time of the exact algorithm. The exact algorithm takes a couple of minutes whereas the approximation algorithms take only a couple of seconds.

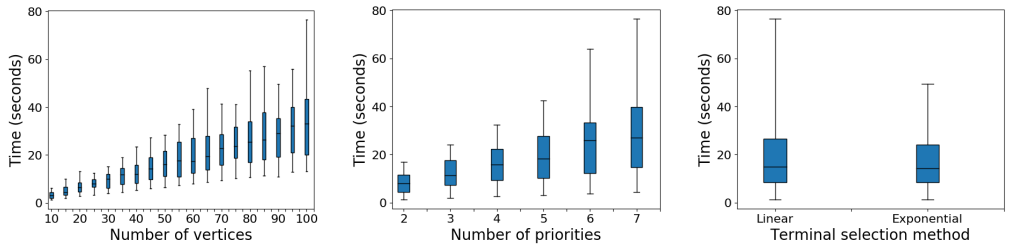


Figure 19 Experimental running times for computing exact solutions w.r.t. $|V|$, ℓ, and terminal selection method with proportional edge weights on Watts–Strogatz graphs.

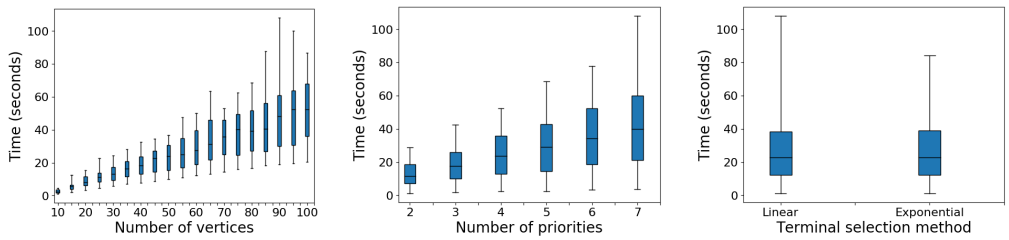


Figure 20 Experimental running times for computing exact solutions w.r.t. $|V|$, ℓ, and terminal selection method with proportional edge weights on Barabási–Albert graphs.

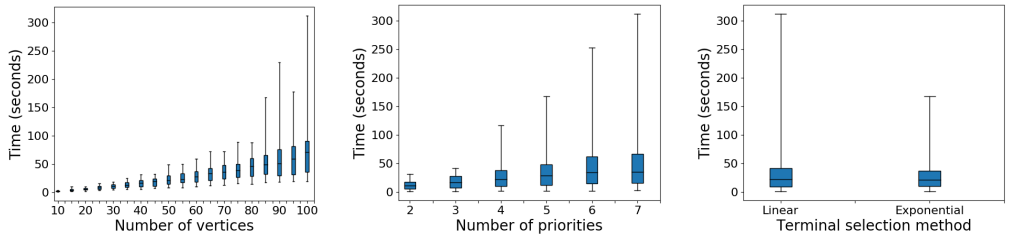


Figure 21 Experimental running times for computing exact solutions w.r.t. $|V|$, ℓ, and terminal selection method with non-proportional edge weights on Erdős–Rényi graphs.

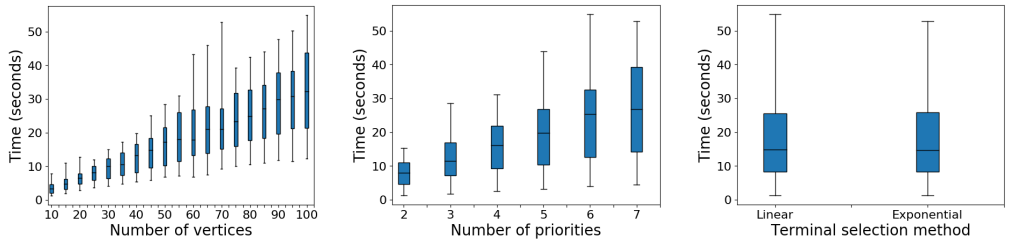


Figure 22 Experimental running times for computing exact solutions w.r.t. $|V|$, ℓ, and terminal selection method with non-proportional edge weights on Watts–Strogatz graphs.

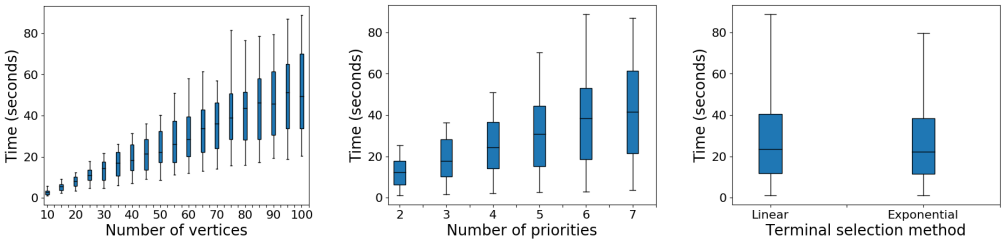


Figure 23 Experimental running times for computing exact solutions w.r.t. $|V|$, ℓ, and terminal selection method with non-proportional edge weights on Barabási–Albert graphs.

Analysis of the Period Recovery Error Bound

Amihood Amir
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
https://u.cs.biu.ac.il/~amir
amir@esc.biu.ac.il

Itai Boneh
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
barbunyaboy2@gmail.com

Michael Itzhaki
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
michaelitzhaki@gmail.com

Eitan Kondratovsky
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
https://u.cs.biu.ac.il/~kondrae
kondrae@cs.biu.ac.il

Abstract

The *recovery problem* is the problem whose input is a *corrupted* text T that was originally *periodic*, and where one wishes to recover its original period. The algorithm's *input* is T without any information about either the period's length or the period itself. An algorithm that solves this problem is called a *recovery algorithm*. In order to make recovery possible, there must be some assumption that not "too many" errors corrupted the initial periodic string. This is called the *error bound*. In previous recovery algorithms, it was shown that a given error bound of $\frac{n}{(2+\epsilon)p}$ can lead to $O(\log_{1+\epsilon} n)$ period candidates, that are *guaranteed* to include the original period, where p is the length of the original period (unknown by the algorithm) and $\epsilon > 0$ is an arbitrary constant.

This paper provides the first analysis of the relationship between the error bound and the number of candidates, as well as identification of the error parameters that still guarantee recovery. We improve the previously known upper error bound on the number of corruptions, $\frac{n}{(2+\epsilon)p}$, that outputs $O(\log_{1+\epsilon} n)$ period candidates. We show how to (1) remove ϵ from the bound, (2) relax the error bound to allow more errors while keeping the candidates set of size $O(\log n)$. It turns out that this relaxation on the previously known upper bound is quite challenging.

To achieve this result we provide what, to our knowledge, is the first known non-trivial lower bound on the *Hamming* distance between two periodic strings. This proof leads to an error bound, that produces a family of period candidates of size $2\log_3 n$. We show that this result is tight and further provide a compact representation of the period candidates. We call this representation the *canonic period seed*.

In addition to providing less restrictive error bounds that guarantee a smaller candidate set, we also provide a *hierarchy* of more restrictive upper error bounds that asymptotically reduces the size of the potential period candidate set.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Sorting and searching

Keywords and phrases Period Recovery, Period Recovery Hierarchy, Hamming Distance

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.5

Funding *Amihood Amir*: Partly supported by ISF grant 1475/18 and BSF grant 2018141.

1 Introduction

Deterministic algorithms live in a "sterile" world: The problem is combinatorially clean, the environment is exact and unchanging, and thus the result is well defined. Reality is seldom so accommodating. Therefore, when applying algorithms to real world problems, one is generally required to approximate a solution.

Such approximations are derived from two sources: (1) Problems that can't be solved efficiently, due to their inherent complexity, and (2) input that has been corrupted by various error-inducing sources. Theoretical Computer Science, in the field of Algorithms Development and Analysis, solves the above first problem by optimization algorithms (see e.g. [11–13, 22]). These are algorithms that come provably close to the optimal solution. The second problem is generally solved using the assumption that the input incurred the *smallest* number of possible corruptions (see e.g. [1, 15–17]). Two examples are the following:

The first solution to the Human Genome Sequencing project [21] used "shotgun sequencing". Since the genome can not be read in its entirety, we are really presented with a "soup" of small subsequences of the genome. These subsequences need to be combined to produce the full sequence. The assumption was that the *shortest common superstring* [20] is the solution, i.e. the shortest string that can be cut into the input subsequences. Of course, there is no absolute guarantee that the original input was, indeed, the shortest. But this is the assumption that was made. Whenever there is statistical support for an assumption on the nature of the output, this support strengthens the result, but one can never be 100% sure that the produced output is indeed the "real" one.

The second example deals with Evolutionary Biology. By 1987, 145 races of humans were identified. The question was, how did the different races evolved? Cann, Stoneking, and Wilson [7] wrote their paper on human evolution, based on mitochondrial DNA. The idea is to fix a gene, and by its differences in the different races, construct a tree depicting the evolution, with races having smaller differences in the gene being closer to each other in the tree than races with greater differences. These evolutionary trees are constructed with the idea of parsimony in mind [8]. Clearly, though, the resulting tree is not the initial one, since different genes cause different trees, which then need to be reconciled [4, 9, 18, 19].

The above two are just examples of our shortcoming in approximating scientific phenomena. The scientific paradigm for reconstructing a phenomenon, at best can produce a measure of confidence, but never guarantee that the result of the algorithm indeed recovers the initial object.

The reconstruction task is the problem in which one has sampled a corrupted text T that was originally periodic, and wishes to recover its original period phenomenon. The input is T without any information about either the period's length nor the period itself. This problem seems doomed since in most cases, even one error can lead to an ambiguity. However, in 2012, Amir et. al [3] introduced the *recovery model*. They have achieved some surprising results. They identified a phenomenon - *periodicity* - where one can get a corrupted input and produce a very small set of solutions (logarithmic in the input size), one of which is **guaranteed** to be the initial uncorrupted source, provided that the number of errors is reasonably bounded. This result was succeeded by additional papers, dealing with various types of error measures [2, 14], and recovering different phenomena [5].

In the *recovery model* for periodic strings, the output is a set of period candidates that must include the original period. Algorithms in such a model assume that the number of errors introduced into the text is limited. The error bound ensures that the recovery algorithm outputs a set of at most $o(n)$ candidates. However, there has not been a systematic study of what types of errors are to be bound, or whether the bound is tight. Nor has the relation between the error bound and the number of potential candidates ever been studied.

One of the topics that this work investigates is different types of error bounds. An error bound *type* is defined by the variables that it limits. We may consider a *universal error bound*, i.e. that in the entire string T there do not appear more than a constant number of errors. We may say that the number of errors is a function of the length of T. Another possibility is that the number of errors is a function of cycles of the period. In other words, how many copies of the period have to pass by without corruption. This last is the type of error considered historically. Some of those variables allow a degree of freedom, some are known while others might be unknown, and even without the ability to be estimated. For example, the historic error bound of [3] has a known value n, which is the length of input T, an unknown value p, the length of the original period (which is not part of the input), a set degree of freedom ϵ, and a parameter c determined by the metric. We can say in an abbreviated manner that previously known error bounds are of (n, p, ϵ, c)-type.

In this paper, we make the first attempt at a systematic analysis of the required error bound for recovery of a periodic string under mismatch errors. The only currently known relations between error bounds and number of candidates are: (1) the trivial bound of $|\Sigma|^{n/2}$ candidates for n possible errors, where Σ is the alphabet, since all possible periodic strings are candidates, and (2) The bound shown in Amir et al. [3]: Let T be an n-length periodic string with period P of length p. For $\epsilon > 0$, if we are guaranteed that there are no more than $\frac{n}{(2+\epsilon)p}$ mismatch errors, then a set of $\log_{1+\epsilon} n$ candidates can be constructed in $O(n \log n)$ time, that is guaranteed to include the original period P.

Amir et al. [3] proved their error bound result for pseudo-local metrics. For simplicity, we consider the *Hamming distance* as the distance metric, i.e. errors counted as the number of replacements. All our results can be easily extended to c-pseudo local metrics.

We desire answers to the following questions:
Past upper bounds depended on p. The first question to ask is whether this dependency is essential. In this paper, we study the case where the upper bound on errors is the number k. We are departing from the historical (n, p, ϵ, c)-type upper bounds, and instead consider k-type bounds, where k is either known or could be estimated. In this case, when one wants to reproduce the original periodic phenomena, the period length is unknown. On the other hand, the estimation of the number of corrupt copies of the period, as high as k, is assumed to be known.

In previous results, there is a "degree of freedom" variable ϵ. it seems that ϵ is not a natural variable to have in the upper bound formula. Its only use was to support the analysis. Indeed, one might want ϵ to be as close to zero as possible to relax upper bound on errors. However, this aim increases the number of candidates. This observation leads to our second question, whether it is possible to improve such analysis and get rid of ϵ without having too many candidates.

The second question leads to a greater task of how significant the upper bound of (n, p, c)-type can be relaxed while ensuring $o(n)$ candidates. This paper uses algebraic techniques to improve the upper bound on errors while preserving the property of $o(n)$ candidates. We give a partial answer, but believe that the question of how can be upper bound be further relaxed, and yet offer $o(n)$ periodic candidates is difficult and open to future research.

This paper successfully answers the above two questions.

However, on the path toward that goal, our first non-trivial insight is that even a single error (when $k = 1$) in text T might result in $\Theta(n)$ indistinguishable candidates. That is, without any additional knowledge on the problem, having a universal upper bound with $o(n)$ candidates is impossible. We further show that if the original period is repeated in the text $k + 1$ times, i.e. there exists a *single* complete uncorrupted occurrence of the original period, then it is still possible to construct an example with $\Theta(\frac{n}{k(k+1)}) = \Theta(n)$ candidates.

The novel combinatorial results are that (1) if the number of original period repetitions is $2k+1$ or higher then there is at most one candidate, i.e. *the original period can be recovered*, and (2) if the period originally repeated $2k$ times or higher ($k \geq 2$) then there are at most 2 indistinguishable candidates. We conclude that when k is known, the only required additional knowledge to find a constant number of period candidates is whether $p \leq \frac{n}{2k+1}$ or $p \leq \frac{n}{2k}$, respectively. Analysing the candidate set size when the number of repetitions is in the range between $k+1$ and $2k$ requires future research.

We highlight a connection between (n,p)-type and k-type upper bounds. Assume the case when k is unknown, but the above assumption about the repetitions holds. Then we should look at all possible k values and add their number of candidates. This leads us to the bounds $\lfloor \frac{n-p}{2p} \rfloor$ and $\lfloor \frac{n}{2p} \rfloor$, respectively. However, there are $\Theta(n)$ different values for k, and we require that the number of candidates must be $o(n)$, therefore we provide a new proof methodology.

In this paper, we analyze the $\lfloor \frac{n}{2p} \rfloor$ upper bound on errors, and prove that the number of candidates is $2\log_3 n$ and that this bound is tight by providing a family of examples. Moreover, we show that these candidates can be represented by a single *canonic period seed*.

Finally, we show a hierarchy of more restricted upper bounds of (n,p,ϵ,c)-type that yield $\Theta(\log^{(i)} n)$ candidates, where $\log^{(i)} n$ is $\log\log\cdots\log n$ i times.

In Fig. 1 we show the known bounds and the results of this paper. The formulae on the horizontal axis are error bounds, we show a hierarchy of candidate set size upper bounds. In various cases we show examples where this upper bound is indeed tight.

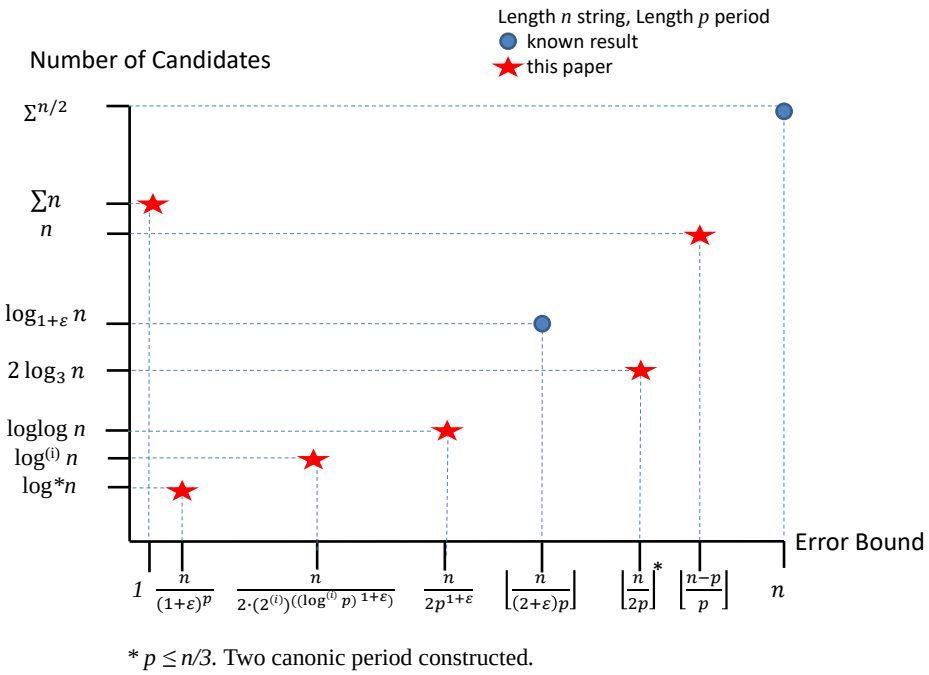


Figure 1 The results of this paper.

This paper is organized as follows: In Section 5, we discuss the k-type upper bounds. In Section 6, we tighten the logarithmic bound on the number of candidates. While in [3] the log base was dependent on ϵ of the allowed error bound, we show that for all ϵ the bound

is $2\log_3 n$. In this section, we introduce a new algebraic method to analyze the distance between periodic strings derived by different seeds. This enables us to further tighten the upper error bounds to $\lfloor \frac{n}{2p} \rfloor$, and construct a **single** *canonic period seed*, where no more than $2\log_3 n$ candidates derived from it, among which the initial period is guaranteed to exist. Finally, in Section 7, we give a hierarchy of upper bounds that produce decreasing number of candidates that include the original period. The hierarchy decreases from $\log n$ via $\log\log n$, $\log^{(i)} n$, to $\log^* n$.

2 Preliminaries

Let Σ be an alphabet. A *string* T over Σ is a finite sequence of letters from Σ. By $T[i]$, for $0 \leq i \leq t-1$, we denote the i^{th} letter of T. The *empty string* is denoted by λ. By $T[i..j]$ we denote the string $T[i]\dots T[j]$ called a *substring* of T (if $i > j$, then the substring is the empty string). A substring is called a *prefix* if $i = 0$ and a *suffix* if $j = t-1$. The prefix of length $j+1$ is denoted by $T[..j]$. While by $T[i..]$ we denote the suffix which starts from index i in T. We will follow the convention of using capital letters for string names, and the same small letter for the length of the string. For example: the length of string T is t. Other notations we use is $T[-i] = T[t-i]$.

An n-length string T is periodic if $T = P^k P'$, where $k \in \mathbb{N}$, $k \geq 2$, and P' is a prefix of P (the prefix might be empty). P^k is the concatenation of P to itself k times. It is clear that P is a substring of T and $p \leq t/2$.

If T is periodic, the shortest P, such that $T = P^k P'$ is called *the period of* T. There is a unique such period by the *periodicity lemma* [10]. The periodicity lemma states that for two different periods of lengths p and q, where $n \geq p + q - gcd(p, q)$, there exists a period of length $gcd(p, q)$, where $gcd(a, b)$ is the greatest common divisor of a and b. A string P is *primitive* if there is no string S such that $P = S^k$ and $k > 1$.

The recovery problem seeks the original period of a corrupted text T. The corruption may have caused T to be non periodic. Thus there may be a number of indistinguishable periods that generate strings of length t. We are seeking an error bound on the distance between these strings and T that forces only a small set of such periods.

▶ **Example 1.** $T = abaabaab$, then $P = aba$ is the period. In our exposition, for the sake of brevity, we may denote this string by $T = P^{2\frac{2}{3}}$. We allow ourselves to use fractions $\frac{x}{y}$ where $\frac{x}{y} \times t$ is an integer. In this example, we could not use $\frac{1}{4}$ because $\frac{1}{4} \times 3$ is not an integer.

Let T and S be two n-length strings, their *Hamming distance*, denoted by $Ham(T, S)$, is the number of mismatches between these strings. The Hamming distance represents the number of substitutions required to convert one string to the other.

Let P be a primitive string of length p. Let T be a n-length string, P^∞ denotes the periodic string of length ℓ, where the value of ℓ is clear from the context. For example, in the expression $Ham(T, P^\infty)$, both operands T and P^∞ should be of equal lengths, $\ell = n$. P is called a *period candidate* with bound e if $Ham(T, P^\infty) \leq e$. If $\ell = n$, we denote $T_P = P^\infty$. This notation simplifies the expressions when dealing with two different periodic candidates of T. For example, when observing $Ham(T_P, T_Q)$, where P and Q are two different periods.

Let P be a periodic candidate of T. The substring ranges of the form $[i, i+p-1]$, where $i = 1, p+1, 2p+1, \dots, (\lfloor \frac{n}{p} \rfloor - 1)p + 1$ denote *full occurrences* of P in T. A *full occurrence* $[i, i+p-1]$ is called an *exact occurrence* of P if $P = T[i..i+p-1]$, otherwise it is a *corrupted occurrence*.

3 A universal Error Bound Does not Allow Recovery

We begin by proving that a universal error bound, even if it is a single error in the string, does not suffice for $o(n)$ candidates. We describe an example of $\Omega(n)$ indistinguishable candidates when there is only a single corruption in some periodic text. Then we generalize it to any number of corruptions $k \in \mathbb{N}$.

▶ **Example 2.** Let $T = a^{2\ell}ba^{4\ell+1}$, where $n = 6\ell + 2$. We show that there are $\frac{n}{6}$ indistinguishable period candidates.

The original periodic source of the text T is one of the following.
$C = \{a^{2\ell}b\ a^{2\ell}b\ a^{2\ell}, a^{2\ell}ba\ a^{2\ell}ba\ a^{2\ell-2}, a^{2\ell}ba^2\ a^{2\ell}ba^2\ a^{2\ell-4}, \ldots, a^{2\ell}ba^{\ell}\ a^{2\ell}ba^{\ell}\}$
$= \{(a^{2\ell}b)^{2+\frac{2\ell}{2\ell+1}}, (a^{2\ell}ba)^{2+\frac{2\ell-2}{2\ell+2}}, (a^{2\ell}ba^2)^{2+\frac{2\ell-4}{2\ell+3}}, \ldots, (a^{2\ell}ba^{\ell})^2\}$

Each such possible source text has a different period. And for each such source, the number of the introduced corruptions is exactly 1. The second b is replaced by an a.

It is clear that $|C| = O(n)$, because of the following reason. Observe that the third occurrence of the period is not complete, only a suffix of it occurs. It begins with $a^{2\ell}$ and for each successive item, the length of the suffix of the third period occurrence decreases by 2, until it becomes the empty string. Thus, $|C| = \ell + 1 \approx \frac{n}{6}$.

It is easy to generalize the example to the case where k corruptions are allowed and there are $k + 1$ full occurrences of the period, in other words, we still have one uncorrupted periodic occurrence. In this case, the constructed example would has $\Omega(\frac{n}{k(k+1)}) = \Omega(n)$ indistinguishable period candidates.

▶ **Example 3.** $T = a^{\ell k}ba^{\ell k^2+k-\ell-1}$, where $n = \ell k^2 + \ell k + k$, $k \geq 2$, and $\ell \geq k$. We show that there are $\Omega(\frac{n}{k(k+1)})$ indistinguishable period candidates.

The original periodic source of the text T is one of the following.
$C = \{(a^{\ell k}b)^k\ a^{\ell k}, (a^{\ell k}ba)^k\ a^{\ell k-k}, (a^{\ell k}ba^2)^k\ a^{\ell k-2k}, \ldots, (a^{\ell k}ba^{\ell})^k\}$
$= \{(a^{\ell k}b)^{k+\frac{\ell k}{\ell k+1}}, (a^{\ell k}ba)^{k+\frac{\ell k-k}{\ell k+2}}, (a^{\ell k}ba^2)^{k+\frac{\ell k-2k}{\ell k+3}}, \ldots, (a^{\ell k}ba^{\ell})^k\}$

Thus, $|C| = \ell + 1 = \Theta(\frac{n}{k(k+1)})$.

4 Results

Our main contributions are the following.

▶ **Theorem 4.** *Let k be a fixed integer value. Let P be a period for which $Ham(T, T_P) \leq k$, and P has at least $2k + 1$ full occurrences in T. Then the number of possible candidates for P is at most 1.*

▶ **Theorem 5.** *Let $k \geq 2$ be a fixed integer value. Let P be a period for which $Ham(T, T_P) \leq k$, and P has at least $2k$ full occurrences in T. Then the number of possible candidates for P is at most 2.*

▶ **Theorem 6.** *Let P be a period for which $Ham(T, T_P) \leq \lfloor \frac{n}{2p} \rfloor$, and $p \leq \frac{t}{3}$. Then the number of possible candidates for P is at most $2\log_3(n) = O(\log n)$.*

5 k-Type Upper Error Bounds

In this section, we prove Theorems 4 and 5. Doing so requires a few new lemmas. We begin by examining the case where $k = 1$. Then we generalize our results to any $k \in \mathbb{N}$.

Let P, Q be two period candidates of T, with lengths p, q, respectively. Without loss of generality, assume $p > q$. We recall a useful lemma that was proven when P fully occurs at least twice.

▶ **Lemma 7** ([3]). *For any two periods P and Q, if they both fully occur at least twice, then $Ham(T_P, T_Q) \geq 2$.*

In [3], it is claimed that for the general case $Ham(T_P, T_Q) \geq \frac{n}{p}$. Our Corollary 10 is exactly this result. However, the previous proof fails to handle properly the case where $\left\lfloor \frac{n}{p} \right\rfloor$ is odd. We take care of this missed case.

▶ **Lemma 8.** *Let T be a text and assume there is at most one corruption error in T. Further assume that T is a corruption of an original periodic string T_P where the period P fully occurs at least 3 times in T (neither the period nor its length is part of the input) . Then there is a single period candidate for the original text.*

Proof. Let us assume in contradiction that there are P and Q two possible period candidates for T, $P \neq Q$. We begin by showing that $p \neq q$. If $p = q$, there are three occurrences of P and Q which are of the same length, but $Ham(T_P, T_Q) \leq 2$. It means that one full occurrence of P is equal to the corresponding full occurrence of Q, namely, $P = Q$. As a consequence, $p \neq q$. The proof is divided into three cases.

Case 1. When $q \mid p$. Let $k = \frac{p}{q}$.

P is primitive, against each full occurrence of P there are Q^k. That is, each full occurrence of P in T_P should cause at least 1 mismatch with T_Q, otherwise P is not primitive, thus $Ham(T_P, T_Q) \geq 3$. However, one error is assumed thus, $Ham(T_P, T_Q) \leq Ham(T, T_P) + Ham(T, T_Q) = 2$. Thus, we got a contradiction.

Case 2. When $q \mid 2p$ and $q \nmid p$.

First, we observe that q must be even. Let $Q = Q_P Q_S$, where Q_P and Q_S are exactly the first and last half of Q of lengths $q/2$. We observe the prefix of length $\frac{3q}{2}$ of the three first full occurrences of P. It is easy to see that against the first and third occurrence of P in T_P, there are $Q_P Q_S Q_P$ in T_Q. On the other hand, against the second full occurrence of P there are $Q_S Q_P Q_S$. It is clear that $Q_P \neq Q_S$, otherwise Q is not primitive. Thus, we must have at least 3 mismatches between T_P and T_Q, in contradiction.

Case 3. Otherwise, when $q \nmid 2p$ and $q \nmid p$.

From the alignment lemma, the number of mismatches between T_P and T_Q is at least 4, in contradiction. ◀

▶ **Corollary 9.** *For any two periods P and Q that fully occur at least three times, then $Ham(T_P, T_Q) \geq 3$.*

▶ **Corollary 10.** *Let $k \geq 2$. For any two periods P and Q that fully occur at least k times, then $Ham(T_P, T_Q) \geq k$.*

Proof. If k is even then use Lemma 7 on all disjoint consecutive pairs of full occurrences of P against the rotations of Q. If k is odd, use Lemma 7 for all full occurrences of P except its last three full occurrences. Handle these occurrences by Corollary 9. ◀

We now have the tools to prove the main theorems. The proofs can be found in the full version of this paper.

▶ **Theorem 4.** *Let k be a fixed integer value. Let P be a period for which $Ham(T, T_P) \leq k$, and P has at least $2k + 1$ full occurrences in T. Then the number of possible candidates for P is at most 1.*

▶ **Theorem 5.** *Let $k \geq 2$ be a fixed integer value. Let P be a period for which $Ham(T, T_P) \leq k$, and P has at least $2k$ full occurrences in T. Then the number of possible candidates for P is at most 2.*

6 Tighter Bounds for $2\log_3 n$ Candidates

In this section we relax the constraints of Amir et al. [3] regarding the number of allowed mismatches, and yet provide a much smaller candidate set. The main tool in achieving this is a string combinatorics theorem that improves the best known lower bound on the Hamming distance between two periodic strings.

6.1 Lower bound on the Hamming distance between strings

We start by improving the best known lower bound on the number of errors between two periodic strings. We provide a new nontrivial expression that tightens the lower bound and give tight examples. Our formula results from a careful analysis of the *Turning Points*, *Windows*, and *Adjacency Strings* of the two strings.

▶ **Definition 11.** *A string P is called* Non-trivial if it contains at least two distinct characters.

▶ **Theorem 12.** *Let P, Q be non-periodic, non-trivial strings of lengths p, q, respectively, s.t. $q < p, q \nmid p$, and let $p = aq + b, \quad 0 < b < q$, then*

1. $\forall m, 4 \leq m < \frac{q}{\gcd(p,q)}, \quad Ham(P^m, Q^\infty) \geq m(a+1) - 2 + \left\lfloor \frac{mb}{q} \right\rfloor$
2. $\forall m, 2 \leq m < \frac{q}{\gcd(p,q)}, \quad Ham(P^m, Q^\infty) \geq m(a+1) - 2$

In the next subsections, we provide the proof of Theorem 12 in-depth.
For pedagogical reasons and for simplicity and comprehensibility, we will prove the theorem for strings of co-prime lengths and binary alphabet, $\Sigma = \{\alpha, \beta\}$, and then show a reduction from the general case to the simplified cases.

Troughout the proof, we will assume w.l.o.g that $p > q$ and that α appears in Q at least as many times as β. We will also let a, b satisfy $p = aq + b, \quad a, b > 0$.

We begin with definitions that will help in understanding the motive and correctness of the proof and methods.

6.1.1 Groundwork

▶ **Definition 13** (Index mapping function)**.** *The* index mapping function *is defined as follows:*

$$\delta_m : [0, p-1] \to [0, q-1], \quad \textit{where} \quad \delta_m(i) = i + (m-1)p \mod q$$

The index mapping function maps the index of the m_{th} occurrence of $P[i]$ in P^m to the corresponding index in Q^∞. Note, that the mapping function is also dependent on the lengths of P, Q. We refrain from indexing the function with these symbols for simplicity's sake.

▶ **Definition 14** (Adjacency string)**.** *Let P, Q be strings of co-prime lengths, and $p > q$. The adjacency string $\tilde{Q}_p$ is a string of length q that satisfies $\forall i, 0 \leq i < q, \tilde{Q}_p[i] = Q[i \cdot p \mod q]$. The adjacency subset-string $\tilde{P}_q$ is a multi-string of length q that satisfies*

$$\forall i, 0 \leq i < q, \tilde{P}_q[i] = \{P[j] | j \equiv_q i \cdot p\}$$

A *subset string* is a string where each string-position might contain several characters. The term was originally defined at [6]. Note that in our case, the same character can recur multiple times at a single position.

The motivation behind this representation is to encapsulate the index-mapping function. It holds that $\forall m, i, j$ where $j \equiv_q i \cdot p$ implies $\tilde{Q}_p[(j+m) \mod q] = Q[\delta_{m+1}(i)]$. In words, it means that a character in $\tilde{P}$ that aligns against $\tilde{Q}_p[i]$, will align against $\tilde{Q}_p[i+1]$ in its next recurrence. $\tilde{P}$ was defined for mere convenience; $Q[i] = \tilde{Q}_p[j] \rightarrow P[i + n \cdot q] \in \tilde{P}_q[j]$, for all $i + n \cdot q < p$ [1].

We will often omit the subscript and simply write $\tilde{Q}, \tilde{P}$. Fig. 2 shows an example of an adjacency string. In this example, one can see that if a character σ in P is aligned with $\tilde{Q}[i]$, the next occurrence of σ will align with $\tilde{Q}[(i+1) \mod q]$.

P=0123456 $\quad \widetilde{P}_4$ ={0,4}3{2,6}{1,5}
Q=abcd $\quad \widetilde{Q}_7$ =adcb

P^4 =0123456012345601234560123456
Q^7 =abcdabcdabcdabcdabcdabcdabcd

Figure 2 Example for adjacency string, Definition 14.

During the proof, we refer to the symbol "β" as *Black node*, and to "α" as *White node*.

▶ **Definition 15.** *The* m-forward window $W_m^f(i)$ *is the multi-set of characters in* Q *that align against* $\tilde{P}[i]$ *in the next* m *repetitions of* P*, which is* $\{\tilde{Q}[i], ..., \tilde{Q}[i+m-1]\}$
The m-backward window $W_m^b(i)$ *is the multi-set of characters in* P *that align against* $\tilde{Q}[i]$ *in the next* m *repetitions of* P*, which is* $\{\tilde{P}[i], ..., \tilde{P}[i-m+1]\}$

In Fig. 3 we see an example for backwards and forwards windows. One can see that $W_3^b(2)$ contains all the characters that touch $\tilde{Q}[2]$ in the next 3 repetitions of P, and that $W^f(0)$ contains all the characters that touch $\tilde{P}[0]$ in the next 3 repetitions. It is also apparent that W_3^b contains the character '0' twice.

P =01234067 $\quad \widetilde{P}_5$ ={0,0}3{1,6}4{2,7}
Q =abcde $\quad \widetilde{Q}_7$ =adbec

P^3 =01234067|01234067|01234067
Q^∞=abcdeabc|deabcdea|bcdeabcd
$W_3^b(2) = \{\widetilde{P}[2], \widetilde{P}[1], \widetilde{P}[0]\} = \{1, 6, 3, 0, 0\}$

P^3 =01234067|01234067|01234067
Q^∞=abcdeabc|deabcdea|bcdeabcd
$W_3^f(0) = \{\widetilde{Q}[0], \widetilde{Q}[1], \widetilde{Q}[2]\} = \{a, d, b\}$

Figure 3 Example for backward and forward windows, Definition 15.

[1] The converse is not true for, as the mapping function maps to characters in Q.

▶ **Definition 16** (Heavy index). *We say that i is a* heavy index *if it satisfies* $|\tilde{P}[i]| = \left\lceil \frac{p}{q} \right\rceil = a+1$.

We will call the last b characters of P *heavy characters*.

▶ **Corollary 17.** *$\forall m, i$, the number of characters in $W_m^b(i)$ are at least $am + \left\lfloor \frac{mb}{q} \right\rfloor$.*

Proof. Considering $\tilde{P}$ has at least a characters at every index, and the number of heavy indices in any m repetitive recurrences is at least $\left\lfloor \frac{mb}{q} \right\rfloor$, giving the required result. The heavy indices are distributed equally in $\tilde{P}$ because in P, the if the heavy indices are $0, 1, ..., j-1$, then on the next recurrence they will align with $j, j+2, ..., 2(j-1)$, and so on, and by abstract algebra the distribution of heavy indices in $\tilde{P}$ is equal, though not probabilistic. ◀

▶ **Definition 18 (Turning points).** *Let $t_1, t_2, ..., t_\ell$ be the indices such that $\tilde{Q}_p[t_i] \neq \tilde{Q}_p[t_i + 1 \mod q]$. We call these indices* Turning points.

This definition will help in counting mismatches; Turning points create mismatches, as a character that matches a turning point will create a mismatch in the next recurrence of P.

See Fig. 4 for an example of Turning points.

```
P  =0101010101010101
Q  =0100010011011
Q̃7 =0001101111000
```

Figure 4 Example for Turning points, Definition 18.

▶ **Corollary 19.** *ℓ is even.*

Proof. Given that turning points are the only indices that satisfy $q[t_i] \neq q[t_{i+1}]$, and therefore if ℓ is odd, then it can be expressed as $2\ell' + 1$, and

$q[t_1] \neq q[t_2] ... \neq q[t_{2\ell'+1}] \neq q[t_1] \rightarrow$
$q[t_1] = q[t_3] = ... = q[t_{2\ell'+1}] \neq q[t_1] \rightarrow$
$q[t_1] \neq q[t_1]$ ◀

▶ **Corollary 20.** $\ell \geq 2$

Proof. Let us assume that $\ell = 0$. Since there are no turning points, there is no index s.t. $\tilde{Q}_p[i] \neq \tilde{Q}_p[i+1]$, and consequently the string Q is trivial, a contradiction.
This means $\ell \geq 1$, and by using Corollary 19, $\ell \geq 2$. ◀

▶ **Corollary 21.** $Ham(P^2, Q^\infty) \geq \sum_{i=1}^{\ell} |\tilde{P}[i]| \geq a\ell$.

Proof. Let i be a turning point. By the definition of adjacency string, all the characters in $\tilde{P}[i]$ will align in the next two repetitions against $\tilde{Q}[i], \tilde{Q}[(i+1) \mod q]$. By the definition of turning point, $\tilde{Q}[i] \neq \tilde{Q}[(i+1) \mod q]$ and accordingly each character in $\tilde{P}[i]$ will cause exactly one mismatch. Hence, the minimal number of mismatches is $\sum_{i=1}^{\ell} |\tilde{P}[i]|$. Considering that $\forall i, \tilde{P}[i]$ contains at least a characters, the expression is at least $a\ell$. ◀

Given the above facts, we can now proceed to the proof of Theorem 12.

6.1.2 Co-prime proof

In this subsection, we constrain p, q to be co-prime, namely, $\gcd(p, q) = 1$.

Proof. We will divide the proof to 3 cases, and treat each of them separately.

▶ **Case 1** ($\ell \geq 4$.). When $m = 2$, it holds that

$$Ham(P^2, Q^\infty) \geq 4a = 2(a+1) - 2 + 2a > 2(a+1) - 2 + \left\lfloor \frac{2b}{q} \right\rfloor.$$

We can,accordingly, assume that $m \geq 3$.

Using Corollary 17, at least $4\left\lfloor \frac{mb}{q} \right\rfloor$ indices in the m-backward windows of the turning points are heavy; So using Corollary 21, every $m+1$ repetitions we get at least that many mismatches in addition to the already-calculated number. Set $e_m = \left\lfloor \frac{mb}{q} \right\rfloor$. Given that $m \geq 3$, it holds that either $e_m \geq 1$, or $e_{m+2} \leq 1$ [2], and thus $e_{m+2} \leq 4e_m + 1$.

Using the above results, we show that $Ham(P^{2m}, Q^\infty)$ causes more mismatches than allowed for $Ham(P^{2m+1}, Q^\infty)$.

$$\begin{aligned} Ham(P^{2m}, Q^\infty) &\geq 4am + 4e_{2m-1} \\ &\geq 4am + e_{2m+1} - 1 \geq 4am - 1 + e_{2m+1} \\ &\geq (2m+1)(a+1) + 2am - a - 2m - 2 + e_{2m+1} \\ &\geq (2m+1)(a+1) + 2(m-1)(a-1) - 3 + a + e_{2m+1} \\ &\geq (2m+1)(a+1) - 2 + e_{2m+1} \end{aligned}$$

⌟

In the rest of the cases, there are only two turning points. We denote, for simplicity, t_w, t_b as the turning points from white to black and from black to white, respectively.

▶ **Case 2.** $[\ell = 2, \quad (t_b - t_w) \mod q = 1]$ In words, it means that there is only one occurrence of the character β in Q. Let i_b be an index such that $\tilde{Q}[i_b] = \beta$. By the assumptions, this index is unique.

Let $W = W_m^b(i_b)$. Let $c = |W|$, and set c_w to be the number of white characters in W. The number of black characters in the window, denoted by c_b satisfies $c_b = c - c_w$. Using Corollary 17, we can claim $c \geq ma + \left\lfloor \frac{mb}{q} \right\rfloor$.

If $c_b = 0$, then the black character that must exist in P (since it is not trivial), did not align against the only black character of Q, and thus created m mismatches, so we can assume w.l.o.g that at least one black character appears in the window, as it reduces the number of errors by at least 1.

Putting it all together and maximizing c_w to be $c - 1$, leads to

$$\begin{aligned} Ham(P^m, Q^\infty) &\leq c_w \cdot 1 + (c - c_w) \cdot (m-1) \\ &\leq_1 (c-1) \cdot 1 + m(c - (c-1))(m-1) \\ &= m + c - 2 \\ &= m + am + \left\lfloor \frac{mb}{q} \right\rfloor - 2 \\ &= m(a+1) - 2 + \left\lfloor \frac{mb}{q} \right\rfloor \end{aligned} \tag{1}$$

⌟

[2] If $\frac{3b}{q} < 1$, then $\frac{5b}{q} < 2$

▶ **Case 3** ($\ell = 2, \quad (t_b - t_w) \mod q \neq 1$)**.** First, define $d = (t_b - t_w) \mod q$.

▶ **Corollary 22.** *The number of mismatches caused by each character on index i in the adjacency string $\tilde{P}$ after m repetitions is the minimum between the number of white nodes and black nodes in the m-forward windows of i.*

Proof. Each character creates a mismatch by either aligning with a white index or a black index, so the smallest number of mismatches a character can cause is the minimum between the black and white indices in its next occurrences. ◀

We consider three cases here; the first is where $m \geq d+1$. The second is where $m \leq d$, and also $m \geq 4$. The last is $m \leq d, m \in \{2,3\}$.

▶ **Case 3.1** ($m \geq d+1$)**.** Recall that there are at least as many white characters as black. Mark $W_w = W^b_{m-1}(t_w)$, and $W_b = W^b_2(t_w+2)$. Since $d \geq 2$ and $q \geq m+1$, there are no overlaps between the windows, and all the indices in W_b are black.

In W_w, all of the characters will align against both t_w, t_w+1, and hence create a mismatch. Note that each of the indices $\{t_w - 1, ..., t_w - m + 3\}$ will also align against $t_w - 1$ and $t_w + 2$, hence creating two mismatches. Thus, the total sum of mismatches is at least $a(2+2((t_w-1)-(t_w-m+3)+1)) = 2a(m-2)$, as each index of $\tilde{P}$ has at least a characters.

Considering that $m > d$, $t_w + 1$ will align against $t_b + 1$ and $t_w + 2$ will align against $t_b + 2$, and as a consequence we get at least 2 mismatches for each index, regardless of m. Therefore the total number of mismatches is at least $2a$, which leads to a total sum of $2a(m-1)$ mismatches in both windows. Evaluating this value:

$$\begin{aligned}
&2a(m-1) = 2am - 2a \\
&= am + am - 2a \\
&= m(a+1) + a(m-2) - m \\
&\geq m(a+1) - 2 \text{ (by minimizing } m \text{ to 2)}
\end{aligned}$$

We now show $\left\lfloor \frac{mb}{q} \right\rfloor$ additional mismatches. The number of heavy indices in the m-backwards window of $t_w + 1$ is at least $\left\lfloor \frac{mb}{q} \right\rfloor$, and all of the characters in the window create at least one mismatch, leading to $\left\lfloor \frac{mb}{q} \right\rfloor$ additional mismatches. ⌟

▶ **Case 3.2** ($4 \leq m, \quad m \leq d$)**.** First, it is trivial (yet crucial) to see that $d \geq 4$. Using the previous method, we consider $W^b_{m-1}(t_w)$. Given that $m \leq d$, and using Corollary 22, any character at position $t_w - i$ will cause $\min(i+1, m-i-1)$ mismatches, so we have $1 + 2 + ... + \lfloor \frac{m}{2} \rfloor + ... + 2 + 1 \geq a(1 + 2(m-3) + 1) = 2a(m-2)$ mismatches. The same claim can be made for $W^b_{m-1}(t_b)$, summing up to at least $4a(m-2)$ mismatches after m repetitions.

$$\begin{aligned}
&Ham(P^m, Q^\infty) \geq 4a(m-2) \\
&\geq m(a+1) - m + 3am - 8a \geq m(a+1) + m(3a-1) - 8a \\
&\geq m(a+1) + 4(3a-1) - 8a \geq m(a+1) + 12a - 8a - 4 \\
&\geq m(a+1) + 4(a-1) \geq m(a+1)
\end{aligned}$$

We now need to find $\left\lfloor \frac{mb}{q} \right\rfloor - 2$ additional mismatches. The inequality $\left\lfloor \frac{(m-1)b}{q} \right\rfloor \geq \left\lfloor \frac{mb}{q} \right\rfloor - 1$ holds, and accordingly we have an additional $2\left\lfloor \frac{(m-1)b}{q} \right\rfloor \geq 2(\left\lfloor \frac{mb}{q} \right\rfloor - 1) \geq \left\lfloor \frac{mb}{q} \right\rfloor - 2$ mismatches. ⌟

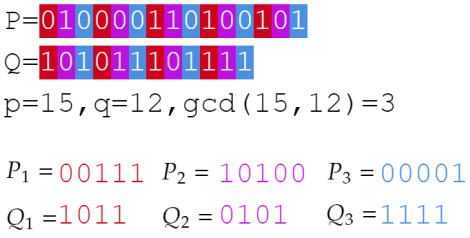


Figure 5 Example for strings decomposition (Definition 25).

▶ **Case 3.3** ($m \leq d, \quad 2 \leq m \leq 3$). In this case, we only have to show that $Ham(P^m, Q^\infty) \geq m(a+1) - 2$.
If $m = 2$, then we have $2a = m(a+1) - 2$ mismatches directly from Corollary 21.
If $m = 3$, then $t_w, t_w - 1, t_b, t_b - 1$ will all create at least one mismatch, which results in $4a$ mismatches, and

$$4a = 3a + a = 3(a+1) - 2 + a - 1 \geq 3(a+1) - 2 = m(a+1) - 2$$ ⌟

◀

6.1.3 Non-divisible proof

The case of $gcd(p, q) = 1$ was proven in Subsection 6.1.2. We now prove that the theorem is true for the more general version, where $q \nmid p$. Let $gcd(p, q) = g > 1$, and let $q' = \frac{q}{g}, p' = \frac{p}{g}$, and $p' = a'q' + b'$.

▶ **Corollary 23.** $\left\lfloor \frac{mb}{q} \right\rfloor = \left\lfloor \frac{mb'}{q'} \right\rfloor$

Proof.

$$\left\lfloor \frac{mb}{q} \right\rfloor = \left\lfloor \frac{m(p \mod q)}{q} \right\rfloor = \left\lfloor \frac{m(g(a'q' + b') \mod gq')}{gq'} \right\rfloor = \left\lfloor \frac{m(gb' \mod gq')}{gq'} \right\rfloor = \left\lfloor \frac{mgb'}{gq'} \right\rfloor = \left\lfloor \frac{mb'}{q'} \right\rfloor$$

◀

▶ **Corollary 24.** $a' = a$

We will now decompose P and Q to smaller strings of co-prime lengths.

▶ **Definition 25** (Decomposed strings). *Given strings P, Q, $\gcd(p, q) = g$, the decomposed string of P at index i is $P_i = P[i]P[i+g]...P[i+(p'-1)g]$. The decomposed strings of Q are defined respectively.*

▶ **Corollary 26.** *$\exists i$ s.t. P_i is non-trivial.*

Proof. If such an index does not exist, for each index i, P_i is trivial, hence $\forall i, P_i = c^{p'}$, and accordingly $P = (P[0]...P[g-1])^{p'}$, and P is periodic, a contradiction to our assumptions. The same claim can be made about Q. ◀

▶ **Definition 27.** *Let P, Q be strings, and let i, j be the minimal indices such that P_i and P_j are not trivial. We will say P, Q are* aliens *if $i \neq j$, and we will say P, Q are* similar *if $i = j$. If $\gcd(p, q) = 1$, then P, Q are aliens regardless.*

The latter definition is rather synthetic, since we only consider the minimal indices.

▶ **Case 1** (P, Q are aliens). Let i, j be the minimal indices s.t. P_i, Q_j are non-trivial.
In this case, $Ham(P^m, Q^\infty) \geq Ham(P_i^m, c_i^\infty) + Ham(c_j^{p'm}, Q_j^\infty)$, where $P_i = c_i^{p'}, Q_j = c_j^{q'}$.
Now,

$$Ham(P_i^m, c_i^\infty) = m \cdot Ham(P_i, c_i^{p'}) \geq m \tag{1}$$

$$\begin{aligned} Ham(c_j^{p'm}, Q_j^\infty) &\geq m \cdot Ham(c_j^{p'}, Q_j^{\left\lfloor \frac{p'm}{q'} \right\rfloor}) \\ &\geq \left\lfloor \frac{p'm}{q'} \right\rfloor = \left\lfloor \frac{m(a'q'+b')}{q'} \right\rfloor \\ &= ma' + \left\lfloor \frac{mb'}{q'} \right\rfloor = ma + \left\lfloor \frac{mb}{q} \right\rfloor \end{aligned} \tag{2}$$

So,

$$Ham(P^m, Q^\infty) \geq Ham(P_i^m, c_i^\infty) + Ham(c_j^{p'm}, Q_j^\infty) \geq m + am + \left\lfloor \frac{mb}{q} \right\rfloor = m(a+1) + \left\lfloor \frac{mb}{q} \right\rfloor \quad ⌟$$

▶ **Corollary 28.** *If P, Q are alien strings of non co-prime lengths, then $Ham(P^m, Q^\infty) \geq m(a+1) + \left\lfloor \frac{mb}{q} \right\rfloor$, for all $m > 0$.*

▶ **Case 2** (P, Q are similar). Let i, j be the minimal indices s.t. P_i, Q_j are not trivial. It holds that

$$Ham(P^m, Q^\infty) \geq Ham(P_i^m, Q_i^\infty) \geq m(a'+1) - 2 + \left\lfloor \frac{mb'}{q'} \right\rfloor = m(a+1) - 2 + \left\lfloor \frac{mb}{q} \right\rfloor$$

Which is exactly the required expression, with the LCM occurring more often. ⌟

6.2 Candidates for periods

▶ **Definition 29.** *Let T be a text, and P a string. We say that P is a* periodic seed *(or* seed*) of T, if $Ham(T, P^\infty) = k \to t \geq 2kp$ and P is non-periodic.*

▶ **Lemma 30.** *Let P, Q be non-trivial strings of co-prime lengths (i.e., $\gcd(p, q) = 1$), then $Ham(P^q, Q^p) \geq d(p-2) + q$, where d is the number of occurrences of the less frequent character in either P or Q.*

Proof. Let P, Q be non-trivial strings of co-prime lengths p, q. Let σ_S be the number of occurrences of a character σ in a string S.
In the LCM, i.e., when the strings P, Q appear enough times to perfectly align with each other, each characters of P aligns with each character of Q exactly once (using basic abstract algebra properties). It means that $Ham(P^q, Q^p) = \alpha_P\beta_Q + \beta_P\alpha_Q$.
Seeing that P, Q are both binary strings implies $\beta_P = p - \alpha_P$, and $\beta_Q = q - \alpha_Q$.
Assume $\alpha_Q \geq \beta_Q$, and that $\beta_Q \geq d \geq 1$.

$$\begin{aligned} Ham(P^q, Q^p) &= \alpha_P\beta_Q + \beta_P\alpha_Q \\ &= \alpha_P(q - \alpha_Q) + (p - \alpha_P)\alpha_Q \\ &\geq_1 (p-1)(q - \alpha_Q) + 1\alpha_Q \\ &= (p-1)d + (q-d) \\ &= d(p-2) + q \end{aligned}$$

(1) is true since $a_Q \geq b_Q$, so be minimizing b_P we minimize the expression. ◀

▶ **Lemma 31.** *For alien strings Q, P s.t. $q \leq \frac{p}{4}$, $\gcd(p,q) = 1$ and $p = aq + b, 2 \leq m < q$,*

$$Ham(P^{nq+m}, Q^{\infty}) \geq n(p+q-2) + m\left\lceil \frac{p}{q} \right\rceil - 2 + \left\lfloor \frac{mb}{q} \right\rfloor$$

Proof. Using Theorem 12, we only need to show this is true when $d \geq 2, m \leq d$, where d is $(t_b - t_w) \mod q$, as defined. Because the case where $d = 1 \vee m \geq 4 \vee m > d$ is proved regardless of m, d and when $m \leq 1$ the expression can simply evaluate to 0. If $q \leq \frac{p}{4}$, then either $m \geq 4$ or $n \geq 1$. One case is already proven, so we can presume $n \geq 1$. Naturally, $m \leq 3$, as otherwise the lemma is trivially proven.
By Lemma 30, the distance at the LCM is $d(p-2)+q$. Consider the distance at the LCM in the original expression: $p+q-2$. The difference between these values is $(d-1)(p-2) \geq p-2$. This means the number of mismatches is increased by at least $p-2$ every LCM.

By Theorem 12, $Ham(P^m,Q^{\infty}) \geq m(a+1)$–2, and we attempt to show that $Ham(P^m,Q^{\infty}) \geq m(a+1) - 2 + \left\lfloor \frac{mb}{q} \right\rfloor$, leaving us to show that the additional $p-2$ mismatches are greater than $\left\lfloor \frac{mb}{q} \right\rfloor$. Evaluate:

$$p - 2 = aq + b - 2 \geq b > \left\lfloor \frac{mb}{q} \right\rfloor$$ ◀

▶ **Theorem 32.** *Let T be a text, and let P be a seed of T such that $p \leq \frac{t}{4}$, then for all strings Q s.t. $q < p, q \nmid p$, then Q can not be a seed of T.*

Proof. Let P be a seed of T of length $p \leq \frac{t}{4}$, and assume Q is a seed of T, and $q < p, q \nmid p$. In this proof, we use the same notations as in Theorem 31.
Set $occ_p = \left\lfloor \frac{t}{p} \right\rfloor = nq + m, 0 \leq m < q$, and set $p = aq + b, 1 \leq b < q$. $nq + m \geq 4$.
Obviously, $t < (nq + m + 1)p$. Therefore the maximum number of mismatches between T and P^{∞}, marked as k_p is at most $\left\lfloor \frac{nq+m}{2} \right\rfloor$. The number of occurrences of Q in T is $occ_q = \left\lfloor \frac{t}{q} \right\rfloor = \left\lfloor \frac{(nq+m+1)p-1}{q} \right\rfloor$, and thus the number of mismatches between T and Q^{∞}, marked as k_q accordingly is at most $\left\lfloor \frac{occ_q}{2} \right\rfloor$.

By the triangle inequality, $Ham(P^{nq+m}, Q^{\infty}) \leq Ham(T, P^{\infty}) + Ham(T, Q^{\infty}) \leq k_p + k_q$. We will show that $Ham(P^{nq+m}, Q^{\infty}) > k_p + k_q$, which will lead to a contradiction.

We split the proof into two cases - in the first case P, Q are aliens, and in the second, P, Q are similar (See Definition 27).

6.2.1 Proof for alien strings

Begin by evaluating the minimal number of errors:

$$Ham(P^{nq+m}, Q^{\infty}) \geq n(p+q-2) + m(a+1) - 2 + \left\lfloor \frac{mb}{q} \right\rfloor$$

Since we required $p \leq \frac{t}{4}$, then $n \geq 1$ or $m \geq 4$. We consider both cases.

▶ **Case 1** ($n = 0, m \geq 4$)**.** Begin by re-evaluating the previous expressions by setting $n = 0$, which will lead to significantly shorter expressions.

$$Ham(P^{nq+m}, Q^{\infty}) = n(p+q-2) + m(a+1) - 2 + \left\lfloor \frac{mb}{q} \right\rfloor = m(a+1) - 2 + \left\lfloor \frac{mb}{q} \right\rfloor \quad (1)$$

$$occ_p = m \quad (2)$$

$$occ_q = \frac{\left\lfloor \frac{(m+1)p-1}{q} \right\rfloor}{2} \quad (3)$$

Again split into two cases:

1. $b \geq \frac{q}{2}$
2. $b < \frac{q}{2}$

▶ **Case 1.1** ($b \geq \frac{q}{2}$)**.** In this case, $\left\lfloor \frac{mb}{q} \right\rfloor \geq \left\lfloor \frac{mq}{2q} \right\rfloor = \lfloor \frac{m}{2} \rfloor$. Seeing that the latter is exactly the value of k_p, let's us subtract $\lfloor \frac{m}{2} \rfloor$ from both sides of the equation, which leads to the following inequality:

$$m(a+1) - 2 > k_q = \left\lfloor \frac{\left\lfloor \frac{(m+1)p-1}{q} \right\rfloor}{2} \right\rfloor$$

Given that we required $b \geq \frac{q}{2}$, we will evaluate $\left\lfloor \frac{(m+1)p-1}{q} \right\rfloor$ in worst-case settings in terms of mismatches, i.e. $b = q - 1$.

$$\left\lfloor \frac{(m+1)p-1}{q} \right\rfloor \leq \left\lfloor \frac{(m+1)((a+1)q-1)-1}{q} \right\rfloor = \left\lfloor \frac{q(m+1)(a+1)-m-2}{q} \right\rfloor \leq (a+1)(m+1)-1$$

What left to prove is $m(a+1) - 2 > \left\lfloor \frac{(a+1)(m+1)-1}{2} \right\rfloor$.

$$\begin{aligned} m(a+1) - 2 > \left\lfloor \frac{(a+1)(m+1)-1}{2} \right\rfloor &\rightarrow 2m(a+1) - 4 > (a+1)(m+1) - 1 \\ &\rightarrow 2am + 2m - 4 > am + a + m + 1 - 1 \\ &\rightarrow am + m - a - 4 > 0 \\ &\rightarrow a(m-1) + m - 4 > 0 \\ &\rightarrow 2m - 5 > 0 \rightarrow m \geq 3 \end{aligned}$$

⌟

▶ **Case 1.2** ($b < \frac{q}{2}$)**.** In this case, we suppose that $\left\lfloor \frac{mb}{q} \right\rfloor = 0$, and again evaluate $\left\lfloor \frac{(m+1)p-1}{q} \right\rfloor$ in worst-case settings ($b = \frac{q}{2}$).

$$(m+1)p - 1 \leq (m+1)((a+0.5)q - 1) - 1 \leq (m+1)(a+0.5)q - 1$$

Using the above result, the inequality $\left\lfloor \frac{(m+1)p-1}{q} \right\rfloor \leq (m+1)(a+0.5) - 1$ holds, and thereby the resulting inequality is

$$m(a+1) - 2 > \left\lfloor \frac{(m+1)(a+0.5)-1}{2} \right\rfloor + \left\lfloor \frac{m}{2} \right\rfloor$$

Further evaluation leads to

$$\begin{aligned} m(a+1) - 2 > \left\lfloor \frac{(m+1)(a+0.5)-1}{2} \right\rfloor + \left\lfloor \frac{m}{2} \right\rfloor \\ \rightarrow 4m(a+1) - 8 > (m+1)(2a+1) - 2 + 2m \\ \rightarrow 4am + 4m > 2am + m + 2a + 2m + 1 + 8 - 2 \\ \rightarrow 2am + m - 2a \geq 8 \\ \rightarrow 2a(m-1) + m \geq 8 \\ \rightarrow 2 \cdot 3 + 4 \geq 8 \qquad \text{(Setting } a = 1, m = 4\text{)} \end{aligned}$$

⌟

▶ **Case 2** ($n \geq 1, m \in \{0,1\}$)**.** This case is rather complicated. Begin by examining the case $m = 0$ which is the easier one.

▶ **Case 2.1** ($m = 0$)**.** Recall that $nq + m \geq 4 \rightarrow nq \geq 3 \rightarrow q \geq 3 \vee n \geq 2$. We require $nq \geq 3$ and not $nq \geq 4$, so we will also be able to use the expression when $m = 1$.
In this case, the number of occurrences of Q in T is $occ_q \leq np + a$, so we need to show that $n(p + q - 2) > \lfloor \frac{np+a}{2} \rfloor + \lfloor \frac{nq}{2} \rfloor$. We will multiply it by 2 to obtain

$$\begin{aligned} &2n(p+q-2) > np + a + nq \\ &\rightarrow n(p+q-4) \geq a+1 \\ &\rightarrow n((a+1)q-3) \geq a+1 \\ &\rightarrow_1 n(2q-3) \geq 2 \rightarrow q \geq 3 \vee n \geq 2 \end{aligned}$$

(1) - Because we are incrementing a increments the left hand-side by qn, and the right hand-side by only 1 ($qn > 1$).

▶ **Corollary 33.** $Ham(Q^{a+m+1}, P^2) \geq 2m, \quad m < a$

Proof. First, define $P_{\rightarrow i}$ as the string cyclic shift of P i positions to the right.
Consider the hamming distance $Ham(Q^{m+1}, Q^{m+1}_{\rightarrow(q-b)})$. The first string is aligned with the first repetition of P, the other is aligned with the second repetition of P.
Since there are m full repetitions of Q in both strings, there are at least $2m$ mismatches. Using the triangle inequality:

$$2m \leq Ham(Q^{m+1}, Q^{m+1}_{\rightarrow(q-b)}) \leq Ham(Q^{m+1}, P) + Ham(P, Q^{m+1}_{\rightarrow(q-b)}) \leq Ham(Q^{a+m+1}, P^2)$$

◀

▶ **Corollary 34.** *If Q is a seed of a text with tail size $t = t - (nq+1)p$ s.t. $t \geq 2q - b$, then Q is also the seed of a text with tail size $2q - b - 1$.*

The proof is trivial: any additional occurrence of Q in T will cause at-least 2 more verified mismatches. We allow one mismatch for 2 occurrences so it is always possible to reduce the number of mismatches by 2 and the number of occurrences by 1.

▶ **Lemma 35.** *If $Ham(Q^{a+1}, P^2) = 0$, then $b \geq 2$*

Proof. By definition, $q(a+1) > p$. If $Ham(Q^{a+1}, P^2) = 0$, then $P = Q^a Q[..b]$, and that $P[..q-b] = Q[b..] \rightarrow Q[..q-b] = Q[b..]$.
Assume $b = 1$. This means $Q[..q-1] = Q[1..]$, or

$$Q[0] = Q[1], Q[1] = Q[2], ...Q[-2] = Q[-1]$$

Which implies that Q is a trivial periodic string, contradiction. ◀

▶ **Case 2.2** ($m = b = 1$)**.** In this case, Q can occur at most a times in the tail without creating an extra mismatch. Thus, we will suppose it occurs exactly a times in the tail.
$n(p + q - 2) > \lfloor \frac{np+a}{2} \rfloor + \lfloor \frac{nq+1}{2} \rfloor$

This is almost exactly the same as the case where $m = 0$, with the difference of k_p, which changed from $\lfloor \frac{nq}{2} \rfloor$ to $\lfloor \frac{nq+1}{2} \rfloor$. If either n or q are even, it is exactly the same expression as before, and accordingly we presume $q \neq 2 \rightarrow q \geq 3 \rightarrow p \geq 4$.

Evaluating the expression in a similar manner to case 2.1 will lead to the inequality $n(2q-3) \geq 3$, and by setting $q = 3, n = 1$, we get $1 \cdot (2 \cdot 3 - 3) = 3 \geq 3$.

▶ **Case 2.3** ($m = 1, b > 1$)**.** This case is trivial. Requiring $b \geq 2$ results in $p \geq q + 2$, which in turn causes an additional mismatch on the LCM expression $p + q - 2$, but creates only one additional occurrence for Q, and the problem can be reduced to previous case. ⌟

▶ **Case 3** ($n \geq 1, m \geq 2$)**.** This is the simplest case of the three. We have $p > q \geq 3$, as $m < q < p$, and $m \geq 2$.

Proof. The length of T is at most $(nq + m + 1)p - 1$. The string Q occurs in nqp characters exactly np times. And $(m + 1)p - 1 = (m + 1)(aq + b) - 1 = maq + mb + aq + b - 1$, which contains $a(m + 1) + \left\lfloor \frac{b(m+1)-1}{q} \right\rfloor$ instances of Q.
The inequality $\left\lfloor \frac{b(m+1)-1}{q} \right\rfloor \leq \left\lfloor \frac{bm}{q} \right\rfloor + 1$ holds, and as a result we claim that the number of occurrences of Q in T is $\left\lfloor \frac{b(m+1)-1}{q} \right\rfloor + 1$.

The expression $\left\lfloor \frac{bm}{q} \right\rfloor$ also appears in $Ham(P^m, Q^\infty)$, so we can subtract it from both sides, which will simplify the expression to $n(p+q-2)+m(a+1)-2 > \left\lfloor \frac{np+a(m+1)+1}{2} \right\rfloor + \left\lfloor \frac{nq+m}{2} \right\rfloor$. Multiply it by 2, and get the following:

$$
\begin{aligned}
&2n(p + q - 2) + 2m(a + 1) - 4 > np + a(m + 1) + 1 + nq + m \\
&\to n(p + q - 4) + 2am + 2m - 4 > am + a + 1 + m \\
&\to n((a + 1)q - 3) + am - a + m \geq 6 \\
&\to_1 n(2q - 3) + 2m - 1 \geq 6 \\
&\to_2 3n + 2m \geq 7 \to 3 \cdot 1 + 2 \cdot 2 \geq 7
\end{aligned}
$$

(1) - Setting a to minimum ($a = 1$)
(2) - Setting q to minimum ($q = 3$) ◀

⌟

6.2.2 Proof for similar strings

We now present the proof of the theorem when $\gcd(p, q) > 1$, and $i = j$.

We use the definition of P_i, Q_i, p', q' as usual, and also define T_i as $T[i]T[i + g][T + 2g]$....
Because t is not necessarily divisible by g some T_i might be shorter than the others.
Let i be the index such that both P_i, Q_i are non-trivial. The inequality $\forall j, T_j \geq 4p'$ holds as $t \geq 4pg$.
As k_p is the maximal number of full occurrences of P in T, the number of occurrences of P_i in T_i is at least k_p. Proof for k_q is the same.

We now have Q_i, P_i, T_i, where $P_i \leq \frac{|T_i|}{4}$ and $gcd(P_i, Q_i) = 1$, and therefore P_i and Q_i cannot both be seeds of T_i, or in other words, $Ham(p_i^{nq+m}, q_i^\infty) > k_q + k_p$. In contrast, we know that $Ham(p^{nm+q}, q^\infty) \geq Ham(p_i^{nq+m}, q_i^\infty) > k_q + k_p$, and P, Q cannot both be seeds. ◀

▶ **Definition 36** (Binary Renaming Function)**.** *A binary renaming function is a function from general alphabet to binary alphabet, $\delta : \Sigma \to \{a, b\}$. The generalized renaming function $\delta^* : \Sigma^* \to \{a, b\}^*$ is defined in the standard way:*

1. $\delta^*(\epsilon) = \epsilon$
2. $\delta^*(P) = \delta(P[0])\delta^*(P[1..])$

▶ **Lemma 37.** *Let P, Q be non-trivial strings over Σ. W.l.o.g $p \geq q$. $\forall \delta^*, m$, $Ham(P^m, Q^\infty) \geq Ham(\delta^*(P)^m, \delta^*(Q)^\infty)$*

Proof. Let e_i be an indicator on mismatch on index i between P^m and Q^∞, and e'_i be defined respectively for $\delta^*(P^m)$, $\delta^*(Q^\infty)$. Since $e_i = 0$ indicates a match, then $e'_i = 0$ as well, which implies $e'_i \leq e_i$. The contrary is not true. The direct implication is our lemma. ◀

▶ **Corollary 38.** *Theorem 12 is true for general alphabet.*

▶ **Corollary 39.** *Theorem 43 is true for general alphabet.*

Proof. Assume there are $P, Q \in \Sigma^*$ that contradict 43. This means that $Ham(T, P^\infty) + Ham(T, Q^\infty) \leq k_p + k_q$, but using Lemma 37,

$$\exists \delta^*, Ham(T, P^\infty) + Ham(T, Q^\infty) \geq Ham(\delta^*(T), \delta^*(P)^\infty) + Ham(\delta^*(T), \delta^*(Q)^\infty) >_1 k_q + k_p$$

And (1) is true directly from Theorem 43. ◀

6.3 The Number of Candidates

▶ **Lemma 40.** *Let T be text, and let P, Q be periodic seeds of T, such that $p > q, q \nmid p, p \leq \frac{t}{3}$. Then, $\exists g$ s.t. $p = 3g, q = 2g$.*

The proof appears in the full version of the paper,

▶ **Corollary 41.** *Given a text T, there is at most one periodic seed P of length $\frac{t}{4} < p \leq \frac{t}{3}$.*

▶ **Corollary 42.** *If P, Q are seeds of a text T and $p = q$ then $Ham(P, Q) \leq 1$*

Proof. Assume $Ham(P, Q) \geq 2$, and that $\ell = \left\lfloor \frac{t}{p} \right\rfloor$. Using the definition of a seed, $Ham(P^\ell, Q^\ell) \leq Ham(T, P^\infty) + Ham(T, Q^\infty) \leq \ell$, at the same time $Ham(P^\ell, Q^\ell) = 2\ell$, contradiction. ◀

▶ **Theorem 43.** *There are at most $2\log_3 n$ candidates of length $\ell \leq \frac{t}{3}$*

Proof. We begin by proving a useful lemma regarding the number of candidates of divisible length.

▶ **Lemma 44.** *If there are two candidates of length g, there is at most one candidate of length $2g$.*

The proof appears in the full version of the paper.

▶ **Corollary 45.** *If all of the periods of length $\ell \leq n$ divide n, then there are at most $2\log_3 n$ periods of length $\leq n$.*

The proof appears in the full version of the paper. ◀

▶ **Corollary 46.** *There are at most 2 periods that are canonical.*

Proof. As we have proved, if there are 3 different periods, then the length of the shortest period, divides the length of the longer periods, and by simple induction the minimal period's length divides the lengths of the rest of the periods, and is canonical to them.
If there are no 3 different periods then we can call all of the periods canonical. ◀

▶ **Corollary 47** (Generalized gcd-theorem)**.** *Let T be a text, and let P, Q be seeds of T that occur at least 3 full times in T, then T has a seed of length* $\gcd(p, q)$.

7 The Error Upper Bounds Hierarchy

The previously known [3] upper error bound was $\lfloor \frac{n}{(2+\epsilon)p} \rfloor$ and led to $O(\log_{1+\epsilon} n)$ period candidates. To achieve a hierarchy of upper bounds, we follow similar techniques to those of [3]. The innovation is the new *Hamming distance* error bounds that we define. All the upper bounds in this section are of (n, p, ϵ)-type. We do not get rid of ϵ at the moment. However, we feel that by methods similar to those of Section 6, the ϵ can be eliminated from the error bounds.

▶ **Lemma 48.** *Let T be the input text. For any period P, assume that $Ham(T, T_P) \leq \frac{n}{(2+\epsilon)\cdot(2^{(i)})((\log^{(i)} p)^{1+\epsilon})}$. Then there are $O(\log^{(i+1)} n)$ candidates for P, for any $i \geq 0$. Note that, for $i = 0$, the upper bound is $Ham(T, T_P) \leq \frac{n}{(2+\epsilon)\cdot p^{1+\epsilon}}$*

▶ **Lemma 49.** *Let T be the input text. For any period P, assume that $Ham(T, T_P) \leq \frac{n}{(2+\epsilon)^{p+1}}$. Then there are $O(\log^* n)$ candidates for P*

▶ **Lemma 50.** *Lemmas 48 and 49 have tight examples.*

8 Conclusions and Open Problems

We showed that $\lfloor \frac{n}{2p} \rfloor$ is a tight upper error bound and it results in $2\log_3 n$ candidates if P fully occurs at least 3 times in T. Otherwise, if P fully occurs twice (maybe with a tail) then we provide an example with $\Omega(n)$ candidates.

It is easy to show that our result can be generalized to *c-pseudo local metrics* [3]. Such a metric Δ behaves like *Hamming distance* in the manner that for every pair of equal length strings T and S, $\Delta(T, S) \leq c \cdot Ham(T, S)$. An example of such a metric is *swap distance*, where the distance counted by the number of swaps of two consecutive letters (each letter can only participate in a single swap). It is easy to verify that $\lfloor \frac{n}{2cp} \rfloor$ is a tight upper bound, and the result of *canonic period seed* can also be applied to such metrics. By a similar fashion, replacing $(2+\epsilon)$ by $(2c+\epsilon)$ in the hierarchy upper bounds allows the hierarchy generalization to *c-pseudo local metrics*.

In this paper, corruptions are of the replacement type. That is, the error count is the number of changed letters. It is of interest to analyze different types of error distance metrics besides the *Hamming Distance*. One example of such a metric, that has been considered in the recovery algorithms literature, is the *Edit Distance* [2, 14].

It is of interest to apply the ideas of section 6 to the $\log^{(i)}$ hierarchy and eliminate the dependence on ϵ.

Finally, can the hierarchy be extended to the other side? Come up with tight error bounds that lead to a set of n^ϵ period candidates, where $\epsilon < 1$.

References

1 S. Aluru, A. Apostolico, and S. V. Thankachan. Efficient alignment free sequence comparison with bounded mismatches. In *Proc. 19th Research in Computational Molecular Biology Conference, RECOMB*, pages 1–12, 2015.

2 A. Amir, M. Amit, G.M.Landau, and D. Sokol. Period recovery of strings over the hamming and edit distances. *Theortetical Computer Science*, 710:2–18, 2018.

3 A. Amir, E. Eisenberg, A. Levy, E. Porat, and N. Shapira. Cycle detection and correction. *ACM Trans. Alg.*, 9(1):13, 2012.

4 A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees - metrics and efficient algorithms. *Proc. FOCS 94*, pages 758–769, 1994.

5 A. Amir, A. Levy, M. Lewenstein, R. Lubin, and B. Porat. Can we recover the cover? In *Proc. 28st Annual Symposium on Combinatorial Pattern Matching (CPM)*, LIPICS, 2017.

6 A. Amir, M. Lewenstein, and E. Porat. Approximate subset matching with "don't care"s. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 305–306, 2001.

7 R.L. Cann, M. Stoneking, and A.C. Wilson. Mitochondrial DNA and human evolution. *Nature*, 325(6099):31–36, 1987.

8 M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Proc. 25th Annual ACM Symposium on the Theory of Computing*, pages 137–145, 1993.

9 M. Farach, T. M. Przytycka, and M. Thorup. Computing the agreement of trees with bounded degrees. *Proc. 3rd European Symposium on Algorithms*, pages 381–393, 1995.

10 N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.*, 16:109–114, 1965.

11 S. Har-Peled and S. Mahabadi. Proximity in the age of distraction: Robust approximate nearest neighbor search. In *Proc. 28th ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1–15, 2017.

12 S. Heydrich and A. Wiese. Faster approximation schemes for the two-dimensional knapsack problem. In *Proc. 28th ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 79–98, 2017.

13 C. Kalaitzis. An improved approximation guarantee for the maximum budgeted allocation problem. In *Proc. 27th ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1048–1066, 2016.

14 T. Kociumaka, J. Radoszewski, W. Rytter, J. Straszyński, T. Waleń, and W. Zuba. Faster recovery of approximate periods over edit distance. In *Proc. 25th International Symposium on String Processing and Information Retrieval (SPIRE)*, LNCS, pages 233–240. Springer, 2018.

15 L.A.B. Kowada, D. Doerr, S. Dantas, and J. Stoye. New genome similarity measures based on conserved gene adjacencies. In *Proc. 20th Research in Computational Molecular Biology Conference, RECOMB*, pages 204–224, 2016.

16 A. Ojewole, J.D. Jou, V.G. Fowler, and B.R. Donald. Bbk* (branch and bound over k*): A provable and efficient ensemble-based algorithm to optimize stability and binding affinity over large sequence spaces. In *Proc. 21st Research in Computational Molecular Biology Conference, RECOMB*, pages 157–172, 2017.

17 A. Sobih, A. I. Tomescu, and V. Mäkinen. Metaflow: Metagenomic profiling based on whole-genome coverage analysis with min-cost flows. In *Proc. 20th Research in Computational Molecular Biology Conference, RECOMB*, pages 111–121, 2016.

18 M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, 1993.

19 M. A. Steel and D. Penny. Distributions of tree comparison metrics - some new results. *Syst. Biol.*, 42:126–141, 1993.

20 E. Ukkonen. A linear-time algorithm for finding approximate shortest common superstrings. *Algorithmica*, 5:313–323, 1990.

21 J. C. Venter and Celera Genomics Corporation. The sequence of the human genome. *Science*, (291):1304–1351, 2001.

22 C. Wulff-Nilsen. Approximate distance oracles for planar graphs with improved query time-space tradeoff. In *Proc. 27th ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 351–362, 2016.

Approximation of the Diagonal of a Laplacian's Pseudoinverse for Complex Network Analysis

Eugenio Angriman
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
angrimae@hu-berlin.de

Maria Predari
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
predarim@hu-berlin.de

Alexander van der Grinten
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
avdgrinten@hu-berlin.de

Henning Meyerhenke
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
meyerhenke@hu-berlin.de

Abstract

The ubiquity of massive graph data sets in numerous applications requires fast algorithms for extracting knowledge from these data. We are motivated here by three electrical measures for the analysis of large small-world graphs $G = (V, E)$ – i. e., graphs with diameter in $\mathcal{O}(\log |V|)$, which are abundant in complex network analysis. From a computational point of view, the three measures have in common that their crucial component is the diagonal of the graph Laplacian's pseudoinverse, $\mathbf{L}^\dagger$. Computing $\mathrm{diag}(\mathbf{L}^\dagger)$ exactly by pseudoinversion, however, is as expensive as dense matrix multiplication – and the standard tools in practice even require cubic time. Moreover, the pseudoinverse requires quadratic space – hardly feasible for large graphs. Resorting to approximation by, e. g., using the Johnson-Lindenstrauss transform, requires the solution of $\mathcal{O}(\log |V|/\epsilon^2)$ Laplacian linear systems to guarantee a relative error, which is still very expensive for large inputs.

In this paper, we present a novel approximation algorithm that requires the solution of only one Laplacian linear system. The remaining parts are purely combinatorial – mainly sampling uniform spanning trees, which we relate to $\mathrm{diag}(\mathbf{L}^\dagger)$ via effective resistances. For small-world networks, our algorithm obtains a $\pm\epsilon$-approximation with high probability, in a time that is nearly-linear in $|E|$ and quadratic in $1/\epsilon$. Another positive aspect of our algorithm is its parallel nature due to independent sampling. We thus provide two parallel implementations of our algorithm: one using OpenMP, one MPI + OpenMP. In our experiments against the state of the art, our algorithm (i) yields more accurate approximation results for $\mathrm{diag}(\mathbf{L}^\dagger)$, (ii) is much faster and more memory-efficient, and (iii) obtains good parallel speedups, in particular in the distributed setting.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Graph algorithms analysis; Theory of computation → Parallel algorithms; Mathematics of computing → Solvers

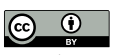
Keywords and phrases Laplacian pseudoinverse, electrical centrality measures, uniform spanning tree, effective resistance, parallel sampling

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.6

Related Version `https://arxiv.org/abs/2006.13679`

Funding This work is partially supported by German Research Foundation (DFG) grant ME 3619/3-2 within Priority Programme 1736 *Algorithms for Big Data* and by DFG grant ME 3619/4-1 (*Accelerating Matrix Computations for Mining Large Dynamic Complex Networks*).

Acknowledgements We thank our colleague Fabian Brandt-Tumescheit for his technical support for the experiments.



1 Introduction

Massive graph data sets are abundant these days in numerous applications. Extracting knowledge from these data thus requires fast algorithms. One common matrix to represent a graph $G = (V, E)$ with n vertices and m edges in algebraic algorithms is its Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Here, $\mathbf{D}$ is the diagonal degree matrix with $\mathbf{D}[u, u]$ being the (possibly weighted) degree of vertex $u \in V$. The matrix $\mathbf{A}$ is the (possibly weighted) adjacency matrix of G. It is well-known that $\mathbf{L}$ does not have full rank and is thus not invertible. Its Moore-Penrose pseudoinverse [25] $\mathbf{L}^\dagger$, in turn, has numerous applications in physics and engineering [60] as well as applied mathematics [25] and graph (resp. matrix) algorithms [38].

We are motivated by one particular class of applications: *electrical centrality measures* for the analysis of small-world networks – i. e., graphs whose diameter is bounded by $\mathcal{O}(\log n)$. Many important real-world networks (social, epidemiological, information, biological, etc.) have the small-world feature [48]. Centrality measures, in turn, belong to the most widely used network analysis concepts and indicate the importance of a vertex (or edge) in the network [13]. Numerous measures exist, some based on shortest paths, others consider paths of arbitrary lengths. Electrical centrality measures fall into the latter category. They exploit the perspective of graphs as electrical networks (see e. g., [41]). One well-known of such measures is *electrical closeness centrality*, a. k. a. *current-flow closeness* or *information centrality* [16], $c^{el}(\cdot)$. It is the reciprocal of the average effective resistance $\mathbf{r}(u, \cdot)$ between u and all other vertices:

$$c^{el}(u) := \frac{n-1}{\sum_{v \in V \setminus \{u\}} \mathbf{r}(u, v)}. \tag{1}$$

In an electrical network corresponding to G, $\mathbf{r}(u, v)$ is the potential (voltage) difference across terminals u and v when a unit current is applied between them [23]. It can be computed by solving $\mathbf{L}\mathbf{x} = \mathbf{e}_u - \mathbf{e}_v$ for $\mathbf{x}$, where $\mathbf{e}_z$ is the canonical unit vector for vertex z. Then, $\mathbf{r}(u, v) = \mathbf{x}[u] - \mathbf{x}[v]$, also see Section 2.1.

Effective resistance also plays a major role in two other electrical measures we consider here, *normalized random-walk betweenness* [47] and *Kirchhoff index centrality* [39]. Also note that effective resistance is a graph metric with numerous other applications, well beyond its usage in electrical centralities (cf. Refs. [2, 23]). A straightforward way to compute electrical closeness (or the other two measures) would be to compute $\mathbf{L}^\dagger$. Without exploiting structure, this takes $\mathcal{O}(n^\omega)$ time, where $\omega < 2.38$ is the exponent for fast matrix multiplication. The standard tools in practice even require cubic time, cf. [52]. $\mathbf{L}^\dagger$ is also in general a dense matrix (also for sparse $\mathbf{L}$). Thus, full (pseudo)inversion is clearly limited to small inputs.

Conceptually similar to inversion would be to solve $\Theta(n)$ Laplacian linear systems. In situations with lower accuracy demands, fewer linear systems suffice: using the Johnson-Lindenstrauss transform (JLT) in connection with a fast Laplacian solver such as Ref. [19], one gets a relative approximation guarantee by solving $\mathcal{O}(\log n/\epsilon^2)$ systems [57] in $\tilde{\mathcal{O}}(m \log^{1/2} n \log(1/\epsilon))$ time each, where $\tilde{\mathcal{O}}(\cdot)$ hides a $\mathcal{O}((\log\log n)^{3+\delta})$ factor for $\delta > 0$.

As pointed out previously [15], the (only) relevant part of $\mathbf{L}^\dagger$ for computing electrical closeness is its diagonal (we will see that this is true for other measures as well). Numerical methods for sampling-based approximation of the diagonal of implicitly given matrices do exist [9]. Yet, for our purpose, they solve $\mathcal{O}(\log n/\epsilon^2)$ Laplacian linear systems as well to obtain an ϵ-approximation with high probability, see Section 2.2 for more details.

While this number of Laplacian linear systems can be solved in parallel, their solution can still be time-consuming in practice, in part due to high constants hidden in the $\mathcal{O}$-notation.

Contribution and Outline. We propose a new algorithm for approximating $\text{diag}(\mathbf{L}^\dagger)$ of a Laplacian matrix $\mathbf{L}$ that corresponds to weighted undirected graphs (Section 3). Our main technique is the approximation of effective resistances between a pivot vertex $u \in V$ and all other vertices of G. It is based on sampling uniform (= random) spanning trees (USTs). The resulting algorithm is highly parallel and (almost) purely combinatorial – it relies on the connection between Laplacian linear systems, effective resistances, and USTs.

For small-world graphs, our algorithm obtains an absolute $\pm\epsilon$-approximation guarantee with high probability in (sequential) time $\mathcal{O}(m \log^4 n \cdot \epsilon^{-2})$. In particular, compared to using the fastest theoretical Laplacian solvers in connection with JLT, our approach is off by only a polylogarithmic factor. Probably more importantly, after some algorithm engineering (Section 4), our algorithm performs much better than the state of the art, already in our sequential experiments (Section 5): (i) it is much faster and more memory-efficient, (ii) it yields a maximum absolute error that is one order of magnitude lower, and (iii) results in a more accurate complete centrality ranking of elements of $\text{diag}(\mathbf{L}^\dagger)$. Due to good parallel speedups, we can even compute a reasonably accurate diagonal of $\mathbf{L}^\dagger$ on a small-scale cluster with 16 compute nodes in less than 8 minutes for a graph with $\approx$ 13.6M vertices and $\approx$ 334.6M edges. **Material omitted due to space constraints can be found in the appendix of the full version of this paper [4].**

2 Preliminaries

2.1 Problem Description and Notation

We type vectors and matrices in bold font. As input we consider simple, finite, connected undirected graphs $G = (V, E)$ with n vertices, m edges, and non-negative edge weights $\mathbf{w} \in \mathbb{R}^m_{\geq 0}$. For the complexity analysis, we usually assume that the diameter of G is $\mathcal{O}(\log n)$, but our algorithm would also work correctly without this assumption.

Graphs as electrical networks. We interpret G as an electrical network in which every edge $e \in E$ represents a resistor with resistance $1/\mathbf{w}[e]$. In this context, it is customary to fix an arbitrary orientation $E^\pm$ of the edges in E and to define a unit s-t-current flow (also called electrical flow) in this network as a function of the edges (written as vector) $\mathbf{f} \in \mathbb{R}^{|E^\pm|}_{\geq 0}$. Whenever possible, we use $\mathbf{f}[u, v]$ as shorthand notation for $\mathbf{f}[\{u, v\}]$ or $\mathbf{f}[(u, v)]$. Note that $\mathbf{f}[e] = -\mathbf{f}[-e]$ for $e \notin E^\pm$. This sign change in the s-t current when the flow direction is changed, is required to adhere to Kirchhoff's current law on flow conservation:

$$\sum_{w \in \delta^+(v)} \mathbf{f}[v, w] - \sum_{u \in \delta^-(v)} \mathbf{f}[u, v] = \begin{cases} 1 & \text{if } v = s \\ -1 & \text{if } v = t \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $\delta^+(v)$ [$\delta^-(v)$] is the set of edges having v as head [tail] in the orientation we choose in $E^\pm$. Such a flow also adheres to Kirchhoff's voltage law (sum in cycle is zero when considering flow directions) and Ohm's law (potential difference = resistance $\cdot$ current), cf. [14, 43]. The effective resistance between two vertices u and v, $\mathbf{r}(u, v)$, is defined as the potential difference between u and v when a unit current is injected into G at u and extracted at v, comp. [14, Ch. IX]. To compute $\mathbf{r}(u, v)$, let $\mathbf{e}_z$ be the canonical unit vector for vertex z, i. e., $\mathbf{e}_z(z) = 1$ and $\mathbf{e}_v = 0$ for all vertices $v \neq z$. Then,

$$\mathbf{r}(u, v) = (\mathbf{e}_u - \mathbf{e}_v)^T \mathbf{L}^\dagger (\mathbf{e}_u - \mathbf{e}_v) = \mathbf{L}^\dagger[u, u] - 2\mathbf{L}^\dagger[u, v] + \mathbf{L}^\dagger[v, v] \tag{3}$$

or, equivalently, $\mathbf{r}(u,v) = \mathbf{x}[u] - \mathbf{x}[v]$, where $\mathbf{x}$ is the solution vector of the Laplacian linear system $\mathbf{L}\mathbf{x} = \mathbf{e}_u - \mathbf{e}_v$. The Laplacian pseudoinverse, $\mathbf{L}^\dagger$, can be expressed as $\mathbf{L}^\dagger = (\mathbf{L} + \frac{1}{n}\mathbf{J})^{-1} - \frac{1}{n}\mathbf{J}$ [60], where $\mathbf{J}$ is the $n \times n$-matrix with all entries being 1.

Also note that the effective resistance between the endpoints of an edge $e \in E$ equals the probability that e is an edge in a uniform spanning tree (UST), i. e., a spanning tree selected uniformly at random among all spanning trees of G, cf. [14, Ch. II].

Electrical Closeness. The combinatorial counterpart of electrical closeness is based on shortest-path distances $\mathrm{dist}(u,v)$ for vertices u and v in G: $c^c(u) := (n-1)\,/\,f^c(u)$, where the denominator is the *combinatorial farness* of u:

$$f^c(u) := \sum_{v \in V \setminus \{u\}} \mathrm{dist}(u,v). \tag{4}$$

Electrical farness $f^{el}(\cdot)$ is defined analogously to combinatorial farness – shortest-path distances in Eq. (4) are replaced by effective resistances $\mathbf{r}(u,v)$. Closeness centrality (both combinatorial and electrical) are not defined for disconnected graphs due to infinite distances. We can get around this, however: a combinatorial generalization for closeness called Lin's index (cf. [10]) can be adapted to the electrical case, too. Thus, our assumption of G being connected is no limitation.

Normalized Random-Walk Betweenness. Classical betweenness, based on shortest paths, is one of the most popular centrality measures. The betweenness of vertices using random-walk routing instead of shortest paths is given by the normalized random-walk betweenness (NRWB) [47]. This measure counts each random walk passing through a vertex only once. By mapping the random walk problem to current flowing in a network, Ref. [47] obtains a closed-form expression of NRWB and provides an analysis of its scaling bahavior as a function of n. Then, the NRWB $c_b(\cdot)$ of a vertex v is:

$$c_b(v) = \frac{1}{n} + \frac{1}{n-1} \sum_{t \neq v} \frac{\mathbf{M^{-1}}[t,t] - \mathbf{M^{-1}}[t,v]}{\mathbf{M^{-1}}[t,t] + \mathbf{M^{-1}}[v,v] - 2\mathbf{M^{-1}}[t,v]}, \tag{5}$$

where $\mathbf{M} := \mathbf{L} + \mathbf{P}$, with $\mathbf{P}$ the projection operator onto the zero eigenvector of the Laplacian $\mathbf{L}$ such that $\mathbf{P}[i,j] = 1/n$. We show in Section 3.4 how to simplify this expression.

We also consider the Kirchhoff index and related centrality measures. Their description can be found in Appendix B.1.

2.2 Related Work

Solving Laplacian Systems. A straightforward approach to compute electrical closeness and related centralities is to compute $\mathbf{L}^\dagger$, by solving a number of Laplacian systems. Brandes and Fleischer [16] computed electrical closeness from the solution of n linear systems using conjugate gradient (CG) in $\mathcal{O}(mn\sqrt{\kappa})$ time, where κ is the condition number of the appropriately preconditioned Laplacian matrix.[1] Later, Spielman and Srivastava proposed an approximation algorithm for computing effective resistance distances [57]. The main ingredients of the algorithm are a dimension reduction with Johnson-Lindenstrauss [29] and the use of a fast Laplacian solver for $\mathcal{O}(\log n/\epsilon^2)$ Laplacian systems. The algorithm approximates

[1] Brandes and Fleischer provide a rough estimate of κ as $\Theta(n)$, leading to a total time of $\mathcal{O}(mn^{1.5})$.

effective resistance values for all edges within a factor of $(1 \pm \epsilon)$ in $\mathcal{O}(I(n,m) \log n/\epsilon^2)$ time, where $I(n,m)$ is the running time of the Laplacian solver, assuming that the solution of the Laplacian systems is exact. With an approximate Laplacian solution, the algorithm yields a $(1+\epsilon)^2$-approximation. Significant progress in the development of fast Laplacian solvers with theoretical guarantees [19, 30, 32, 33, 37] has resulted in the currently best one running in $\mathcal{O}(m \log^{1/2} n \log(1/\epsilon))$ time (up to polylogarithmic factors) [19]. Parallel algorithms for solving linear systems on the more general SDD matrices also exist in the literature [12, 51]. To date, the fastest algorithms for electrical closeness and spanning edge centrality extend the idea of Spielman and Srivastava [11, 27, 45]. (Similar ideas are used for centrality measures based on the Kirchhoff index, see Appendix B.2). Since theoretical Laplacian solvers rely on heavy graph-theoretic machinery such as low-stretch spanning trees, one rather uses multigrid solvers [11, 34, 40] in practice instead.

Diagonal Estimation. Recall from the introduction that the diagonal (or in case of the Kirchhoff index even only its sum, the trace – see Appendix B.1) of $\mathbf{L}^\dagger$ is enough to compute electrical closeness and related measures, also see Eq. (6) below. Algorithms that approximate the diagonal (or the trace) of matrices that are only implicitly available often use iterative methods [56], sparse direct methods [3, 28], Monte Carlo [22] or deterministic probing techniques [9, 59]. A popular approach is the standard Monte-Carlo method for the trace of $\mathbf{A}$, due to Hutchinson [22]. The idea is to estimate the trace of $\mathbf{A}$ by observing the action of $\mathbf{A}$ (in terms of matrix-vector products) on a sufficiently large sample of random vectors r_k. In our case, this would require to solve a large number of Laplacian linear systems with vectors r_k as right-hand sides. Avron and Toledo [6] proved that the method takes $\mathcal{O}(\log n/\epsilon^2)$ samples to achieve a maximum error ϵ with probability at least $1-\delta$. The approach from Hutchinson [22] has been extended by Bekas et al. [9] for estimating the diagonal of $\mathbf{A}$. Finally, Barthelmé et al. [8] have recently proposed a combinatorial algorithm to approximate the trace (not the diagonal) of the inverse of a matrix closely related to the Laplacian. Their algorithm can be seen as a special case of our algorithm in the situation where a universal vertex exists.

Normalized Random Walk Betweenness. Along with the introduction of the measure, Ref. [47] provided numerical evaluations of Eq. (5) on various graph models. However, no algorithm to compute the measure without (pseudo)inverting $\mathbf{L}$ has been proposed yet.

3 Approximation Algorithm

3.1 Overview

In order to compute the electrical closeness for all vertices in $V(G)$, the main challenge is obviously to compute their electrical farness, $f^{el}(\cdot)$. Recall from the introduction that the diagonal of $\mathbf{L}^\dagger$ is sufficient to compute $f^{el}(\cdot)$ for all vertices simultaneously (comp. Ref. [15, Eq. (15)] with a slightly different definition of electrical closeness). This follows from Eq. (3) and the fact that each row/column in $\mathbf{L}^\dagger$ sums to 0:

$$f^{el}(u) := \sum_{v \in V \setminus \{u\}} \mathbf{r}(u,v) = n \cdot \mathbf{L}^\dagger[u,u] + \mathrm{tr}(\mathbf{L}^\dagger) - 2 \sum_{v \in V} \mathbf{L}^\dagger[u,v] = n \cdot \mathbf{L}^\dagger[u,u] + \mathrm{tr}(\mathbf{L}^\dagger), \tag{6}$$

since the trace $\mathrm{tr}(\cdot)$ is the sum over the diagonal entries.

We are interested in an approximation of $\mathrm{diag}(\mathbf{L}^\dagger)$, since we do not necessarily need exact values for our particular applications. To this end, we propose an approximation algorithm for which we give a rough overview first. Our algorithm works best for small-world networks – thus, we focus on this important input class. Let G be unweighted for now; we discuss the extension to weighted graphs in Section 3.4.

1. Select[2] a pivot vertex $u \in V$ and solve the linear system $\mathbf{Lx} = \mathbf{e}_u - \frac{1}{n} \cdot \mathbf{1}$, where $\mathbf{1} = (1, \ldots, 1)^T$. The solution $\mathbf{x}$ is $\mathbf{L}^\dagger[:, u]$, the column of $\mathbf{L}^\dagger$ corresponding to u [60].
2. Throughout the rest of this paper we denote $V' := V \setminus \{u\}$. As a direct consequence from Eq. (3), the diagonal entries $\mathbf{L}^\dagger[v, v]$ for all $v \in V'$ can be computed as:

$$\mathbf{L}^\dagger[v, v] = \mathbf{r}(u, v) - \mathbf{L}^\dagger[u, u] + 2\mathbf{L}^\dagger[v, u]. \tag{7}$$

3. It remains to approximate these $n - 1$ effective resistance values $\mathbf{r}(\cdot, \cdot)$. In order to do so, we employ Kirchhoff's theorem, which connects electrical flows with spanning trees [14, Ch. II]. To this end, let N be the total number of spanning trees of G; moreover, let $N_{s,t}(a, b)$ be the number of spanning trees in which the unique path from s to t traverses the edge $\{a, b\}$ *in the direction from a to b.*

   ▶ **Theorem 1** (Kirchhoff, comp. [14]). *Let $\mathbf{f}[a, b] := (N_{s,t}(a, b) - N_{s,t}(b, a))/N$. Distribute current flows on the edges of G by sending a current of size $\mathbf{f}[a, b]$ from a to b for every edge $\{a, b\}$. Then there is a total current of size 1 from s to t satisfying Kirchhoff's laws.*

   As a result of Theorem 1, the effective resistance between s and t is the potential difference between s and t induced by the current-flow given by $\mathbf{f}$. Vice versa, since the current flow is induced by potential differences (Ohm's law), one simply has to add the currents on a path from s to t to compute $\mathbf{r}(s, t)$ (see Eq. (8) in Section 3.2). Actually, as a proxy for the current flows, we use the (approximate) $N(\cdot)$-values mentioned in Theorem 1.
4. It is impractical to compute exact values for N (e. g., by Kirchhoff's matrix-tree theorem [24], which would require the determinant or all eigenvalues of $\mathbf{L}^\dagger$) or $N(\cdot)$ for large graphs. Instead, we obtain approximations of the desired values via sampling: we sample a set of uniform spanning trees and determine the $N(\cdot)$-values by aggregation over all sampled trees. This approach provides a probabilistic absolute approximation guarantee.

The pseudocode of the algorithm, in adjusted order, is shown and discussed as Algorithm 1 in Appendix A.1. Components and properties of Algorithm 1 are explained in the remainder of Section 3; reading Section 3.2 while/before studying the pseudocode is recommended.

Note that Steps 2-4 of the algorithm are entirely combinatorial. Step 1 may or may not be combinatorial, depending on the Laplacian solver used. Corresponding implementation choices are discussed in Section 4.

3.2 Effective Resistance Approximation by UST Sampling

Extending and generalizing work by Hayashi et al. [27] on spanning edge centrality, our main idea is to compute a sufficiently large sample of USTs and to aggregate the $N(\cdot)$-values of the edges in these USTs. Given G and an electrical flow with source u and sink v, recall that the effective resistance between them is the potential difference $\mathbf{x}[u] - \mathbf{x}[v]$, where $\mathbf{x}$ is the solution vector in $\mathbf{Lx} = \mathbf{e}_u - \mathbf{e}_v$. Since $\mathbf{x}$ is a potential and the electrical flow $\mathbf{f}$ results from its difference, $\mathbf{r}(u, v)$ can be computed given *any* path $\langle u = v_0, v_1, \ldots, v_{k-1}, v_k = v \rangle$ as:

[2] As we will see later on, one can improve the empirical running time when u is not arbitrary, but chosen so as to have low eccentricity, i. e., the length of its longest shortest path. The correctness and the asymptotic time complexity of the algorithm are not affected by the selection, though.

$$\begin{aligned}\mathbf{r}(u,v) &= \sum_{i=0}^{k-1} \mathbf{f}[v_i, v_{i+1}] \\ &= 1/N \sum_{i=0}^{k-1} \left(N_{u,v}(v_i, v_{i+1}) - N_{u,v}(v_{i+1}, v_i)\right). \end{aligned} \tag{8}$$

Recall that the sign of the current flow changes if we traverse an edge against the flow direction. This is reflected by the second summand in the sum of Eq. (8). Since we can choose any path from u to v, for efficiency reasons we use *one shortest* path $P(v)$ per vertex $v \in V'$. We compute these paths with one breadth-first search (BFS) with root u, resulting in a tree B_u whose edges are considered as implicitly directed from the root to the leaves. For each vertex $v \in V'$, we maintain an estimate $R[v]$ of $\mathbf{r}(u,v)$, which is initially set to 0 for all v. After all USTs have been processed, we divide all entries of R by τ, the number of sampled trees, i. e., τ takes the role of N in Eq. (8).

Sampling USTs. In total, we sample τ USTs, where τ depends on the desired approximation guarantee and is determined later. The choice of the UST algorithm depends on the input: for general graphs, the algorithm by Schild [53] with time complexity $\mathcal{O}(m^{1+o(1)})$ is the fastest. Among others, it uses a sophisticated shortcutting technique using fast Laplacian solvers to speed up the classical Aldous-Broder [1,17] algorithm. For unweighted small-world graphs, however, Wilson's simple algorithm using loop-erased random walks is in $\mathcal{O}(m \log n)$, as outlined in Appendix A.2. Thus, for our class of inputs, Wilson's algorithm is preferred.

Data structures. When computing the contribution of a UST T to $N(\cdot)$, we need to update for each edge $e = (a,b) \in E(T)$ its contribution to $N_{u,v}(a,b)$ and $N_{u,v}(b,a)$, respectively – for exactly every vertex v for which (a,b) [or (b,a)] lies on $P(v)$. Hence, the algorithm that aggregates the contribution of UST T to R will need to traverse $P(v)$ for each vertex $v \in V$. To this end, we represent the BFS tree B_u as an array of parent pointers for each vertex $v \in V$. On the other hand, the tree T can conveniently be represented by storing a child and a sibling for each vertex $v \in V$. Compared to other representations (such as adjacency lists), this data structure can be constructed and traversed with low constant overhead.

Tree Aggregation. After constructing a UST T, we process it to update the intermediate effective resistance values $R[\cdot]$. Note that we can discard T afterwards and do not have to store the full sample, which has a positive effect on the memory footprint of our algorithm. The aggregation algorithm is shown as Algorithm 2 in Appendix A.1. Recall that we need to determine for each vertex v and each edge $(a,b) \in P(v)$, whether (a,b) or (b,a) occurs on the unique u-v path in T. To simplify this test, we root T at u; hence, it is enough to check if (a,b) [or (b,a)] appears above v in T. For general graphs, the test still incurs quadratic overhead in running time (in particular, the number of vertex-edge pairs that need to be considered is $f^c(u) = \sum_{v \in V'} |P(v)| = \mathcal{O}(n^2)$). We remark that, perhaps surprisingly, a bottom-up traversal of T does not improve on this, either; it is similarly difficult to determine all $R[v]$ that a given $(a,b) \in E(T)$ contributes to (those v form an arbitrary subset of descendants of b in T). However, we can exploit the fact that on small-world networks, the depth of B_u can be controlled, i. e., $f^c(u)$ is sub-quadratic. To accelerate the test, we first compute a DFS data structure for T, i. e., we determine discovery and finish timestamps for all vertices in V, respectively. For an arbitrary $v \in V$ and $(a,b) \in V \times V$, this data structure allows us

to answer in constant time (i) whether either (a, b) or (b, a) is in T and (ii) if $(a, b) \in E(T)$, whether v appears below (a, b) in T. Finally, we loop over all $v \in V$ and all $e = (a, b) \in P(v)$ and aggregate the contribution of T to $N_{u,v}(a, b)$. To do so, we add [subtract] 1 to [from] $R[v]$ if e has the same [opposite] direction in B_u. If e is not in B_u, $R[v]$ does not change.

3.3 Algorithm Analysis

The choice of the pivot u has an effect on the time complexity of our algorithm. The intuitive reason is that the BFS tree B_u should be shallow in order to have short paths to the root u, which is achieved by a u with small eccentricity. The proofs of this subsection can be found in Appendices A.3 and A.4. Regarding aggregation, we obtain:

▶ **Lemma 2.** *Tree aggregation (Algorithm 2 in Appendix A.1) has time complexity $\mathcal{O}(f^c(u))$, which can be bounded by $\mathcal{O}(n \cdot \mathrm{ecc}(u)) = \mathcal{O}(n \cdot \mathrm{diam}(G))$.*

In high-diameter networks, the farness of u can become quadratic (consider a path graph) and thus problematic for large inputs. In small-world graphs, however, we obtain $\mathcal{O}(n \log n)$ per aggregation. We continue the analysis with the main algorithmic result.

▶ **Theorem 3.** *Let G be an undirected and unweighted graph with n vertices, m edges, diameter $\mathrm{diam}(G)$ and Laplacian $\mathbf{L} = \mathbf{L}(G)$. Then, our diagonal approximation algorithm (Algorithm 1 in Appendix A.1) computes an approximation of $\mathrm{diag}(\mathbf{L}^\dagger)$ with absolute error $\pm\epsilon$ with probability $1 - \delta$ in time $\mathcal{O}(m \cdot \mathrm{ecc}^3(u) \cdot \epsilon^{-2} \cdot \log(m/\delta))$. For small-world graphs and with $\delta := 1/n$ to get high probability, this yields a time complexity of $\mathcal{O}(m \log^4 n \cdot \epsilon^{-2})$.*

Thus, for small-world networks, we have an approximation algorithm whose running time is nearly-linear in m (i. e., linear up to a polylogarithmic factor), quadratic in $1/\epsilon$, and logarithmic in $1/\delta$. By choosing a "good" pivot u, it is often possible to improve the running time of Algorithm 1 by a constant factor (i. e., without affecting the $\mathcal{O}$-notation). In particular, there are vertices u with $\mathrm{ecc}(u)$ as low as $\frac{1}{2}\mathrm{diam}(G)$. A discussion on the algorithm's parallelization in the work-depth model can be found in Appendix A.1.2.

▶ Remark 4. If G has constant diameter, Algorithm 1 has time complexity $\mathcal{O}(m \log n \cdot \epsilon^{-2})$ to obtain an absolute ϵ-approximation guarantee. This is faster than the best JLT-based approximation (which provides a relative guarantee instead).

3.4 Generalizations

In this section we show how our algorithm can be adapted to work for weighted graphs and for normalized random-walk betweenness as well. The extensions to Kirchhoff-related indices are presented in Appendix B.3.

Extension to Weighted graphs. For an extension to weighted graphs, we need a weighted version of Kirchhoff's theorem. To this end, the weight of a spanning tree T is defined as the product of the weights (= conductances) of its edges. Then, let N^* be the sum of the weights of all spanning trees of G; also, let $N^*_{s,t}(a, b)$ be the sum of the weights of all spanning trees in which the unique path from s to t traverses the edge $\{a, b\}$ in the direction from a to b.

▶ **Theorem 5** (comp. [14], p. 46)**.** *There is a distribution of currents satisfying Ohm's law and Kirchhoff's laws in which a current of size 1 enters at s and leaves at t. The value of the current on edge $\{a, b\}$ is given by $(N^*_{s,t}(a, b) - N^*_{s,t}(b, a))/N^*$.*

Consequently, our sampling approach needs to estimate N^* as well as the $N^*(\cdot)$-values. It turns out that no major changes are necessary. Wilson's algorithm also yields a UST for weighted graphs (if its random walk takes edge weights for transition probabilities into account) [62]. Yet, the running time bound for Wilson needs to mention the graph volume, $\mathrm{vol}(G)$, explicitly now: $\mathcal{O}(\mathrm{ecc}(u) \cdot \mathrm{vol}(G))$. The weight of each sampled spanning tree can be accumulated during each run of Wilson. It has to be integrated into Algorithm 2 by adding [subtracting] the tree weight in Line 9 [Line 12] instead of 1. For the division at the end, one has to replace τ by the total weight of the sampled trees. Finally, the tree B_u remains a BFS tree. The eccentricity and farness of u then still refer in the analysis to their unweighted versions, respectively, as far as B_u is concerned.

To conclude, the only important change regarding bounds happens in Theorem 3. In the time complexity, m is replaced by $\mathrm{vol}(G)$.

Normalized Random-Walk Betweenness. Ref. [47] proposes normalized random-walk betweenness as a measure for the influence of a vertex in the network, but the paper does not provide an algorithm (beyond implicit (pseudo)inversion). We propose to compute normalized random-walk betweenness with Algorithm 1 and derive (proof in Appendix A.5):

▶ **Lemma 6.** *Normalized random-walk betweenness $c_b(v)$ (Eq. (5)) can be rewritten as:*

$$c_b(v) = \frac{1}{n} + \frac{\mathrm{tr}(\mathbf{L}^\dagger)}{(n-1)f^{el}(v)}. \tag{9}$$

Hence, since Algorithm 1 approximates the diagonal of $\mathbf{L}^\dagger$ and both trace and electrical farness depend only on the diagonal, the following proposition holds:

▶ **Proposition 7.** *Let $G = (V, E)$ be a small-world graph as in Theorem 3. Then, Algorithm 1 approximates with high probability $c_b(v)$ for all $v \in V$ with absolute error $\pm\epsilon$ in $\mathcal{O}(m \log^4 n \cdot \epsilon^{-2})$ time.*

4 Engineering Aspects and Parallelization

Important engineering decisions concern the choice of the UST sampling algorithm, decomposition of the input graph into biconnected components, selection of the pivot u, and the linear solver used for the initial linear system. For these aspects, we refer the reader to Appendix C.1.

Our implementation uses OpenMP for shared memory and MPI+OpenMP for distributed memory. We assume that the entire graph fits into main memory (even in the distributed case). Hence, we can parallelize Algorithm 1 to a large extent by sampling and aggregating multiple USTs in parallel. In particular, we turn the main sampling loop into a **parallel for** loop. We also solve the initial Laplacian system using a shared-memory parallel Conjugate Gradient (CG) solver (see Appendix C.1). Note that we do not employ parallelism in the other steps of the algorithm. In particular, the BFS to compute B_u is executed sequentially. We also do not parallelize over the loops in Algorithm 2 to avoid nested parallelism with multiple invocations of Algorithm 2. We note that, in contrast to the theoretical work-depth model, solving the initial Laplacian system and performing the BFS are not the main bottlenecks in practice. Instead, sampling and aggregating USTs together consume the majority of CPU time (see Figure 5 in Appendix E.1 of the full version [4]). More details regarding shared and distributed memory, in particular load balancing for the distributed case, are discussed in Appendix C.2.

5 Experiments

5.1 Settings

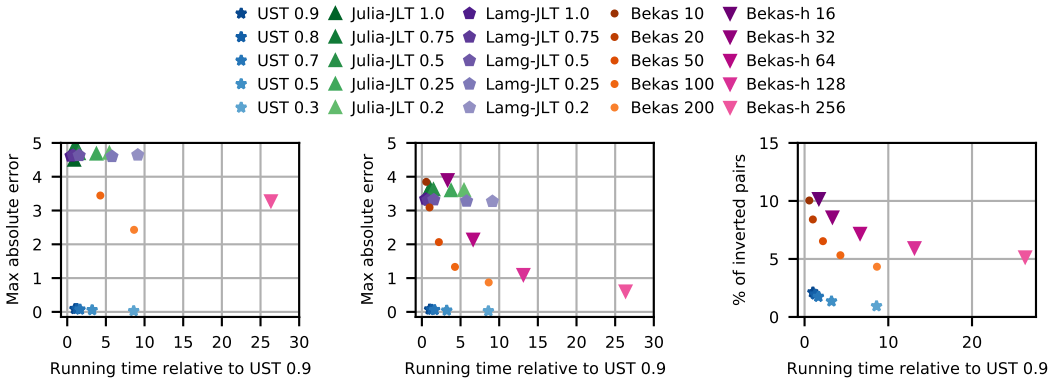
We conduct experiments to demonstrate the performance of our approach compared to the state-of-the-art competitors. Unless stated otherwise, we implemented all algorithms in C++, using the NetworKit [58] graph APIs. Our own algorithm is labelled UST in the sections below. All experiments were conducted on a cluster with 16 Linux machines, each equipped with an Intel Xeon X7460 CPU (2 sockets, 12 cores each), and 192 GB of RAM. To ensure reproducibility, all experiments were managed by the SimexPal [5] software. We executed our experiments on the graphs in Tables 2, 3, 4, and 5 (see Appendix D.2 of the full version of this paper [4] for further details on input graphs). All of them are unweighted and undirected. They have been downloaded from the public KONECT [35] repository and reduced to their largest connected component.

Quality Measures and Baseline. To evaluate the diagonal approximation quality, we measure the maximum absolute error ($\max_i \mathbf{L}^\dagger_{ii} - \widetilde{\mathbf{L}^\dagger_{ii}}$) on each instance, and we take both the maximum and the arithmetic mean over all the instances. Since for some applications [48, 49] a correct ranking of the entries is more relevant than their scores, in our experimental analysis we compare complete rankings of the elements of $\widetilde{\mathbf{L}^\dagger}$. Note that the lowest entries of $\mathbf{L}^\dagger$ (corresponding to the vertices with highest electrical closeness) are distributed on a significantly narrow interval. Hence, to achieve an accurate electrical closeness ranking of the top k vertices, one would need to solve the problem with very high accuracy. For this reason, all approximation algorithms we consider do not yield a precise top-k ranking, so that we (mostly) consider the complete ranking.

Using `pinv` in NumPy or Matlab as a baseline would be too expensive in terms of time (cubic) and space (quadratic) on large graphs (see Appendix D.2 of the full version [4]). Thus, as quality baseline we employ the LAMG solver [40] (see also next paragraph) as implemented within NetworKit [11] in our experiments (with 10^{-9} tolerance). The results in Table 6 in Appendix E.4 indicate that the diagonal obtained this way is sufficiently accurate.

Competitors in Practice. In practice, the fastest way to compute electrical closeness so far is to combine a dimension reduction via the Johnson-Lindenstrauss lemma [29] (JLT) with a numerical solver. In this context, Algebraic MultiGrid (AMG) solvers exhibit better empirical running time than fast Laplacian solvers with a worst-case guarantee [45]. For our experiments we use JLT combined with LAMG [40] (named Lamg-jlt); the latter is an AMG-type solver for complex networks. We also compare against a Julia implementation of JLT together with the fast Laplacian solver proposed by Kyng et al. [36], for which a Julia implementation is already available in the package Laplacians.jl.[3] This solver generates a sparse approximate Cholesky decomposition for Laplacian matrices with provable approximation guarantees in $\mathcal{O}(m \log^3 n \log(1/\epsilon))$ time; it is based purely on random sampling (and does not make use of graph-theoretic concepts such as low-stretch spanning trees or sparsifiers). We refer to the above implementation as Julia-jlt throughout the experiments. For both Lamg-jlt and Julia-jlt, we try different input error bounds (they correspond to the respective numbers next to the method names in Figure 1). This is a relative error, since these algorithms use numerical approaches with a relative error guarantee, instead of an absolute one (see Appendix E of the full version [4] for results in terms of different quality measures).

[3] `https://github.com/danspielman/Laplacians.jl`



(a) Maximum of the maximum absolute errors.

(b) Arithmetic mean of the maximum absolute error.

(c) Geometric mean of the percentage of inverted pairs in the full ranking of $\mathrm{diag}(\mathbf{L}^{\dagger})$.

Figure 1 Quality results over the instances of Table 2 (full version [4]). All runs are sequential.

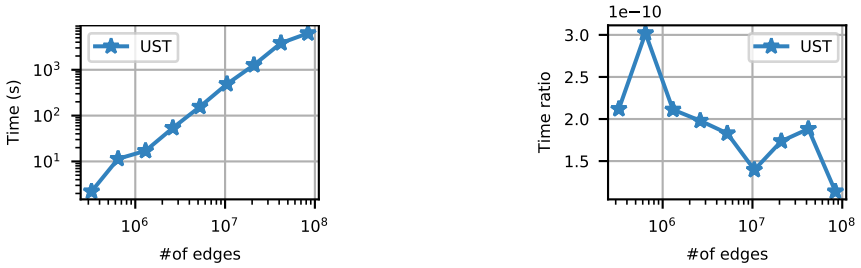
Finally, we compare against the diagonal estimators due to Bekas et al. [9], one based on random vectors and one based on Hadamard rows. To solve the resulting Laplacian systems, we use LAMG in both cases. In our experiments, the algorithms are referred to as Bekas and Bekas-h, respectively. Excluded competitors are discussed in Appendix D.1 (full version [4]).

5.2 Running Time and Quality

Figure 1a shows that, in terms of maximum absolute error, every configuration of UST achieves results with higher quality than the competitors. Even when setting $\epsilon = 0.9$, UST yields a maximum absolute error of 0.09, and it is 8.3× faster than Bekas with 200 random vectors, which instead achieves a maximum absolute error of 2.43. Furthermore, the running time of UST does not increase substantially for lower values of ϵ, and its quality does not deteriorate quickly for higher values of ϵ. Regarding the average of the maximum absolute error, Figure 1b shows that, among the competitors, Bekas-h with 256 Hadamard rows achieves the best precision. However, UST yields an average error of 0.07 while also being 25.4× faster than Bekas-h, which yields an average error of 0.62. Note also that the number next to each method in Figure 1 corresponds to different values of absolute (for UST) or relative (for Lamg-jlt, Julia-jlt) error bounds, and different numbers of samples (for Bekas, Bekas-h). For Bekas-h the number of samples needs to be a multiple of four due to the dimension of Hadamard matrices.

In Figure 1c we report the percentage of inverted pairs in the full ranking of $\widetilde{\mathbf{L}^{\dagger}}$. Note that JLT-based approaches are not depicted in this plot, because they yield $> 15\%$ of rank inversions. Among the competitors, Bekas achieves the best time-accuracy trade-off. However, when using 200 random vectors, it yields 4.3% inversions while also being 8.3× slower than UST with $\epsilon = 0.9$, which yields 2.1% inversions only.

For validation purposes, we also measure how well the considered algorithms compute the *set* (not the ranking) of top-k vertices, i. e., those with highest electrical closeness centrality, with $k \in \{10, 100\}$. For each algorithm we only consider the parameter settings that yields the highest accuracy. JLT-based approaches appear to be very accurate for this purpose, as their top-k sets achieve a Jaccard index of 1.0. As expected (due to its absolute error guarantee), UST performs slightly worse: on average, it obtains 0.95 for $k = 10$ and 0.98 for $k = 100$, which still shows a high overlap with the ground truth.



(a) Running time of UST w. r. t. #of edges.

(b) Ratio of running time of UST w. r. t. its theoretical running time (see Theorem 3).

Figure 2 Scalability of UST on random hyperbolic graphs ($\epsilon = 0.3$, 1×24 cores).

The memory consumption is shown and discussed in more detail in Appendix E.3 (full version [4]). In summary, as our algorithm can discard each UST after its aggregation, it is rather space-efficient and requires less memory than the competitors.

5.3 Parallel Scalability

The log-log plot in Figure 4a (Appendix E.1, full version [4]) shows that on shared-memory UST achieves a moderate parallel scalability w. r. t. the number of cores; on 24 cores in particular it is 11.9× faster than on a single core. Even though the number of USTs to be sampled can be evenly divided among the available cores, we do not see a nearly-linear scalability: on multiple cores the memory performance of our NUMA system becomes a bottleneck. Therefore, the time to sample a UST increases and using more cores yields diminishing returns. Limited memory bandwidth is a known issue affecting algorithms based on graph traversals in general [7, 42]. Finally, we compare the parallel performance of UST indirectly with the parallel performance of our competitors. More precisely, assuming a perfect parallel scalability for our competitors Bekas and Bekas-h on 24 cores, UST would yield results 4.1 and 12.6 times faster, respectively, even with this strong assumption for the competition's benefit.

UST scales better in a distributed setting. In this case, the scalability is affected mainly by its non-parallel parts and by synchronization latencies. The log-log plot in Figure 4b shows that on up to 4 compute nodes the scalability is almost linear, while on 16 compute nodes UST achieves a 15.1× speedup w. r. t. a single compute node.

Figure 5 (Appendix E.1, full version [4]) shows the fraction of time that UST spends on different tasks depending on the number of cores. We aggregated over "Sequential Init." the time spent on memory allocation, pivot selection, solving the linear system, the computation of the biconnected components, and on computing the tree B_u. In all configurations, UST spends the majority of the time in sampling, computing the DFS data structures and aggregating USTs. The total time spent on aggregation corresponds to "UST aggregation" and "DFS" in Figure 5, indicating that computing the DFS data structures is the most expensive part of the aggregation. Together, sampling time and total aggregation time account for 99.4% and 95.3% of the total running time on 1 core and 24 cores, respectively. On average, sampling takes 66.8% of this time, while total aggregation takes 31.2%. Since sampling a UST is on average 2.2× more expensive than computing the DFS timestamps and aggregation, faster sampling techniques would significantly improve the performance of our algorithm.

Table 1 Running time of UST on large real-world networks (16×24 cores).

Network	$\|V\|$	$\|E\|$	Time (s) $\epsilon = 0.3$	Time (s) $\epsilon = 0.9$
petster-carnivore	601,213	15,661,775	16.8	4.8
soc-pokec-relationships	1,632,803	22,301,964	55.5	9.5
soc-LiveJournal1	4,843,953	42,845,684	277.0	75.5
livejournal-links	5,189,808	48,687,945	458.4	80.6
orkut-links	3,072,441	117,184,899	71.8	19.9
wikipedia_link_en	13,591,759	334,590,793	429.9	88.3

5.4 Scalability to Large Networks

Results on Synthetic Networks. The log-log plots in Figure 2a show the average running time of UST (1×24 cores) on networks generated with the random hyperbolic generator from von Looz et al. [61].[4] For each network size, we take the arithmetic mean of the running times measured for five different randomly generated networks. Our algorithm requires 184 minutes for the largest inputs (with up to 83.9 million edges). Interestingly, Figure 2b shows that the algorithm scales slightly better than our theoretical bound predicts. In Figure 6 (Appendix E.2, full version [4]) we present results on an additional graph class, namely R-MAT graphs. On these instances, the algorithm exhibits a similar running time behavior; however, the comparison to the theoretical bound is less conclusive.

Results on Large Real-World Networks. In Table 1 we report the performance of UST in a distributed setting (16×24 cores) on large *real-world* networks. With $\epsilon = 0.3$ and $\epsilon = 0.9$, UST always runs in less than 8 minutes and 1.5 minutes, respectively.

6 Conclusions

We have proposed a new parallel algorithm for approximating $\mathrm{diag}(\mathbf{L}^\dagger)$ of Laplacian matrices $\mathbf{L}$ corresponding to small-world networks. Compared to the main competitors, our algorithm is about one order of magnitude faster, it yields results with higher quality in terms of absolute error and ranking of $\mathrm{diag}(\mathbf{L}^\dagger)$, and it requires less memory. The gap between the theoretical bounds and the much better empirical error yielded by our algorithm suggests that tighter bounds on the number of samples are a promising direction for future work. So is an improvement of the running time for high-diameter graphs, both in theory and practice.

References

1 David J. Aldous. The random walk construction of uniform spanning trees and uniform labelled trees. *SIAM J. Discret. Math.*, 3(4):450–465, November 1990. doi:10.1137/0403039.

2 Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. Graph clustering using effective resistance. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 41:1–41:16, Cambridge, Massachusetts, USA, 2018. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2018.41.

[4] The random hyperbolic generator generates networks with a heavy-tailed degree distribution. We set the average degree to 20 and the exponent of the power-law distribution to 3.

3 Patrick Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and François-Henry Rouet. Parallel computation of entries of a^{-1}. *SIAM J. Scientific Computing*, 37(2):C268–C284, 2015. doi:10.1137/120902616.

4 Eugenio Angriman, Maria Predari, Alexander van der Grinten, and Henning Meyerhenke. Approximation of the diagonal of a laplacian's pseudoinverse for complex network analysis, 2020. arXiv:2006.13679.

5 Eugenio Angriman, Alexander van der Grinten, Moritz von Looz, Henning Meyerhenke, Martin Nöllenburg, Maria Predari, and Charilaos Tzovas. Guidelines for experimental algorithmics: A case study in network analysis. *Algorithms*, 12(7):127, 2019.

6 Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM*, 58(2):8:1–8:34, 2011. doi:10.1145/1944345.1944349.

7 David A Bader, Guojing Cong, and John Feo. On the architectural requirements for efficient execution of graph algorithms. In *2005 International Conference on Parallel Processing (ICPP'05)*, pages 547–556, Oslo, Norway, 2005. IEEE, IEEE.

8 Simon Barthelmé, Nicolas Tremblay, Alexandre Gaudillière, Luca Avena, and Pierre-Olivier Amblard. Estimating the inverse trace using random forests on graphs, 2019. arXiv:1905.02086.

9 C. Bekas, E. Kokiopoulou, and Y. Saad. An estimator for the diagonal of a matrix. *Appl. Numer. Math.*, 57(11-12):1214–1229, November 2007. doi:10.1016/j.apnum.2007.01.003.

10 Elisabetta Bergamini, Michele Borassi, Pierluigi Crescenzi, Andrea Marino, and Henning Meyerhenke. Computing top-k closeness centrality faster in unweighted graphs. *TKDD*, 13(5):53:1–53:40, 2019. doi:10.1145/3344719.

11 Elisabetta Bergamini, Michael Wegner, Dimitar Lukarski, and Henning Meyerhenke. Estimating current-flow closeness centrality with a multigrid laplacian solver. In *Proc. 7th SIAM Workshop on Combinatorial Scientific Computing, CSC 2016*, pages 1–12. SIAM, 2016. doi:10.1137/1.9781611974690.ch1.

12 Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. Near linear-work parallel sdd solvers, low-diameter decomposition, and low-stretch subgraphs, 2011. doi:10.1145/1989493.1989496.

13 Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014. doi:10.1080/15427951.2013.865686.

14 Béla Bollobás. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer, 2002. doi:10.1007/978-1-4612-0619-4.

15 Enrico Bozzo and Massimo Franceschet. Resistance distance, closeness, and betweenness. *Social Networks*, 35(3):460–469, 2013. doi:10.1016/j.socnet.2013.05.003.

16 Ulrik Brandes and Daniel Fleischer. Centrality measures based on current flow. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science, STACS 2005*, volume 3404 of *LNCS*, pages 533–544. Springer, 2005. doi:10.1007/978-3-540-31856-9_44.

17 A. Broder. Generating random spanning trees. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, page 442–447, USA, 1989. IEEE Computer Society. doi:10.1109/SFCS.1989.63516.

18 Ashok K Chandra, Prabhakar Raghavan, Walter L Ruzzo, Roman Smolensky, and Prasoon Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6(4):312–340, 1996.

19 Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving sdd linear systems in nearly mlog1/2n time. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 343–352, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591833.

20 Wendy Ellens, Flora Spieksma, P. Mieghem, A. Jamakovic, and Robert Kooij. Effective graph resistance. *Linear Algebra and its Applications*, 435:2491–2506, November 2011. doi:10.1016/j.laa.2011.02.024.

21 Josh Ericson, Pietro Poggi-Corradini, and Hainan Zhang. Effective resistance on graphs and the epidemic quasimetric. *Involve, a Journal of Mathematics*, 7(1):97–124, 2013.

22 Hutchinson M. F. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *J. Commun. Statist. Simula.*, 19(2):433–450, 1990.

23 Arpita Ghosh, Stephen Boyd, and Amin Saberi. Minimizing effective resistance of a graph. *SIAM Rev.*, 50(1):37–66, February 2008. doi:10.1137/050645452.

24 Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013.

25 Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins University Press, 1996.

26 Gaël Guennebaud, Benoît Jacob, et al. Eigen v3, 2010. URL: http://eigen.tuxfamily.org.

27 Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. Efficient algorithms for spanning tree centrality. In *IJCAI*, pages 3733–3739. IJCAI, 2016.

28 Mathias Jacquelin, Lin Lin, and Chao Yang. Pselinv – a distributed memory parallel algorithm for selected inversion: The non-symmetric case. *Parallel Computing*, 74:84–98, 2018. Parallel Matrix Algorithms and Applications (PMAA'16). doi:10.1016/j.parco.2017.11.009.

29 William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

30 Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 911–920. ACM, 2013.

31 Douglas Klein and Milan Randic. Resistance distance. *Journal of Mathematical Chemistry*, 12:81–95, December 1993. doi:10.1007/BF01164627.

32 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for SDD linear systems. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 590–598. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.85.

33 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD linear systems. *SIAM J. Comput.*, 43(1):337–354, 2014. doi:10.1137/110845914.

34 Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 115(12):1638–1646, 2011.

35 Jérôme Kunegis. KONECT: the koblenz network collection. In Leslie Carr, Alberto H. F. Laender, Bernadette Farias Lóscio, Irwin King, Marcus Fontoura, Denny Vrandecic, Lora Aroyo, José Palazzo M. de Oliveira, Fernanda Lima, and Erik Wilde, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume*, pages 1343–1350. International World Wide Web Conferences Steering Committee / ACM, 2013. doi:10.1145/2487788.2488173.

36 R. Kyng and S. Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582. IEEE, October 2016. doi:10.1109/FOCS.2016.68.

37 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 842–850, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2897518.2897640.

38 Huan Li, Richard Peng, Liren Shan, Yuhao Yi, and Zhongzhi Zhang. Current flow group closeness centrality for complex networks? In Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia, editors, *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 961–971. ACM, 2019. doi:10.1145/3308558.3313490.

39 Huan Li and Zhongzhi Zhang. Kirchhoff index as a measure of edge centrality in weighted networks: Nearly linear time algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2377–2396. SIAM, SIAM, 2018.

40 Oren E Livne and Achi Brandt. Lean algebraic multigrid (LAMG): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

41 L. Lovász. Random walks on graphs: A survey. In D. Miklós, V. T. Sós, and T. Szőnyi, editors, *Combinatorics, Paul Erdős is Eighty*, volume 2, pages 353–398. János Bolyai Mathematical Society, Budapest, 1996.

42 Andrew Lumsdaine, Douglas Gregor, Bruce Hendrickson, and Jonathan Berry. Challenges in parallel graph processing. *Parallel Processing Letters*, 17(01):5–20, 2007.

43 Russell Lyons and Yuval Peres. *Probability on Trees and Networks*. Cambridge University Press, USA, 1st edition, 2017.

44 Clémence Magnien, Matthieu Latapy, and Michel Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *Journal of Experimental Algorithmics (JEA)*, 13:1–10, 2009.

45 Charalampos Mavroforakis, Richard Garcia-Lebron, Ioannis Koutis, and Evimaria Terzi. Spanning Edge Centrality: Large-scale Computation and Applications. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015*, pages 732–742. ACM, 2015.

46 Brendan D McKay et al. *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University Tennessee, USA, 1981.

47 Onuttom Narayan and Iraj Saniee. Scaling of random walk betweenness in networks. In Luca Maria Aiello, Chantal Cherifi, Hocine Cherifi, Renaud Lambiotte, Pietro Lió, and Luis M. Rocha, editors, *Complex Networks and Their Applications VII*, pages 41–51, Cham, 2019. Springer International Publishing.

48 Mark Newman. *Networks (2nd Ed.)*. Oxford university press, 2018.

49 Kazuya Okamoto, Wei Chen, and Xiang-Yang Li. Ranking of closeness centrality for large-scale social networks. In *International workshop on frontiers in algorithmics*, pages 186–195. Springer, Springer, 2008.

50 Melissa E. O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, September 2014.

51 Richard Peng and Daniel A. Spielman. An efficient parallel solver for sdd linear systems, 2014. doi:10.1145/2591796.2591832.

52 Gyan Ranjan, Zhi-Li Zhang, and Daniel Boley. Incremental computation of pseudo-inverse of laplacian. In Zhao Zhang, Lidong Wu, Wen Xu, and Ding-Zhu Du, editors, *Combinatorial Optimization and Applications*, pages 729–749, Cham, 2014. Springer International Publishing.

53 Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 214–227, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188852.

54 John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, USA, 2004.

55 Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Ann. Math. Statist.*, 21(1):124–127, March 1950.

56 Roger B. Sidje and Yousef Saad. Rational approximation to the fermi-dirac function with applications in density functional theory. *Numerical Algorithms*, 56(3):455–479, March 2011. doi:10.1007/s11075-010-9397-6.

57 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. doi:10.1137/080734029.

58 Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: A tool suite for large-scale complex network analysis. *Network Science*, 4(4):508–530, 2016. doi:10.1017/nws.2016.20.

59 Jok M. Tang and Yousef Saad. A probing method for computing the diagonal of a matrix inverse. *Numerical Linear Algebra with Applications*, 19(3):485–501, 2012.

60 Piet Van Mieghem, Karel Devriendt, and H Cetinay. Pseudoinverse of the laplacian and best spreader node in a network. *Physical Review E*, 96(3):032311, 2017.

61 Moritz von Looz, Mustafa Safa Özdayi, Sören Laue, and Henning Meyerhenke. Generating massive complex networks with hyperbolic geometry faster in practice. In *2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, September 13-15, 2016*, pages 1–6. IEEE, 2016. doi:10.1109/HPEC.2016.7761644.

62 David Bruce Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 296–303, New York, NY, USA, 1996. ACM. doi:10.1145/237814.237880.

A Algorithmic Details and Omitted Proofs

A.1 Our Approximation Algorithm in More Detail

Overall algorithm. Algorithm 1 already receives the pivot vertex u as input. Lines 4 to 10 approximate the effective resistances. To do so, Lines 4 to 7 perform initializations: first, the estimate of the effective resistance is set to 0 for all vertices. Then the accuracy η of the linear solver is computed so as to ensure an absolute ϵ-approximation for the whole algorithm. The BFS tree B_u with root u realizes shortest paths between u and all other vertices. The sample size τ depends on the parameters ϵ and δ, among others.

The first `for`-loop does the actual sampling and aggregation (the latter with Algorithm 2). Afterwards, Lines 11 to 14 fill the u-th column and the diagonal of $\mathbf{L}^\dagger$ – to the desired accuracy. Apart from that, the algorithm's high-level ideas have been provided already in Section 3.1.

Algorithm 1 Approximation algorithm for $\mathrm{diag}(\mathbf{L}^\dagger)$.

1: **function** ApproxDiagLpinv(G, u, ϵ, δ)
2: **Input:** Undirected small-world graph $G = (V, E)$, pivot $u \in V$, error bound $\epsilon > 0$, probability $0 < \delta < 1$
3: **Output:** $\mathrm{diag}(\widetilde{\mathbf{L}^\dagger})$, i. e., an (ϵ, δ)-approximation of $\mathrm{diag}(\mathbf{L}^\dagger)$
4: $R[v] \leftarrow 0 \ \forall v \in V$ ▷ $\mathcal{O}(n)$
5: Pick constant $\kappa \in (0, 1)$ arbitrarily; $\eta \leftarrow \frac{\kappa\epsilon}{3\sqrt{mn \log n \operatorname{diam}(G)}}$
6: Compute BFS tree B_u of G with root u ▷ $\mathcal{O}(n + m)$
7: $\tau \leftarrow \mathrm{ecc}(u)^2 \cdot \lceil \log(2m/\delta)/(2(1-\kappa)^2\epsilon^2) \rceil$ ▷ $\mathcal{O}(1)$
8: **for** $i \leftarrow 1$ to τ **do** ▷ τ times
9: Sample UST T_i of G with root u ▷ $\mathcal{O}(m \log n)$
10: $R \leftarrow$ Aggregate(T_i, R, B_u) ▷ $\mathcal{O}(n \log n)$
11: Solve $\mathbf{L}\mathbf{x} = \mathbf{e}_u - \frac{1}{n} \cdot \mathbf{1}$ for $\mathbf{x}$ (accuracy: η) ▷ $\tilde{\mathcal{O}}(m \log^{1/2} n \log(1/\eta))$
12: $\widetilde{\mathbf{L}^\dagger}[u, u] \leftarrow \mathbf{x}[u]$ ▷ $\mathcal{O}(1)$
13: **for** $v \in V'$ **do** ▷ All iterations: $\mathcal{O}(n)$
14: $\widetilde{\mathbf{L}^\dagger}[v, v] \leftarrow R[v]/\tau - \mathbf{x}[u] + 2\mathbf{x}[v]$
15: **return** $\mathrm{diag}(\widetilde{\mathbf{L}^\dagger})$

▶ Remark 8. Due to the fact that Laplacian linear solvers provide a relative error guarantee (and not an absolute $\pm\epsilon$ guarantee), the (relative) accuracy η for the initial Laplacian linear system (Lines 5 and 11) depends in a non-trivial way on our guaranteed absolute error ϵ. For details, see the proofs in Appendix A.4.

We also remark that the value of the constant κ does not affect the asymptotic running time (nor the correctness) of the algorithm. However, it does affect the empirical running time by controlling which fraction of the error budget is invested into solving the initial linear system vs. UST sampling.

A.1.1 Aggregation algorithm

Algorithm 2 Aggregation of T's contribution to $R[\cdot]$.

```
1:  function AGGREGATE(T, R, B_u)
2:      Input: spanning tree T, array of effective resistance estimates R, shortest-path tree B_u
3:      Output: R updated with T's contribution
4:      {α, Ω} ← DFS(T)                          ▷ α(v), Ω(v): discovery/finish times of v
5:      for v ∈ V' do
6:          for (a, b) ∈ P(v) obtained from B_u do
7:              if parent(b) = a then
8:                  if α(b) < α(v) and Ω(v) < Ω(b) then
9:                      R[v] ← R[v] + 1
10:             else if parent(a) = b then
11:                 if α(a) < α(v) and Ω(v) < Ω(a) then
12:                     R[v] ← R[v] − 1
13:     return R
```

Algorithm 2 depicts the pseudocode of the tree aggregation algorithm that is discussed in Section 3.2. Here, $\alpha(\cdot)$ and $\Omega(\cdot)$ denote our DFS discovery and finish timestamps, respectively. The test whether (a, b) [or (b, a)] is in T is carried out in Line 7 [or Line 10, respectively]. If that is indeed the case, Line 8 [or Line 11] checks whether v is below (a, b) [or (b, a), respectively]. If that is the case, the effective resistance estimate is adapted in Line 9 [Line 12].

A.1.2 Parallelism

Algorithm 1 can be parallelized by sampling and aggregating USTs in parallel. This yields a work-efficient parallelization in the work-depth model. The depth of the algorithm is dominated by (i) computing the BFS tree B_u, (ii) sampling each UST (Line 9) and (iii) solving the Laplacian linear system (Line 11). With current algorithms, the latter two procedures have depth $\mathcal{O}(m \log n)$ and $\tilde{\mathcal{O}}(m \log^{\frac{1}{2}} n \log(1/\eta))$, respectively (simply by executing them sequentially). We note that parallelizing the loops of Algorithm 2 results in a depth of $\mathcal{O}(n)$ for Algorithm 2; however, this does not impact the depth of Algorithm 1. We also note that by using a parallel Laplacian solver, the depth of solving the initial linear system becomes polylogarithmic [51]. Nevertheless, real-world implementations show a good parallelization behavior by parallelizing only Algorithm 1 (see Sections 4 and 5); consequently, we do not focus on parallelizing the sampling itself.

A.2 Wilson's UST Algorithm

Given a path P, its loop erasure is a simple path created by removing all cycles of P in chronological order. Wilson's algorithm grows a sequence of sub-trees of G, in our case starting with u as root of T. Let $M = \{v_1, \ldots, v_{n-1}\}$ be an enumeration of $V \setminus \{u\}$. Following the order in M, a random walk starts from every unvisited v_i until it reaches (some vertex in) T and its loop erasure is added to T.

▶ **Proposition 9** ([62], comp. [27]). *For a connected and unweighted undirected graph $G = (V, E)$ and a vertex $u \in V$, Wilson's algorithm samples a uniform spanning tree of G with root u. The expected running time is the mean hitting time of G, $\sum_{v \in V'} \pi_G(v)\kappa_G(v, u)$, where $\pi_G(v)$ is the probability that a random walk stays at v in its stationary distribution and where $\kappa_G(v, u)$ is the commute time between v and u.*

▶ **Lemma 10.** *Let G be as in Proposition 9. Its mean hitting time can be rewritten as $\sum_{v \in V'} \deg(v) \cdot \mathbf{r}(u, v)$, which is $\mathcal{O}(\mathrm{ecc}(u) \cdot m)$. In small-world graphs, this is $\mathcal{O}(m \log n)$.*

Proof of Lemma 10. First, we replace $\pi_G(v)$ by $\frac{\deg(v)}{\mathrm{vol}(G)}$ in the sum [41]. By using the well-known relation $\kappa_G(v, u) = \mathrm{vol}(G) \cdot \mathbf{r}(v, u)$ [18], the volumes cancel and we obtain $\sum_{v \in V'} \deg(v) \cdot \mathbf{r}(u, v)$. We can bound this from above by $\sum_{v \in V'} \deg(v) \cdot \mathrm{dist}(u, v) \leq \mathrm{vol}(G) \cdot \mathrm{ecc}(u)$, because effective resistance is never larger than the graph distance [21]. In unweighted graphs, $\mathrm{vol}(G) = 2m$ and in undirected graphs $\mathrm{ecc}(u) \leq \mathrm{diam}(G)$ for all $u \in V$, which proves the claim. ◀

A.3 Proof of Lemma 2

Proof of Lemma 2. DFS in T takes $\mathcal{O}(n)$ time since T is a spanning tree. Furthermore, Algorithm 2 loops over $\mathcal{O}(f^c(u))$ vertex-edge pairs (as $|P(v)| = \mathrm{dist}(u, v)$), with $\mathcal{O}(1)$ query time spent per pair. Since no path to the root in B_u is longer than $\mathrm{ecc}(u)$, we obtain $\mathcal{O}(n \cdot \mathrm{ecc}(u))$, which is by definition $\mathcal{O}(n \cdot \mathrm{diam}(G))$. ◀

A.4 Proof of Theorem 3

The proof of our main theorem makes use of Hoeffding's inequality. In the inequality's presentation, we follow Hayashi et al. [27].

▶ **Lemma 11.** *Let $X_1, \ldots, X_\tau$ be independent random variables in $[0, 1]$ and $X = \sum_{i \in [\tau]} X_i$. Then for any $0 < \epsilon < 1$, we have*

$$Pr[|X - \mathbb{E}[X]| > \epsilon\tau] \leq 2\exp(-2\epsilon^2\tau). \tag{10}$$

Before we can prove Theorem 3, we need auxiliary results on the equivalence of norms. For this purpose, let $\|\mathbf{x}\|_{\mathbf{L}} := \sqrt{\mathbf{x}^T\mathbf{L}\mathbf{x}}$ for any $\mathbf{x} \in \mathbb{R}^n$. Note that $\|\cdot\|_{\mathbf{L}}$ is a norm on the subspace of $\mathbb{R}^n$ with $\mathbf{x} \perp \mathbf{1}$. We show that:

▶ **Lemma 12.** *Let $G = (V, E)$ be a connected undirected graph with n vertices and m edges. Moreover, let $\mathbf{L}$ be its Laplacian matrix and λ_2 the second smallest eigenvalue of $\mathbf{L}$. The volume of G, $\mathrm{vol}(G)$, is the sum of all (possibly weighted) vertex degrees.*

For any $\mathbf{x} \in \mathbb{R}^n$ with $\mathbf{x} \perp \mathbf{1}$ we have:

$$\sqrt{\lambda_2} \cdot \|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_{\mathbf{L}} \leq \sqrt{2\,\mathrm{vol}(G)} \cdot \|\mathbf{x}\|_\infty. \tag{11}$$

Proof. Since $\mathbf{L}$ is positive semidefinite, it can be seen as a Gram matrix and written as $\mathbf{K}^T\mathbf{K}$ for some real matrix $\mathbf{K}$. The second smallest eigenvalue of $\mathbf{K}$ is then $\sqrt{\lambda_2}$ and we can write:

$$\sqrt{\lambda_2} \cdot \|\mathbf{x}\|_\infty \leq \sqrt{\lambda}_2 \cdot \|\mathbf{x}\|_2 = \|\sqrt{\lambda}_2 \cdot \mathbf{x}\|_2 \leq \|\mathbf{K}\mathbf{x}\|_2 = \|\mathbf{x}\|_{\mathbf{L}}. \tag{12}$$

The first, second, and last (in)equality in Eq. (12) follow from basic linear algebra facts, respectively. The third inequality follows from the Courant-Fischer theorem, since the eigenvector corresponding to the smallest eigenvalue 0, $\mathbf{1}$, is excluded from the subspace of $\mathbf{x}$ (comp. for example Ch. 3.1 of Ref. [54].)

Using the quadratic form of the Laplacian matrix, we get:

$$\|\mathbf{x}\|_{\mathbf{L}} = \left(\sum_{\{i,j\}\in E} \mathbf{w}(u,v)(\mathbf{x}[i]-\mathbf{x}[j])^2 \right)^{1/2} \tag{13}$$

$$\leq \left(\frac{1}{2}\operatorname{vol}(G)\cdot(2\|\mathbf{x}\|_\infty)^2 \right)^{1/2} = \sqrt{2\operatorname{vol}(G)}\cdot\|\mathbf{x}\|_\infty \tag{14}$$

◀

We are now in the position to prove our main result:

Proof of Theorem 3. Solving the initial linear system with the solver by Cohen et al. [19] takes $\tilde{\mathcal{O}}(m\log^{1/2} n\cdot\log(1/\eta))$ time to achieve a relative error bound of $\|\tilde{\mathbf{x}}-\mathbf{x}\|_{\mathbf{L}} \leq \eta\|\mathbf{x}\|_{\mathbf{L}}$. Here, $\mathbf{x}$ is the true solution, $\tilde{\mathbf{x}}$ the estimate, and $\|\mathbf{x}\|_{\mathbf{L}} = \sqrt{\mathbf{x}^T\mathbf{L}\mathbf{x}}$. To make this error bound compatible with the absolute error we pursue, we first note that $\sqrt{\lambda_2}\cdot\|\tilde{\mathbf{x}}\|_\infty \leq \|\tilde{\mathbf{x}}\|_{\mathbf{L}} \leq \sqrt{2\operatorname{vol}(G)}\cdot\|\tilde{\mathbf{x}}\|_\infty$ (Lemma 12), where λ_2 is the second smallest eigenvalue of $\mathbf{L}$. We may use Lemma 12, as $\tilde{\mathbf{x}}$ and $\mathbf{x}$ are both perpendicular to $\mathbf{1}$ (since the image of $\mathbf{L}$ is perpendicular to its kernel, which is $\mathbf{1}$). It is known that $\lambda_2 \geq 4/(n\cdot\operatorname{diam}(G))$ [46]. Hence, if we set $\eta := \frac{\kappa\epsilon}{3\cdot\sqrt{mn\log n}\operatorname{diam}(G)}$, we get for small-world graphs:

$$\begin{aligned}
\|\tilde{\mathbf{x}}-\mathbf{x}\|_\infty &\leq \frac{1}{\sqrt{\lambda_2}}\cdot\|\tilde{\mathbf{x}}-\mathbf{x}\|_{\mathbf{L}} \leq \frac{\eta}{\sqrt{\lambda_2}}\cdot\|\mathbf{x}\|_{\mathbf{L}} \\
&\leq \frac{\eta}{\sqrt{4/(n\cdot\operatorname{diam}(G))}}\cdot\|\mathbf{x}\|_{\mathbf{L}} \leq \frac{\eta\sqrt{n\log n}}{2}\cdot 2\sqrt{m}\cdot\|\mathbf{x}\|_\infty \\
&= \frac{\kappa\epsilon}{3\sqrt{mn\log n}\operatorname{diam}(G)}\cdot\sqrt{mn\log n}\cdot\|\mathbf{x}\|_\infty \\
&\leq \frac{\kappa\epsilon}{3\operatorname{diam}(G)}\|\mathbf{x}\|_\infty \leq \frac{\kappa\epsilon}{3\operatorname{diam}(G)}\operatorname{diam}(G) \leq \frac{\kappa}{3}\epsilon.
\end{aligned}$$

The second last inequality follows from the fact that $\mathbf{x}$ expresses potentials scaled by $1/n$, arising from $n-1$ (scaled) effective resistance problems fused together. The maximum norm of $\mathbf{x}$ can thus be bounded by $(n-1)\frac{1}{n}\mathbf{r}(u,v) \leq \operatorname{diam}(G)$, because $\mathbf{r}(\cdot,\cdot)$ is bounded by the graph distance.

Taking Eq. (7) into account, this means that the maximum error of a diagonal value in $\widetilde{\mathbf{L}^\dagger}$ as a consequence from the linear system can be bounded by $\kappa\epsilon$. The resulting running time for the solver is then $\tilde{\mathcal{O}}(m\log^{1/2} n\log(\frac{3\sqrt{mn\log n}\operatorname{diam}(G)}{\kappa\epsilon})) = \tilde{\mathcal{O}}(m\log^{1/2} n\log(n/\epsilon))$.

The main bottleneck is the loop that samples τ USTs and aggregates their contribution in each iteration. According to Lemma 10, sampling takes $\mathcal{O}(m\log n)$ time per tree in small-world graphs. Aggregating a tree's contribution is less expensive (Lemma 2).

Let us determine next the sample size τ that allows the desired guarantee. To this end, let $\epsilon' := (1-\kappa)\epsilon$ denote the possible absolute error for the effective resistance estimates. Plugging $\tau := \operatorname{ecc}(u)^2\cdot\lceil\log(2m/\delta)/(2(\epsilon')^2)\rceil$ into Hoeffding's inequality (Lemma 11), yields for each single edge $e\in E$ and its estimated electrical flow $\tilde{\mathbf{f}}(e)$: $Pr[\tilde{\mathbf{f}}(e) = \mathbf{f}(e)\pm\epsilon'/\operatorname{ecc}(u)] \geq 1-\delta/m$. Using the union bound, we get that $Pr[\tilde{\mathbf{f}}(e) = \mathbf{f}(e)\pm\epsilon'/\operatorname{ecc}(u)]$ for all $e\in E$ at the same time holds with probability $\geq 1-\delta$. Since for all v the path length $|P(v)|$ is bounded by $\operatorname{ecc}(u)$, another application of the union bound yields that $Pr[R[v] = \mathbf{r}(u,v)\pm\epsilon'] \geq 1-\delta$. ◀

A.5 Proof of Lemma 6

Proof. Recall that the normalized random-walk betweenness is expressed as follows (Eq. (5)):

$$c_b(v) = \frac{1}{n} + \frac{1}{n-1} \sum_{t \neq v} \frac{\mathbf{M^{-1}}[t,t] - \mathbf{M^{-1}}[t,v]}{\mathbf{M^{-1}}[t,t] + \mathbf{M^{-1}}[v,v] - 2\mathbf{M^{-1}}[t,v]}$$

where $\mathbf{M} := \mathbf{L} + \mathbf{P}$ with $\mathbf{L}$ the Laplacian matrix and $\mathbf{P}$ the projection operator onto the zero eigenvector of the Laplacian such that $\mathbf{P}[i,j] = 1/n$. We also have $\mathbf{L}^\dagger := (\mathbf{L} + \mathbf{P})^{-1} - \mathbf{P}$ and thus we can replace $\mathbf{M}^{-1}$ with $\mathbf{L}^\dagger + \mathbf{P}$. Then, for the numerator of Eq. (5) we have:

$$\begin{aligned}
&\sum_{t \neq v} (\mathbf{M^{-1}}[t,t] - \mathbf{M^{-1}}[t,v]) = \\
&\sum_{t \neq v} (\mathbf{L}^\dagger[t,t] - \mathbf{P}[t,t] - \mathbf{L}^\dagger[t,v] + \mathbf{P}[t,v]) = \\
&\sum_{t \neq v} (\mathbf{L}^\dagger[t,t] - \mathbf{L}^\dagger[t,v]) = \operatorname{tr}(\mathbf{L}^\dagger) - \mathbf{L}^\dagger[v,v] - \sum_{t \neq v} \mathbf{L}^\dagger[t,v] = \\
&\operatorname{tr}(\mathbf{L}^\dagger). \qquad (15)
\end{aligned}$$

The second equality holds because $\sum_{l \neq v} (\mathbf{P}[t,v] - \mathbf{P}[t,t]) = 0$ for all $t, v \in V$. The final equality holds since $\sum_{t \neq v}^{n} \mathbf{L}^\dagger[t,v] = -\mathbf{L}^\dagger[v,v]$ for all $v \in V$.

Then, for the denominator we have:

$$\begin{aligned}
&(n-1) \sum_{t \neq v} (\mathbf{M^{-1}}[t,t] + \mathbf{M^{-1}}[v,v] - 2\mathbf{M^{-1}}[t,v]) = \\
&(n-1) \sum_{t \neq v} (\mathbf{L}^\dagger[t,t] + \mathbf{P}[t,t] + \mathbf{L}^\dagger[v,v] + \mathbf{P}[v,v] - 2\mathbf{L}^\dagger[t,v] - 2\mathbf{P}[t,v]) = \\
&(n-1) \sum_{t \neq v} (\mathbf{L}^\dagger[t,t] + \mathbf{L}^\dagger[v,v] - 2\mathbf{L}^\dagger[t,v]) = \\
&(n-1) f^{el}(v). \qquad (16)
\end{aligned}$$

The second equality holds since $\sum_{t \neq v} \mathbf{P}[t,t] + \mathbf{P}[v,v] - 2\mathbf{P}[t,v] = 0$ for all $t, v \in V$ and the last equality due to Eq. (3) and the definition of electrical farness. The claim follows from combining Eqs. (15) and (16). ◀

B Kirchhoff Index and Related Centralities

B.1 Description

The sum of the effective resistance distances over all pairs of vertices is an important measure for network robustness known as the Kirchhoff index $\mathcal{K}(G)$ or (effective) graph resistance [20, 31]. The Kirchhoff index is often computed via the closed-form expression $\mathcal{K}(G) = n \operatorname{tr}(\mathbf{L}^\dagger)$ [31], where the trace is the sum of the diagonal elements. Li and Zhang [39] recently adapted the Kirchhoff index to obtain two edge centrality measures for $e \in E$: (i) $\mathcal{C}_\theta(e) := n \operatorname{tr}(\mathbf{L}^\dagger \setminus_\theta e)$, where $\mathbf{L} \setminus_\theta e$ corresponds to a graph in which edge e has been down-weighted according to a parameter θ and (ii) $\mathcal{C}^\Delta_\theta(e) := \mathcal{C}_\theta(e) - \mathcal{K}(G)$, which quantifies the difference of the Kirchhoff indices between the new and the original graph.

B.2 Related Work

To calculate the Kirchhoff edge centralities, Ref. [39] uses techniques such as partial Cholesky factorization [36], fast Laplacian solvers and the Hutchinson estimator. For $\mathcal{C}_\theta^\Delta(e)$, which is the more interesting measure in our context, they propose an ϵ-approximation algorithm that approximates $\mathcal{C}_\theta^\Delta(e)$ for all edges in $\mathcal{O}(m\epsilon^{-2}\theta^{-2}\log^{2.5} n \log(1/\epsilon))$ time (up to polylogarithmic factors). The algorithm uses the Sherman-Morrison formula [55], which gives a fractional expression of $(\mathbf{L}^\dagger \setminus_\theta e - \mathbf{L}^\dagger)$. The numerator is approximated by the Johnson-Lindenstrauss lemma, and the denominator by effective resistance estimates for all edges.

B.3 Kirchhoff Index and Edge Centralities

It is easy to see that Algorithm 1 can approximate Kirchhoff Index, exploiting the expression $\mathcal{K}(G) = n\,\mathrm{tr}(\mathbf{L}^\dagger)$ [31]. As a direct consequence, we have:

▶ **Proposition 13.** *Let G be a small-world graph as in Theorem 3. Then, Algorithm 1 approximates with high probability $\mathcal{K}(G)$ with absolute error $\pm\epsilon$ in $\mathcal{O}(m\log^4 n \cdot \epsilon^{-2})$ time.*

We also observe that we can use a component of Algorithm 1 to approximate $\mathcal{C}_\theta^\Delta(e)$. Recall that $\mathcal{C}_\theta^\Delta(e) = \mathcal{C}_\theta(e) - \mathcal{K}(G) = n(\mathrm{tr}(\mathbf{L}^\dagger \setminus_\theta e) - \mathrm{tr}(\mathbf{L}^\dagger))$. Using the Sherman-Morrison formula, as done in Ref. [39], we have:

$$\mathcal{C}_\theta^\Delta(e) = n(1-\theta)\frac{\mathbf{w}(e)\,\mathrm{tr}(\mathbf{L}^\dagger \mathbf{b}_e \mathbf{b}_e^\top \mathbf{L}^\dagger)}{1-(1-\theta)\mathbf{w}(e)\mathbf{b}_e^\top \mathbf{L}^\dagger \mathbf{b}_e}, \tag{17}$$

where $\mathbf{b}_e$ for $e = (u, v)$ is the vector $\mathbf{e}_u - \mathbf{e}_v$.

Ref. [39] approximates $\mathcal{C}_\theta^\Delta(e)$ with an algorithm that runs in $\mathcal{O}(m\theta^{-2}\log^{2.5} n \log(1/\epsilon)\,\mathrm{poly}(\log\log n)\cdot\epsilon^{-2})$ time. The algorithm is dominated by the denominator of Eq. (17), which runs in $\mathcal{O}(m\theta^{-2}\log^{2.5} n\,\mathrm{poly}(\log\log n)\cdot\epsilon^{-2})$. For the numerator of Eq. (17), they use the following Lemma:

▶ **Lemma 14** (paraphrasing from Ref. [39])**.** *Let $\mathbf{L}$ be a Laplacian matrix and ϵ a scalar such that $0 < \epsilon \leq 1/2$. There is an algorithm that achieves an ϵ-approximation of the numerator of Eq. (17) with high probability in $\mathcal{O}(m\log^{1.5} n \log(1/\epsilon)\cdot\epsilon^{-2})$ time.*

The algorithm in Lemma 14 uses the Monte-Carlo estimator with $\mathcal{O}(\epsilon^{-2}\log n)$ random vectors $\mathbf{z}_i$ to calculate the trace of the implicit matrix $\mathbf{y}_i^\top \mathbf{b}_e \mathbf{b}_e^\top \mathbf{y}_i$, where $\mathbf{y}_i$ is the approximate solution of $\mathbf{y}_i := \mathbf{L}^\dagger \mathbf{z}_i$ – derived from solving the corresponding linear system involving $\mathbf{L}$. For each system, the Laplacian solver runs in $\mathcal{O}(m\log^{1/2} n \log(1/\epsilon))$ time.

We notice that a UST-based sampling approach works again for the denominator: The denominator is just $1-(1-\theta)\mathbf{w}(e)\mathbf{r}(e)$, where $e \in E$ ($\mathbf{r}(e) = \mathbf{b}_e^\top \mathbf{L}^\dagger \mathbf{b}_e$). Approximating $\mathbf{r}(e)$ for every $e \in E$ then requires sampling USTs and counting for each edge e the number of USTs it appears in. Moreover, we only need to sample $q = \lceil 2\epsilon^{-2}\log(2m/\delta)\rceil$ to get an ϵ-approximation of the effective resistances for all edges (using Theorem 8 in Ref. [27]). Since $\mathbf{r}(e)$ are approximate, we need to bound their approximation when subtracted from 1. Following Ref. [39], we use the fact that $0 < \theta < 1$ and that for each edge $\mathbf{w}(e)\mathbf{r}(e)$ is between 0 and 1, bounding the denominator. The above algorithm can be used to approximate the denominator of Eq. (17) with absolute error $\pm\epsilon$ in $\mathcal{O}(m\log^2 n \cdot \epsilon^{-2})$ time. Combining the above algorithm and Lemma 14, it holds that:

▶ **Proposition 15.** *Let $G = (V, E)$ be a small-world graph as in Theorem 3. Then, there is an algorithm (using Lemma 14 and our Wilson-based sampling algorithm) that approximates with high probability $\mathcal{C}_\theta^\Delta(e)$ for all $e \in E$ with absolute error $\pm\epsilon$ in $\mathcal{O}(m\log^2 n \log(1/\epsilon)\cdot\epsilon^{-2})$ time.*

C Detailed Engineering Aspects

C.1 UST Generation, Pivot Selection, and the Linear System

Wilson's algorithm [62] using loop-erased random walks is the best choice in practice for UST generation and also the fastest asymptotically for unweighted small-world graphs. A fast random number generator is required for this algorithm; our code uses PCG32 [50] for this purpose. For our implementation we use a variant of Wilson's algorithm to sample each tree, proposed by Hayashi et al. [27]: first, one computes the biconnected components of G, then applies Wilson to each biconnected component, and finally combines the component trees to a UST of G. In each component, we use a vertex with maximal degree as root for Wilson's algorithm. Using this approach, Hayashi et al. [27] experienced an average performance improvement of around 40% on sparse graphs compared to running Wilson directly.

As a consequence of Theorem 3, the pivot vertex u should be chosen to have low eccentricity. As finding the vertex with lowest eccentricity with a naive APSP approach would be too expensive, we compute a lower bound on the eccentricity for all vertices of the graph and choose u as the vertex with the lowest bound. These bounds are computed using a strategy analogous to the double sweep lower bound by Magnien et al. [44]: we run a BFS from a random vertex v, then another BFS from the farthest vertex from v, and so on. At each BFS we update the bounds of all the visited vertices; an empirical evaluation has shown that 10 iterations yield a reasonably accurate approximation of the vertex with lowest eccentricity.

As a result from preliminary experiments, we use a general-purpose Conjugate Gradient (CG) solver for the single (sparse) Laplacian linear systems, together with a diagonal preconditioner. We choose the implementation of the C++ library Eigen [26] for this purpose and found that the accuracy parameter $\kappa = 0.3$ yields a good trade-off between the CG and UST sampling steps.

C.2 Parallel Implementation

Shared memory. Our implementation uses OpenMP for shared-memory parallelism. We aggregate $R[\cdot]$ in thread-local vectors and perform a final parallel reduction over all $R[\cdot]$. We found that on the graphs that we can handle in shared memory, no sophisticated load balancing strategies are required to achieve reasonable scalability.

Distributed memory. We provide an implementation of our algorithm for replicated graphs in distributed memory that exploits hybrid parallelism based on MPI + OpenMP. On each compute node, we take samples and aggregate $R[\cdot]$ as in shared memory. Compared to the shared-memory implementation, however, our distributed-memory implementation exhibits two main peculiarities: (i) we still solve the initial Laplacian system on a single compute node only; we interleave, however, this step with UST sampling on other compute nodes, and (ii) we employ explicit load balancing. The choice to solve the initial system on a single compute node only is done to avoid additional communication among nodes. In fact, we only expect distributed CG solvers to outperform this strategy for inputs that are considerably larger than the largest graphs that we consider. Furthermore, since we interleave this step of the algorithm with UST sampling on other compute nodes, our strategy only results in a bottleneck on input graphs where solving a single Laplacian system is slower than taking *all* UST samples – but these inputs are already "easy".

For load balancing, the naive approach would consist of statically taking $\lceil \tau/p \rceil$ UST samples on each of the p compute nodes. However, in contrast to the shared-memory case, this does not yield satisfactory scalability. In particular, for large graphs, the running

time of the UST sampling step has a high variance. To alleviate this issue, we use a simple *dynamic* load balancing strategy: periodically, we perform an asynchronous reduction (`MPI_Iallreduce`) to calculate the total number of UST samples taken so far (over all compute nodes). Afterwards, each compute node calculates the number of samples that it takes before the next asynchronous reduction (unless more than τ samples were taken already, in which case the algorithm stops). We compute this number as $\lceil \tau/(b \cdot p^\xi) \rceil$ for fixed constants b and ξ. We also overlap the asynchronous reduction with additional sampling to avoid idle times. Finally, we perform a synchronous reduction (`MPI_Reduce`) to aggregate $R[\cdot]$ on a single compute node before outputting the resulting diagonal values. By parameter tuning [5], we found that choosing $b = 25$ and $\xi = 0.75$ yields the best parallel scalability.

Cutting Polygons into Small Pieces with Chords: Laser-Based Localization

Esther M. Arkin
Stony Brook University, NY, USA
esther.arkin@stonybrook.edu

Rathish Das
Stony Brook University, NY, USA
rathish.das@stonybrook.edu

Jie Gao
Rutgers University, Piscataway, NJ, USA
jg1555@rutgers.edu

Mayank Goswami
Queens College of CUNY, New York, NY, USA
mayank.goswami@qc.cuny.edu

Joseph S. B. Mitchell
Stony Brook University, NY, USA
joseph.mitchell@stonybrook.edu

Valentin Polishchuk
Linköping University, Norrköping, Sweden
valentin.polishchuk@liu.se

Csaba D. Tóth
California State University Northridge,
Los Angeles, CA, USA
Tufts University, Medford, MA, USA
csaba.toth@csun.edu

Abstract

Motivated by indoor localization by tripwire lasers, we study the problem of cutting a polygon into small-size pieces, using the chords of the polygon. Several versions are considered, depending on the definition of the "size" of a piece. In particular, we consider the area, the diameter, and the radius of the largest inscribed circle as a measure of the size of a piece. We also consider different objectives, either minimizing the maximum size of a piece for a given number of chords, or minimizing the number of chords that achieve a given size threshold for the pieces. We give hardness results for polygons with holes and approximation algorithms for multiple variants of the problem.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Packing and covering problems; Mathematics of computing → Combinatorial algorithms; Theory of computation → Computational geometry

Keywords and phrases Polygon partition, Arrangements, Visibility, Localization

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.7

Related Version A full version of the paper is available at `https://arxiv.org/abs/2006.15089`.

Funding This research was partially supported by NSF grants CCF-1439084, CCF-1540890, CCF-1617618, CCF-1716252, CCF-1725543, CCF-1910873, CCF-2007275, CNS-1618391, CNS-1938709, CRII-1755791, CSR-1763680, DMS-1737812, DMS-1800734, and OAC-1939459. The authors also acknowledge partial support from the US-Israel Binational Science Foundation (project 2016116), the DARPA Lagrange program, the Sandia National Labs and grants by the Swedish Transport Administration and the Swedish Research Council.

Acknowledgements We thank Peter Brass for technical discussions and for organizing an NSF-funded workshop where these problems were discussed and this collaboration began.

1 Introduction

Indoor localization is a challenging and important problem. While GPS technology is very effective outdoors, it generally performs poorly inside buildings, since GPS depends on line-of-sight to satellites. Thus, other techniques are being considered for indoor settings. One of the options being investigated for localization and tracking is to use one-dimensional

tripwire sensors [16] such as laser beams, video cameras with a narrow field of view [31], and pyroelectric or infrared sensors [12, 14]. In these approaches, multiple sensors emitting directional signal beams are deployed in an environment, with the beams inducing an arrangement that cuts the domain into cells, allowing one to track the movement of a mobile target from one cell to another when it crosses the signal beam. Since the accuracy of the localization depends on the sizes of the cells, it is desirable to cut the polygon into *small* pieces. With such beam deployment, one can also ensure that no "large" object can be "hidden" in the domain, since any such object will necessarily intersect one of the beams.

In the literature there have been studies of target localization and tracking using such "tripwire" sensors. Zheng, Brady, and Agarwal [32] consider general models of "boundary sensors" that are triggered when an object crosses them. They assume that the position of the sensors is already given and consider the signal processing problem of determining the location and trace of a target by the spatial and temporal sequence of the laser beams crossed by the target. In this paper, we focus on the problem of optimizing the placement of signal beam sensors to minimize the ambiguity of target location within each cell.

Problem Formulation and Notation. We study various versions of the laser cutting problem. The input polygon, denoted by P, is a closed polygonal domain (i.e., a connected compact set in $\mathbb{R}^2$ with piecewise linear boundary) having a total of n vertices, r of which are reflex (having internal angle greater than π). The terms "cut" and "laser" will be used interchangeably to denote a chord of P, i.e., a maximal line segment in P whose relative interior lies in the interior of P. The *measure* (or *size*) of a cell in the arrangement will be (a) the cell's area, (b) its diameter (defined as the maximum Euclidean distance between two points of the cell), or (c) the radius of the largest inscribed disk within the cell.

For each measure, we consider two formulations of the optimization problem:

- MinMeasure: Given a positive integer k, determine how to place k laser beams in P to minimize the maximum measure, δ, of a cell in the arrangement of the lasers.
- Min-LaserMeasure: Given $\delta > 0$, determine the smallest number of laser beams to cut P into cells each of measure at most δ.

In Min-LaserMeasure, no generality is lost by taking the cell size bound, δ, to be 1. We assume that the optimal solution is greater than a constant c; otherwise, the problem can be solved optimally in $O(n^{\text{poly}(c)})$ time (in the real RAM model of computation, standard for geometric algorithms) by reducing it to a mathematical program whose variables are the locations of the lasers endpoints on the boundary of P (the space of the variables would be split into regions of fixed combinatorial types for all the lasers, and in each region, the measures for the cells of the partition of P will be explicitly written and optimized – since each cell has $\text{poly}(c) = O(1)$ complexity, the optimization problem will be of constant size). It may be interesting to investigate also the opposite scenario and obtain efficient algorithms for minimizing the measures using a small given number of lasers. Further variants of the problem may be defined. One possible requirement is to use only axis-aligned lasers – in fact, with this restriction (of primarily theoretical interest) we obtain better approximations than for the more general case of unrestricted-orientation lasers.

Results. We give hardness results and approximation algorithms for several variants of the problems, using a variety of techniques. Specifically,

- Section 2 proves hardness of our problems in polygons *with holes*: we show that it is NP-hard to decide whether one can split the domain into pieces of measure at most δ, using a given number k of lasers (this holds for any of the measures, which implies that

both MinMeasure and Min-LaserMeasure are hard for polygons with holes). Our hardness reductions hold using axis-parallel lasers, as well, which implies that the problem is hard with or without the restriction to axis-aligned lasers.

- Section 3.1 gives an $O(\log r)$-approximation for Min-LaserArea in *simple* polygons. The algorithm "unrefines" the ray shooting subdivision by Hershberger and Suri [17], merging the triangles bottom-up along the decomposition tree; the merging stops whenever the next merge would create a cell of area greater than δ, implying that the boundaries between the merged cells can be charged to disjoint parts of P of area more than δ. The lasers are then put along the cell boundaries of the coarsened subdivision; since the subdivision is obtained by cutting out $O(1)$ children from parents in a tree on the original subdivision (where the children were separated from parents by polygonal chains of $O(1)$ complexity), we can charge these $O(1)$ lasers to the intersection of OPT with a region of area more than δ. The remaining large pieces in the coarsened subdivision (e.g., triangles of area more than δ in the initial triangulation) are cut with a suitable grid of lasers, which is within a constant factor of optimal subdivision for each piece. The $O(\log n)$ approximation factor then follows from the fact that each laser could pass through $O(\log n)$ cells of the original subdivision (the subdivision's core property). To bring the approximation factor down to $O(\log r)$ we decompose P into convex pieces with a decomposition whose stabbing number is $O(\log r)$ (a result, which may be of independent interest) and use the same scheme as with the Hershberger–Suri decomposition.
- In Section 3.2 we present a bi-criteria approximation to the diameter version for *simple* polygons: if k lasers can cut P into pieces of diameter at most δ, we find a cutting with at most $2k$ lasers into $O(\delta)$-diameter pieces. In Section 3.3 we use the bi-criteria algorithm to give a constant-factor approximation to MinDiameter. Both algorithms use only axis-aligned lasers, yielding the same approximation guarantees for the versions with general-direction lasers and with axis-aligned lasers.
- Section 4 gives a constant-factor approximation to Min-LaserDiameter and Min-LaserArea in *simple* polygons under the restriction that the lasers are axis-aligned. The algorithms are based on "histogram decomposition" with constant stabbing number and solving the problems in each histogram separately.
- In Section 5 we give a bi-criteria approximation to the diameter version in polygons *with holes* under the restriction that lasers are axis-parallel. The algorithm is similar to the one for simple polygons in that they both use a grid; however, everything else is different: in simple polygons we place lasers along grid lines, while in polygons with holes the grid lines just subdivide the problem (in fact, we consider the vertical and the horizontal strips separately). More importantly, even though we place axis-aligned lasers in both simple and nonsimple polygons, for the former we approximate cutting with arbitrary-direction lasers, while for the latter only cuttings with axis-aligned lasers (approximating cuttings with general-direction lasers in polygons with holes is open). We use the bi-criteria algorithm to give a constant-factor approximation to MinDiameter in polygons with holes – this part is the same as for simple polygons.
- Section 6 gives an $O(\log \mathrm{OPT})$-approximation for Min-LaserCircle in polygons *with holes*. The algorithm is based on a reduction to the SetCover problem.

Table 1 summarizes our results. The running times of our algorithms depend on the output complexity, which may depend on the size (area, perimeter, etc.) of P. Some of our algorithms can be straightforwardly made to run in strongly-polynomial time, producing a strongly-polynomial-size representation of the output; for others, such conversion – which in general is outside our scope – is not easily seen. Many versions of the problem still remain

open. For simple polygons, despite considerable attempts, we have neither hardness results nor polynomial-time algorithms to compute an optimal solution; all of our positive results are approximation algorithms.

Table 1 Approximations for simple polygons. The results marked with asterisks apply also to polygons with holes (either directly or with a similar/extended algorithm).

	Axis-Parallel Lasers		Unrestricted-Direction Lasers	
	Min-LaserMeasure	MinMeasure	Min-LaserMeasure	MinMeasure
Area	$O(1)$ § 4	OPEN	$O(\log r)$ § 3.1	OPEN
Diameter	$O(1)$ § 4	$O(1)$* § 3.3, § 5	bi-critreria § 3.2	$O(1)$ § 3.3
In-circle radius	$O(\log \mathrm{OPT})$* § 6	OPEN	$O(\log \mathrm{OPT})$* § 6	OPEN

Related Previous Work. Previous results on polygon decomposition [21] use models that do not support laser cuts or are restricted to convex bodies. For example, *Borsuk's conjecture* [5, 18, 19] seeks to partition a convex body of unit diameter in $\mathbb{R}^d$ into the minimum number of pieces of diameter less than one. Conway's *fried potato problem* [3, 9] seeks to minimize the maximum in-radius of a piece after a given number of *successive* cuts by hyperplanes for a convex input polyhedron in $\mathbb{R}^d$. Croft et al. [9, Problem C1] raised a variant of the problem in which a convex body is partitioned by an *arrangement* of hyperplanes (i.e., our problem in $\mathbb{R}^d$), but no results have been presented.

Equipartition problems ask to partition convex polygons into convex pieces all having the same area or the same perimeter (or other measures) [2, 4, 20, 22, 27, 29]. In these problems, the partition is not restricted to chords (or hyperplanes). Topological methods are used for existential results in this area, and very few algorithmic results are known [1]. Another related problem is the family of so-called *cake cutting problems* [13, 28], in which an infinite straight line "knife" is used to cut a convex "cake" into (convex) pieces that represent a "fair" division into portions. In contrast, we are interested in cutting *nonconvex* polygons into connected pieces.

In [6] several variants of Chazelle's result from [8] were explored, including cutting the polygon along a chord to get approximately equal areas of the two resulting parts. Yet another related problem is that of "shattering" with arrangements [11], in which one seeks to isolate objects in cells of an arrangement of a small number of lines, but without consideration of the size of the cells (as is important in our problem).

2 Hardness in Polygons with Holes

We show that for all three measures (area, diameter, the radius of the largest inscribed circle) it is NP-hard to decide whether a given polygon P *with holes* can be divided into pieces of small measure using a given number of lasers, both for unrestricted-orientation and axis-aligned lasers. However, it is currently open whether these problems remain NP-hard for simple polygons.

We prove hardness by reduction from the 3SAT problem. Our polynomial-time reduction is similar to previous reductions for line cover problems, which are geometric variants of set cover [23]. In particular, Megiddo and Tamir [25] proved that the LineCover problem is NP-complete: Given n points in the plane and an integer k, decide whether the points can be covered by k lines. Hassin and Megiddo [15] proved hardness for MinimumHittingHorizontalUnitSegments problem: Given n horizontal line segments in the plane, each of

unit length, and an integer k, decide whether there exists a set of k axis-parallel lines that intersects all n segments. Our reduction is based on the idea of Hassin and Megiddo, but requires some adjustments to generate a subdivision of a polygon.

▶ **Theorem 1.** *In a polygon with holes, both MinArea and Min-LaserArea are NP-hard (with or without the axis-aligned lasers restriction).*

Proof. We reduce from 3-SAT. Let Φ be a boolean formula in 3CNF with m clauses $c_1, \ldots, c_m$, and n variables $x_1, \ldots, x_n$. We construct an orthogonal polygon P with holes and an integer k such that Φ is satisfiable if and only if P can be subdivided into regions of area at most 2 using k lasers. (The reduction goes through with or without the restriction that all lasers are axis-parallel).

We construct a polygon P from the rectangle $B = [0, 7m + 2] \times [0, 3n + 4]$ by carving rectangular "rooms" connected by narrow corridors. The rooms are pairwise disjoint and they each have area of 2. The corridors are axis-parallel, run between opposite sides of the bounding box B, and their width is $1/(100 \max\{m, n\})$. See Fig. 1 for an illustration.

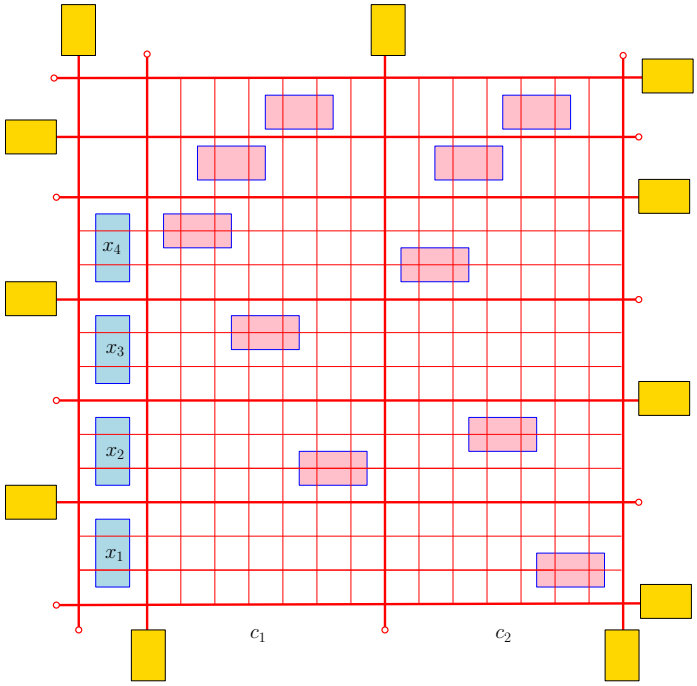


Figure 1 An example for the rooms and corridors for $\Phi = (x_2 \vee \overline{x}_3 \vee \overline{x}_4) \wedge (x_1 \vee \overline{x}_2 \vee x_4)$. The rooms corresponding to variables are in blue. The rooms corresponding to clauses (five rooms per clause) are in pink. The corridors are shown in red, some of which are connected to additional rooms (shown in yellow). The polygon is composed of the union of all the rooms and corridors.

Variable rooms. For each variable x_i, $i = 1, \ldots, n$, create one room: $[\frac{1}{2}, \frac{3}{2}] \times [3(i-1) + \frac{1}{2}, 3i - \frac{1}{2}]$. Note that all rooms are to the left of the line $x = 2$.

Clause rooms. For each clause c_j, $j = 1, \ldots, m$, create five rooms. All five rooms have size 2×1 and lie between the lines $x = 7(j-1)+2$ and $x = 7j+2$. Three out of five rooms are aligned with the variable rooms. Suppose c_j contains the variables x_i, $x_{i'}$, and $x_{i''}$, where $i < i' < i''$. If x_i is nonnegated, then create the room $[7(j-1) + \frac{1}{2}, 7(j-1) + \frac{5}{2}] \times [3(i-1) + \frac{1}{2}, 3i - \frac{3}{2}]$;

otherwise create the room $[7(j-1)+\frac{1}{2}, 7(j-1)+\frac{5}{2}]\times[3(i-1)+\frac{3}{2}, 3i-\frac{1}{2}]$. We create a room for $x_{i'}$ (resp., $x_{i''}$) analogously, shifted by a horizontal vector $(0,2)$ (resp;., $(0,4)$). Note that the x-projections of these rectangles do not overlap. Two additional rooms lie above the variable rooms: $[7(j-1)+\frac{3}{2}, 7(j-1)+\frac{7}{2}]\times[2n+\frac{1}{2}, 2n+\frac{3}{2}]$ and $[7(j-1)+\frac{7}{2}, 7(j-1)+\frac{11}{2}]\times[2n+\frac{5}{2}, 2n+\frac{7}{2}]$.

Corridors and separator gadgets. Create narrow corridors along the vertical lines $x = 0, 2, 3, \ldots, 7m$ and horizontal lines $y = 0, 1, 2, \ldots, 3n$, $y = 3n+2$, and $y = 3n+4$. Add rectangular rooms of area 2 at one end of some of the corridors. Specifically, we add rooms to the corridors at $x = 0$ and $x = 7j+2$ for $j = 0, 1, \ldots, m$ alternately at the top and bottom endpoints; and similarly for the corridors at $y = 3i$ for $i = 0, 1, \ldots, n$, $y = 3n+2$, and $y = 3n+4$, alternately at the left and right endpoints. Altogether, $m+n+5$ corridors have rooms at their endpoints.

Finally, we set the parameter $k = 3m+2n+5$. This completes the description of an instance corresponding to the Boolean formula Φ.

Equivalence. Let $\tau : x_i \to \{\texttt{true}, \texttt{false}\}$ be a satisfying truth assignment for Φ. We show that P can be subdivided by k lasers into regions of area at most 2. Place lasers at all horizontal and vertical lines that have additional rooms at their endpoints; this requires $m+n+5$ lasers. These lasers subdivide P into subpolygons that each intersect at most one room. For $i = 1, \ldots, n$, if $\tau(x_i) = \texttt{true}$, then place a horizontal laser at $y = 3(i-1)+1$ (along the bottom corridor touching room for x_i), otherwise at $y = 3(i-1)+2$ (along the top corridor touching room for x_i). These lasers split each variable room into two rectangles of area $\frac{1}{2}$ and $\frac{3}{2}$. For $j = 1, \ldots, m$, we place two vertical lasers that subdivide the rooms associated with clause c_j. Since τ is a satisfying truth assignment, the rooms corresponding to true literals are already split by horizontal lasers. As can easily be checked, the remaining (at most 4) rooms can be split using two vertical lasers. Now P is subdivided into pieces that each intersect at most one room, and contains at most 1.5 area of each room. Since the corridors are narrow, the area of each piece is less than 2, as required. We have used n horizontal lasers for the variables, and $2m$ vertical lasers for clauses. Overall, we have used $(m+n+5)+n+2m = 3m+2n+5$ lasers.

Suppose now that $k = 3m+2n+5$ lasers can subdivide P into polygons of area at most 2. We show that Φ is satisfiable. The area of each room is about 2, so they each intersect at least one laser. Each variable room requires at least one laser; and the n variable rooms jointly require n lasers (as no laser can intersects two variable rooms). Each clause is associated with two rooms above the line $y = 3n$; which jointly require two lasers. Overall these rooms require $2m$ lasers.

Note that a laser that intersects a clause rooms above $y = 3n$ or a variable room cannot intersect any room at the end of corridors. We are left with at most $k-(n+2m) = m+n+5$ lasers to split these rooms. Since we have precisely $m+n+5$ rooms at the end of the corridors, and no laser can intersect two such rooms, there is a unique laser intersecting each of these rooms. As argued above, for $i = 1, \ldots, n$, the room associated with x_i intersects only one laser. If this laser intersects the corridor at $y = 3(i-1)+1$, then let $\tau(x_i) = \texttt{true}$, otherwise $\tau(x_i) = \texttt{false}$. For $j = 1, \ldots, m$, there are two lasers that intersect the two rooms associated with c_j above $y = 3n$. These two lasers cannot intersect all three rooms associated with c_j below $y = 3n$. Consequently, at least one of these rooms intersects a laser coming from a variable room. Hence each clause contains a true literal, and Φ is satisfiable. ◀

The proofs of the following two theorems are presented in the full version of the paper.

▶ **Theorem 2.** *In polygons with holes, both MinDiameter and Min-LaserDiameter are NP-hard (with or without the axis-aligned lasers restriction).*

▶ **Theorem 3.** *In polygons with holes, both MinCircle and Min-LaserCircle are NP-hard (with or without the axis-aligned lasers restriction).*

3 Decomposition Algorithms for Simple Polygons

In this section, we present approximation results for decomposing a simple polygon P by lasers of arbitrary orientations (recall that n denotes the total number of vertices of P and r is the number of reflex vertices). We describe an $O(\log r)$-approximation for Min-LaserArea (Section 3.1), a bi-criteria algorithm for diameter (Section 3.2), and a $O(1)$-approximation for MinDiameter (Section 3.3).

3.1 Min-LaserArea

Given a simple polygon P and a threshold δ, we wish to find the minimum number of lasers that subdivide P into pieces, each of area at most 1. We start with the easy $O(1)$-approximation in the special case when P is a convex polygon (Proof in the full version).

▶ **Lemma 4.** *For every convex polygon P, we can find a set of $k = O(\sqrt{\text{area}(P)})$ lasers that subdivide P into pieces, each of area at most 1, in $O(k+n)$ time. Every decomposition into pieces of area at most 1 requires $\Omega(\sqrt{\text{area}(P)})$ lasers.*

Overview. We give a brief overview of our approximation algorithm for a simple polygon P. The basic idea is to decompose P into convex pieces, and use Lemma 4 to further decompose each convex piece. There are two problems with this naïve approach: (1) a laser in an optimal solution may intersect several convex pieces (i.e., the sum of lower bounds for the convex pieces is not a global lower bound); and (2) the lasers used for a convex decomposition are not accounted for. We modify the basic approach to address both of these problems.

We use the Hershberger–Suri triangulation (as a convex subdivision). For a simple polygon P with n vertices, Hershberger and Suri [17] construct a Steiner triangulation into $O(n)$ triangles such that every chord of P intersects $O(\log n)$ triangles. We can modify their construction to produce a Steiner decomposition into a set $\mathcal{C}$ of convex cells (rather than triangles) such that each laser intersects $O(\log r)$ convex cells, where r is the number of reflex vertices of P. Thus, each laser of OPT can help partition $O(\log r)$ convex cells; this factor dominates the approximation ratio of our algorithm.

A convex cell $C \in \mathcal{C}$ is *large* if $\text{area}(C) > 1$, otherwise it is *small*. We decompose each large convex cell using Lemma 4. We can afford to place $O(1)$ lasers along the boundary of a large cell. We cannot afford to place lasers on the boundaries of all small cells. If we do not separate the small cells, however, they could merge into a large (nonconvex) region, so we need *some* separation between them. In the algorithm below, we construct such separators recursively by carefully unrefining the Hershberger–Suri triangulation. The unrefined subdivision is no longer a triangulation, but we maintain the properties that (i) each cell is bounded by $O(1)$ lasers within each pseudotriangle (and an arbitrary number of consecutive edges of P), and (ii) every chord of P intersects $O(\log n)$ cells.

Basic properties of the Hershberger–Suri triangulation. Given a simple polygon P with n vertices, Hershberger and Suri [17] construct a Steiner-triangulation in two phases (see Fig. 2 for an example): First, they subdivide P into $O(n)$ pseudotriangles (i.e., simple

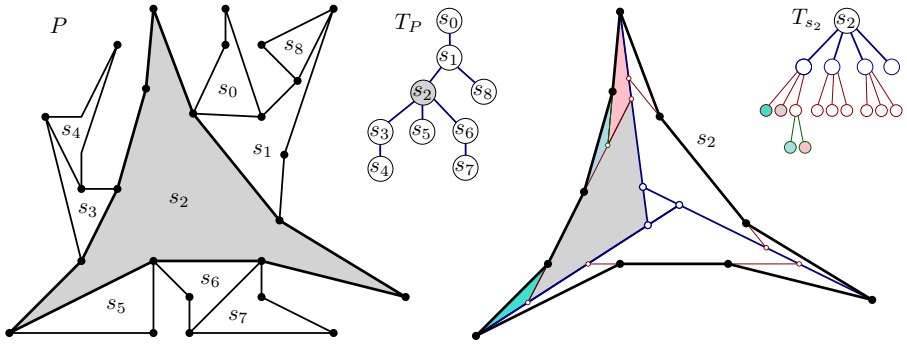


Figure 2 Left: A simple polygon P, decomposed into pseudotriangles, and the dual graph T_P. Right: A pseudotriangle s_2 is recursively subdivided into Steiner triangles, with recursion tree T_{s_2}.

polygons with precisely three convex vertices) using $O(n)$ noncrossing diagonals of P; and then subdivide each pseudotriangle into Steiner triangles. The runtime of their algorithm, as well as the number of Steiner triangles, is $O(n)$. Let $\mathcal{S}$ denote the set of pseudotriangles produced in the first phase; and let T_P be the *dual tree* of the pseudotriangles, in which each node corresponds to a pseudotriangle, and two nodes are adjacent if and only if the corresponding pseudotriangles share an edge (a diagonal of P). Note that the degree of T_P is not bounded by a constant (it is bounded by n), as a pseudotriangle may be adjacent to arbitrarily many other pseudotriangles. We consider T_P to be a rooted tree, rooted at an arbitrary pseudotriangle. Then every nonroot pseudotriangle s in $\mathcal{S}$ has a unique edge incident to the parent of s; we call this edge the *parent edge of* s.

Hershberger and Suri subdivide each pseudotriangle $s \in S$ recursively: In each step, they use $O(1)$ line segments to subdivide a pseudotriangle into $O(1)$ pseudotriangles, which are further subdivided recursively until they obtain triangles. Let us denote by T_s the recursion tree for s. Each vertex $v \in T_s$ represents a region $R_v \subset s$: The root of T_s represents s, and the leaves represent the Steiner triangles in s. The recursive subdivision maintains the following two properties: (a) Every edge of s is incident to a unique region in each level of T_s, (b) For each node $v \in T_s$, the boundary between R_v and $s \setminus R_v$ is a polyline with $O(1)$ edges (that is, R_v is bounded by $O(1)$ line segments inside s, and some sequence of consecutive edges of s).

Algorithm. We are ready to present an approximation algorithm for Min-LaserArea. Given a simple polygon Q, we begin by computing the Hershberger–Suri triangulation, the pseudotriangles $\mathcal{S}$, the dual tree T_P, and a recursion tree T_s for each pseudotriangle $s \in \mathcal{S}$. We then process the pseudotriangles in a bottom-up traversal of T_P.

Within each pseudotriangle $s \in \mathcal{S}$, we unrefine the Steiner triangulation of s by merging some of the cells into one cell (the resulting larger cells need not be triangular or convex). Initially, each node $v \in T_s$ corresponds to a region $R_v \subseteq s$. However, if we do not place lasers along the edges of s, then R_v may be adjacent to (and merged with) other cells that are outside the pseudotriangle s, along the boundary of s. Since we have an upper bound on the total area of each cell in the final decomposition, we need to keep track of the area of the region on both sides of an edge of the pseudo-triangulation. In the course of unrefinement algorithm for all $s \in \mathcal{S}$, we compute nonnegative *weights* $w(\cdot)$ for all edges of the pseudotriangulation. The weights are used for bookkeeping purposes. Specifically, the edges of P have zero weight. In a bottom-up traversal of T_P, when we start processing a pseudotriangle s, the *weights* $w(e)$

have already been computed for all edges of the pseudo-triangle s except the parent edge of s. The weight $w(e)$ for the parent edge e of s is determined when we have computed the unrefined subdivision of s; and $w(e)$ will be the area of the unrefined cell in s adjacent to the parent edge. A node $v \in T_s$ initially corresponds to a region R_v within the pseudotriangle s, but in the final decomposition of P, the node is part of some larger cell $\widehat{R}_v \subseteq P$, with $\text{area}(\widehat{R}_v) = \text{area}(R_v) + \sum_e w(e)$, where the summation is over all edges of s on the boundary of R_v, and $w(e)$ denotes the area of the cell on the opposite side of e.

As the weight of the parent edge is not available yet when we unrefine s, we modify the recursion tree T_s as follows: We choose the root to be the leaf $v_0 \in T_s$ adjacent to the parent edge of s, and reverse the parent-child relation on all edges of T_s along the s-v_0 path. We denote the modified recursion tree T'_s (Fig. 3 (left)). For all nodes v along the s-v_0 path (including s and v_0), we redefine the corresponding regions of the nodes in T'_s as follows. We denote by $R_v(T_s)$ and $R_v(T'_s)$ the regions corresponding to node v in trees T_s and T'_s, respectively. We set $R_{v_0}(T'_s) := s$ and for all other nodes v along the s-v_0 path (including s), we set $R_v(T'_s) := s \setminus R_u(T_s)$, where u is the parent of v in T'_s. With a slight abuse of notation, we set $R_v = R_v(T'_s)$ for all $v \in T'_s$ for the remainder of the algorithm. Note that $\text{area}(R_v)$ monotonically decreases with the depth in T'_S.

In a bottom-up traversal of T_P, consider every $s \in \mathcal{S}$. We proceed with two phases (see Fig. 3 for an example).

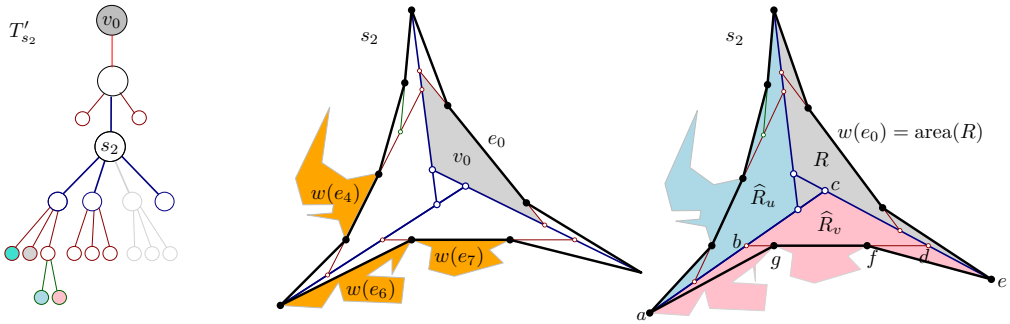


Figure 3 Left: The modified recursion tree T'_{s_2}. Middle: pseudotriangle s_2 with the initial Steiner triangulation, edge weights representing the areas of adjacent regions in the descendants of s_2, and the parent edge e_0 of s_2. Right: The unrefined subdivision of s_2 into R_u, R_v, and R; larger cells $\widehat{R}_u$ and $\widehat{R}_v$ (blue and pink), and the weight $w(e_0) = \text{area}(R)$ of the parent edge of s_2 (gray).

Phase 1 of the algorithm is an unrefinement process, that successively merges small cells of the Hershberger–Suri triangulation (no lasers are involved). We initialize three variables:

$$R := s, \qquad T := T'_s, \qquad U_s := \emptyset,$$

where $R \subseteq s$ is the region yet to be handled, T is a subtree of T'_s corresponding to the region R, and U_s is the set of interior-disjoint faces in s produced by the unrefinement process. While $\text{area}(R) > 1$, do the following:

- Find a lowest node $v \in T$ for which $\text{area}(\widehat{R}_v) > 1$,
- Set $U_s := U_s \cup \{\widehat{R}_v\}$,
- Set $R := R \setminus R_v$,
- Delete the subtree rooted at v from T, and
- For all ancestors u of v, set $\widehat{R}_u := \widehat{R}_u \setminus \widehat{R}_v$.

When the while loop ends, define the weight of the parent edge of s to be $\text{area}(R)$.
Phase 2 of the algorithm positions lasers in a pseudotriangle s as follows.
For every region $\widehat{R}_v \in U_s$, do:

- Step 1. Place lasers along all edges of the boundary between $\widehat{R}_v$ and $s \setminus \widehat{R}_v$, and the boundaries between R_v and $R_{v'}$ for all children v' of v. For example in Fig. 3 (right), two lasers are placed along the edges (a, c) and (c, e) that disconnect $\widehat{R}_v$ from s_2. Also, a laser that is placed along edge (b, d) that separates the children of R_v.
- Step 2. If $\text{area}(R_v) \geq 1$ (which means R_v has not merged with any other region in Phase 1, i.e., $\widehat{R}_v = R_v$ hence $\widehat{R}_v$ is convex), subdivide $\widehat{R}_v$ by $\Theta(\sqrt{\text{area}(R_v)})$ lasers according to Lemma 4.

This completes the description of our algorithm.

▶ **Theorem 5.** *Let P be a simple polygon with n vertices, and let k^* be the minimum number of lasers that subdivide P into pieces of area at most 1. We can find an integer k with $k^* \leq k \leq O(k^* \log n)$ in $O(n)$ time, and a set of k lasers that subdivide P into pieces of area at most 1 in output-sensitive $O(k + n)$ time.*

Proof. Phase 1 of our algorithm (unrefinement) subdivides each pseudotriangle $s \in \mathcal{S}$ into regions such that each region corresponds to a subtree rooted at some node v of the recursion tree T'_s. Node v corresponds to a region $R_v \subset s$, and a possibly larger region $\widehat{R}_v \subset P$ which is the union of R_v and adjacent regions in the descendant pseudotriangles of s adjacent to R_v. Phase 1 of the algorithm ensures that $\text{area}(\widehat{R}_v) > 1$ (therefore, $\widehat{R}_v$ must intersect at least one laser in OPT), but for all children v' of v in T'_s, we have $\text{area}(\widehat{R}_{v'}) \leq 1$.

In Step 1, the algorithm uses $O(1)$ lasers for each $v \in U_s$ to separate $\widehat{R}_v$ from $s \setminus \widehat{R}_v$. Recall that the recursion tree T_s has bounded degree. Consequently, we use $O(1)$ lasers to separate $\widehat{R}_{v'}$ from $\widehat{R}_v \setminus \widehat{R}_{v'}$ for all children v' of v. These polylines subdivide $\widehat{R}_{v'}$ into smaller regions of area at most 1. Overall, we have used $O(1)$ lasers for each of these nodes $v \in T'_s$, $s \in \mathcal{S}$. Note that each region $\widehat{R}_v$ is the union of triangles from the Hershberger–Suri Steiner triangulation, and so each laser in OPT intersects $O(\log n)$ such regions. Consequently, we use $O(k^* \log n)$ lasers in Step 1.

Finally, consider the lasers used in Step 2 for subdividing the triangles $t \in T$ with $\text{area}(t) > 1$. By Lemma 4, each such triangle intersects at least $\Omega(\sqrt{\text{area}(t)})$ lasers in any valid solution; and conversely each laser of an optimal solution intersects $O(\log n)$ regions in T. Consequently, the number of lasers uses in Step 2 is $\sum_{t \in T} O(\sqrt{\text{area}(t)}) \leq O(k^* \log n)$.

It remains to show that the algorithm runs in $O(n + k)$ time. The Hershberger-Suri Steiner triangulation can be computed in $O(n)$ time [17]. It consists of $O(n)$ triangles, hence the combined size of the dual tree T_p, and the recursion trees T_s, $s \in \mathcal{S}$, is also $O(n)$. The unrefinement algorithm is done in a single traversal of these trees, spending $O(1)$ time at each node. For each large cell (triangle) of the Hershberger-Suri triangulation, by Lemma 4, we can compute a minimum bounding box and the *number* of lasers used by the algorithm in $O(1)$ time. Computing all k lasers requires $O(k)$ additional time. ◀

An $O(\log r)$ Approximation for Min-LaserArea in Simple Polygons. We can improve the approximation ratio in Theorem 5 from $O(\log n)$ to $O(\log r)$, where r is the number of reflex vertices of P, if we replace the Hershberger–Suri triangulation with a convex decomposition. (Hershberger and Suri decompose P into *triangles* to support ray shooting queries, but for our purposes a decomposition into convex cells suffices.)

Let $(v_1, \ldots, v_n)$ be the n vertices of P; assume they are in general position. Let R be the set of reflex vertices of P. For every reflex vertex $v \in R$, the angle bisector of the interior angle at v hits some edge $a_v b_v$ of P. Let $L = \{v, a_v, b_v : v \in R\}$, that is, L is the set of all reflex vertices of P, and both endpoints of the edges hit by the angle bisectors of reflex angles. Clearly, $|L| \leq 3r$.

▶ **Lemma 6.** *There is a simple polygon $Q \subset P$ whose vertex set is L, and every connected component of $P \setminus Q$ is a convex polygon. The polygon Q can be computed in $O(n \log n)$ time.*

Proof. We describe an algorithm that decomposes P along noncrossing diagonals into a collection $\mathcal{P}$ of convex polygons and their complement $P \setminus (\bigcup_{A \in \mathcal{P}} A)$ will be polygon Q. Initially $\mathcal{P} = \{P\}$ and $Q = \emptyset$.

The algorithm has two steps. In the first step, a collection $\mathcal{P}$ of convex polygons is created such that the vertex set of the complement $P \setminus (\bigcup_{A \in \mathcal{P}} A)$ is L. However, $P \setminus (\bigcup_{A \in \mathcal{P}} A)$ is not necessarily connected. In the second step, the connected components of $P \setminus (\bigcup_{A \in \mathcal{P}} A)$ are merged into a simple polygon Q (a single connected component) with the same vertex set L.

First step. While there is a nonconvex polygon $P' \in \mathcal{P}$, we replace P' with one or more smaller polygons in $\mathcal{P}$ as follows. Let v be a reflex vertex of P'. Since $P' \subset P$, vertex v is also a reflex vertex of P. Denote by $\vec{r}_v$ the angle bisector of P at v. Note that $\vec{r}_v$ enters the interior of P' at v; denote by ab the edge of P' where $\vec{r}_v$ first exits P'. Let $P(v)$ be the geodesic triangle formed by the edge ab and the shortest paths from v to a and to b, respectively. Update $\mathcal{P}$ by replacing P' with the polygons in $P' \setminus P(v)$. See Figure 4 for an example. In the course of the algorithm, every polygon in $\mathcal{P}$ is formed by a sequence of consecutive vertices of the input polygon P.

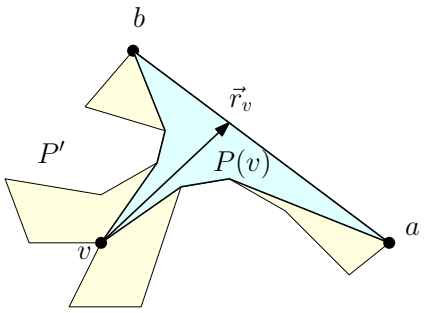


Figure 4 Replace P' by four polygons (in yellow), after taking out the geodesic triangle $P(v)$ where v is a reflex vertex.

We claim that in each iteration of the algorithm, all vertices of $P(v)$ are in L. Clearly, v is a reflex vertex in P', hence a reflex vertex of P, as well. Similarly, the interior vertices of the shortest paths from v to a and to b are reflex vertices in P', hence in P. It remains to show that $a, b \in L$. If ab is an edge of P, then $a, b \in L$ by the definition of L. Otherwise, ab is a diagonal of P, and so it is an edge of a geodesic triangle $P(v')$ of some previous iteration of the algorithm – by induction, they are in L, as well. At the end of the while loop, all polygons in $\mathcal{P}$ are convex, however, the complement $P \setminus (\bigcup_{A \in \mathcal{P}} A)$ is not necessarily connected. See Figure 5 for an illustration.

Second step. While there is a (convex) polygon $P' \in \mathcal{P}$ incident to three or more vertices in L, we replace P' with smaller polygons: In particular, let $V(P')$ be the vertex set of P'. If $|V(P') \cap L| \geq 3$, then replace P' with the polygons in $P' \setminus \text{conv}(V(P') \cap L)$, where conv(.) stands for the convex hull. In each iteration, all polygons in $\mathcal{P}$ remain convex. At the end of the while loop, every polygon in $\mathcal{P}$ is incident to exactly two vertices in L, and $P \setminus (\bigcup_{A \in \mathcal{P}} A)$ is a simple polygon with vertex set L. ◀

▶ **Lemma 7.** *Every simple polygon P on n vertices, r of which are reflex, can be decomposed into convex faces such that every chord of P intersects $O(\log r)$ faces. Such a decomposition can be computed in $O(n \log n)$ time.*

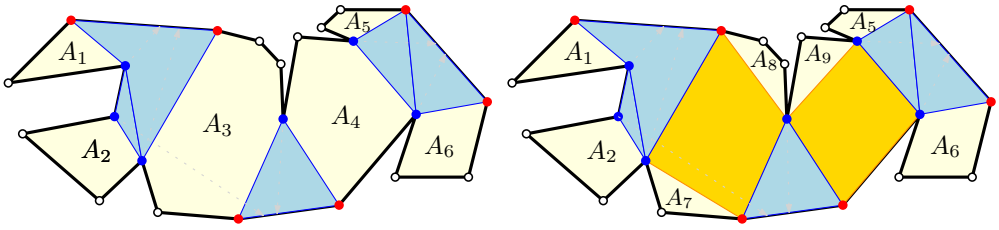


Figure 5 A simple polygon P, the vertices in L are blue (reflex) or red (hit by angle bisector). Left: The first step produces convex polygons $\mathcal{P} = \{A_1, \ldots, A_6\}$, but $P \setminus \bigcup_{i=1}^{6} A_i$ is disconnected. Right: The second step merges $P \setminus \bigcup_{i=1}^{6} A_i$ into a simple polygon Q. As $|A_3 \cap L| \geq 3$ and $|A_4 \cap L| \geq 3$, the second step creates $\text{conv}(V(A_3) \cap L)$ and $\text{conv}(V(A_4) \cap L)$ (shown in deep yellow) which merges $P \setminus \bigcup_{i=1}^{6} A_i$ into a single connected component Q.

Proof. We can compute the set L of up to $3r$ vertices and a simple polygon $Q \subset P$ described in Lemma 6 in $O(n \log n)$ time. We then compute the Hershberger–Suri triangulation for Q, which is a Steiner triangulation of $O(r)$ triangles such that every chord of Q intersects $O(\log r)$ triangles [17]. This triangulation of Q, together with the convex polygons in $P \setminus Q$, form a subdivision of P into convex faces.

We claim that every chord of P intersects at most $O(\log r)$ faces: at most $O(\log r)$ triangles in Q and at most two convex sets in $P \setminus Q$. If a chord ℓ of P intersects three components of $P \setminus Q$, say C_1, C_2, C_3 in this order, then ℓ crosses the boundary of C_2 twice, so C_2 must have at least two edges on the boundary between C_2 and Q. However, by Lemma 6, every edge of Q is either an edge or a diagonal of P. Therefore the boundary between Q and a component of $P \setminus Q$ is a single diagonal of P. Thus ℓ intersects at most two components of $P \setminus Q$; moreover $\ell \cap Q$ is a chord of Q, so it intersects $O(\log r)$ triangles inside Q. ◀

By performing the algorithm on the convex subdivision in Corollary 7, the approximation ratio improves to $O(\log r)$.

▶ **Theorem 8.** *Let P be a simple polygon with n vertices, r of which are reflex, and let k^* be the minimum number of lasers needed to subdivide P into pieces of area at most 1. We can find an integer k with $k^* \leq k \leq O(k^* \log r)$ in $O(n \log n)$ time, and a set of k lasers that subdivide P into pieces of area at most 1 in output-sensitive $O(k + n \log n)$ time.*

3.2 Bi-Criteria Approximation for Diameter

For the diameter version in a simple polygon, we describe a bi-criteria approximation algorithm (Theorem 11). We start from deriving a lower bound for the minimum number of lasers in a decomposition into pieces of diameter at most δ (for bi-criteria approximation algorithm we use general δ, instead of δ =1, because we will scroll over δ when using the algorithm to get an approximation for MinDiameter).

Consider the infinite set of vertical lines, $\mathcal{L}_V$, evenly spaced with separation δ; that is, $\mathcal{L}_V = \{x = i\delta : i \in \mathbb{Z}\}$. The lines in $\mathcal{L}_V$ decompose P into a set $\mathcal{P}_V$ of simple polygons, that we call *cells*. By construction, the orthogonal projection of each cell to the x-axis is an interval of length at most δ. (More precisely, we consider the polygon P to be a closed set in the plane. Subtracting the union of vertical lines $\mathcal{L}_V$ from P results in a set of connected components; the closures of these components are the simple polygons in $\mathcal{P}_V$). The polygons in $\mathcal{P}_V$ are faces in the arrangement of the lines in $\mathcal{L}_V$ and the edges of P; the planar dual of this decomposition is a tree, whose nodes are the faces $\mathcal{P}_V$ and whose edges are dual to the vertical lines.

If the projection of polygon $Q \in \mathcal{P}_V$ onto the x-axis is an interval of length δ (which means it extends from $x = i\delta$ to $x = (i+1)\delta$, for some integer i), we say that Q is a *full-width* cell; otherwise the projection of Q onto the x-axis is of length less than δ, and we say that Q is a *narrow* cell. (It may be that P itself is a narrow cell if, e.g., P does not intersect any of the vertical lines L_V.)

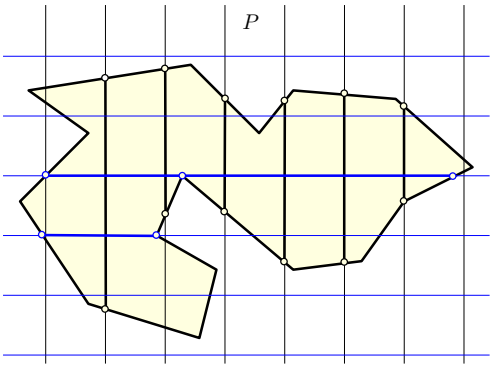


Figure 6 P is subdivided by a grid; the lasers are thick.

The intersection of the lines in $\mathcal{L}_V$ with P is a set of vertical chords of P. Let C_V be the set of these chords. While there is a chord $\ell \in C_V$ that lies on the boundary of some narrow cell, remove ℓ from C_V (thereby merging the cells on the two sides of ℓ into one cell). As a result, all remaining chords lie on the boundary between full-width cells. Let C'_V be the resulting set of chords, and let $k_V = |C'_V|$ denote their cardinality. Since any two full-width cells of $\mathcal{P}_V$ that are in adjacent vertical strips remain separated by a chord in C'_V, the x-extent of each face in the new decomposition of P is at most 3δ. We summarize this below.

▶ **Proposition 9.** *The remaining k_V chords C'_V, $k_V \geq 0$, subdivide P into a set $\mathcal{Q}$ of $k_V + 1$ polygons, each of which intersects at most two lines in $\mathcal{L}_V$, consequently its projection to the x-axis is an interval of length less than 3δ. Further, the dual graph of this decomposition is a tree (with k_V edges and $k_V + 1$ nodes).*

If $k_V = 0$, then there is just one cell, $\mathcal{Q} = \{P\}$. If $k_V \geq 1$, then each $Q \in \mathcal{Q}$ includes at least one full-width cell, since the only lasers remaining are those separating one full-width cell from an adjacent full-width cell sharing the laser.

Thus, the boundary of each $Q \in \mathcal{Q}$ includes at least two distinct (simple) paths connecting a point on one line of $\mathcal{L}_V$ to a point on an adjacent line of $\mathcal{L}_V$. Each of these paths has length at least δ. The endpoints of such a path are at distance at least δ away from each other. In any laser cutting of P into pieces of diameter at most δ, each such path contains a laser endpoint in its interior or at both endpoints (if the path is a horizontal line segment). In any case, each of these paths contains a laser endpoint in its interior or at its left endpoint. Thus overall, there must be at least $2|\mathcal{Q}| = 2(k_V + 1)$ endpoints of lasers. This implies that $k^* \geq k_V + 1$, where k^* is the minimum number of lasers in order to achieve pieces of diameter at most δ. Therefore we conclude,

▶ **Lemma 10.** *If $k_V \geq 1$, then $k^* \geq k_V + 1$.*

Now, we consider the set of horizontal lines, $\mathcal{L}_H = \{y = j\delta : j \in \mathbb{Z}\}$, and apply the above process to polygon P, yielding horizontal chords C_H, and then a subset $C'_H \subseteq C_H$ of chords after merging cells (removing lasers that separate a *full-height* cell from an adjacent *short*

cell). The result is a decomposition of P into $k_H+1 = |C'_H|+1$ pieces, each having projection onto the y-axis of length less than 3δ. Analogously to Lemma 10, we get $k^* \geq k_H + 1$ if $k_H \geq 1$.

If we now overlay the vertical chords C'_V and the horizontal chords C'_H, the resulting arrangement decomposes P into pieces each of which is a simple polygon having projections onto both the x- and the y-axis of lengths less than 3δ; thus, the resulting pieces each have diameter less than $3\delta\sqrt{2}$. The total number of lasers is $k_V + k_H \leq 2(k^* - 1)$.

▶ **Theorem 11.** *Let P be a simple polygon with n vertices, and let k^* be the minimum number of lasers that decompose P into pieces each of diameter at most δ for a fixed $\delta > 0$. One can compute a set of at most $2(k^* - 1)$ axis-aligned lasers that decompose P into pieces each of diameter at most $3\sqrt{2}\delta$ in time polynomial in n and $\mathrm{diam}(P)/\delta$.*

3.3 $O(1)$-Approximation for MinDiameter in Simple Polygons

In this section we consider the problem of minimizing the maximum diameter of a cell in the arrangement of k lasers, for a given number k. Our $O(1)$-approximation algorithm repeatedly decreases the x- and y- separation in the bi-criteria solution from Theorem 11 until the number of placed lasers is about to jump over $2k$; then, the number of lasers is halved while increasing the diameter by a constant factor.

Specifically, let $\ell(\delta)$ denote the number of lasers used in the end of the bi-criteria algorithm with the x- and y-separation between consecutive vertical and horizontal lines being δ. Our algorithm to approximate the diameter achievable with k lasers is as follows:

- Initialize $\delta = \mathrm{diam}(P)$, and $\varepsilon > 0$.
- While $\ell(\delta) \leq 2k$, set $\delta = \delta/(1+\varepsilon)$ and recompute $\ell(\delta)$.
- Let δ_0 be such that $\ell(\delta_0) \leq 2k$ but $\ell(\delta_0/(1+\varepsilon)) > 2k$.
- Let C_V and C_H be the $\ell(\delta_0) \leq 2k$ vertical and horizontal lasers, resp., found by the bi-criteria algorithm.
- Partition C_V into lasers along $x = i\delta_0$ for even i and the rest (odd i); let C'_V be a smallest part. Similarly, let C_H be a smaller part when we partition C_H into two subsets of lines where $y = i\delta_0$ is an even or odd multiple of δ_0.
- Return the lasers in $C'_V \cup C'_H$.

▶ **Theorem 12.** *Let P be a simple polygon with n vertices, and let δ^* be the optimal diameter achievable with k lasers. For every $\varepsilon > 0$, one can compute a set of at most k lasers that partition P into pieces each of diameter at most $4\sqrt{2}(1+\varepsilon)\delta^*$ in time polynomial in n, $\mathrm{diam}(P)/\delta^*$, and ε.*

The proof of Theorem 12 is presented in the full version of this paper.

4 Axis-Parallel Lasers

In this section we study Min-LaserDiameter and Min-LaserArea under the constraint that all lasers must be axis-parallel (the edges of P may have arbitrary orientations). The algorithms for both problems start with a "window partitioning" P into "(pseudo-)histograms" of stabbing number at most three, and are then tuned to the specific measures to partition the histograms. We use a simple sweepline algorithm for the diameter, and a dynamic program for the area. The main result is:

▶ **Theorem 13.** *Let P be a simple polygon with n vertices and let k^* be the minimum number of axis-parallel lasers needed to subdivide P into pieces of area (diameter) at most 1. There is an algorithm that finds $O(k^*)$ axis-parallel lasers that subdivide P into pieces of area (diameter) at most 1 in time polynomial in n and $\mathrm{area}(P)$ ($\mathrm{diam}(P)$).*

4.1 Reduction to Histograms

A *histogram* is a simple polygon bounded by an axis-parallel line segment, the *base*, and an x- or y-monotone polygonal chain between the endpoints of the base.

The *window partition* of a simple polygon was originally used for the design of data structures that support link distance queries [24, 30]. In this section, we use the axis-parallel version, which partitions a simple polygon P into histograms such that every axis-parallel chord of P intersects at most 3 histograms. Window partitions for orthogonal polygons can be computed by a standard recursion [24, 30]; we use a modified version where we recurse until the remaining subpolygons are below the size threshold 1. This modification guarantees termination on all simple polygons (not only orthogonal polygons).

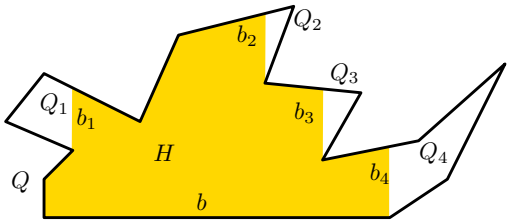


Figure 7 Window partition of a polygon Q with a horizontal base b into a maximal histogram H (colored gold) and four polygons Q_1, Q_2, Q_3, and Q_4 (in white). If the sizes (areas/diameters) of Q_1, Q_2, Q_3, and Q_4 are each at most 1, then Q is a pseudo-histogram.

Window Partition Algorithm. Given a simple polygon P, let b be an axis-parallel chord of P that subdivides P into two simple polygons P_1 and P_2 with a common base b. Let $\mathcal{S} = \{(P_1, b), (P_2, b)\}$ be a set of tuples where each tuple has a polygon and its axis-parallel base, and let $\mathcal{H} = \emptyset$ be the set of histograms. While $\mathcal{S}$ contains a pair (Q, b), where the size (e.g., the diameter) of Q is more than 1, do the following:

1. compute the maximal histogram[1] H of base b in Q, and add (H, b) to $\mathcal{H}$; see Figure 7;
2. update $\mathcal{S}$ by replacing (Q, b) with the pairs (Q_i, b_i), where the polygons Q_i are the connected components of $Q \setminus H$, and b_i is the boundary between Q_i and H.

Return $\mathcal{H}$ and $\mathcal{S}$.

Pseudo-histograms. Let T_1 and T_2 be the recursion trees of the algorithm, rooted at P_1 and P_2, respectively. Let $T = T_1 \cup T_2$. Each node $v \in T$ corresponds to a polygon $Q_v \subset P$. Every nonleaf node $v \in T$ also corresponds to a histogram $H_v \subset Q_v$; it is possible that $\text{size}(H_v) \leq 1$ but $\text{size}(Q_v) > 1$ (the size is area or diam based on the problem). For a leaf $v \in T$, we have either $\text{size}(Q_v) \leq 1$, or $H_v = Q_v$ and $\text{size}(H_v) > 1$. The polygons Q_v at leaf nodes and the histograms H_v at nonleaf nodes jointly form a subdivision of P.

Every node $v \in T$ in the recursion tree corresponds to a polygon-base pair (Q_v, b_v). For any subset $U \subset V(T)$, where $V(T)$ is the set of vertices of T, the bases $\{b_u : u \in U\}$ decompose P into simply connected cells. For every $u \in U$, there is a cell P_u in the decomposition such that $H_u \subset P_u \subset Q_u$. Since every axis-parallel chord of P crosses at most 2 bases, it can intersect at most 3 polygons in such a decomposition.

[1] Without loss of generality, assume b is horizontal. H can be obtained by taking all points of Q reachable through a vertical line from points on b.

In a bottom-up traversal of T, we can find a subset $U \subset V(T)$ such that $\{b_u : u \in U\}$ decomposes P into polygons P_u, $u \in U$, such that $\text{size}(P_u) > 1$ but the size of every component of $P_u \setminus H_u$ is at most 1. Each polygon P_u consists of a histogram H_u with base b_u, and subpolygons (*pockets*) of size at most 1 attached to some edges of H_u orthogonal to b_u. We call each such polygon P_u a *pseudo-histogram*. See Figure 7.

4.2 $O(1)$-Approximation for Min-LaserDiameter in Histograms

We start with an $O(1)$-approximation for histograms, and then extend our algorithm to pseudo-histograms and simple polygons. Without loss of generality, we assume that the base is horizontal.

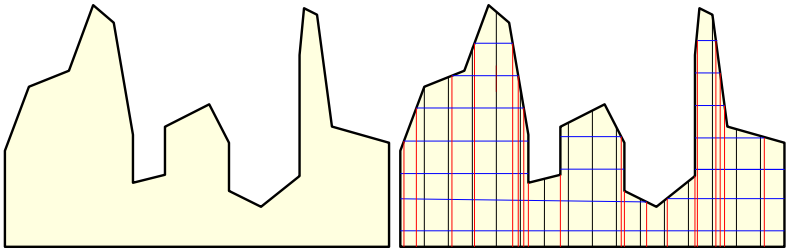


Figure 8 Left: A histogram polygon with a horizontal base. Right: lasers introduced in Phase 1 are shown in black. Horizontal (vertical) lasers introduced in Phase 2 are shown in blue (red).

▶ **Theorem 14.** *There is an algorithm that, given a histogram P with n vertices, computes an $O(1)$-approximation for the axis-parallel Min-LaserDiameter problem in time polynomial in n and $\text{diam}(P)$.*

Proof. We first describe the algorithm.

Algorithm. We are given a histogram P with a horizontal base ab. If $\text{diam}(P) \leq 1$, halt. Otherwise, do the following:

1. Subdivide ab into $\lceil 2|ab| \rceil$ intervals which all, except possibly one, have length $1/2$ and place vertical lasers on the boundary between consecutive intervals.
2. Sweep P with a horizontal line L top down, and maintain the set of cells formed by all lasers in step one and the line L. When the diameter of a cell C above L is precisely 1, place a horizontal laser pq along the bottom side of C, where $p, q \in \partial P$, and place two vertical lasers at p and q, respectively.

Analysis. Let OPT denote the set of lasers in an optimal solution, and let $k^* = |\text{OPT}|$. Denote by ALG the set of lasers computed by the algorithm; let ALG^1 be the number of vertical lasers computed in Phase 1, and let ALG^2_h and ALG^2_v be the set of horizontal and vertical lasers computed in Phase 2. Clearly, $|\text{ALG}^2_v| \leq 2|\text{ALG}^2_h|$. Therefore it is enough to prove that $|\text{ALG}^1| = O(k^*)$ and $|\text{ALG}^2_h| = O(k^*)$.

First we show that $|\text{ALG}^1| = O(k^*)$. The vertical lasers in OPT subdivide the base ab into segments of length at most 1. Therefore, $k^* \geq \lfloor |ab| \rfloor$. Combined with $k^* \geq 1$, this readily implies that $|\text{ALG}^1| = \lceil 2|ab| \rceil - 1 = O(k^*)$.

Next we prove that $|\text{ALG}^2_h| \leq 2k^*$ using a charging scheme. Specifically, we charge every laser in ALG^2_h to a laser in OPT such that each laser in OPT is charged at most twice. Recall for each laser $pq \in \text{ALG}^2_h$ placed by the algorithm, there is a cell $C = C_{pq}$ such that $\text{diam}(C_{pq}) = 1$ and pq contains the bottom edge of C_{pq}. Since $\text{diam}(C_{pq}) = 1$, the cell C_{pq}

intersects some laser in OPT; we charge pq to one of these lasers. Denote by $\mathrm{OPT}_h(C_{pq})$ and $\mathrm{OPT}_v(C_{pq})$, resp., the horizontal and vertical lasers in OPT that intersect C_{pq}. The charging scheme is described by the following rules:

(a) If $\mathrm{OPT}_h(C_{pq}) \neq \emptyset$, then charge pq to the lowest laser in $\mathrm{OPT}_h(C_{pq})$;
(b) else, if C_{pq} intersects ∂P, then charge pq to a laser in $\mathrm{OPT}_v(C_{pq})$ that is closest to $C_{pq} \cap \partial P$;
(c) else, if there is no horizontal laser in OPT that lies above pq, then charge pq to an arbitrary laser in $\mathrm{OPT}_v(C_{pq})$;
(d) else, charge pq to the lowest horizontal laser in OPT that lies above pq.

It remains to prove that each laser in OPT is charged at most once for each the four rules. Since (a) and (d) charge to horizontal lasers, and (b) and (c) charge to vertical lasers in OPT, then each laser in OPT is charged by at most two of the rules. In each case, we argue by contradiction. Assume that a laser $\ell \in \mathrm{OPT}$ is charged twice by one of the rules, that is, there are two lasers $pq, rs \in \mathrm{ALG}_h^2$, that are charged to ℓ by the same rule. The width of cells C_{pq} and C_{rs} is at most $1/2$, because of the spacing of the vertical lasers in ALG^1. Since $\mathrm{diam}(C_{pq}) = \mathrm{diam}(C_{rs}) = 1$, they each have height at least $\sqrt{3}/2$. Without loss of generality, we may assume that the algorithm chooses pq before rs.

(a) In this case, ℓ is the lowest horizontal laser in OPT that intersect C_{pq} and C_{rs}, respectively. Since pq is above rs, laser pq intersects the interior of C_{rs}, contradicting the assumption that C_{rs} is a cell formed by the arrangement of all lasers in ALG.
(b) In this case, ℓ is a vertical laser that intersects both C_{pq} and C_{rs}, and also intersect ∂P. When the algorithm places a horizontal laser at pq, it also places vertical lasers from p and q to the base of P. These three lasers separate ∂P from the portion of ℓ below pq. This contradict the assumption that C_{rs} is a cell formed by the arrangement of all lasers in ALG.
(c) In this case, both C_{pq} and C_{rs} intersects a vertical laser $\ell \in \mathrm{OPT}$, and they both lie above the highest horizontal laser that crosses ℓ. Consequently, they both intersect the two highest cells, say C_{left} and C_{right}, on the two sides of ℓ in the arrangement formed by OPT. The combined height of C_{pq} and C_{rs} is at least $\sqrt{3}$. Therefore, the height of C_{left} and C_{right} is at least $\sqrt{3} > 1$, contradicting the assumption that $\mathrm{diam}(C_{\mathrm{left}}) \leq 1$ and $\mathrm{diam}(C_{\mathrm{right}}) \leq 1$.
(d) In this case, ℓ is the lowest horizontal laser in OPT that lies above C_{pq} and C_{rs}, respectively. Let C_{below} be the cell of the arrangement of OPT that lies below ℓ. The combined height of C_{pq} and C_{rs} is at least $\sqrt{3}$. Therefore, the height of C_{below} is at least $\sqrt{3} > 1$, contradicting the assumption that $\mathrm{diam}(C_{\mathrm{below}}) \leq 1$. ◀

Adaptation to Pseudo-Histograms. In a laser cutting of P into pieces of diameter at most 1, each pseudo-histogram P_u intersects a laser, since $\mathrm{diam}(P_u) > 1$. An adaptation of the algorithm in Section 4.2 can find an $O(1)$-approximation for Min-LaserDiameter in each P_u. As noted above, each laser intersect at most 3 pseudo-histograms, hence the union of lasers in the solutions for pseudo-histograms is an $O(1)$-approximations for P.

The sweepline algorithm in Section 4.2 can easily be adapted to subdivide a pseudo-histogram P_u. Recall that P_u consists of a histogram H_u and pockets of diameter at most 1. We run steps 1 and 2 of the algorithm for the histogram H_u with two minor changes in step 2: (1) we compute the *critical* diameters with respect to P_u (rather than H_u), and (2) when the diameter of a cell C above a chord pq of P_u is precisely 1, we place up to *four* vertical lasers: at intersection points of L with ∂P_u the ∂H_u (the vertical lasers through $pq \cap \partial H_u$ cut possible pockets that intersect pq). The analysis of the sweepline algorithm is analogous to Section 4.2, and yields the following result.

▶ **Theorem 15.** *There is an algorithm that, given a simple polygon P with n vertices,computes an $O(1)$-approximation for the axis-parallel* Min-LaserDiameter *problem in time polynomial in n and* $\mathrm{diam}(P)$.

4.3 Discretization of the Solution Space in a Histogram Polygon

Consider a histogram polygon P having n vertices. We assume that the vertices are in general position, in the sense that no three vertices are collinear. We show that there is a discrete set of candidate orthogonal chords such that lasers chosen from this set yield an $O(1)$ approximation for minimizing the number of lasers, subject to a target measure bound 1 on the obtained pieces. This is useful for finding an $O(1)$ approximation for Min-LaserArea by dynamic programming.

We prove the following lemma in the full version, which extends to pseudo-histograms.

▶ **Lemma 16.** *For a histogram P, let k^* be the minimum number of axis-parallel lasers that subdivide P into pieces of area (diameter) at most 1. We can find a set C of $O(n + \mathrm{area}(P))$ ($O(n + per(P))$) chords of P, such that $O(k^*)$ lasers from C can subdivide P into pieces of area (diameter) at most 1.*

4.4 $O(1)$-Approximation for Min-LaserArea

We now consider the Min-LaserArea with axis-parallel lasers chosen from a discrete set to achieve pieces of area at most 1. An $O(1)$-approximation algorithm is based on the window partition method described earlier, allowing us to reduce to the case of subdividing a pseudo-histogram, for which we give a dynamic program to choose lasers from the discrete candidate set. In the full version, we describe the algorithm using area as the measure. With slight modifications, the algorithm also applies to the measure of diameter, allowing us to solve Min-LaserDiameter in pseudo-histograms (albeit in a higher polynomial time bound than stated in Theorem 14).

5 Diameter Measure in Polygons with Holes and Axis-Parallel Lasers

5.1 Bi-Criteria Approximation for Diameter

In this section we give a bi-criteria approximation for the diameter version in a polygon with holes when lasers are constrained to be axis-parallel. The approach is similar to the algorithm for simple polygons and lasers of arbitrary orientations (cf. Section 3.2) in that both use grid lines, but they differ significantly to handle holes in a polygon when the lasers are axis-parallel. Particularly, in simple polygons we place lasers along grid lines, while in polygons with holes the grid lines just divide the problem into sub-problems.

Lasers in Vertical Strips. Consider the infinite set of equally spaced vertical lines $\mathcal{L}_V = \{x = i\delta : i \in \mathbb{Z}\}$, for some $\delta > 0$. The lines subdivide P into a set $\mathcal{P}_V$ of polygons (possibly with holes), that we call *strips*. (Unlike Section 3.2, we do not place lasers along the lines in $\mathcal{L}_V$; we use the strips for a divide-and-conquer strategy.) The projection of any strip on the x-axis has length at most δ; we say that a strip is *full-width* if its projection has length exactly δ. Let $\mathcal{F}_V \subset \mathcal{P}_V$ denote the set of full-width strips, and let $F \in \mathcal{F}_V$ be a full-width strip.

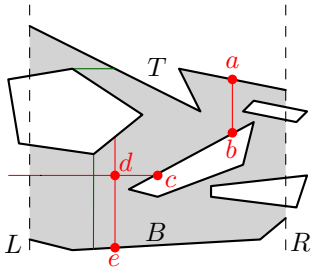


Figure 9 F is shaded, the holes are white. The L-R separating path $\gamma = abcde$ (vertices marked with red disks) alternates between holes and lasers (red) in the interior of P; $a_F(\gamma) = 2$ as there are two links ab and de in path γ whose extensions are fully contained in F. ab and cde are the maximal rectilinear subpaths of γ through the free space. The minimum-link path $\gamma(F)$ (darkgreen) also alternates between holes and free space.

The leftmost (resp., rightmost) points of F lie on a line $L = \{x = i\delta\}$ (resp., a line $R = \{x = (i+1)\delta\}$) for some $i \in \mathbb{Z}$ (see Fig. 9). Consequently, the outer boundary of F contains two simple paths between L and R; we denote them by T (top) and B (bottom).

Since the distance between L and R is δ, in any laser cutting of P into cells of diameter at most δ, there exists a T-B path $\gamma \subset F$ along the boundaries of cells that separates L and R. Since γ is disjoint from the interior of the cells, it must follow lasers in the interior of P. We may assume, w.l.o.g., that γ follows any laser at most once; otherwise we could shortcut γ along the laser. Since the lasers are axis-aligned, γ is an alternating sequence of subpaths that are either in ∂P or rectilinear paths through the interior of F; we call any such T-B path an *alternating* path.

An axis-aligned segment s, fully contained in F, is *associated* with F if it remains fully contained in F after it is maximally extended within P (i.e., if both endpoints of the supporting chord of P are on the boundary of F). For example, any *vertical* segment $s \subset F$ is associated with F (because T and B belong to the boundary of F). Let $a_F(\gamma)$ be the number of associated links of γ (i.e., the number of edges whose supporting chords are fully contained in F). Let $|\gamma|$ denote the total number of the (axis-aligned) edges in γ. A key observation is the following.

▶ **Lemma 17.** $|\gamma| \leq 3\, a_F(\gamma)$.

Proof. Let π be a (maximal) rectilinear subpath of γ through the free space, i.e., a part of γ whose endpoints lie on the boundary of P. If π is a single horizontal link, then the link is associated with F (because if any of its two ednpoints is outside F, then γ protrudes through L or R, not separating them). Otherwise (i.e., if π contains vertical links), the number of the vertical links is at least $1/3$ of the total number of links in π. The lemma follows by summation over all subpaths of γ. ◀

Our algorithm computes an alternating path $\gamma(F)$ with the minimum number of links and places one laser along every link of $\gamma(F)$ (the horizontal lasers may extend beyond F). To find $\gamma(F)$, we can build the *critical graph* of F, whose vertices are T, B, and components of ∂P within the strip F (including holes in the strip), and in which the weight of the edge between two vertices is the axis-parallel link distance between them. The weight of an edge between vertices i and j can be found by in polynomial time by standard wave propagation techniques [10, 26], i.e., by successively labeling the areas reachable with k links from i for increasing k, until j is hit by the wave. After the critical graph is built, $\gamma(F)$ is found as the shortest T-B path in the graph.

By minimality of $\gamma(F)$, the number links $|\gamma(F)|$ in it (and hence the number of lasers we place) is at most $|\gamma(F)|$. Let $k_V = \sum_{F \in \mathcal{F}_V} |\gamma(F)|$ be the total number of lasers placed in all full-width strips in $\mathcal{F}_V$, and let k^* be the minimum number of axis-parallel lasers in a laser cutting of P into cells of diameter at most δ. An immediate consequence of Lemma 17 is the following.

▶ **Corollary 18.** $k_V \leq 3k^*$.

Proof. As the links of $\gamma(F)$ follow lasers, at least $a_F(\gamma)$ lasers are fully contained in F. ◀

The k_V lasers placed in full-width strips subdivide P into polygonal pieces; let Q be one such piece.

▶ **Lemma 19.** *The length of the x-projection of Q on the x-axis is at most 2δ.*

Proof. We prove that Q intersects at most one line in $\mathcal{L}_V$. Suppose, to the contrary, that Q intersects two consecutive lines $\ell_1 : x = i\delta$ and $\ell_2 = x = (i+1)\delta$. Let λ be a shortest path in Q between points in $Q \cap \ell_1$ and $Q \cap \ell_2$, respectively. By minimality, λ lies in the strip between ℓ_1 and ℓ_2. Consequently, λ is contained in some full-width strip $F \subset \mathcal{F}_V$. However, the path $\gamma(F)$ intersects every path in F between $F \cap \ell_1$ and $F \cap \ell_2$; in particular, it intersects λ. Since we have placed a laser along every segment of $\gamma(F)$ in the interior of P, λ intersects a laser, contradicting the assumption that $\lambda \subset Q$. ◀

Lasers in Horizontal Strips. Similarly, we consider the set of horizontal lines $\mathcal{L}_H = \{y = j\delta : j \in \mathbb{Z}\}$ and apply the above process to P, yielding horizontal chords C_H that subdivide the polygon into horizontal strips (polygons, possibly with holes). We again work only with *full-height* strips, whose boundary intersect two consecutive lines in $\mathcal{L}_H$. In each full-height strip, we find a minimum-interior-link rectilinear path that separates the boundary points along the two lines in $\mathcal{L}_H$, and place lasers along the links of the path. Let k_H be the number of lasers over all full-height strips.

Putting Everything Together. We overlay the k_V lasers in full-width strips with the k_H lasers in full-height strips. The resulting arrangement partitions P into polygonal pieces (possibly with holes). The x- and y-projection of each piece has length at most 2δ by Lemma 19; thus, each piece has diameter less than $2\delta\sqrt{2}$. By Corollary 18, the total number of lasers used in the arrangement is $k_V + k_H \leq 6k^*$. We obtain the following theorem.

▶ **Theorem 20.** *Let P be a polygon with holes of diameter* $\operatorname{diam}(P)$ *having n vertices, and let k^* be the minimum number of laser cuts that partition P into pieces each of diameter at most δ for a fixed $\delta > 0$. In time polynomial in n and* $\operatorname{diam}(P)/\delta$*, one can compute a set of at most $6k^*$ lasers that subdivide P into pieces each of diameter at most $2^{3/2}\delta$.*

5.2 $O(1)$-Approximation to MinDiameter

Similarly to Section 3.3, we can use the bi-criteria algorithm to derive a constant-factor approximation for minimizing the maximum diameter of a cell in the arrangement of a given number k of axis-parallel lasers. Our $O(1)$-approximation algorithm (Theorem 21, proof in the full version) repeatedly decreases the x- and y- separation in the bi-criteria solution from Theorem 20 until the number of placed lasers is about to jump over $6k$; then, the number of lasers is decreased by a factor of 6 while increasing the diameter by a constant factor.

▶ **Theorem 21.** *Let δ^* be the minimum diameter achievable with k axis-parallel lasers. For every $\varepsilon > 0$, one can compute a set of at most k axis-parallel lasers that partition P into pieces each of diameter at most $12\sqrt{2}(1+\varepsilon)\delta^*$ in time polynomial in n,* $\operatorname{diam}(P)/\delta^*$*, and ε.*

6 $O(\log \mathrm{OPT})$-approximation for Min-LaserCircle

This section considers the radius of the largest inscribed circle as the measure of cell size; in particular, in Min-LaserCircle the goal is to split the polygon P (which may have holes) into pieces so that no piece contains a disk of radius 1. We give an $O(\log \mathrm{OPT})$-approximation algorithm for Min-LaserCircle based on reducing the problem to SetCover. The following reformulation is crucial for the approximation algorithm:

▶ **Observation 22.** *A set of lasers splits P into pieces of in-circle radius at most 1 if and only if every unit disk that lies inside P is hit by a laser.*

▶ **Theorem 23.** *For a polygon P with n vertices (possibly with holes), Min-LaserCircle admits an $O(\log \mathrm{OPT})$-approximation in time polynomial in n and $\mathrm{area}(P)$.*

Proof. We lay out a regular square grid of points at spacing of $\sqrt{2}$. The set of grid points within P is denoted by G. We may assume $|G| = O(\mathrm{area}(P))$ by a suitable (e.g., uniformly random) shift. Due to the spacing, every unit-radius disk in P contains a point of G (possibly on its boundary).

Consider an optimal set L^* of lasers that hit all unit disks that are contained within P. Replace each laser (chord) $c \in L^*$ with up to four anchored chords of the same homotopy type as c with respect to the vertices of P and the points G, obtained as follows: Shift the chord c vertically down (up), while keeping its endpoints on the same pair of edges of P, until it becomes incident to a point in G or a vertex of P, then rotate the chord clockwise (counterclockwise) around this point until it becomes incident to another point in G or a vertex of P. Since every unit disk within P contains a point of G, any unit disk within P that is intersected by c is also intersected by one of the shifted and rotated copies of c. This means that we can construct a candidate set, C, of $O((n + \mathrm{area}(P))^2)$ chords that can serve as lasers in an approximate solution, giving up at most a factor 4 of optimal. Further, in the arrangement of the segments C within P, any unit disk is intersected by some set of chords of C, thereby defining a combinatorial type for each unit disk in P. (Two disks are of the same type if they are intersected by the same subset of chords in C; one way to define the type is to associate it with a cell in the arrangement of lines drawn parallel to each chord $c \in C$ at distance 2 from c on each side of c. While the center of the disk is in one cell of the arrangement, the disk intersects the same chords.) Let D be the polynomial-size ($O(|C|^2)$) set of disks, one "pinned" (by two segments, from the set C and the set of edges of P) disk per combinatorial type. By construction, any set of chords from C that meets all disks of D must meet all unit disks within P.

We thus formulate a discrete set cover instance in which the "sets" correspond to the candidate set C of chords, and the "elements" being covered are the disks D. Since there are constant-size sets of disks that cannot be shattered, the VC dimension of the set system is constant, and an $O(\log \mathrm{OPT})$-approximate solution for the set cover can be found in time polynomial in the size of the instance [7]. ◀

The same algorithm works for the version in which the lasers are restricted to be axis-aligned (the only change is that the candidate set C consists of axis-aligned chords through points of the grid G and vertices of P).

References

1 Bogdan Armaselu and Ovidiu Daescu. Algorithms for fair partitioning of convex polygons. *Theoretical Computer Science*, 607:351–362, 2015. doi:10.1016/j.tcs.2015.08.003.

2 Imre Bárány, Pavle Blagojević, and András Szűcs. Equipartitioning by a convex 3-fan. *Advances in Mathematics*, 223(2):579–593, 2010. doi:10.1016/j.aim.2009.08.016.

3 Aandrás Bezdek and Károly Bezdek. A solution of Conway's fried potato problem. *Bulletin of the London Mathematical Society*, 27(5):492–496, 1995. doi:10.1112/blms/27.5.492.

4 Pavle V. M. Blagojević and Günter M. Ziegler. Convex equipartitions via equivariant obstruction theory. *Israel Journal of Mathematics*, 200(1):49–77, 2014. doi:10.1007/s11856-014-1006-6.

5 Karol Borsuk. Drei Sätze über die n-dimensionale euklidische Sphäre. *Fundamenta Mathematicae*, 20:177–190, 1933. doi:10.4064/fm-20-1-177-190.

6 Prosenjit Bose, Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, and Anil Maheshwari. Polygon cutting: Revisited. In *Proc. Japanese Conference on Discrete and Computational Geometry (JCDCG)*, volume 1763 of *LNCS*, pages 81–92. Springer, 1998. doi:10.1007/978-3-540-46515-7_7.

7 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete and Computational Geometry*, 14(4):463–479, 1995. doi:10.1007/BF02570718.

8 Bernard Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 339–349, 1982. doi:10.1109/SFCS.1982.58.

9 Hallard T. Croft, Kenneth J. Falconer, and Richard K. Guy. *Unsolved Problems in Geometry*. Springer-Verlag, New York, 1991. doi:10.1007/978-1-4612-0963-8.

10 Gautam Das and Giri Narasimhan. Geometric searching and link distance. In *Proc. 2nd Workshop on Algorithms and Data Structures (WADS)*, volume 519 of *LNCS*, pages 261–272. Springer, 1991. doi:10.1007/BFb0028268.

11 Robert Freimer, Joseph S. B. Mitchell, and Christine Piatko. On the complexity of shattering using arrangements. Technical report, Cornell University, 1991.

12 Unnikrishnan Gopinathan, David J. Brady, and Nikos Pitsianis. Coded apertures for efficient pyroelectric motion tracking. *Opt. Express*, 11(18):2142–2152, 2003. doi:10.1364/OE.11.002142.

13 Roser Guàrdia and Ferran Hurtado. On the equipartition of plane convex bodies and convex polygons. *Journal of Geometry*, 83(1):32–45, 2005. doi:10.1007/s00022-005-0006-0.

14 Steven C. Gustafson. Intensity correlation techniques for passive optical device detection. In *Infrared Technology for Target Detection and Classification*, volume 302, pages 66–70. International Society for Optics and Photonics, SPIE, 1982. doi:10.1117/12.932632.

15 Refael Hassin and Nimrod Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30(1):29–42, 1991. doi:10.1016/0166-218X(91)90011-K.

16 Tian He, Qiuhua Cao, Liqian Luo, Ting Yan, Lin Gu, John Stankovic, and Tarek Abdelzaher. Electronic tripwires for power-efficient surveillance and target classification. In *Proc. 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004)*. ACM Press, 2004. doi:10.1145/1031495.1031558.

17 John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18(3):403–431, 1995. doi:10.1006/jagm.1995.1017.

18 Thomas Jenrich and Andries E. Brouwer. A 64-dimensional counterexample to Borsuk's Conjecture. *Electr. J. Comb.*, 21(4):P4.29, 2014. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v21i4p29.

19 Jeff Kahn and Gil Kalai. A counterexample to Borsuk's conjecture. *Bull. Amer. Math. Soc.*, 29:60–62, 1993. doi:10.1090/S0273-0979-1993-00398-7.

20 Roman Karasev, Alfredo Hubard, and Boris Aronov. Convex equipartitions: the spicy chicken theorem. *Geometriae Dedicata*, 170(1):263–279, 2014. doi:10.1007/s10711-013-9879-5.

21 J. Mark Keil. Polygon decomposition. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 491–518. North Holland/Elsevier, 2000. doi:10.1016/b978-044482537-7/50012-7.

22 Irina Kostitsyna. *Balanced partitioning of polygonal domains*. PhD thesis, Stony Brook University, Stony Brook, NY, 2015.

23 Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005. doi:10.1007/s00454-004-1108-4.

24 Anil Maheshwari, Jörg-Rüdiger Sack, and Hristo N. Djidjev. Link distance problems. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, chapter 12, pages 519–558. North-Holland, 2000. doi:10.1016/b978-044482537-7/50013-9.

25 Nimrod Megiddo and Arie Tamir. On the complexity of locating linear facilities in the plane. *Operations Research Letters*, 1(5):194–197, 1982. doi:10.1016/0167-6377(82)90039-6.

26 Joseph S.B. Mitchell, Valentin Polishchuk, and Mikko Sysikaski. Minimum-link paths revisited. *Computational Geometry*, 47(6):651–667, 2014. doi:10.1016/j.comgeo.2013.12.005.

27 R. Nandakumar and N. Ramana Rao. Fair partitions of polygons: An elementary introduction. *Proceedings-Mathematical Sciences*, 122(3):459–467, 2012. doi:10.1007/s12044-012-0076-5.

28 Jack Robertson and William Webb. *Cake-cutting algorithms: Be fair if you can.* AK Peters/CRC Press, 1998.

29 Pablo Soberón. Balanced convex partitions of measures in $\mathbb{R}^d$. *Mathematika*, 58(1):71–76, 2012. doi:10.1112/S0025579311001914.

30 Subhash Suri. On some link distance problems in a simple polygon. *IEEE Trans. Robotics and Automation*, 6(1):108–113, 1990. doi:10.1109/70.88124.

31 Samuel Zahnd, Patrick Lichisteiner, and Tobi Delbruck. Integrated vision sensor for detecting boundary crossings. In *2003 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, 2003. doi:10.1109/ISCAS.2003.1205986.

32 Yunhui Zheng, David J. Brady, and Pankaj K. Agarwal. Localization using boundary sensors: An analysis based on graph theory. *ACM Trans. Sen. Netw.*, 3(4), 2007. doi:10.1145/1281492.1281496.

Set Cover with Delay – Clairvoyance Is Not Required

Yossi Azar
Tel Aviv University, Israel
azar@tau.ac.il

Ashish Chiplunkar
Indian Institute of Technology Delhi, India
ashishc@iitd.ac.in

Shay Kutten
Technion – Israel Institute of Technology, Haifa, Israel
kutten@technion.ac.il

Noam Touitou
Tel Aviv University
noamtouitou@mail.tau.ac.il

Abstract

In most online problems with delay, clairvoyance (i.e. knowing the future delay of a request upon its arrival) is required for polylogarithmic competitiveness. In this paper, we show that this is not the case for set cover with delay (SCD) – specifically, we present the first non-clairvoyant algorithm, which is $O(\log n \log m)$-competitive, where n is the number of elements and m is the number of sets. This matches the best known result for the classic online set cover (a special case of non-clairvoyant SCD). Moreover, clairvoyance does not allow for significant improvement – we present lower bounds of $\Omega(\sqrt{\log n})$ and $\Omega(\sqrt{\log m})$ for SCD which apply for the clairvoyant case.

In addition, the competitiveness of our algorithm does not depend on the number of requests. Such a guarantee on the size of the universe alone was not previously known even for the clairvoyant case – the only previously-known algorithm (due to Carrasco et al.) is clairvoyant, with competitiveness that grows with the number of requests.

For the special case of vertex cover with delay, we show a simpler, deterministic algorithm which is 3-competitive (and also non-clairvoyant).

2012 ACM Subject Classification Theory of computation; Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms

Keywords and phrases Set Cover, Delay, Clairvoyant

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.8

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.08543`.

Funding *Yossi Azar*: Supported in part by the Israel Science Foundation (grant No. 1506/16).
Ashish Chiplunkar: Supported by Pankaj Gupta Young Faculty fellowship.
Shay Kutten: Supported by the Ministry of Science and Technology together with the JSPS, as well as the BSF.

1 Introduction

In problems with delay, requests are released over a timeline. The algorithm must serve these requests by performing some action, which incurs a cost. While a request is pending (i.e. has been released but not yet served), the request accumulates delay cost. The goal of the algorithm is to minimize the sum of costs incurred in serving requests and the delay costs of requests.

There are two variants of such problems. In the clairvoyant variant, the delay function of a request (which determines the delay accumulation of that request over time) is revealed to the algorithm upon the release of the request. In the non-clairvoyant variant, at any point in time the algorithm is only aware of delay accumulated up to that point.

Most online problems with delay do not admit competitive non-clairvoyant algorithms – namely, there exist lower bounds for competitiveness which are polynomial in the size of the input space (e.g. the number of points in the metric space upon which requests are released). This is the case, for example, in the multilevel aggregation problem [10, 14], the facility location problem [7] and the service with delay problem [6]. However, these problems do admit *clairvoyant* algorithms which are polylog-competitive. An additional such problem is that of matching with delay (presented in [20]), for which the only known algorithms are for when all requests have an identical, linear delay function (and are in particular clairvoyant). Rather surprisingly, we show in this paper that the online set cover with delay problem *does* admit a competitive non-clairvoyant algorithm.

In the online set cover with delay problem (SCD), a universe of elements and a family of sets are known in advance. Requests then arrive over time on the elements, and accumulate delay cost until served by the algorithm. The algorithm may choose to buy a set at any time, at a cost specific to that set (and known in advance to the algorithm). Buying a set serves all pending requests (requests released but not yet served) on elements of that set; future requests on those elements, that have yet to arrive when the set is bought, must be served separately at a future point in time. For that reason, a set may be bought an unbounded number of times over the course of the algorithm. The goal of an algorithm is to minimize the sum of the total buying cost and the total delay cost. We note that one could also consider the problem in which sets are bought permanently, and cover future requests; however, it is easy to see that this problem is equivalent to the classic online set cover, and is thus of no additional interest. In the full version of this paper, we show that this problem is a special case of our problem.

As a variant of set cover, the SCD problem is very general, capturing many problems. Nevertheless, we give two possible motivations for the problem.

Summoning experts. consider a company which occasionally requires the help of experts. At any time, a problem may arise which requires external assistance in some field, and negatively impacts the performance of the company while unresolved. At any time, the company may hire any one of a set of experts to come to the company, solve all standing problems in that expert's fields of expertise, and then leave. The company aims to minimize the total cost of hiring experts, as well as the negative impact of unresolved problems.

Cluster-covering with delay. suppose antennas generate data requests over time, which must be satisfied by an external server, with a cost to leaving a request pending. To satisfy an update request by an antenna, the server sends the data to a center antenna which transmits it at a certain radius, at a certain cost (which depends on the center antenna and the radius). All requests on antennas inside that radius are served by that transmission. This problem is a covering problem with delay costs, which can be described in terms of SCD. As an SCD instance, the elements are the antennas, and the sets are pairs of a center antenna and a (reasonable) transmission radius (the number of sets is quadratic in the number of antennas).

Carrasco et al. [15] provided a clairvoyant algorithm for the SCD problem, which is $O(\log N)$ competitive (where N is the number of the requests). However, as the number of requests becomes large, the competitive ratio of this algorithm tends to infinity – even for a

very small universe of elements and sets. Thus, this algorithm does not provide a guarantee in terms of the underlying input space, as we would like. In addition, their algorithm has exponential running time (through making oracle calls which compute optimal solutions for NP-hard problems).

In this paper, we present the first algorithm for SCD which is polylog-competitive in the size of the universe, which is also the first algorithm for the problem which runs in polynomial time. Surprisingly, this algorithm is also non-clairvoyant, showing that the SCD problem admits non-clairvoyant competitive algorithms. Our randomized algorithm is $O(\log n \log m)$-competitive, where n is the number of elements and m is the number of sets. In this paper, we show a reduction from the classic online set cover to SCD, which implies (due to [27]) that our upper bound is tight for a polynomial-time, non-clairvoyant algorithm for SCD.

While our algorithm is optimal for the non-clairvoyant setting, one could wonder if there exists a *clairvoyant* algorithm which performs significantly better – especially considering the aforementioned problems, in which the gap between the clairvoyant and non-clairvoyant cases is huge. We answer this in the negative – namely, we show lower bounds of $\Omega(\sqrt{\log n})$ and $\Omega(\sqrt{\log m})$ on the competitiveness of any randomized clairvoyant algorithm, showing that there is no large gap which clairvoyant algorithms could bridge. Nevertheless, a quartic gap still exists, e.g. in the case that $m = \Theta(n)$. We conjecture that the gap is in fact quadratic, and leave this as an open problem.

In this paper, we also consider the problem of vertex cover with delay (denoted VCD). In the VCD problem, vertices of graph are given, with a buying cost associated with each vertex. Requests on the edges of the graph arrive over time, and accumulate delay until served by buying a vertex touching the edge (at the cost of that vertex's price). This problem corresponds to SCD where every element is in exactly two sets.

1.1 Our Results

We denote as before the number of elements in an SCD instance by n, and the number of sets by m. We also define $k \leq m$ to be the maximum number of sets to which a specific element may belong. We consider arbitrary (nondecreasing) continuous delay functions (not only linear functions).

In this paper, we present:

1. An $O(\log k \cdot \log n)$-competitive, randomized, non-clairvoyant algorithm for SCD, based on rounding of a newly-designed $O(\log k)$-competitive algorithm for the fractional version of SCD. The competitive ratio of this algorithm is tight – we show a reduction from (classic) online set cover to non-clairvoyant SCD.
2. Lower bounds of $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ on competitiveness for **clairvoyant** SCD, showing that clairvoyance cannot improve competitiveness beyond a quadratic factor.
3. A simple, deterministic, non-clairvoyant algorithm for vertex cover with delay (VCD) which is 3-competitive.

Our randomized algorithm for SCD is the first (sub-polynomial competitive) non-clairvoyant algorithm for this problem. Moreover, this is the first algorithm which is polylog-competitive in the size of the universe (even among clairvoyant algorithms).

In the process of obtaining our $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ lower bounds, we in fact obtain an $\Omega(\sqrt{\log m})$ lower bound (which immediately implies $\Omega(\sqrt{\log k})$ since $k \leq m$). The lower bounds also apply for the unweighted setting. These lower bounds improve over the lower bound of $\Omega(\log \log n)$ given in [15][1].

For VCD, while our algorithm is 3-competitive, note that there is a lower bound of 2. The lower bound uses a graph with a single edge which is requested multiple times; this graph corresponds to the TCP acknowledgment problem, analyzed in [19].

▶ Remark 1. While our $O(\log k \cdot \log n)$-competitive algorithm is presented for the case in which the sets and elements are known in advance, it can easily be modified for the case in which each element, as well as which of the sets contain it, becomes known to the algorithm only after the arrival of a request on that element. Moreover, the algorithm can in fact operate in the original setting of Carrasco et al. [15], as it does not need to know the family of sets itself, but rather the family of *restrictions* of the sets to the elements that have already arrived. This can be done through standard doubling techniques applied to $\log n$ and $\log k$ (i.e. squaring of n and k).

1.2 Our Techniques

In the course of designing a non-clairvoyant algorithm for the SCD problem, we also consider a fractional version of SCD. In this version, an algorithm may choose to buy a fraction of a set at any moment. Buying a fraction of a set partially serves requests present on an element of that set, which causes them to accumulate less future delay. As with the original version, a request is only served by fractions bought after its arrival. Hence, the sum of fractions bought for a single set over time is unbounded (i.e. a set may be bought many times).

In the **fractional $O(\log k)$-competitive algorithm**, each request that can be served by a set contributes some amount to the buying of that set. This amount depends exponentially on the delay accumulated by that request, as well as the delay of previous requests. Typically in algorithms with exponential contributions, these contributions are summed. Interestingly, our algorithm instead chooses the maximum of the contributions of the requests as the buying function of the set. The choice of maximum over sum is crucial to the proof (using sum instead of maximum would lead to a linear competitive ratio).

The analysis of this algorithm is based on dual fitting: we first present a linear programming representation of the fractional SCD problem, then use a feasible solution to the dual problem to charge the delay of the algorithm to the optimum. This is the reason for using the maximum in the buying function; each quantity satisfies a different constraint in the dual, and choosing the maximum satisfies all constraints. We then charge the buying cost of the algorithm to $O(\log k)$ times its delay, which concludes the analysis.

Next, we design a randomized competitive algorithm for the integer version of SCD using *2-level* randomized rounding of the fractional algorithm. At the *top level*, we construct a **randomized $O(\log k \cdot \log N)$-competitive algorithm for the integer version**, with N the number of requests. The top-level rounding consists of maintaining for each set a random threshold, and buying the set when the total buying of that set in the *fractional* algorithm exceeds the threshold. In addition, special service of a request is performed in the probabilistically unlikely event that the request is half-served in the fractional algorithm

[1] The lower bound of [15] shows $\Omega(\log N)$-competitiveness, but relies on a universe which is exponentially larger than the number of requests. As they mention in their paper, this therefore translates to an $\Omega(\log \log n)$ lower bound on competitiveness.

but is still pending in the rounding. Since in our problem we may buy a set an unbounded number of times, we require use of multiple subsequent thresholds. To analyze this, we make use of Wald's equation for stopping time.

We add the *bottom level* to improve the $O(\log k \cdot \log N)$-competitive algorithm to a **randomized $O(\log k \cdot \log n)$-competitive algorithm for the integer version**. The bottom level partitions time into phases for each element separately, and aggregates requests on that element that are released in the same phase. The competitive ratio of the resulting algorithm is asymptotically optimal for solving non-clairvoyant SCD in polynomial time, as shown by the reduction from the classic online set cover to non-clairvoyant SCD given in the full version of this paper.

Perhaps the most novel techniques in this paper are used for **the lower bounds of $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$** for the clairvoyant case. The lower bounds are obtained by a recursive construction. Given a recursive instance for which any algorithm has a lower bound on the competitive ratio, we amplify that bound by duplicating every set in the recursive instance into two sets, one slightly more expensive than the other. Both sets perform the same function with respect to the recursive instance, but the algorithm also has an incentive to choose the expensive family of sets, since they serve some additional requests. If the algorithm chooses to buy a lot of expensive sets, the optimum releases another copy of the recursive instance, serviceable only by expensive sets. This forces the algorithm to buy the expensive sets twice; the optimum only buys them once. If, on the other hand, the algorithm chooses the inexpensive sets, it misses the opportunity to serve the additional requests and the recursive instance simultaneously, and must serve them separately.

The recursive description of our construction for the lower bounds is significantly more natural than its iterative description. Few lower bounds in online algorithms have this property – another such lower bound is found in [8].

The 3-competitive deterministic algorithm for VCD is simple and based on counters. This algorithm is only $k + 1$ competitive for general SCD, and is thus significantly worse than the previous randomized algorithm that we have shown for general SCD.

1.3 Other Related Work

A different problem called online set cover is considered in [3], in which the algorithm accumulates value for every element that arrives on a bought set, and aims to maximize total value. This problem appears to be fundamentally different from the online set cover in which we minimize cost, in both techniques and results.

The problem of set cover in the online setting has seen much additional work, e.g. in [22, 9, 18, 29, 1]. The set cover problem has also been studied in the streaming model (e.g. [21, 16]), stochastic model (e.g. [24]), dynamic model (e.g. [23]), and in the context of universal algorithms (e.g. [25, 22]) and communication complexity (e.g. [28]).

There are known inapproximability results for the (offline) set cover and vertex cover problems. In [17] it is shown that the offline set cover problem is unlikely to be approximable in polynomial time to within a factor better than $\ln n$. For the offline vertex cover, it is shown in [26] that it is NP hard to approximate within a factor better than 2, assuming the *Unique Games Conjecture*. These results apply to our SCD and VCD problems, as an instance of offline set cover (or vertex cover) can be released at time 0. Of course, these inapproximability results do not constitute lower bounds for the online model, in which unbounded computation is allowed – unlike the information-theoretic lower bound of $\Omega(\sqrt{\log n})$ for SCD which is given in this paper.

The field of online problems with delay over time has been of interest recently. This includes the problems of min-cost perfect matching with delays [20, 5, 2, 12, 11, 4], online service with delay [6, 13, 7] and multilevel aggregation [10, 14, 7].

Paper Organization

In Section 3, we present and analyze a fractional non-clairvoyant algorithm for SCD. In Section 4, we show how to round the previous algorithm in a non-clairvoyant manner to obtain our algorithm for the original (integral) SCD. In Section 5, we show lower bounds for clairvoyant SCD. In the full version of this paper, we show that the algorithm obtained in Section 4 is optimal for the non-clairvoyant case. In Section 6, we give a simple, deterministic, non-clairvoyant algorithm for vertex cover with delay.

2 Preliminaries

We denote the sets by $\{S_i\}_{i=1}^m$, with m the number of sets. We denote by n the number of elements. We define k to be the minimal number for which every element belongs to at most k sets. Requests q_j arrive on the elements. We denote the arrival time of request q_j by r_j, and write (with a slight abuse of notation) $q_j \in S_i$ if the element on which q_j has been released belongs to the set S_i.

Each request q_j has an arbitrary momentary delay function $d_j(t)$, defined for $t \geq r_j$. The accumulated delay of the request at time $t \geq r_j$ is defined to be $\int_{r_j}^{t} d_j(t)\,\mathrm{d}t$. At any time in which a request is pending, its momentary delay is added to the cost of the algorithm; that is, the algorithm incurs a cost of $\int_{r_j}^{\tau_j} d_j(t)\,\mathrm{d}t$ (the accumulated delay of q_j at time τ_j) for every request q_j, where τ_j is the time in which q_j is served. Each set S_i has a price $c(S_i) \geq 1$ which the algorithm must pay when it decides to buy the set. Buying a set serves all pending requests which belong to the set (but does not affect future requests). The *buying cost* of an online algorithm ON is $\mathrm{Cost}^p_{\mathrm{ON}} = \sum_i n_i \cdot c(S_i)$, where n_i is the number of times S_i has been bought by the algorithm. The *delay cost* of ON is $\mathrm{Cost}^d_{\mathrm{ON}} = \sum_j \int_{r_j}^{\tau_j} d_j(t)\,\mathrm{d}t$, where τ_j is the time in which q_j is served by the algorithm (τ_j is ∞ if q_j is never served by the algorithm)[2].

Overall, the cost of ON for the problem is $\mathrm{Cost}_{\mathrm{ON}} = \mathrm{Cost}^p_{\mathrm{ON}} + \mathrm{Cost}^d_{\mathrm{ON}}$

3 The Non-Clairvoyant Algorithm for Fractional SCD

We first describe a fractional relaxation of the (integer) set cover with delay problem. In this fractional relaxation, a set can be bought in parts. A fractional algorithm determines for each set S_i a nonnegative momentary buying function $x_i(t)$. The total buying cost a fractional online algorithm F incurs is $\mathrm{Cost}^p_F = \sum_i c(S_i) \cdot \int_0^\infty x_i(t)\,\mathrm{d}t$.

In the fractional version, a request can be partially served. Under a fractional algorithm F, for any request q_j, and any set S_i such that $q_j \in S_i$, the set S_i covers q_j at a time $t \geq r_j$ by the amount $\int_{r_j}^{t} x_i(t')\,\mathrm{d}t'$ (which is obviously nondecreasing as a function of t). The total amount by which q_j is covered at time t is

$$\gamma_j(t) = \sum_{i|q_j \in S_i} \int_{r_j}^{t} x_i(t')\,\mathrm{d}t'.$$

[2] We solve the more general problem in which the algorithm doesn't have to serve all requests (observe that the adversary can still force the algorithm to serve all requests by adding infinite delay at time infinity). This allows the problem to capture additional problems (e.g. prize-collecting problems, in which a penalty could be paid to avoid serving a specific request)

If at time t we have $\gamma_j(t) \geq 1$, then q_j is considered served, and the algorithm does not incur delay. However, if $\gamma_j(t) < 1$, the algorithm F incurs delay proportional to the uncovered fraction of q_j. Formally, at time t the request q_j incurs $d_j^F(t)$ delay in F, where

$$d_j^F(t) = \begin{cases} d_j(t) \cdot (1 - \gamma_j(t)) & \text{if } \gamma_j(t) < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The delay cost of the algorithm is $\text{Cost}_F^d = \sum_j \int_{r_j}^{\infty} d_j^F(t)\,\mathrm{d}t$. The total cost of the fractional algorithm is thus $\text{Cost}_F = \text{Cost}_F^p + \text{Cost}_F^d$.

Description of the algorithm. We now describe an online algorithm called ONF for the fractional problem.

We define a total order $\preceq$ on requests, such that for any two requests q_{j_1}, q_{j_2} if $r_{j_1} < r_{j_2}$ we have $q_{j_1} \prec q_{j_2}$ (we break ties arbitrarily between requests with equal arrival time).

At any time t, the algorithm does the following.

1. For every request q_j, evaluate $d_j^{\text{ONF}}(t)$ by its definition in Equation 3.1.
2. For every set S_i and request $q_j \in S_i$, define
$$D_i^j(t) = \sum_{j' | q_{j'} \in S_i \wedge q_{j'} \preceq q_j} d_{j'}^{\text{ONF}}(t).$$
3. For every set S_i and request $q_j \in S_i$, define
$$x_i^j(t) = \frac{1}{k} \cdot \left(\frac{\ln(1+k)}{c(S_i)} \cdot D_i^j(t) \right) \cdot e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_j}^{t} D_i^j(t')\,\mathrm{d}t'}.$$
4. Buy every set S_i according to $x_i(t)$, such that
$$x_i(t) = \max_j x_i^j(t).$$

This completes the description of the algorithm.

The intuition for the algorithm is that at any time t, the amount $\int_{r_j}^{t} D_i^j(t')\,\mathrm{d}t'$ is delay incurred by the algorithm until time t that the optimum possibly avoided by buying S_i at time r_j, and thus the algorithm wishes to minimize this amount. Thus, the request q_j places some "demand" on the algorithm to buy S_i. Since this is true for any $q_j \in S_i$, the algorithm chooses the maximum of the demands as the buying function of S_i.

This demand $x_i^j(t)$ placed on the algorithm by q_j to buy S_i is related to $\int_{r_j}^{t} D_i^j(t')\,\mathrm{d}t'$. If we wanted to make the total buying proportional to $\int_{r_j}^{t} D_i^j(t')\,\mathrm{d}t'$, it would sound reasonable to set $x_i^j(t)$ to be its derivative, namely $D_i^j(t)$. However, this would only be $\Omega(k)$-competitive, as demonstrated in Figure 3.1. We thus want the total buying to be proportional to an expression exponential in $\int_{r_j}^{t} D_i^j(t')\,\mathrm{d}t'$, which underlies the definition of $x_i^j(t)$ in our algorithm.

Denoting $X_i^j(t) = \int_{r_j}^{t} x_i^j(t')\,\mathrm{d}t'$, note that

$$X_i^j(t) = \frac{1}{k} \cdot \left[e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_j}^{t} D_i^j(t')\,\mathrm{d}t'} - 1 \right]. \tag{3.2}$$

In the rest of this section, we prove the following theorem.

▶ **Theorem 2.** *The algorithm for fractional SCD described above is $O(\log k)$-competitive.*

We now analyze the algorithm for fractional SCD and prove Theorem 2.

In this figure, there are $k-1$ elements, where each element is contained in k sets of cost 1, one central set (which contains all elements) and $k-1$ peripheral sets (each contains exactly one element). Consider $k-1$ requests, one on each element, all arriving at time 0. Their delay functions are identical, and go to infinity as time progresses.
Consider an algorithm which buys sets linearly to the delay - that is, $x_i(t) = \max_j D_i^j(t) = \sum_{j|q_j \in S_i} d_j^{\text{ONF}}(t)$. The momentary delay of every request contributes equally to the buying functions of the k containing sets. Thus, the total fraction bought of peripheral sets is exactly $k-1$ times the total fraction bought of the central set. Consider the point in time in which all requests are half-covered (through symmetry, this happens for all requests at the same time, and must happen since the requests gather infinite delay). We have that the central set was bought for a fraction of exactly $\frac{1}{4}$ (which can again be seen through symmetry of the requests and their delay). Thus, the peripheral sets were bought for a fraction of $\frac{k-1}{4}$, for a total of $\frac{k}{4}$. Consider that the optimal solution costs 1, as the optimum buys the central set at time 0.

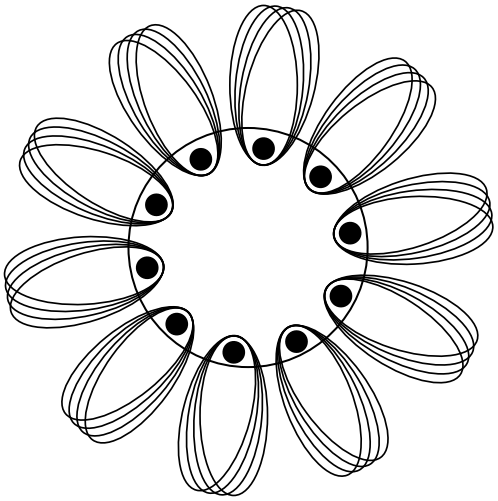


Figure 3.1 Linear Buying $\Omega(k)$ Example.

3.1 Charging Buying Cost to Delay

In this subsection we prove the following lemma.

▶ **Lemma 3.** $Cost^p_{\text{ONF}} \leq 2\ln(1+k) \cdot Cost^d_{\text{ONF}}$.

Proof. The proof is by charging the momentary buying cost at any time t to the $2\ln(1+k)$ times the momentary delay incurred by ONF at t. Let q_j be some request released by time t. For every i such that $q_j \in S_i$, we charge some amount $z_i^j(t)$ to $d_j^{\text{ONF}}(t)$. Denote by j_i the request in S_i such that

$$x_i(t) = x_i^{j_i}(t).$$

If $q_j \preceq q_{j_i}$, we choose

$$z_i^j(t) = \frac{\ln(1+k)}{k} \cdot d_j^{\text{ONF}}(t) \cdot e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_{j_i}}^{t} D_i^{j_i}(t')\,\mathrm{d}t'}.$$

Otherwise, we choose $z_i^j(t) = 0$. Note that for every set S_i we have $\sum_{j|q_j \in S_i} z_i^j(t) = c(S_i) \cdot x_i(t)$, and thus the entire buying cost is charged.

The total buying cost charged to a request q_j at time t is $\sum_{i|q_j \in S_i} z_i^j(t)$. We show that for any j we have

$$\sum_{i|q_j \in S_i} z_i^j(t) \leq 2\ln(1+k) \cdot d_j^{\text{ONF}}(t).$$

Summing the previous equation over requests q_j and integrating over time yields the lemma.

If $d_j^{\text{ONF}}(t) = 0$ we have $z_i^j(t) = 0$ for every i, as required. From now on, we assume that $d_j^{\text{ONF}}(t) > 0$.

Denote by $T_j = \{i | q_j \in S_i \text{ and } z_i^j > 0\}$. We have

$$\begin{aligned}\sum_{i|q_j \in S_i} z_i^j(t) &= \sum_{i \in T_j} z_i^j(t) \\ &= \ln(1+k) \cdot d_j^{\text{ONF}}(t) \cdot \sum_{i \in T_j} \frac{1}{k} \cdot e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_{j_i}}^t D_i^{j_i}(t')\,\mathrm{d}t'}.\end{aligned}$$

Now note that

$$\begin{aligned}\frac{1}{k} \cdot e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_{j_i}}^t D_i^{j_i}(t')\,\mathrm{d}t'} &= \frac{1}{k} + X_i^{j_i}(t) \\ &\leq \frac{1}{k} + \int_{r_{j_i}}^t x_i(t')\,\mathrm{d}t' \\ &\leq \frac{1}{k} + \int_{r_j}^t x_i(t')\,\mathrm{d}t'\end{aligned}$$

where the equality is due to equation 3.2, the first inequality is due to the definition of $X_i^{j_i}(t)$ and since $x_i(t) \geq x_i^{j_i}(t)$, and the last inequality is due to $q_j \preceq q_{j_i}$.

Thus

$$\sum_{i|q_j \in S_i} z_i^j(t) \leq \ln(1+k) \cdot d_j^{\text{ONF}}(t) \cdot \sum_{i \in T_j} \left(\frac{1}{k} + \int_{r_j}^t x_i(t')\,\mathrm{d}t'\right) \leq 2\ln(1+k) \cdot d_j^{\text{ONF}}(t)$$

where the last inequality follows from $|T_j| \leq k$, and from $\sum_{i|q_j \in S_i} \int_{r_j}^t x_i(t')\,\mathrm{d}t' \leq 1$ (due to the assumption that $d_j^{\text{ONF}}(t) > 0$). ◀

3.2 Charging Delay to Optimum

In this subsection, we charge the delay of the algorithm to the optimum via dual fitting.

3.2.1 Linear Programming Formulation

We formulate a linear programming instance for the fractional problem, and observe its dual instance.

Primal. In the primal instance, the variables are:

- $x_i(t)$ for a set S_i and time t, which is the fraction by which the algorithm buys S_i at time t.
- $p_j(t)$ for a request q_j and time $t \geq r_j$, which is the fraction of q_j not covered by bought sets at time t.

The LP instance is therefore:

Minimize:

$$\sum_i \int_0^\infty c(S_i) \cdot x_i(t)\,\mathrm{d}t + \sum_j \int_{r_j}^\infty p_j(t) \cdot d_j(t)\,\mathrm{d}t$$

under the constraints:

$$\forall j, t \geq r_j : p_j(t) + \sum_{i|q_j \in S_i} \int_{r_j}^t x_i(t')\,\mathrm{d}t' \geq 1$$

$$p_j(t) \geq 0\,,\; x_i(t) \geq 0.$$

Dual. Maximize:

$$\sum_j \int_{r_j}^\infty y_j(t)\,\mathrm{d}t$$

under the constraints:

$$\forall i, t : \sum_{j|q_j \in S_i \wedge r_j \leq t} \int_t^\infty y_j(t')\,\mathrm{d}t' \leq c(S_i) \tag{C1}$$

$$\forall j, t \geq r_j : y_j(t) \leq d_j(t) \tag{C2}$$

$$y_j(t) \geq 0.$$

▶ Remark 4. As we chose to consider time as continuous, the linear program described here has an infinite number of variables and constraints. This is merely a choice of presentation, as discretizing time would yield a standard, finite LP. Nevertheless, weak duality for this infinite LP (the only duality property used in this paper) holds (see e.g. [30]).

3.2.2 Charging Delay to Optimum via Dual Fitting

We now charge the delay of the fractional algorithm to the cost of the optimum.

▶ **Lemma 5.** $Cost^d_{\mathrm{ONF}} \leq Cost_{OPT}$.

Proof. The proof is by finding a solution to the dual problem, such that the goal function value of the solution is equal to the delay of the algorithm.

For every request q_j and time t, we assign $y_j(t) = d_j^{\mathrm{ONF}}(t)$. This assignment satisfies that the goal function is the total delay incurred by the algorithm.

Note that the C2 constraints trivially hold, since $d_j^{\mathrm{ONF}}(t) \leq d_j(t)$ for any j, t. Now observe the C1 constraints. For any time t and a set S_i, the resulting C1 constraint is implied by the C1 constraint of time r_j and the set S_i, with q_j being the last request released by time t. We thus restrict ourselves to C1 constraints of time r_j for some j.

For a request q_j and a set S_i, we need to show:

$$\sum_{j'|q_{j'} \in S_i \wedge q_{j'} \preceq q_j} \int_{r_j}^\infty d_{j'}^{\mathrm{ONF}}(t')\,\mathrm{d}t' \leq c(S_i).$$

Using the definition of $D_i^j(t)$, we need to show:

$$\int_{r_j}^{\infty} D_i^j(t)\,\mathrm{d}t \le c(S_i).$$

Define t_0 to be the minimal time (possibly ∞) such that for all $t \ge t_0$ we have $D_i^j(t) = 0$. We must have that $\int_{r_j}^{t_0} x_i(t)\,\mathrm{d}t \le 1$; otherwise, all requests $q_{j'} \in S_i$ such that $q_{j'} \preceq q_j$ will be completed before t_0, in contradiction to t_0's minimality. Thus we have

$$\begin{aligned} 1 &\ge \int_{r_j}^{t_0} x_i(t)\,\mathrm{d}t \ge \int_{r_j}^{t_0} x_i^j(t)\,\mathrm{d}t \\ &= \frac{1}{k}\left[e^{\frac{\ln(1+k)}{c(S_i)}\int_{r_j}^{t_0} D_i^j(t)\,\mathrm{d}t} - 1\right] \end{aligned}$$

where the second inequality is due to the definition of $x_i(t)$, and the equality is due to equation 3.2. This yields

$$(1+k)^{\frac{1}{c(S_i)}\int_{r_j}^{t_0} D_i^j(t)\,\mathrm{d}t} \le 1+k$$

and thus

$$\int_{r_j}^{\infty} D_i^j(t)\,\mathrm{d}t = \int_{r_j}^{t_0} D_i^j(t)\,\mathrm{d}t \le c(S_i)$$

as required. ◀

We can now prove the main theorem.

Proof of Theorem 2. Using Lemmas 3 and 5, we have

$$\begin{aligned} \mathrm{Cost}_{\mathrm{ONF}} &= \mathrm{Cost}^p_{\mathrm{ONF}} + \mathrm{Cost}^d_{\mathrm{ONF}} \\ &\le (2\ln(1+k)+1)\cdot \mathrm{Cost}^d_{\mathrm{ONF}} \\ &\le (2\ln(1+k)+1)\cdot \mathrm{Cost}_{OPT} \end{aligned}$$

as required. ◀

▶ Remark 6. For the more difficult delay model in which a partially served request q_j incurs delay $d_j^{\mathrm{ONF}}(t) = d_j(t)$ instead of $d_j^{\mathrm{ONF}}(t) = d_j(t)\cdot(1-\gamma_j(t))$ in ONF, this algorithm is still $O(\log k)$ competitive against the fractional optimum in the easier delay model. The proof is identical.

4 Randomized Algorithm for SCD by Rounding

In this section, we describe a non-clairvoyant, polynomial-time randomized algorithm which is $O(\log k \cdot \log n)$-competitive for integral SCD. Our randomized algorithm uses randomized rounding of the fractional algorithm of Section 3. We describe the rounding in two steps. First, we show a somewhat simpler algorithm which is $O(\log k \cdot \log N)$-competitive. We then modify this algorithm to obtain a $O(\log k \cdot \log n)$-competitive algorithm.

The rounding of the fractional algorithm of section 3 costs the randomized integral algorithm of this section a multiplicative factor of $\log n$ over that fractional algorithm.

Denote by $x_i(t)$ the fractional buying function in the algorithm ONF of Section 3. For a request q_j, we denote by S_{i_j} the least expensive set containing q_j; that is, $i_j = \arg\min_{i|q_j \in S_i} c(S_i)$.

For every request q_j, we denote the total covering of q_j at time t in ONF by $\gamma_j(t)$, where

$$\gamma_j(t) = \sum_{i|q_j \in S_i} \int_{r_j}^{t} x_i(t')\,\mathrm{d}t'.$$

We denote by t_j the first time in which $\gamma_j(t) = \frac{1}{2}$.

$O(\log k \cdot \log N)$-Competitive Rounding

We now describe a randomized integral algorithm, called ONR, which is $O(\log k \cdot \log N)$ competitive with respect to the fractional optimum, with N the total number of requests. We assume a-priori knowledge of N for the algorithm.

The randomized integral algorithm runs the fractional algorithm of Section 3 in the background, and thus has knowledge of the function $x_i(t)$ for every i. The algorithm does the following.

1. At time 0:
  a. For every set S_i, choose Λ_i from the range $[0, \frac{1}{2\ln N}]$ uniformly and independently, and set $\tau_i = 0$.
2. At time t:
  a. For every i, if $\int_{\tau_i}^{t} x_i(t')\,\mathrm{d}t' \geq \Lambda_i$ then:
    i. Buy S_i.
    ii. Assign to Λ_i a new value drawn independently and uniformly from $[0, \frac{1}{2\ln N}]$.
    iii. Assign $\tau_i = t$.
  b. If there exists a pending request q_j such that $t \geq t_j$, buy S_{i_j}.

We refer to the buying of sets at Step 2a as "type a", and to the buying of sets at Step 2b as "type b".

The intuition for the randomized rounding scheme is that we would like the probability of buying a set in a certain interval of time to be proportional to the fraction of that set bought by the fractional algorithm in that interval, independently of the other sets. This is achieved by the "type a" buying. However, using "type a" alone is problematic. Consider, for example, a request on an element in k sets, such that the fractional algorithm buys $\frac{1}{k}$ of each of the sets to cover the request. Since the probability of buying a set is independent of other sets, there exists a probability that the randomized algorithm would not buy any of the k sets, leaving the request unserved. This bears possibly infinite delay cost for the rounding algorithm, which is not incurred by the underlying fractional algorithm.

The "type b" buying solves this problem, by serving a pending request *deterministically* when it is covered in the underlying fractional algorithm, through buying the cheapest set containing that request. This special service for the request might be expensive, but its probability is low, yielding low expected cost. This is ensured by the $2\log N$ "speedup" given to the "type a" buying, through choosing the thresholds Λ_i from $[0, \frac{1}{2\ln N}]$ (rather than $[0, 1]$).

▶ **Theorem 7.** *The randomized algorithm for SCD described above is $O(\log k \cdot \log N)$-competitive.*

The proof of Theorem 7 is given in the full version of this paper.

Improved $O(\log k \cdot \log n)$-Competitive Rounding

By modifying the $O(\log k \cdot \log N)$-competitive randomized rounding, we prove the following theorem.

▶ **Theorem 8.** *There exists a non-clairvoyant, randomized $O(\log k \cdot \log n)$-competitive algorithm for SCD.*

The modified rounding algorithm and its analysis appear in the full version of this paper.

5 Lower Bounds for Clairvoyant SCD

In this section, we show $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ lower bounds on competitiveness for any randomized, clairvoyant algorithm for SCD or fractional SCD. While the lower bounds use instances in which different sets can have different costs, these instances can be modified to obtain instances with identical set costs. This implies that the lower bounds also apply to the unweighted setting. This modification is shown in Subsection 5.2.

This section shows the following theorem.

▶ **Theorem 9.** *Any randomized algorithm for SCD or fractional SCD is both $\Omega(\sqrt{\log k})$-competitive and $\Omega(\sqrt{\log n})$-competitive.*

In proving Theorem 9, we show a lower bound on competitiveness of a deterministic fractional algorithm against an integral optimum. Showing this is enough to prove the theorem, since any randomized online algorithm (fractional or integral) can be converted to a deterministic fractional online algorithm with identical (or lesser) cost. This follows from setting the momentary buying function of a set at a given time to be the expectation of that value in the randomized algorithm. Since the optimum is integral, the bound also holds for integral SCD, as the theorem states. Therefore, we only consider deterministic fractional online algorithms henceforth.

We show our lower bounds by constructing a set of SCD instances, $\{I_i\}_{i=0}^{\infty}$. For each $i \geq 0$, the SCD instance I_i contains 2^i sets and 3^i elements. We show that any algorithm must be $\Omega(\sqrt{i})$-competitive for I_i, which is both $\Omega(\sqrt{\log m})$ and $\Omega(\sqrt{\log n})$. Noting that $k \leq m$, we also have $\Omega(\sqrt{\log k})$ as required.

The instance I_i exists within the time interval $[0, 3^i)$. That is, no request of I_i is released before time 0, and at time 3^i the optimum has served all requests in I_i, and the algorithm has incurred a high enough cost.

We define the sequence $(c_i)_{i=0}^{\infty}$, which is used in the construction of I_i. The sequence is defined recursively, such that $c_0 = 1$ and for any $i \geq 1$, we have that

$$c_i = c_{i-1} + \frac{1}{12c_{i-1}}.$$

We now describe the recursive construction of the instance I_i. We first describe the universe of I_i, which consists of its sets and elements. We then describe the requests of I_i.

Universe of I_i

For the base instance I_0, the universe consists of a single element e and a single set $S = \{e\}$. We have that $c(S) = 1$.

For $i \geq 1$, the recursive construction of I_i using I_{i-1} is as follows. Denote by E_{i-1} the elements in the universe of I_{i-1}, and by H_{i-1} the family of sets in the universe of I_{i-1}. For the construction of I_i, consider three disjoint copies of E_{i-1} and H_{i-1}. For $l \in \{1, 2, 3\}$, we

This figure shows the universes of I_0, I_1 and I_2. In the figure, each element is a point and the sets are the bodies containing them, where each set has a distinct color. The costs of the sets are also shown in the figure. The figure shows how three copies of the set of elements E_{i-1} (of the instance I_{i-1}) appear in I_i – the copy E^1_{i-1} appears at the top of I_i's visualization, the copy E^2_{i-1} appears at the bottom-left, and the copy E^3_{i-1} appears at the bottom-right.

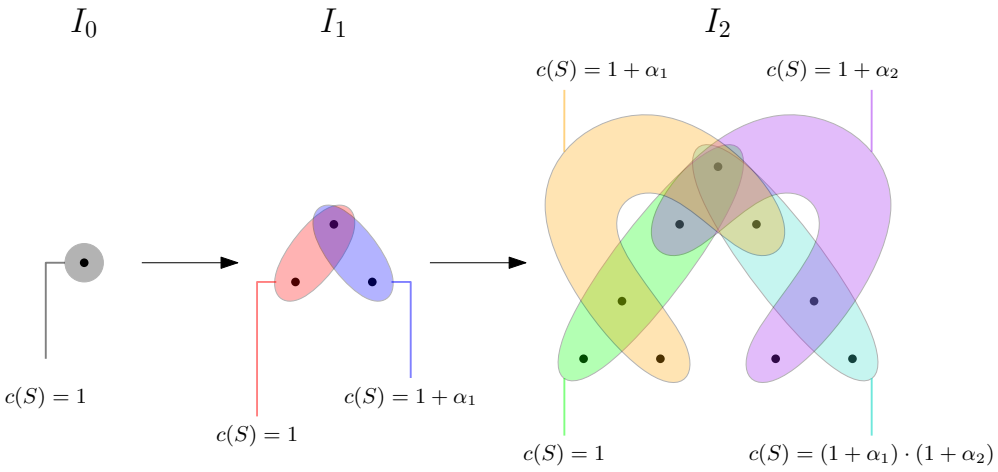


Figure 5.1 The Universes of I_0, I_1 and I_2.

denote by E^l_{i-1} and H^l_{i-1} the l'th copy of E_{i-1} and H_{i-1}, respectively. We denote by S^l the copy of the set $S \in H_{i-1}$ in H^l_{i-1}. Similarly, we denote by e^l the copy of an element $e \in E_{i-1}$ in E^l_{i-1}.

The universe of I_i consists of:

- The elements $E_i = E^1_{i-1} \cup E^2_{i-1} \cup E^3_{i-1}$.
- The family of sets $H_i = \mathcal{T}_1 \cup \mathcal{T}_2$, where $\mathcal{T}_1$ and $\mathcal{T}_2$ are defined below.

We define:

- The family of sets $\mathcal{T}_1 = \{S^1 \cup S^2 | S \in H_{i-1}\}$. A set $T \in \mathcal{T}_1$ formed from $S \in H_{i-1}$ has cost $c(T) = c(S)$.
- The family of sets $\mathcal{T}_2 = \{S^1 \cup S^3 | S \in H_{i-1}\}$. A set $T \in \mathcal{T}_2$ formed from $S \in H_{i-1}$ has cost $c(T) = (1 + \alpha_i) \cdot c(S)$, with $\alpha_i = \frac{1}{2c_{i-1}}$.

Requests of I_i

We first describe a type of request used in our construction. Let S be a set such that there exists an element $e \in S$ such that e is in no other set besides S (we call e unique to S). For times a, b such that $a < b$, we define a request $q^b_a(S)$ that can be released at any time $r \leq a$ on an element unique to S, and satisfies:

1. $\int_r^a d_j(t)\,\mathrm{d}t = 0$
2. $\int_r^b d_j(t)\,\mathrm{d}t \geq c(S)$.

▶ Remark 10. For the degenerate case of set cover with deadlines, when observing a request with deadline at time b, it can be said to accumulate 0 delay until any time before b, and infinite delay until time b. Therefore, deadline requests can function as $q^b_a(S)$ requests. Since all requests used in our construction are $q^b_a(S)$ requests for some a, b, S, our lower bound applies for set cover with deadlines as well.

To use those $q_a^b(S)$ requests, we require the following proposition, which states that a $q_a^b(S)$ request can be released on every S.

▶ **Proposition 11.** *For every set $T \in H_i$, there exists an element $e \in E_i$ unique to T.*

Proof. By induction on i. For the base case, this holds since there is only a single set with a single element. Assuming the proposition holds for I_{i-1}, we show that it holds for I_i by observing that there exists $S \in H_{i-1}$ such that $T = S^1 \cup S^l$ for $l \in \{2, 3\}$. Via induction, there exists an element $e \in E_{i-1}$ such that $e \in S$ and $e \notin S'$ for every $S' \in H_{i-1}$ such that $S' \neq S$. Choosing the element e^l yields the proposition. ◀

Base case of I_0 – at time 0, the request $q_0^1(S)$ is released on the single element e.

Recursive construction of I_i using I_{i-1} – we define $C(I_i)$ to be $\sum_{S \in H_i} c(S)$. We now define the instance I_i:

1. At time 0:
   a. Release $q_{2 \cdot 3^{i-1}}^{3^i}(T)$ for every $T \in \mathcal{T}_2$.
   b. Release I_{i-1} on the elements E_{i-1}^1 (see Remark (a)).
2. At time 3^{i-1}:
   a. If the algorithm has bought sets of $\mathcal{T}_2$ at a total cost of at least $\frac{1}{2} \cdot (1 + \alpha_i) \cdot C(I_{i-1})$, release $(1 + \alpha_i)I_{i-1}$ on the elements E_{i-1}^3 (see Remark (c)).
   b. Otherwise, release I_{i-1} on the elements of E_{i-1}^2 (see Remark (b)).

The construction of I_i includes releasing copies of I_{i-1} on the elements E_{i-1}^l, for $l \in \{1, 2, 3\}$. The following remarks make this well-defined.

▶ Remark (a). **The I_{i-1} on E_{i-1}^1**: every set $S \in H_{i-1}$ forms two sets in H_i, which are $T_1 = S^1 \cup S^2 \in \mathcal{T}_1$ and $T_2 = S^1 \cup S^3 \in \mathcal{T}_2$. The I_{i-1} construction on E_{i-1}^1 treats buying either of these sets as buying the set S. That is, it treats the sum of the momentary buying of T_1 and of T_2 as the momentary buying of S.

▶ Remark (b). **The I_{i-1} on E_{i-1}^2**: in this instance, for every set $S \in H_{i-1}$, the I_{i-1} construction treats buying $T_1 = S^1 \cup S^2 \in \mathcal{T}_1$ as buying S.

▶ Remark (c). **The scaled $(1 + \alpha_i)I_{i-1}$ on E_{i-1}^3**: similarly to Remark 5, in this instance, for every set $S \in H_{i-1}$, the I_{i-1} construction treats buying $T_2 = S^1 \cup S^3 \in \mathcal{T}_2$ as buying S. In addition, since the sets of $\mathcal{T}_2$ are $(1 + \alpha_i)$-times more expensive than the original sets of H_{i-1}, the delays of the jobs in I_{i-1} are also scaled by $1 + \alpha_i$ in order to maintain the I_{i-1} instance. We denote this scaled instance by $(1 + \alpha_i)I_{i-1}$.

5.1 Analysis of Lower Bounds

We show that any online fractional algorithm is at least c_i competitive on I_i with respect to the integral optimum.

▶ **Lemma 12.** *The optimal integral algorithm can serve I_i by time 3^i with no delay cost by buying every set in H_i exactly once, for a total cost of $C(I_i)$.*

Proof. Via induction on i. For the base case of $i = 0$, the optimal algorithm buys the single set S at time 0 and pays $c(S) = C(I_0)$. Now, for $i \geq 1$, suppose the optimum can serve the instance I_{i-1} according to the lemma. We observe the optimum in I_i according to the cases in the release of I_i:

Case 2a: In this case, the optimum could have served I_{i-1} on E^1_{i-1} by time 3^{i-1} by buying each set of $\mathcal{T}_1$ exactly once, with no delay cost. It could then serve $(1+\alpha_i)I_{i-1}$ on E^3_{i-1} by time $2 \cdot 3^{i-1}$ by buying each set of $\mathcal{T}_2$ exactly once, with no delay cost. Since the optimum has bought all of $\mathcal{T}_2$, the requests released on step 1a have also been served before incurring delay. The lemma thus holds for this case.

Case 2b: In this case, the optimum could have served I_{i-1} on E^1_{i-1} by time 3^{i-1} by buying each set of $\mathcal{T}_2$ exactly once, with no delay cost. It could then serve I_{i-1} on E^2_{i-1} by time $2 \cdot 3^{i-1}$ by buying each set of $\mathcal{T}_1$ exactly once, with no delay cost. Since the optimum has bought all of $\mathcal{T}_2$, the requests released on step 1a have again been served before incurring delay. The lemma thus holds for this case as well. ◀

We now analyze the cost of the algorithm.

▶ **Lemma 13.** *Any online algorithm has a cost of at least $c_i \cdot C(I_i)$ on I_i by time 3^i.*

Proof. By induction on i.

For $i = 0$, observe the algorithm at time 1. Denoting by Γ_S the total buying of the single set S by the algorithm by time 1, the algorithm has a cost of at least

$$c(S) \cdot \Gamma_S + (1 - \Gamma_S) \cdot \int_0^1 d_{q_0^1(S)}(t)\,\mathrm{d}t \geq c(S) = C(I_0)$$

where the inequality is due to the definition of $q_0^1(S)$. This finishes the base case of the induction.

For the case that $i \geq 1$, assume that the lemma holds for $i-1$. We show that it holds for i.

Fix any algorithm for I_i. We denote by Γ the total buying cost of the algorithm in the time interval $[0, 3^{i-1})$ for sets of $\mathcal{T}_2$. We again split into cases according to the chosen branch in the construction of I_i.

Case 2a: In this case we have $\Gamma \geq \frac{1}{2} \cdot (1+\alpha_i) \cdot C(I_{i-1})$. From the definition of the first I_{i-1} released, the adversary is oblivious to whether a copy of $S \in H_{i-1}$ came from $\mathcal{T}_1$ or $\mathcal{T}_2$. Using the induction hypothesis, any online algorithm for this instance incurs a cost of at least $c_{i-1} \cdot C(I_{i-1})$ by time 3^{i-1}, including the algorithm in which buying sets from $\mathcal{T}_2$ are replaced with buying the equivalent sets from $\mathcal{T}_1$. Such a modified online algorithm would cost $\frac{\alpha_i}{1+\alpha_i}\Gamma$ less than the current algorithm, which is at least $\frac{\alpha_i}{2} \cdot C(I_{i-1})$. Therefore, the algorithm pays at least $(c_{i-1} + \frac{\alpha_i}{2}) \cdot C(I_{i-1})$ in the interval $[0, 3^{i-1})$.

As for the second instance $(1+\alpha_i)I_{i-1}$, the algorithm must pay at least $(1+\alpha_i) \cdot c_{i-1} \cdot C(I_{i-1})$ by time $2 \cdot 3^{i-1}$ via induction.

Overall, the algorithm pays by time 3^i at least

$$\begin{aligned}
&\left(\left(c_{i-1} + \frac{\alpha_i}{2}\right) \cdot C(I_{i-1})\right) + ((1+\alpha_i) \cdot c_{i-1} \cdot C(I_{i-1})) \\
&\qquad = \left((2+\alpha_i)c_{i-1} + \frac{\alpha_i}{2}\right) \cdot C(I_{i-1}) \\
&\qquad = c_{i-1} \cdot C(I_i) + \frac{\alpha_i}{2} \cdot C(I_{i-1}) \\
&\qquad \geq \left(c_{i-1} + \frac{\alpha_i}{6}\right) \cdot C(I_i) \\
&\qquad = \left(c_{i-1} + \frac{1}{12c_{i-1}}\right) \cdot C(I_i)
\end{aligned}$$

where the inequality is due to $C(I_i) = (2+\alpha_i)C(I_{i-1}) \leq 3C(I_{i-1})$.

Case 2b: In this case we have $\Gamma < \frac{1}{2} \cdot (1+\alpha_i) \cdot C(I_{i-1})$. For the first I_{i-1} instance, the algorithm pays at least $c_{i-1} \cdot C(I_{i-1}) + \Gamma \cdot \frac{\alpha_i}{1+\alpha_i}$ by time 3^{i-1}, similar to the previous case. For the second I_{i-1} instance, released on E^2_{i-1}, the algorithm must pay via induction at least $c_{i-1} \cdot C(I_{i-1})$ by time $2 \cdot 3^{i-1}$. Since sets of $\mathcal{T}_2$ do not satisfy requests in this instance, this cost is either in buying sets of $\mathcal{T}_1$ or in delay of requests from that I_{i-1} instance.

In addition to the two I_{i-1} instances, due to the $q^{3^i}_{2 \cdot 3^{i-1}}(S)$ requests released in step 1a, the algorithm has a cost of at least $\left(\sum_{T \in \mathcal{T}_2} c(T)\right) - \Gamma = (1+\alpha_i)C(I_{i-1}) - \Gamma$ during the interval $[1,3)$ in either buying sets of $\mathcal{T}_2$ in order to finish these requests, or in delay by those requests (using a similar argument to that in the base case). Overall, the algorithm has a cost of at least

$$\begin{aligned}
&\left(c_{i-1} \cdot C(I_{i-1}) + \Gamma \cdot \frac{\alpha_i}{1+\alpha_i}\right) + (c_{i-1} \cdot C(I_{i-1})) + ((1+\alpha_i)C(I_{i-1}) - \Gamma) \\
&\qquad = (2c_{i-1} + 1 + \alpha_i) \cdot C(I_{i-1}) - \frac{1}{1+\alpha_i}\Gamma \\
&\qquad \geq (2c_{i-1} + 1 + \alpha_i) \cdot C(I_{i-1}) - \frac{1}{2}C(I_{i-1}) \\
&\qquad = \left(2c_{i-1} + \frac{1}{2} + \alpha_i\right) \cdot C(I_{i-1}) \\
&\qquad = \left((2+\alpha_i)c_{i-1} + \frac{1}{2} + (1-c_{i-1})\alpha_i\right) \cdot C(I_{i-1}) \\
&\qquad = c_{i-1} \cdot C(I_i) + \left(\frac{1}{2} + \frac{1}{2c_{i-1}} - \frac{1}{2}\right) \cdot C(I_{i-1}) \\
&\qquad \geq \left(c_{i-1} + \frac{1}{6c_{i-1}}\right) \cdot C(I_i) \geq c_i \cdot C(I_i)
\end{aligned}$$

where the fourth equality and the second inequality are due to $C(I_i) = (2+\alpha_i)C(I_{i-1}) \leq 3C(I_{i-1})$, and the fourth equality uses the definition of α_i. ◀

Proof of Theorem 9. Lemmas 12 and 13 immediately imply that any deterministic fractional algorithm is at least c_i-competitive on I_i with respect to the integral optimum. Solving the recurrence in the definition of c_i, we have that $c_i = \Omega(\sqrt{i})$. To observe this, note that for every i, the first index $i' \geq i$ such that $c_{i'} \geq c_i + 1$ is at most $O(c_i)$ larger than i. Since $k \leq m = 2^i$ and $n = 3^i$, this provides lower bounds of $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ for deterministic algorithms for fractional SCD. As stated before, this implies the same lower bound for randomized algorithms for both SCD and fractional SCD. ◀

5.2 Reduction to Unweighted

The lower bound above uses weighted instances, in which sets may have different costs. In this subsection, we describe how to convert a weighted instance to an unweighted instance, in which all set costs are equal. This conversion maintains both the $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ lower bounds on competitiveness. The conversion consists of creating multiple copies of each element, and converting each original set to multiple sets of cost 1. The cost of the original set affects the cardinality of the new sets, such that a set with higher cost turns into smaller sets of cost 1.

We suppose that the costs of all sets are integer powers of 2. This can easily be achieved by rounding the costs to powers of 2 (losing a factor of 2 in the lower bound), and then scaling the instance (both delays and buying costs) by the inverse of the lowest cost.

Denote by $C = 2^M$ the largest cost in the instance. The universe of the unweighted instance is the following:

- For each element e in the original instance, we have C elements in the unweighted instance, denoted by $e_0, ..., e_{C-1}$.
- For each set S, we have $c(S)$ sets in the unweighted instance, labeled $S_0, ..., S_{c(S)-1}$.
- We have that $e_i \in S_j$ if and only if both $e \in S$ and $i \equiv j \mod c(S)$.

Whenever a request q_j arrives in the original instance on an element e with delay function $d_j(t)$, C requests $q_{j,0}, ..., q_{j,C-1}$ arrive in the unweighted instance on the elements $e_0, ..., e_{C-1}$ respectively. For each $0 \leq l \leq C-1$, the request $q_{j,l}$ has the delay function $d_{j,l}(t) = \frac{d_j(t)}{C}$.

For the instance I_i described above, we consider its unweighted conversion, denoted by I'_i. Any fractional online algorithm for I'_i can be converted to a fractional online algorithm for I_i with a cost which is at most that of the original algorithm. This is done by setting the buying function of a set S in I_i to the average of the buying functions of $S_0, ..., S_{c(S)-1}$.

In addition, the integral optimum described in the analysis of I_i can be modified to an integral optimum for I'_i with identical cost. This is by converting each buying of the set S in I_i to buying the sets $S_0, ..., S_{c(S)-1}$ in I'_i.

The aforementioned facts imply that any fractional algorithm is $\Omega(\sqrt{i})$ competitive on I'_i. Note that the parameter k is the same for I_i and I'_i, implying $\Omega(\sqrt{\log k})$-competitiveness on I'_i. In addition, denoting by n' the number of elements in I'_i, we have that $n' = C \cdot n$. Observing the construction of I_i, we have that $n = 3^i$ and $C \leq 2^i$ (Using the fact that $(1 + \alpha_j) \leq 2$ for any j). Therefore, $\log n' \leq \log(6^i)$, yielding that $i = \Omega(\log n')$, and a $\Omega(\sqrt{\log n'})$ lower bound on competitiveness for I'_i.

6 Vertex Cover with Delay

In this section, we show a 3-competitive deterministic algorithm for VCD. Recall that VCD is a special case of SCD with $k = 2$, where k is the maximum number of sets to which an element can belong. In fact, we show a $(k+1)$-competitive deterministic algorithm for SCD, which is therefore 3-competitive for VCD. Recall that since the TCP acknowledgment problem is a special case of VCD with a single edge, the lower bound of 2-competitiveness for any deterministic algorithm on the TCP acknowledgment problem (shown in [19]) applies to VCD as well.

The $(k+1)$-competitive algorithm for SCD, ON, is as follows.

1. For every set S, maintain a counter $z(S)$ of the total delay incurred by ON over requests on elements in S (all $z(S)$ are initialized to 0).
2. If for any S, we have that $z(S) = c(S)$:
   a. Buy S.
   b. Zero the counter $z(S)$.

We denote by $z(S, t)$ the value of $z(S)$ at time t. We prove the following theorem.

▶ **Theorem 14.** *The algorithm* ON *for SCD has a competitive ratio of* $k + 1$. *In particular,* ON *is* 3*-competitive for VCD.*

▶ **Lemma 15.** *The cost of the algorithm is at most* $k + 1$ *times its delay cost.*

Proof. It is sufficient to bound the buying cost in terms of the delay cost. For each purchase of a set S, $z(S)$ must increase from 0 to $c(S)$. A delay for a request contributes to the increase of at most k counters. Thus, the buying cost is at most k times the delay cost. ◀

We are left to bound the delay cost of the algorithm by the adversary's cost.

▶ **Lemma 16.** *For any set* S, *let* T *be a subset of the requests on elements of* S *such that all requests of* T *are unserved at time* t. *Then we have* $\sum_{j|q_j \in T} \int_t^\infty d_j^{\mathrm{ON}}(t')\,\mathrm{d}t' \leq c(S)$.

Proof. Denote by $\hat{t}$ the first time in which all requests in T are served. We have that

$$\sum_{j|q_j \in T} \int_t^{\infty} d_j^{\mathrm{ON}}(t')\,\mathrm{d}t' = \sum_{j|q_j \in T} \int_t^{\hat{t}} d_j^{\mathrm{ON}}(t')\,\mathrm{d}t'.$$

At time t, we have $z(S,t) \geq 0$. Observe that the algorithm never bought S in the time interval $[t, \hat{t})$. Thus, at any time $t'' \in [t, \hat{t})$ we have that

$$z(S,t'') = z(S) + \sum_{j|q_j \in T} \int_t^{t''} d_j^{\mathrm{ON}}(t')\,\mathrm{d}t'.$$

Observe that $z(S,t'') < c(S)$, otherwise the algorithm would have bought S at t', serving all requests in T, in contradiction to the definition of $\hat{t}$. Therefore $\sum_{j|q_j \in T} \int_t^{t''} d_j^{\mathrm{ON}}(t')\,\mathrm{d}t' < c(S)$. The claim follows as t'' approaches $\hat{t}$. ◀

▶ **Lemma 17.** *The delay cost of the algorithm is at most the adversary's cost.*

Proof. We construct a solution to the dual LP from section 3, with a goal function which is the delay cost of the algorithm. This charges the delay cost of the algorithm to the fractional optimum, and thus to the integer optimum as well.

Specifically, we set $y_j(t) = d_j^{\mathrm{ON}}(t)$ for every j, t. Obviously, the C2 constraints hold. In order to show that the C1 constraint for a set S_i and a time t holds, observe that any request $q_j \in S_i$ served in ON before time t has $d_j^{\mathrm{ON}}(t') = 0$ for all $t' \geq t$. Using Lemma 16 for the requests unserved at t concludes the proof. ◀

Proof of theorem 14. The proof of the theorem results directly from lemmas 16 and 17. ◀

Note that this algorithm's competitive ratio is indeed as bad as $k+1$. Consider, for example, a single request in k sets with unit costs, which the optimum solves with cost 1 and the algorithm has cost $k+1$.

References

1. Sebastian Abshoff, Christine Markarian, and Friedhelm Meyer auf der Heide. Randomized online algorithms for set cover leasing problems. In *8th COCOA*, pages 25–34, 2014.
2. Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul M. Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Proceedings of the APPROX/RANDOM*, pages 1:1–1:20, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.1.
3. Baruch Awerbuch, Yossi Azar, Amos Fiat, and Frank Thomson Leighton. Making commitments in the face of uncertainty: How to pick a winner almost every time. In *Proceedings of the Twenty-Eighth STOC*, pages 519–530, 1996. doi:10.1145/237814.238000.
4. Y. Azar and A. Jacob-Fanani. Deterministic Min-Cost Matching with Delays. *ArXiv e-prints*, June 2018. arXiv:1806.03708.
5. Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *28th SODA*, pages 1051–1061, 2017. doi:10.1137/1.9781611974782.67.
6. Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *49th STOC*, pages 551–563, 2017. doi:10.1145/3055399.3055475.
7. Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *Proceedings of the 60th IEEE FOCS*, pages 60–71, 2019. doi:10.1109/FOCS.2019.00013.

8 Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *Proceedings of the Twentieth SODA*, pages 1238–1244, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496904.

9 Kshipra Bhawalkar, Sreenivas Gollapudi, and Debmalya Panigrahi. Online set cover with set requests. In *Proceedings of the APPROX/RANDOM*, pages 64–79, 2014.

10 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In *Proceedings of the 24th ESA*, pages 12:1–12:17, 2016. doi:10.4230/LIPIcs.ESA.2016.12.

11 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. *CoRR*, abs/1804.08097, 2018. arXiv:1804.08097.

12 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. A match in time saves nine: Deterministic online matching with delays. In *15th WAOA*, pages 132–146, 2017. doi:10.1007/978-3-319-89441-6_11.

13 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In *24th SIROCCO*, 2018.

14 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Twenty-Eighth SODA*, pages 1235–1244, 2017. doi:10.1137/1.9781611974782.80.

15 Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In *Proceedings of the LATIN 2018:*, pages 245–259, 2018. doi:10.1007/978-3-319-77404-6_19.

16 Amit Chakrabarti and Anthony Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. In *Proceedings of the Twenty-Seventh SODA*, pages 1365–1373, 2016. doi:10.1137/1.9781611974331.ch94.

17 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.

18 Stefan Dobrev, Jeff Edmonds, Dennis Komm, Rastislav Královic, Richard Královic, Sacha Krug, and Tobias Mömke. Improved analysis of the online set cover problem with advice. *Theor. Comput. Sci.*, 689:96–107, 2017.

19 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. TCP dynamic acknowledgment delay: Theory and practice. In *Proceedings of the Thirtieth STOC*, pages 389–398, 1998. doi:10.1145/276698.276792.

20 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the 48th STOC*, pages 333–344, 2016. doi:10.1145/2897518.2897557.

21 Yuval Emek and Adi Rosén. Semi-streaming set cover - (extended abstract). In *Proceedings of the 41st ICALP*, pages 453–464, 2014. doi:10.1007/978-3-662-43948-7_38.

22 Fabrizio Grandoni, Anupam Gupta, Stefano Leonardi, Pauli Miettinen, Piotr Sankowski, and Mohit Singh. Set covering with our eyes closed. *SIAM J. Comput.*, 42(3):808–830, 2013.

23 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th STOC*, pages 537–550, 2017. doi:10.1145/3055399.3055493.

24 Sungjin Im, Viswanath Nagarajan, and Ruben van der Zwaan. Minimum latency submodular cover. *ACM Trans. Algorithms*, 13(1):13:1–13:28, 2016. doi:10.1145/2987751.

25 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for tsp, steiner tree, and set cover. In *37th STOC*, pages 386–395, 2005. doi:10.1145/1060590.1060649.

26 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.

27 Simon Korman. On the use of randomization in the online set cover problem. Master's thesis, Weizmann Institute of Science, 2005.

28 Noam Nisan. The communication complexity of approximate set packing and covering. In *Proceedings of the ICALP*, pages 868–875, 2002. doi:10.1007/3-540-45465-9_74.

29 Ashwin Pananjady, Vivek Kumar Bagaria, and Rahul Vaze. The online disjoint set cover problem and its applications. In *2015 IEEE INFOCOM*, pages 1221–1229, 2015.

30 Thomas W. Reiland. Optimality conditions and duality in continuous programming ii. the linear problem revisited. *Journal of Mathematical Analysis and Applications*, 1980.

Improved Bounds for Metric Capacitated Covering Problems

Sayan Bandyapadhyay
Department of Informatics, University of Bergen, Norway
sayan.bandyapadhyay@gmail.com

Abstract

In the Metric Capacitated Covering (MCC) problem, given a set of balls $\mathcal{B}$ in a metric space P with metric d and a capacity parameter U, the goal is to find a minimum sized subset $\mathcal{B}' \subseteq \mathcal{B}$ and an assignment of the points in P to the balls in $\mathcal{B}'$ such that each point is assigned to a ball that contains it and each ball is assigned with at most U points. MCC achieves an $O(\log |P|)$-approximation using a greedy algorithm. On the other hand, it is hard to approximate within a factor of $o(\log |P|)$ even with $\beta < 3$ factor expansion of the balls. Bandyapadhyay et al. [SoCG 2018, DCG 2019] showed that one can obtain an $O(1)$-approximation for the problem with 6.47 factor expansion of the balls. An open question left by their work is to reduce the gap between the lower bound 3 and the upper bound 6.47. In this current work, we show that it is possible to obtain an $O(1)$-approximation with only 4.24 factor expansion of the balls. We also show a similar upper bound of 5 for a more generalized version of MCC for which the best previously known bound was 9.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Approximation algorithms

Keywords and phrases Capacitated covering, approximation algorithms, bicriteria approximation, LP rounding

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.9

Funding This work is partly supported by the Research Council of Norway via the project "MULTIVAL".

Acknowledgements I am indebted to Tanmay Inamdar for giving invaluable feedback on this work. I also thank the anonymous reviewers whose suggestions have helped to further improve the quality of the paper.

1 Introduction

In any metric space P with metric d, a ball $B(c, r)$ with center $c \in P$ and radius r is defined as the set of points at a distance at most r from c, i.e., $B(c, r) = \{p \in P \mid d(c, p) \leq r\}$. In the *Metric Capacitated Covering* (MCC) problem, we are given a set of balls $\mathcal{B}$ in the metric space P with metric d. We are also given a capacity parameter $U \in \mathbb{N}$ for the balls. The goal is to find a minimum sized subset $\mathcal{B}' \subseteq \mathcal{B}$ and an assignment $\phi : P \to \mathcal{B}'$ such that for any point $p \in P$, the ball $\phi(p)$ contains p and the number of points assigned to a ball $B \in \mathcal{B}'$ via ϕ is at most U, i.e., $|\phi^{-1}(B)| \leq U$. For $B_i \in \mathcal{B}$, we denote its center and radius by c_i and r_i, respectively.

The greedy algorithm of [28] yields an $O(\log |P|)$-approximation for MCC. Indeed, this approximation factor is tight, which can be proved using the following simple reduction from set cover. For each element, add a point. For each set, add a ball of radius 1. If an element is in a set, then the distance between the center of the corresponding ball and the corresponding point is set to 1. Consider the metric space induced by the centers and the points. The capacity of each ball is set to the total number of elements, say n. Now, if there is a set cover of size k, then all the points can be covered by k balls without violating the

capacities. The converse is also true. As set cover is hard to approximate within a factor of $o(\log n)$ under standard complexity theoretic assumptions [16], it is not possible to find an approximation for MCC which is asymptotically better than $O(\log n)$.

As it is not possible to obtain a $o(\log n)$-approximation for MCC, researchers have focused on obtaining bicriteria approximation. An (α, β) bicriteria approximation for MCC is a solution where the balls can be expanded by a factor of β (i.e., for a ball $B_i \in \mathcal{B}$ and a point p_j assigned to B_i, $d(c_i, p_j) \leq \beta \cdot r_i$) and the size of the solution is at most α times the optimum solution size (that does not expand the balls). From the above reduction, it follows that no $(o(\log n), \beta)$ bicriteria approximation is possible for MCC under standard complexity theoretic assumptions for any $\beta < 3$. This is true, as in the construction for a ball B_i that does not contain a point p_j, the distance between c_i and p_j is at least 3. Thus, with less than 3 factor expansion, B_i cannot contain any more points than before.

On the positive side, Bandyapadhyay et al. [4] obtained an $(O(1), 6.47)$ bicriteria approximation for the problem, i.e., with only a 6.47 factor expansion of the balls it is possible to obtain a constant approximation. Their algorithm is based on rounding of the natural LP relaxation of MCC. One problem that was left open by the work of [4] is to reduce the gap between the lower bound 3 and the upper bound 6.47. Thus, for what possible value of $3 \leq \beta < 6.47$ can one obtain an $(O(1), \beta)$ bicriteria approximation for MCC? They also consider a generalization of MCC – *Metric Monotonic Capacitated Covering* (MMCC). This problem is similar to MCC except each ball B_i has its individual capacity $U_i \in \mathbb{N}$ which must be satisfied if it is chosen in the solution and the capacities are *monotonic* – for any two balls B_i and B_j if the radius of B_i is at least the radius of B_j, then $U_i \geq U_j$. At first glance, this assumption might seem artificial. However, this model has applications in wireless network. In a wireless network, coverage areas of antennas can be modelled using balls. Moreover, it might be economical to invest in capacity of an antenna to serve more clients, if its coverage area is larger. Bandyapadhyay et al. [4] gave an $(O(1), 9)$ bicriteria approximation for MMCC using the same approach.

1.1 Our Results and Techniques

In this paper, we obtain improved results both for MCC and MMCC.

- For MCC, we obtain an $(O(1), 4.24)$ bicriteria approximation, i.e., it is possible to obtain an $O(1)$-approximation with only 4.24 factor expansion of the balls when the capacities are uniform.
- For MMCC, we obtain an $(O(1), 5)$ bicriteria approximation, i.e., it is possible to obtain an $O(1)$-approximation with only 5 factor expansion of the balls when the capacities are monotonic.

Similar to [4] our results are also based on LP rounding. Indeed, our starting point is their rounding algorithm. For the purpose of giving an overview of our technique, let us focus on MMCC. The algorithm in [4] consists of three steps – Preprocessing, Cluster Formation and Selection of Balls. Each of Preprocessing and Selection of Balls incurs an overhead of a factor 3 expansion of the balls, resulting in the 9 factor expansion. In our algorithm we judiciously avoid the preprocessing step to save the factor 3 expansion. At first glance, it is not entirely clear how to do the rounding without preprocessing, as the preprocessed solution has several "nice" properties. Nevertheless, we partition the set of points into two subsets and construct two auxilliary LPs. Using the initial fractional LP solution, we construct two feasible fractional solutions to these two LPs. We round these two solutions independently to obtain two integral solutions corresponding to the two subsets of points. For rounding the

first LP, we use an algorithm similar to the one in [4], but without preprocessing. We show that the constructed fractional LP solution has equally nice properties so that the algorithm in [4] can be extended in this case. For rounding the second LP, we use a rather simple algorithm.

The sets of balls involved in two LPs are not necessarily disjoint, and thus a ball can be selected in both of the solutions. But, taking multiple copies of a ball is not allowed. To resolve this issue, we first identify a subset of balls and allow only these balls to be involved in both solutions. Moreover, we scale down the capacities of these balls by a suitable factor. This ensures that even if a ball is selected in both solutions, the total capacity used by the copies does not exceed the original capacity. Note that the scaling of capacities leads to a new issue that the capacities no longer satisfy the monotonicity property in general. However, we show that it is possible to overcome this issue by considering two classes of balls separately – one whose capacities remain unchanged and the other whose capacities are scaled down.

1.2 Related Work

Considering the hardness of MCC, researchers have studied the Euclidean version of the problem with the goal of obtaining better approximation. The dimension d of the space is assumed to be a constant. One interesting case is when the set $\mathcal{B}$ contains all possible unit balls, which appeared in the Sloan Digital Sky Survey project [25]. Ghasemi and Razzazi [18] have obtained a PTAS for this case. In the general Euclidean case the best known approximation factor is still $O(\log n)$. Bandyapadhyay et al. [4] showed that in this special case of MCC only $1+\epsilon$ expansion of the balls is sufficient to obtain a constant approximation.

MCC is a special version of *Capacitated Set Cover* (CSC). CSC is similar to set cover except each set S_i has a capacity U_i. Moreover, we want to find an assignment of the points to the chosen subfamily of sets such that each element is assigned to a set it is in and at most U_i elements are assigned to each set S_i. CSC is a well-studied problem. Wolsey [28] designed a greedy algorithm for CSC that achieves a tight $O(\log n)$-approximation. Capacitated vertex cover is another special case of CSC, where each element is contained in exactly two sets. A 3-approximation for this problem was given by Chuzhoy and Naor [12]. The approximation factor was subsequently improved to 2 by Gandhi et al. [17]. The generalization where each element belongs to at most a bounded number of sets is also well-studied [20, 29].

The uncapacitated version of MCC (*Metric Uncapacitated Covering* (MUC)), where each set can be assigned with any number of points is another extensively studied problem. Note that the same bicriteria hardness of MCC mentioned above holds even for MUC. But, using a simple LP rounding scheme one can obtain a $(1, 3)$ bicriteria approximation for this problem. The MUC problem in the fixed-dimensional Euclidean space also has received huge attention from the researchers. Brönnimann and Goodrich [7] have designed an $O(1)$-approximation for this problem in the plane. In a celebrated work, Mustafa and Ray [26] improved this result by obtaining a PTAS for the problem. In dimension more than 2, the problem is notoriously hard and the best known approximation is $O(\log n)$. Considering this situation Har-Peled and Lee [19] gave a $(1+\epsilon, 1+\epsilon)$ bicriteria approximation.

Capacitated clustering and facility location problems are another set of interesting and well-studied problems. One such interesting problem is capacitated k-center. $O(1)$-approximations are known both for the uniform [6, 21] and non-uniform [2, 14] version of this problem. Another popular clustering problem is capacitated k-median for which no $O(1)$-approximation is known so far. Seemingly the existing techniques are not capable of handling the combination of the global constraint on the number of centers and the capacity constraint. Indeed, if either of these constraints is allowed to be violated by an

$O(1)$ factor, $O(1)$-approximations are known in those cases [9, 8, 10, 13, 15, 23, 24]. For capacitated facility location $O(1)$-approximations are known based on local search paradigm [1, 5, 11, 22, 27] and rounding of LP [3].

1.3 Paper Outline

In Section 2 we describe the natural LP for MMCC and have some definitions, which will be useful throughout the paper. In Section 3 we give an overview of the algorithm of [4]. Our LP rounding algorithm for MMCC and the analysis appear in Section 4. In Section 5 we show how to modify our algorithm for MMCC in the uniform case to obtain the improved bound. Finally, in Section 6 we conclude with some open problems.

2 Preliminaries

Recall that in MMCC we are given a set of points P and a set of balls $\mathcal{B}$. The capacity of each ball $B_i \in \mathcal{B}$ is U_i. Also, these capacities satisfy monotonicity, i.e., for any two balls B_i and B_j, if $r_i \geq r_j$, $U_i \geq U_j$.

The relaxation of the natural LP for MMCC is shown in the following. In the LP for MMCC, we have a variable y_i for each ball $B_i \in \mathcal{B}$ that indicates whether B_i is in the solution ($y_i = 1$) or not ($y_i = 0$). For each ball B_i and each point $p_j \in P$, there is a variable x_{ij} that indicates whether p_j is assigned to B_i ($x_{ij} = 1$) or not ($x_{ij} = 0$). Constraint 1 ensures that if a point is assigned to a ball, the ball must be selected in the solution. Constraint 2 ensures that the total number of points assigned to B_i is at most U_i. Constraint 3 ensures that each point is assigned to exactly one ball. Constraint 4 ensures that if a point p_j is assigned to a ball B_i, p_j must be contained in B_i. The remaining constraints are relaxed in MMCC-LP, which define the domains of the variables. We note that the LP relaxation for MCC is same as MMCC-LP except there all the U_i are equal.

$$\text{minimize} \quad \sum_{B_i \in \mathcal{B}} y_i \qquad \text{(MMCC-LP)}$$

$$\text{s.t.} \quad x_{ij} \leq y_i \qquad \forall p_j \in P,\ \forall B_i \in \mathcal{B} \qquad (1)$$

$$\sum_{p_j \in P} x_{ij} \leq y_i \cdot U_i \qquad \forall B_i \in \mathcal{B} \qquad (2)$$

$$\sum_{B_i \in \mathcal{B}} x_{ij} = 1 \qquad \forall p_j \in P \qquad (3)$$

$$x_{ij} = 0 \qquad \forall p_j \in P,\ \forall B_i \in \mathcal{B} \text{ such that } p_j \notin B_i \qquad (4)$$

$$x_{ij} \geq 0 \qquad \forall p_j \in P,\ \forall B_i \in \mathcal{B} \qquad (5)$$

$$0 \leq y_i \leq 1 \qquad \forall B_i \in \mathcal{B} \qquad (6)$$

We denote any solution to MMCC-LP by (x, y). To distinguish between two different solutions, we use different annotations with x and y. The cost of (x, y) is defined as, $\text{cost}(x, y) = \sum_{B_i \in \mathcal{B}} y_i$. For an integral solution, the cost is exactly the number of balls in the solution. Consider any solution (x, y) to MMCC-LP. For a ball B_i and a point p_j, if $x_{ij} > 0$, we say B_i *serves* p_j and p_j receives x_{ij} amount of *flow* from B_i. The *flow out* of B_i is the total amount of flow $\sum_{p_j \in P} x_{ij}$ that B_i gives to all the points. Next, we define an operation that we call "reroute". For a point p_j and two balls B_i and B_ℓ, rerouting of f amount of flow for p_j from B_i to B_ℓ means we increase $x_{\ell j}$ by f and decrease x_{ij} by f. For two balls B_i and B_ℓ, rerouting of flow from B_i to B_ℓ means for each point p_j served by B_i,

we reroute x_{ij} amount of flow for p_j from B_i to B_ℓ. Thus, the flow out of B_i becomes 0 after this operation. For a point p_j, a set of balls S and a ball $B_\ell \notin S$, rerouting of f amount of flow from the balls in S to B_ℓ means we increase $x_{\ell j}$ by f and decrease x_{ij} by $f_i \geq 0$ for each $B_i \in S$ such that $\sum_{B_i \in S} f_i = f$.

3 Overview of the Algorithm of [4]

Our algorithm is based on the algorithm of [4]. In this section we give an overview of the algorithm of [4]. Let (x, y) be a feasible solution to MMCC-LP. The LP rounding algorithm of [4] rounds the solution so that y values of all the balls become integral. We note that it is sufficient to obtain such a solution. Indeed, as all the capacities are integral, it is possible to find another solution with the same y values where all the x values are also integral [12]. The algorithm has three major steps. The first step is the preprocessing step. Fix a $0 < \alpha \leq 3/8$. A ball B_i is called *heavy* if $y_i = 1$ and *light* if $0 \leq y_i \leq \alpha$. Let $\mathcal{H}$ and $\mathcal{L}$ be the respective set of heavy and light balls. We note that the sets of heavy and light balls are always defined w.r.t. an LP solution. But, for simplicity we do not explicitly mention that in the notations $\mathcal{H}$ and $\mathcal{L}$. The implicit solution w.r.t. which $\mathcal{H}$ and $\mathcal{L}$ are defined can be easily derived from the context. Now, it might not be true that for all $p_j \in P$, the sum of the y values of the balls in $\mathcal{L}$ that serve p_j is at most α. In the preprocessing step, the algorithm of [4] modifies the computed LP solution to obtain another LP solution such that the above mentioned property is satisfied. In particular, they prove the following lemma.

▶ **Lemma 1** (Lemma 3.1 of [4]). *Given a feasible LP solution $\sigma = (x, y)$, and a parameter $0 < \alpha \leq \frac{3}{8}$, there exists a polynomial time algorithm to obtain another LP solution $\overline{\sigma} = (\overline{x}, \overline{y})$ that satisfies all the constraints of MMCC-LP (Constraints 1-6), except Constraint 4. Additionally, $\overline{\sigma}$ satisfies the following properties.*

1. *Any ball $B_i \in \mathcal{B}$ with non-zero $\overline{y_i}$ is either heavy ($\overline{y_i} = 1$) or light ($0 < \overline{y_i} \leq \alpha$).*
2. *For each point $p_j \in P$, we have that*

$$\sum_{B_i \in \mathcal{L}: \overline{x}_{ij} > 0} \overline{y_i} \leq \alpha, \tag{7}$$

   where $\mathcal{L}$ is the set of light balls with respect to $\overline{\sigma}$.
3. *For any heavy ball B_i, and any point $p_j \in P$ served by B_i, $d(c_i, p_j) \leq 3r_i$.*
4. *For any light ball B_i, and any point $p_j \in P$ served by B_i, $d(c_i, p_j) \leq r_i$.*
5. $\text{cost}(\overline{\sigma}) \leq \frac{1}{\alpha}\text{cost}(\sigma)$.

Note that a point p_j can be fractionally assigned by the algorithm in Lemma 1 to a heavy ball B_i even if $p_i \notin B_i$, but, in this case $d(c_i, p_j)$ must be at most $3r_i$. Hence, a factor 3 expansion of the ball is sufficient for it to serve the point. In summary, the preprocessing step implicitly incurs an expansion factor of 3 for the heavy balls with respect to the new LP solution $\overline{\sigma}$. We also note that the preprocessing algorithm uses the fact that the capacities are monotonic.

The second step of the algorithm is the key step and is called Cluster Formation. In the following, we give an overview of this step. The algorithm maintains an LP solution $\overline{\sigma} = (\overline{x}, \overline{y})$ which is initially the output of the preprocessing step. This solution is essentially altered throughout the step and when the step finishes $\overline{y}_i \in \{0, 1\}$ for all $B_i \in \mathcal{B}$. Each heavy ball B_i forms a cluster which initially consists of itself ($\{B_i\}$). For any light ball B_t, either B_t is opened fully in the solution or it joins a cluster of a heavy ball by rerouting its flow to the heavy ball. The algorithm runs for several iterations until the fate of all these light balls are decided.

In each iteration, every heavy ball uses its available capacity to reroute the flow of as many intersecting light balls as possible to itself. Each such light ball joins the cluster of the heavy ball. From the remaining light balls whose fate are not yet decided, a ball is selected greedily to be included in the solution. Also, for points inside the selected ball, an appropriate amount of flow is rerouted from other balls to this ball to utilize its capacity. We skip the details of this flow rerouting in this overview. This completes the overview of the step.

Note that the flow rerouting from heavy balls to a light ball when the light ball is opened fully, is an essential component of the analysis for obtaining the constant factor guarantee on the size of the solution. Consider a light ball B_t which is selected for opening fully and assume that it serves $k_t \leq U_t$ points. Then, we can set the $\overline{x}_{tj}$ value for each of these k_t points to 1, i.e., we fully assign p_j to B_t. Note that preprocessing ensures that $\sum_{B_i \in \mathcal{L}: \overline{x}_{ij} > 0} \overline{y_i} \leq \alpha$ or $\sum_{B_i \in \mathcal{H}: \overline{x}_{ij} > 0} \overline{y_i} \geq 1 - \alpha$. Thus, when these points are fully assigned to B_t, at least $(1-\alpha)k_t$ amount of flow is rerouted from the heavy balls to B_t which they can now use to reroute flow from other light balls. This argument is essential in the analysis. Now, we have an observation which follows due to the way light balls are added to a cluster.

▶ **Observation 2.** *Consider a cluster of a heavy ball B_h that contains the light balls $B_1, \ldots, B_\ell$. Then, when the Cluster Formation finishes,*

1. *For each $1 \leq i \leq \ell$, there is a point p_j such that $p_j \in B_h \cap B_i$.*
2. *$\sum_{i=1}^{\ell} \sum_{j \in P} \overline{x}_{ij} \leq U_h - \sum_{j \in P} \overline{x}_{hj}$, i.e., the total amount of flow out of the balls in the cluster of B_h is at most U_h.*

The third step is called Selection of Balls. In this step, from each cluster a ball is carefully selected and expanded so that it can serve all the points served by the balls in the cluster. For a cluster of a heavy ball B_h, if it is the largest ball in the cluster then B_h is selected and with three factor expansion it can serve all the points served by the cluster. As during preprocessing the heavy ball might have been expanded by a factor of 3, its total expansion factor is 9. If B_h is not the largest ball, the largest ball B_ℓ is a light ball of the cluster. Then, we select this light ball and expand by a factor of 5 so that it can serve all the points served by the cluster. The light ball can serve the total flow assigned to the cluster, as $U_\ell \geq U_h$ due to monotonicity. This is another place where the monotonicity assumption on the capacities is necessary.

The following lemma that states the guarantee achieved by the above algorithm follows due to the analysis of [4].

▶ **Lemma 3.** *There is a $(6 + 5\alpha)/\alpha$-approximation for MMCC that expands the balls by at most a factor of 9.*

4 The Modified Algorithm for MMCC

In this section, we describe our algorithm. Note that among the 9 factor expansion needed in the algorithm of [4] 3 factor is contributed by the preprocessing step. Our algorithm avoids this preprocessing step to save this factor 3 expansion.

Fix $0 < \alpha \leq 1/60$. We first compute a fractional LP solution $\sigma^* = (x^*, y^*)$ to MMCC-LP. Set $y_i = 1$ if $y_i^* > \alpha$, otherwise $y_i = y_i^*$. Also, set $x = x^*$. Note that $\sigma = (x, y)$ is a feasible solution to MMCC-LP such that $\text{cost}(\sigma) \leq \text{cost}(\sigma^*)/\alpha$. We define the sets $\mathcal{H}$ and $\mathcal{L}$ of heavy and light balls w.r.t. σ in the same way, i.e., $\mathcal{H} = \{B_i \mid y_i = 1\}$ and $\mathcal{L} = \{B_i \mid 0 < y_i \leq \alpha\}$. Note that in σ, any ball that gives some flow to a point is either a heavy or a light ball. We take one copy of the set of heavy balls and two copies of the set of light balls. Let these sets be $\mathcal{H}_1$, $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively.

Next, we partition the point set into two subsets. Let P_1 be the subset of points such that $p_j \in P_1$ if $\sum_{B_i \in \mathcal{L}} x_{ij} \leq 4\alpha$, i.e., p_j gets a flow of at most 4α from the balls in $\mathcal{L}$. Let $P_2 = P \setminus P_1$. Based on these sets P_1, P_2, we are going to construct two LP solutions to two auxilliary LPs and round them independently. Finally, we combine these two solutions to find a solution to MMCC-LP where for each $B_i \in \mathcal{B}$, $y_i \in \{0, 1\}$. Intuitively, we satisfy the demands of these two sets of points independently. The light balls are involved in both of the solutions and they might get opened fully in both of the solutions. However, we are not allowed to open multiple copies of a ball. To avoid this situation we reduce the capacity of the light balls by appropriate factor in the auxilliary LP.

Let the new capacity $U_i' = U_i/10$ for each light ball B_i. The new capacity of each heavy ball B_i remains same as before, i.e., $U_i' = U_i$. At this point the reader might wonder about the value of the scaling factor. We note that it is carefully chosen through back calculation to ensure that the analysis goes through. The first auxilliary LP that we consider is as follows.

$$\text{minimize} \quad \sum_{B_i \in \mathcal{L}_1 \cup \mathcal{H}_1} y_i \tag{AUX-LP1}$$

$$\text{s.t.} \quad x_{ij} \leq y_i \qquad \forall p_j \in P_1,\ \forall B_i \in \mathcal{L}_1 \cup \mathcal{H}_1 \tag{8}$$

$$\sum_{p_j \in P_1} x_{ij} \leq y_i \cdot U_i' \qquad \forall B_i \in \mathcal{L}_1 \cup \mathcal{H}_1 \tag{9}$$

$$\sum_{B_i \in \mathcal{L}_1 \cup \mathcal{H}_1} x_{ij} = 1 \qquad \forall p_j \in P_1 \tag{10}$$

$$x_{ij} = 0 \qquad \forall p_j \in P_1,\ \forall B_i \in \mathcal{L}_1 \cup \mathcal{H}_1 \text{ such that } p_j \notin B_i \tag{11}$$

$$x_{ij} \geq 0 \qquad \forall p_j \in P_1,\ \forall B_i \in \mathcal{L}_1 \cup \mathcal{H}_1 \tag{12}$$

$$0 \leq y_i \leq 1 \qquad \forall B_i \in \mathcal{L}_1 \cup \mathcal{H}_1 \tag{13}$$

Note that the above LP has a variable y_i for each ball B_i in $\mathcal{L}_1 \cup \mathcal{H}_1$, and a variable x_{ij} for each ball B_i in $\mathcal{L}_1 \cup \mathcal{H}_1$ and each point $p_j \in P_1$. We are not going to solve this LP. Instead, we construct a solution to this LP using σ and round it using an algorithm similar to the one in [4]. This LP is used to compare the cost of the rounded solution and the cost of σ^* in the end.

We construct an LP solution $\overline{\sigma} = (\overline{x}, \overline{y})$ from σ in the following manner. For $B_i \in \mathcal{H}_1$, $\overline{y}_i = y_i$. For $B_i \in \mathcal{L}_1$, $\overline{y}_i = 10 \cdot y_i \leq 10\alpha < 1$ ($\alpha \leq 1/60$). For $p_j \in P_1$, $B_i \in \mathcal{L}_1 \cup \mathcal{H}_1$, $\overline{x}_{ij} = x_{ij}$.

▶ **Lemma 4.** *$\overline{\sigma} = (\overline{x}, \overline{y})$ is a feasible solution to AUX-LP1 with cost at most $\text{cost}(\sigma^*)/\alpha$.*

Proof. First note that,

$$\text{cost}(\overline{\sigma}) = \sum_{B_i \in \mathcal{H}_1} y_i + 10 \sum_{B_i \in \mathcal{L}_1} y_i \leq (1/\alpha) \sum_{B_i \in \mathcal{H}_1} y_i^* + 10 \sum_{B_i \in \mathcal{L}_1} y_i^* \leq \text{cost}(\sigma^*)/\alpha.$$

For $p_j \in P_1$, $B_i \in \mathcal{L}_1 \cup \mathcal{H}_1$, $\overline{x}_{ij} = x_{ij} \leq y_i \leq \overline{y}_i$. Thus, Constraint 8 is satisfied.

For $B_i \in \mathcal{H}_1$, $\sum_{p_j \in P_1} \overline{x}_{ij} = \sum_{p_j \in P_1} x_{ij} \leq y_i \cdot U_i = \overline{y}_i \cdot U_i'$. For $B_i \in \mathcal{L}_1$, $\sum_{p_j \in P_1} \overline{x}_{ij} = \sum_{p_j \in P_1} x_{ij} \leq y_i \cdot U_i = (10 \cdot y_i) \cdot (U_i/10) = \overline{y}_i \cdot U_i'$. Thus, Constraint 9 is satisfied.

For $p_j \in P_1$, $\sum_{B_i \in \mathcal{L}_1 \cup \mathcal{H}_1} \overline{x}_{ij} = \sum_{B_i \in \mathcal{L}_1 \cup \mathcal{H}_1} x_{ij} = 1$. Thus, Constraint 10 is satisfied. Also, it is trivial to verify that Constraints 11-13 are also satisfied. Hence, the lemma follows. ◀

Next, we describe our second auxilliary LP. Let us again consider the solution $\sigma = (x, y)$ to MMCC-LP and the set of light balls $\mathcal{L}$ w.r.t. σ. Also, consider the second copy $\mathcal{L}_2$ of the set of light balls. For each point p_j in P_2, define the demand $d_j = \sum_{B_i \in \mathcal{L}_2} x_{ij}$.

$$\text{minimize} \quad \sum_{B_i \in \mathcal{L}_2} y_i \tag{AUX-LP2}$$

$$\text{s.t.} \quad x_{ij} \leq y_i \qquad \forall p_j \in P_2,\ \forall B_i \in \mathcal{L}_2 \tag{14}$$

$$\sum_{p_j \in P_2} x_{ij} \leq y_i \cdot U'_i \qquad \forall B_i \in \mathcal{L}_2 \tag{15}$$

$$\sum_{B_i \in \mathcal{L}_2} x_{ij} \geq d_j \qquad \forall p_j \in P_2 \tag{16}$$

$$x_{ij} = 0 \qquad \forall p_j \in P_2,\ \forall B_i \in \mathcal{L}_2 \text{ such that } p_j \notin B_i \tag{17}$$

$$x_{ij} \geq 0 \qquad \forall p_j \in P_2,\ \forall B_i \in \mathcal{L}_2 \tag{18}$$

$$0 \leq y_i \leq 1 \qquad \forall B_i \in \mathcal{L}_2 \tag{19}$$

Note that the above LP has a variable y_i for each ball B_i in $\mathcal{L}_2$ and a variable x_{ij} for each ball B_i in $\mathcal{L}_2$ and each point $p_j \in P_2$. Again we are not going to solve this LP. Instead, we construct a solution to this LP using σ and round it. This LP is used to compare the cost of the rounded solution and the cost of σ^* in the end.

We construct an LP solution $\hat{\sigma} = (\hat{x}, \hat{y})$ from σ in the following manner. For $B_i \in \mathcal{L}_2$, $\hat{y}_i = 10 \cdot y_i \leq 10\alpha < 1$. For $p_j \in P_2$, $B_i \in \mathcal{L}_2$, $\hat{x}_{ij} = x_{ij}$.

▶ **Lemma 5.** *$\hat{\sigma} = (\hat{x}, \hat{y})$ is a feasible solution to AUX-LP2 with cost at most $10 \cdot \text{cost}(\sigma^*)$.*

Proof. First note that $\text{cost}(\hat{\sigma}) \leq 10 \sum_{B_i \in \mathcal{L}_2} y_i = 10 \sum_{B_i \in \mathcal{L}_2} y^*_i \leq 10 \cdot \text{cost}(\sigma^*)$. For $p_j \in P_2$, $B_i \in \mathcal{L}_2$, $\hat{x}_{ij} = x_{ij} \leq y_i < \hat{y}_i$. Thus, Constraint 14 is satisfied.

For $B_i \in \mathcal{L}_2$, $\sum_{p_j \in P_2} \hat{x}_{ij} = \sum_{p_j \in P_2} x_{ij} \leq y_i \cdot U_i = (10 \cdot y_i) \cdot (U_i/10) = \hat{y}_i \cdot U'_i$. Thus, Constraint 15 is satisfied.

For $p_j \in P_2$, $\sum_{B_i \in \mathcal{L}_2} \hat{x}_{ij} = \sum_{B_i \in \mathcal{L}_2} x_{ij} = d_j$. Thus, Constraint 16 is satisfied. Also, it is trivial to verify that Constraints 17-19 are also satisfied. Hence, the lemma follows. ◀

In the following, we give two algorithms for rounding the two auxilliary LPs. The rounded solution of the first LP satisfies all the constraints except the coverage constraint. The rounded solution of the second LP satisfies all the constraints except the coverage and capacity constraints. Then, we merge these two solutions to obtain a solution for MMCC-LP that does not violate any capacity constraints.

4.1 Rounding the First Auxilliary LP

Note that we are given a feasible LP solution $\overline{\sigma} = (\overline{x}, \overline{y})$ to AUX-LP1 that has the following properties.

1. For any $B_i \in \mathcal{H}_1$, $\overline{y}_i = 1$.
2. For any $B_i \in \mathcal{L}_1$, $\overline{y}_i \leq 10\alpha$.
3. For any $p_j \in P_1$, $\sum_{B_i \in \mathcal{H}_1} \overline{x}_{ij} \geq 1 - 4\alpha$.
4. $\text{cost}(\overline{\sigma}) \leq \text{cost}(\sigma^*)/\alpha$.

Note that Property (3) above states that for any point $p_j \in P_1$, the flow received by p_j from the balls in $\mathcal{H}_1$ is at least $1 - 4\alpha$. We will heavily use this property while performing the rounding. Indeed, we are going to use an algorithm similar to the one in [4] without the preprocessing step. In the algorithm of [4], preprocessing ensures that for any point p_j, the sum of the y values of the light balls that give non-zero flow to p_j is at most α. Note that this might not be true in our case for balls in $\mathcal{L}_1$. At first glance it is not clear how to do the

rounding without this assumption. However, as we show, a similar rounding scheme can be designed using the weaker assumption on the flow mentioned above. Another hurdle to adapt the algorithm of [4] is the monotonicity assumption, which might not be true in our case because of scaling of the capacities. However, we note that only light balls' capacities are scaled by a uniform constant scaling factor. Due to this fact, we show that their algorithm can be modified to handle our case. Next, we describe our rounding algorithm.

The first step in our algorithm is Cluster Formation. In this step, for each ball $B_i \in \mathcal{L}_1$, either B_i is opened fully (added to a set $\mathcal{O}$) and flow from other balls including the balls in $\mathcal{H}_1$ are rerouted to B_i only for points in B_i. Otherwise, B_i joins a cluster of a ball in $\mathcal{H}_1$ to which its entire flow is rerouted. $\mathcal{O}$ is initialized to the empty set. For each ball $B_i \in \mathcal{H}_1$, initialize the cluster of B_i, cluster(B_i) to $\{B_i\}$. During the course of the algorithm, let $\Lambda \subseteq \mathcal{L}_1$ be the set of balls which are not yet added to $\mathcal{O}$ or to a cluster of a ball in $\mathcal{H}_1$. Throughout the algorithm, we maintain the invariant that for any point p_j which is served by a ball in Λ, p_j receives a flow of at least $1 - 4\alpha$ from the balls in $\mathcal{H}_1$. Note that in the beginning of the algorithm this is true, as $\Lambda = \mathcal{L}_1$. At any point, the available capacity of a ball B_i, $AC(B_i) = U'_i - \sum_{j \in P_1} \overline{x}_{ij}$. While the set Λ is non-empty, apply the following steps.

While there is a ball $B_i \in \mathcal{H}_1$ and $B_{i'} \in \Lambda$ such that B_i intersects $B_{i'}$ and $AC(B_i)$ is at least the flow out $\sum_{j \in P_1} \overline{x}_{i'j}$ of $B_{i'}$, reroute the flow from $B_{i'}$ to B_i. Add $B_{i'}$ to cluster(B_i). If Λ becomes empty at this point, go to the Selection of Balls stage.

For any ball $B_j \in \Lambda$, let $\mathcal{A}_j$ be the set of points currently being served by B_j. Also, let $k_j = \min\{U'_j, |\mathcal{A}_j|\}$. We add a ball $B_t \in \Lambda$ to $\mathcal{O}$ such that k_t is the maximum over all k_j for $B_j \in \Lambda$.

Next we assign points up to larger extents to B_t to utilize its capacity. There are three cases.

1. $k_t > 2$. Note that the flow out of B_t, $\sum_{j \in P_1} \overline{x}_{tj} \leq 10\alpha U'_t$. Also, as $\overline{x}_{tj} = x_{tj} \leq y_t \leq \alpha$, $\sum_{j \in P_1} \overline{x}_{tj} \leq \alpha |\mathcal{A}_t| \leq 10\alpha |\mathcal{A}_t|$. Thus, $AC(B_t) \geq (1 - 10\alpha)k_t$. In this case, we arbitrarily select $\lfloor (1 - 10\alpha)k_t \rfloor$ points served by B_t and for each of those points p_ℓ, we reroute the maximum (whole) amount of flow possible from all other balls to B_t. Note that p_ℓ is no longer served by a ball in Λ, and thus the invariant is satisfied.
2. $1 \leq k_t \leq 2$. If $U'_t \geq |\mathcal{A}_t|$, then $|\mathcal{A}_t| = k_t$. In this case, for each of the k_t points served by B_t, we reroute the maximum amount of flow possible from all other balls to B_t. In the other case, $U'_t < |\mathcal{A}_t|$. Now, $AC(B_t) \geq (1 - 10\alpha)U'_t \geq 1 - 10\alpha$. The last inequality follows, as $U'_t \geq 1$. We arbitrarily select a point p_ℓ that is being served by B_t and reroute its flow from Λ to B_t. Let f be the amount of flow that now p_ℓ receives from B_t. Note that $f \leq 4\alpha$. Also, p_ℓ is no longer served by a ball in Λ. Now, $AC(B_t) \geq 1 - 10\alpha - 4\alpha = 1 - 14\alpha$. We reroute $\min\{AC(B_t), 1 - f\}$ amount of flow from $\mathcal{H}_1$ to B_t for p_ℓ. In any case, the points whose flow are routed to B_t in this step are no longer served by a ball in Λ, and thus the invariant is satisfied.
3. $0 < k_t < 1$. Note that, as $|\mathcal{A}_t| \geq 1$, $k_t = U'_t < 1$. Now, $AC(B_t) \geq (1 - 10\alpha)U'_t$. Consider any arbitrary point p_ℓ that is being served by B_t. First, reroute its flow from Λ to B_t. $AC(B_t) \geq (1 - 10\alpha)U'_t - 4\alpha$. Note that after this rerouting, p_ℓ is no longer served by balls in Λ, and thus the invariant is satisfied. Let p_ℓ gets a flow of f_1 from the balls in $\mathcal{H}_1$. By the invariant we maintain, f_1 is at least $1 - 4\alpha$. Reroute $\min\{AC(B_t), f_1\}$ amount of flow of p_ℓ from the balls in $\mathcal{H}_1$ to B_t.

When the while loop terminates each ball in $\mathcal{L}_1$ is either in $\mathcal{O}$ or added to a cluster. For each $B_i \in \mathcal{O}$, we set $\overline{y}_i = 1$ and cluster$(B_i) = \{B_i\}$.

We note that the third case ($0 < k_t < 1$) mentioned above does not occur in the context of [4], as in their case for each ball B_j, both U_j and $|\mathcal{A}_j|$ are at least 1. This case appears to be the bottleneck for our algorithm and leads to a larger constant of approximation as we will describe in the analysis.

The Selection of Balls step is more interesting in our case as the monotonicity property no longer holds in general. For a cluster of a ball in $\mathcal{O}$, we trivially select this ball. Consider the cluster of any ball $B_h \in \mathcal{H}_1$. If B_h is one of the top 10 largest balls in the cluster, then select all the balls larger than B_h and also B_h. Only B_h is expanded by a factor of 3. The flow rerouted from any selected ball of $\mathcal{L}_1$ to B_h in the Cluster Formation step is assigned to it. Note that for the remaining balls of $\mathcal{L}_1$ which are in the same cluster and not chosen, are smaller than B_h, and thus can be covered by a factor 3 expansion of B_h. The remaining flow is assigned to B_h. Otherwise, the top 10 largest balls are selected all of which are in $\mathcal{L}_1$. The flow rerouted from any selected ball to B_h in the Cluster Formation step is assigned to the ball. Now consider the remaining flow assigned to the cluster. Also consider a point p_j which receives a part of this flow and not in any of the selected balls. Then, by 5 factor expansion, any selected ball can cover p_j. We expand each selected ball by 5 factor and the remaining flow is assigned arbitrarily to selected balls respecting their capacity.

4.1.1 Analysis

Let I be the number of iterations of the outermost while loop. Also, let L_t be the ball of $\mathcal{L}_1$ added to $\mathcal{O}$ at iteration $1 \leq t \leq I$. For a ball $B_i \in \mathcal{H}_1$, let $F(L_t, B_i)$ be the amount of flow rerouted from B_i to L_t. Let $F_t = \sum_{B_i \in \mathcal{H}_1} F(L_t, B_i)$. The next lemma states that when L_t is added to $\mathcal{O}$ sufficient amount of flow is rerouted from the balls in $\mathcal{H}_1$ to L_t irrespective of the value of k_t.

▶ **Lemma 6.** *For $1 \leq i \leq I$, $F_t \geq k_t/60$ for $\alpha \leq 1/60$.*

Proof. To compute the flow rerouted from balls in $\mathcal{H}_1$ to B_t we refer to the three cases mentioned in Cluster Formation. In the first case, for $\lfloor (1-10\alpha)k_t \rfloor$ points, the flow is rerouted from $\mathcal{H}_1$ to B_t. Note that by the invariant we maintain, for each such point p_ℓ, p_ℓ receives at least $1-4\alpha$ amount of flow from the balls in $\mathcal{H}_1$. It follows that, at least $1-4\alpha$ amount of flow is rerouted for p_ℓ and $F_t \geq (1-4\alpha)\lfloor (1-10\alpha)k_t \rfloor \geq (14/15)\lfloor (5/6)k_t \rfloor \geq (14/15)(1/14)k_t = k_t/15 \geq k_t/60$. The second inequality follows as $\alpha \leq 1/60$ and the third inequality follows as $k_t > 2$.

In the second case, using the same argument as above, the amount of flow rerouted from $\mathcal{H}_1$ to B_t is at least $1-14\alpha$. As $k_t \leq 2$, F_t is at least $(1-14\alpha)k_t/2 \geq (23/60)k_t \geq k_t/60$. The first inequality is true for $\alpha \leq 1/60$.

In the third case, again using the same argument as above, the amount of flow rerouted from $\mathcal{H}_1$ to B_t is at least $\min\{(1-10\alpha)k_t - 4\alpha, 1-4\alpha\}$. As $k_t < 1$, $1-4\alpha \geq (1-4\alpha)k_t$. Thus, $F_t \geq (1-10\alpha)k_t - 4\alpha$. As $U_t \geq 1$, $k_t = U'_t \geq 1/10$, and hence $F_t \geq (1-10\alpha)/10 - 4\alpha = 1/10 - 5\alpha \geq 1/60$. The last inequality follows from the fact that $\alpha \leq 1/60$. ◀

Define the y-credit of a ball $B_i \in \mathcal{H}_1$ as $Y(L_t, B_i) = F(L_t, B_i)/k_t$. At any moment during the Cluster Formation stage, define the y-accumulation of B_i as $\tilde{y}(B_i) = \sum_{L_t \in \mathcal{O}} Y(L_t, B_i) - \sum_{B_i \in \mathcal{L}_1 \cap \text{cluster}(B_i)} \overline{y}_i$. The y-credit $Y(L_t, B_i)$ of B_i can be seen as a normalized load it transfers to L_t. The y-accumulation $\tilde{y}(B_i)$ is basically the difference between the total y-credit received by B_i and the sum of normalized flows of the balls absorbed by B_i. The next lemma gives a lower bound on the available capacities of the balls in $\mathcal{H}_1$, which is similar to Lemma 3.3 of [4].

▶ **Lemma 7.** *Consider a ball $B_i \in \mathcal{H}_1$ and any integer $1 \leq t \leq I$. Suppose the balls $L_1, \ldots, L_t$ have been added to $\mathcal{O}$ so far. Then, $AC(B_i) \geq \tilde{y}(B_i)k_t$.*

Proof. For any ball $B_i \in \mathcal{H}_1$, we prove the claim using induction on iteration number. In the base case, just after addition of L_1, $AC(B_i) \geq F(L_1, B_i) = Y(L_1, B_i)k_1 = \tilde{y}(B_i)k_1$. Now, suppose the claim is true for any $t-1$. We show that the claim is true for t as well.

Consider the iteration t. Note that $AC(B_i) \geq \tilde{y}(B_i)k_{t-1}$. Suppose a subset of balls have joined cluster of B_i. Let B_p be the first ball joined, which serves k points. To distinguish between the old and new value of $\tilde{y}(B_i)$, we refer to the new value by $\tilde{y}(B_i)'$. After B_p's joining to cluster of B_i, $\tilde{y}(B_i)' = \tilde{y}(B_i) - \overline{y}_p$. Now, the total flow out of B_p is at most $\min\{\overline{y}_p k, \overline{y}_p U_p'\} = \overline{y}_p \min\{k, U_p'\} \leq \overline{y}_p k_{t-1}$. Thus, $AC(B_i) \geq \tilde{y}(B_i)k_{t-1} - \overline{y}_p k_{t-1} = \tilde{y}(B_i)'k_{t-1}$. Using the same argument it can be shown that after each subsequent addition of a ball to cluster of B_i the claim is true.

In the next step, L_t is added to $\mathcal{O}$. Let $\tilde{y}(B_i)$ be the y-accumulation before this. After this addition, the new y-accumulation $\tilde{y}(B_i)' = \tilde{y}(B_i) + Y(L_t, B_i)$. If $\tilde{y}(B_i) \leq 0$, the new available capacity $A_i' \geq Y(L_t, B_i)k_t \geq \tilde{y}(B_i)'k_t$. Otherwise, $\tilde{y}(B_i) > 0$, the new available capacity by the induction hypothesis is, $A_i' = AC(B_i) + Y(L_t, B_i)k_t \geq \tilde{y}(B_i)k_{t-1} + Y(L_t, B_i)k_t \geq (\tilde{y}(B_i) + Y(L_t, B_i))k_t = \tilde{y}(B_i)'k_t$. ◀

The next lemma shows that for any ball $B_i \in \mathcal{H}_1$, y-accumulation is bounded, which is similar to Lemma 3.4 of [4].

▶ **Lemma 8.** *At any point, for any ball $B_i \in \mathcal{H}_1$, $\tilde{y}(B_i) < 1 + 10\alpha$.*

Intuitively, if the y-accumulation of B_i exceeds the bound, it must be due to selection of a ball L_t in $\mathcal{L}_1$. However, one can show that B_i had enough available capacity to absorb the flow from L_t. Hence, the bound follows.

The following lemma gives an upper bound on the number of balls of $\mathcal{L}_1$ that are fully opened.

▶ **Lemma 9.** *At the end of the Cluster Formation stage, $|\mathcal{O}| \leq 60((1+10\alpha)|\mathcal{H}_1| + \sum_{B_i \in \mathcal{L}_1} \overline{y}_i)$.*

Proof.

$$\begin{aligned}
\sum_{B_i \in \mathcal{H}_1} \tilde{y}(B_i) &= \sum_{B_i \in \mathcal{H}_1} \sum_{L_t \in \mathcal{O}} Y(L_t, B_i) - \sum_{B_i \in \mathcal{H}_1} \sum_{B_i \in \mathcal{L}_1 \cap \text{cluster}(B_i)} \overline{y}_i \\
&\geq \sum_{B_i \in \mathcal{H}_1} \sum_{L_t \in \mathcal{O}} F(L_t, B_i)/k_t - \sum_{B_i \in \mathcal{L}_1} \overline{y}_i \\
&= \sum_{t=1}^{I} F_t/k_t - \sum_{B_i \in \mathcal{L}_1} \overline{y}_i \\
&\geq |\mathcal{O}|/60 - \sum_{B_i \in \mathcal{L}_1} \overline{y}_i \qquad (F_t \geq k_t/60 \text{ by Lemma 6})
\end{aligned}$$

Also, by Lemma 8, $\sum_{B_i \in \mathcal{H}_1} \tilde{y}(B_i) \leq (1 + 10\alpha)|\mathcal{H}_1|$. It follows that, $|\mathcal{O}| \leq 60((1 + 10\alpha)|\mathcal{H}_1| + \sum_{B_i \in \mathcal{L}_1} \overline{y}_i)$. ◀

We obtain the following bound on the cost of the rounded solution.

▶ **Lemma 10.** *When the algorithm terminates the total cost of the solution is at most $10|\mathcal{H}_1| + |\mathcal{O}| \leq (70 + 600\alpha)\text{cost}(\sigma^*)/\alpha$.*

Proof. We note that from a heavy balls' cluster at most 10 balls are selected and all the balls in $\mathcal{O}$ are selected. Now, by Lemma 9,

$$\begin{aligned}10|\mathcal{H}_1| + |\mathcal{O}| &\leq 10|\mathcal{H}_1| + 60((1+10\alpha)|\mathcal{H}_1| + \sum_{B_i \in \mathcal{L}_1} \overline{y}_i) \\ &\leq (70 + 600\alpha)(|\mathcal{H}_1| + \sum_{B_i \in \mathcal{L}_1} \overline{y}_i) \\ &\leq (70 + 600\alpha)\text{cost}(\sigma^*)/\alpha\end{aligned}$$

◀

The following lemma shows that 5 factor expansion is sufficient to serve the points assigned to each cluster.

▶ **Lemma 11.** *Using factor 5 expansion of the balls the flow of any cluster can be assigned to the chosen balls without violating the capacities.*

Proof. It is clear from the algorithm that the coverage constraints are satisfied by expanding the balls by at most a factor of 5. Here we consider the capacity constraints. Note that in the first case the capacities of the selected light balls are trivially satisfied. Also, the remaining flow assigned to B_h must have an amount at most U_h due to the way balls are added to a cluster. Thus, its capacity constraint is satisfied. In the other case, let the total amount of flow rerouted from the selected 10 light balls to B_h in Cluster Formation step be f. Also, let B_ℓ be the smallest radius ball among these 10 balls. Thus, the available capacity of all these balls is at least $10U'_\ell - f$. Note that $U_h \leq U_\ell$, as B_ℓ is larger than B_h. Now, as each light balls' capacity is reduced to a factor 10 of the original capacity and the capacity of B_h remains unchanged, $U_h \leq 10U'_\ell$. Hence, the available capacity of all these 10 balls is at least $U_h - f$. As the remaining flow is at most $U_h - f$, it follows that the capacity constraints of these balls are satisfied. ◀

We summarize our findings in the following lemma.

▶ **Lemma 12.** *The solution $(\overline{x}, \overline{y})$ satisfies all the Constraints of AUX-LP1 except Constraint 11. Moreover,*

1. $\overline{y}_i = 1$ *for all* $B_i \in \mathcal{H}_1 \cup \mathcal{O}$ *and* $\overline{y}_i = 0$ *for all other balls.*
2. *For any* $p_j \in P_1$, $\sum_{B_i \in \mathcal{H}_1 \cup \mathcal{O}} \overline{x}_{ij} = 1$.
3. *For any point* $p_j \in P_1$, *if* $\overline{x}_{ij} > 0$, $d(c_i, p_j) \leq 5 \cdot r_i$.
4. $\text{cost}((\overline{x}, \overline{y})) \leq (70 + 600\alpha)\text{cost}(\sigma^*)/\alpha$.

4.2 Rounding the Second Auxilliary LP

Note that we are given a feasible LP solution $\hat{\sigma} = (\hat{x}, \hat{y})$ to AUX-LP2 that has the following properties.

1. For any $B_i \in \mathcal{L}_2$, $\hat{y}_i \leq 10\alpha$.
2. For any $p_j \in P_2$, $\sum_{B_i \in \mathcal{L}_2} \hat{x}_{ij} \geq 4\alpha$.
3. For any $p_j \in P_2$ and $B_i \in \mathcal{L}_2$, $\hat{x}_{ij} \leq \alpha$.
4. $\text{cost}(\hat{\sigma}) \leq 10 \cdot \text{cost}(\sigma^*)$.

First, we create a new solution to AUX-LP2 from $\hat{\sigma}$ which has cost at most two times that of $\hat{\sigma}$. We denote the new solution as well by $\hat{\sigma}$. Thus, for distinction, we denote the old values by $\hat{y}'_i$ and $\hat{x}'_{ij}$. For each y variable, its new value is twice the old value. Thus, $\hat{y}_i = 2\hat{y}'_i \leq 20\alpha < 1$. The last inequality follows for $\alpha \leq 1/60$. And, for each x variable, its new value is twice the old value. Thus, $\hat{x}_{ij} = 2\hat{x}'_{ij} \leq 2\alpha$. Note that, now, some points might receive flow of more than 1. We adjust the $\hat{x}$ values of these points so that each such point receives 1 amount of flow. We obtain the following lemma.

▶ **Lemma 13.** *There is a feasible LP solution $\hat{\sigma} = (\hat{x}, \hat{y})$ to AUX-LP2 that has the following properties.*

1. *For any $B_i \in \mathcal{L}_2$, $\hat{y}_i \leq 20\alpha$.*
2. *For any $p_j \in P_2$, $\sum_{B_i \in \mathcal{L}_2} \hat{x}_{ij} \geq 8\alpha$.*
3. *For any $p_j \in P_2$ and $B_i \in \mathcal{L}_2$, $\hat{x}_{ij} \leq 2\alpha$.*
4. *$\text{cost}(\hat{\sigma}) \leq 20 \cdot \text{cost}(\sigma^*)$.*

Proof. First note that $\text{cost}(\hat{\sigma}) \leq 20 \cdot \text{cost}(\sigma^*)$, as the values of the y variables are doubled. Next, we show that $\hat{\sigma}$ is feasible.

As the y variables are doubled and $\hat{x}_{ij} \leq 2\hat{x}'_{ij}$, $\hat{x}_{ij} \leq \hat{y}_i$. Thus, Constraint 14 is satisfied.

For $B_i \in \mathcal{L}_2$, $\sum_{p_j \in P_2} \hat{x}_{ij} \leq \sum_{p_j \in P_2} 2\hat{x}'_{ij} = 2\sum_{p_j \in P_2} \hat{x}'_{ij} \leq 2\hat{y}'_i \cdot U'_i = \hat{y}_i \cdot U'_i$. Thus, Constraint 15 is satisfied.

As we do not decrease the x variables, unless a point gets more than 1 amount of flow, Constraint 16 is also satisfied. Also, it is trivial to verify that Constraints 17-19 are also satisfied.

Properties 1, 3, and 4 follows immediately. Also, Property 2 follows from the fact that previously each point received a flow of at least 4α from the balls in $\mathcal{L}_2$. Hence, the lemma follows. ◀

We start with the fractional solution $\hat{\sigma} = (\hat{x}, \hat{y})$ and round it so that $\hat{y}$ becomes integral. Throughout our algorithm we modify $\hat{\sigma}$ over several steps to finally obtain the desired solution. Thus whenever we refer to $\hat{\sigma}$ we refer to its current value. For any $p_j \in P_2$, let $\delta_j = \sum_{B_i \in \mathcal{L}_2} \hat{x}_{ij}$. Note that $\delta_j \geq 8\alpha$. Let S and $\mathcal{O}'$ be two disjoint sets of balls which are initialized to $\mathcal{L}_2$ and $\emptyset$, respectively. Throughout we also maintain that $\sum_{B_i \in S \cup \mathcal{O}'} \hat{x}_{ij} = \delta_j$. Note that this is true in the beginning. Our algorithm is as follows.

While there is a point $p_j \in P_2$ such that $\sum_{B_i \in S} \hat{x}_{ij} > \alpha$, we do the following.

Let S_j be the set of balls in S that give flow to p_j, i.e., $S_j = \{B_i \in S : \hat{x}_{ij} > 0\}$. Note that as $\sum_{B_i \in S_j} \hat{x}_{ij} = \sum_{B_i \in S} \hat{x}_{ij} > \alpha$, $\sum_{B_i \in S_j} \hat{y}_i \geq \sum_{B_i \in S_j} \hat{x}_{ij} > \alpha$. Find $T \subseteq S_j$ such that $\alpha \leq \sum_{B_i \in T} \hat{y}_i \leq 21\alpha$. Such a subset can always be found using a linear scan of S_j, as $\sum_{B_i \in S_j} \hat{y}_i > \alpha$ and $\hat{y}_i \leq 20\alpha$ for all $B_i \in S_j$. Let B_t be the largest ball in T. Set $\hat{y}_t = 1$ and $\hat{y}_i = 0$ for each $B_i \in T$. Add B_t to $\mathcal{O}'$. Remove all balls in T from S. Reroute the flow from all balls in $T \setminus \{B_t\}$ to B_t.

▶ **Lemma 14.** *During the course of the above algorithm, the solution $\hat{\sigma}$ has cost at most $20 \cdot \text{cost}(\sigma^*)/\alpha$ and satisfies all the constraints of AUX-LP2 except Constraint 17. Moreover, for a point $p_j \in P_2$, if $\hat{x}_{ij} > 0$, $d(c_i, p_j) \leq 3 \cdot r_i$.*

Proof. First, we prove the feasibility of $\hat{\sigma}$ using induction on the iteration number. In the beginning, the claim holds. Now, consider a particular iteration. Note that the balls for which the $\hat{y}$ values are changed are in T and the points for which the $\hat{x}$ values are changed are the set of points P' that receive flow from a ball in T. It is sufficient to show that the constraints concerning these balls and points hold. Constraint 14 is satisfied as for each such point p_j, and the ball B_t, $\hat{x}_{tj} \leq \delta_j \leq 1 = \hat{y}_t$ and for a ball $B_i \in T \setminus \{B_t\}$, $\hat{x}_{ij} = 0$. Now, we argue that the capacity constraint of the ball B_t is satisfied. Note that in the beginning of the iteration, the total flow out of balls in T to all points is at most

$$\sum_{B_i \in T} \hat{y}_i \cdot U'_i \leq U'_t \sum_{B_i \in T} \hat{y}_i \leq U'_t \cdot 21\alpha < U'_t.$$

The first inequality follows from the fact that B_t is the largest ball in T and all the capacities of the balls in $\mathcal{L}_1$ are scaled by the same factor. The last inequality follows, as $\alpha \leq 1/60$.

Now, as this total flow is served by B_t the claim holds. Constraint 16 is also satisfied for all the points in P', as the flow is only rerouted from a ball to B_t. The other constraints except 17 are trivial to verify.

Note that whenever we set $\hat{y}_t = 1$, we also set $\hat{y}_i = 0$ for each $B_i \in T \setminus \{B_t\}$. Thus for each ball B_t we can charge all the balls in T. As $\sum_{B_i \in T} \hat{y}_i \geq \alpha$, the cost blow up is at most a factor of $1/\alpha$. Thus, the cost is at most $20 \cdot \text{cost}(\sigma^*)/\alpha$.

Whenever we reassign flow from balls in $T \setminus \{B_t\}$ to B_t, for a point $p_j \in P_2$, it holds that if $\hat{x}_{tj} > 0$, $d(c_t, p_j) \leq 3 \cdot r_t$. This is true, as B_t is the largest ball in T. As we remove B_t from S, no flow is ever rerouted again from or to B_t. Hence, the claim continues to hold for all points. ◀

Now, note that when the while loop of the above algorithm terminates, it holds that for any $p_j \in P_2$, $\sum_{B_i \in S} \hat{x}_{ij} \leq \alpha$. Thus, $\sum_{B_i \in \mathcal{O}'} \hat{x}_{ij} \geq \delta_j - \alpha \geq 7\alpha$. Using this fact, we compute a solution (x', y') to AUX-LP2 (that violates Constraint 17 and Constraint 15). For any ball B_i in $\mathcal{O}'$, set $y'_i = 1$. For any $p_j \in P_2$ and B_i in $\mathcal{O}'$, set $x'_{ij} = \min\{(1/(7\alpha)) \cdot \hat{x}_{ij}, 1\}$. All the other x' and y' values are set to zero. Note that, now, each point receives a flow of at least 1. We adjust the x' values so that each point receives exactly 1 amount of flow. We obtain the following lemma.

▶ **Lemma 15.** *The solution (x', y') satisfies all the constraints of AUX-LP2 except Constraint 17 and Constraint 15. Moreover,*

1. *$y'_i = 1$ for all $B_i \in \mathcal{O}'$ and $y'_i = 0$ for all $B_i \notin \mathcal{O}'$.*
2. *For any $p_j \in P_2$, $\sum_{B_i \in \mathcal{O}'} x'_{ij} = 1$.*
3. *For any $B_i \in \mathcal{O}'$, $\sum_{p_j \in P_2} x'_{ij} \leq (1/(7\alpha)) \cdot U'_i$.*
4. *For any point $p_j \in P_2$, if $x'_{ij} > 0$, $d(c_i, p_j) \leq 3 \cdot r_i$.*
5. *$\text{cost}((x', y')) \leq 20 \cdot \text{cost}(\sigma^*)/\alpha$.*

4.3 Combining the Two LP solutions

Next, we compose the two rounded solutions obtained in Lemma 12 and 15 to construct a solution for the original instance. In the new solution $(\tilde{x}, \tilde{y})$ we fully open the balls in $\mathcal{H}_1 \cup \mathcal{O} \cup \mathcal{O}'$. Also we keep all the x values unchanged. Note that a ball B_i of $\mathcal{L}_1 (= \mathcal{L}_2)$ can be opened in both solutions. However, as we had changed its capacity before, the total capacity that it can use is at most $U'_i + (1/(7\alpha)) \cdot U'_i \leq (1 + 1/(7\alpha))U_i/10 < U_i$. The last inequality follows by setting $\alpha = 1/60$. The total cost of the new solution is at most $(90 + 600\alpha)\text{cost}(\sigma^*)/\alpha \leq 6000 \cdot \text{cost}(\sigma^*)$. Hence, we obtain the following lemma.

▶ **Lemma 16.** *The solution $(\tilde{x}, \tilde{y})$ satisfies all the Constraints of MMCC-LP except Constraint 11. Moreover,*

1. *For any point $p_j \in P_1$, if $\overline{x}_{ij} > 0$, $d(c_i, p_j) \leq 5 \cdot r_i$.*
2. *$\text{cost}((\tilde{x}, \tilde{y})) \leq 6000 \cdot \text{cost}(\sigma^*)$.*

We note that by selecting different values of the parameters throughout the algorithm one can improve the constant in the approximation factor. However, as our main goal is to show any $O(1)$-approximation we did not pursue this.

▶ **Theorem 17.** *There is an $O(1)$-approximation for MMCC by expanding the balls by a factor of at most 5.*

5 Uniform Capacitated Case

The algorithm in the uniform case is same except the Selection of Balls step. The next lemma shows that the Selection of Balls can be performed with only 4.24 factor expansion of the balls.

▶ **Lemma 18.** *Using factor* 4.24 *expansion of the balls the flow of any cluster can be assigned to the chosen balls without violating the capacities.*

Proof. Consider any cluster of a heavy ball $B_h \in \mathcal{H}_1$. Let $c = (1+\sqrt{5})/2$. If B_h is one of the top 10 largest balls in the cluster, then select all the balls larger than B_h and also B_h. Only B_h is expanded by a factor of 3. The flow rerouted from any selected ball of $\mathcal{L}_1$ to B_h is assigned to the selected ball. Note that for the remaining balls of $\mathcal{L}_1$ which are in the same cluster and not chosen, are smaller than B_h and thus can be covered by a factor 3 expansion of B_h. The remaining flow is assigned to B_h. Note that in this case the capacities of the selected light balls are trivially satisfied. Also, the remaining flow assigned to B_h must have an amount at most U_h. Thus, the capacity constraint of B_h is satisfied.

Now, suppose B_h is not one of the top 10 largest balls. Let B_ℓ be the 10^{th} largest ball of this cluster. Also, let r_h and r_ℓ be the radius of B_h and B_ℓ, respectively. Now, there can be two cases (i) $r_h \geq r_\ell/c$ or (ii) $r_h < r_\ell/c$. In the first case, we select the top 9 largest balls all of which are in $\mathcal{L}_1$ and also B_h. The flow rerouted from any selected ball (except B_h) to B_h is assigned to the selected ball. Now consider the remaining flow assigned to the cluster. Also consider a point p_j which receives a part of this flow and not in any of the balls selected from $\mathcal{L}_1$. Then, by triangle inequality, the distance between p_j and the center c_h of B_h is at most $r_h + 2r_\ell \leq r_h + 2cr_h \leq 4.24r_h$. We expand B_h by the factor 4.24 and assign the remaining flow to B_h. Selected balls which are in $\mathcal{L}_1$ are not expanded. The capacity constraints are also satisfied due to the same reason mentioned above.

In the second case, the top 10 largest balls are selected all of which are in $\mathcal{L}_1$. The flow rerouted from any selected ball to B_h is assigned to the selected ball. Now consider the remaining flow assigned to the cluster. Also consider a point p_j which receives a part of this flow and not in any of the selected balls. Let $B_t = B(c_t, r_t)$ be a selected ball. Then, by triangle inequality, the distance between p_j and c_t is at most $r_t + 2r_h + 2r_\ell \leq r_t + 2r_\ell/c + 2r_\ell \leq (3 + 2/c)r_t \leq 4.24r_t$. The second last inequality follows, as r_ℓ is the smallest of the selected balls. We expand each selected ball by the factor 4.24. The remaining flow is assigned arbitrarily to selected balls respecting their capacity. Let the total amount of flow rerouted from the selected 10 light balls to B_h in Cluster Formation step be f. The total available capacity of all these balls is at least $10U'_\ell - f$, as B_ℓ is the smallest radius ball among these 10 balls. Now, as the capacity of each ball of $\mathcal{L}_1$ is reduced to a factor 10 of the original capacity and the capacity of B_h remains unchanged, $U_h \leq 10U'_\ell$. Hence, the available capacity of all these 10 balls is at least $U_h - f$. As the remaining flow is at most $U_h - f$, it follows that the capacity constraints of these balls are satisfied. ◀

▶ **Theorem 19.** *There is an $O(1)$-approximation for MCC by expanding the balls by a factor of at most* 4.24.

6 Conclusion

In this paper, we improve the expansion factor of the balls for MCC and MMCC to 4.24 and 5, respectively, in the context of obtaining constant approximation. Our approximation factor is a large constant. But, it is possible to improve this factor by setting different values

of parameters in the algorithm. Note that the lower bound on the expansion factor is still 3. So, one obvious problem is to reduce the gap further. Another interesting problem is to design a true constant approximation for the Euclidean version of MCC, which does not expand the balls. We note that this problem is open even in the plane.

Note that if the capacities are not monotonic, no $(O(1), O(1))$-approximation is known. On the other hand, the lower bound on the expansion factor even in this case is $3 - \epsilon$, similar to the uniform capacity case. So, a very natural and interesting direction of research is to study this most general version of the problem.

References

1 Ankit Aggarwal, Anand Louis, Manisha Bansal, Naveen Garg, Neelima Gupta, Shubham Gupta, and Surabhi Jain. A 3-approximation algorithm for the facility location problem with uniform capacities. *Math. Program.*, 141(1-2):527–547, 2013.

2 Hyung-Chan An, Aditya Bhaskara, Chandra Chekuri, Shalmoli Gupta, Vivek Madan, and Ola Svensson. Centrality of trees for capacitated k-center. *Math. Program.*, 154(1-2):29–53, 2015. doi:10.1007/s10107-014-0857-y.

3 Hyung-Chan An, Mohit Singh, and Ola Svensson. Lp-based algorithms for capacitated facility location. *SIAM J. Comput.*, 46(1):272–306, 2017.

4 Sayan Bandyapadhyay, Santanu Bhowmick, Tanmay Inamdar, and Kasturi Varadarajan. Capacitated covering problems in geometric spaces. *Discrete & Computational Geometry*, pages 1–31, 2019.

5 Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-approximation for capacitated facility location. In Leah Epstein and Paolo Ferragina, editors, *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, volume 7501 of *Lecture Notes in Computer Science*, pages 133–144. Springer, 2012.

6 Judit Bar-Ilan, Guy Kortsarz, and David Peleg. How to allocate network centers. *J. Algorithms*, 15(3):385–415, 1993. doi:10.1006/jagm.1993.1047.

7 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

8 Jaroslaw Byrka, Krzysztof Fleszar, Bartosz Rybicki, and Joachim Spoerhase. Bi-factor approximation algorithms for hard capacitated k-median problems. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 722–736. SIAM, 2015.

9 Jaroslaw Byrka, Bartosz Rybicki, and Sumedha Uniyal. An approximation algorithm for uniform capacitated k-median problem with 1+\epsilon capacity violation. In Quentin Louveaux and Martin Skutella, editors, *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, volume 9682 of *Lecture Notes in Computer Science*, pages 262–274. Springer, 2016.

10 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002.

11 Fabián A. Chudak and David P. Williamson. Improved approximation algorithms for capacitated facility location problems. *Math. Program.*, 102(2):207–222, 2005.

12 Julia Chuzhoy and Joseph Naor. Covering problems with hard capacities. *SIAM J. Comput.*, 36(2):498–515, 2006.

13 Julia Chuzhoy and Yuval Rabani. Approximating k-median with non-uniform capacities. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 952–958. SIAM, 2005.

14 Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. LP rounding for k-centers with non-uniform hard capacities. In *FOCS*, pages 273–282, 2012.

15 H. Gökalp Demirci and Shi Li. Constant approximation for capacitated k-median with (1+epsilon)-capacity violation. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 73:1–73:14, 2016.

16 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

17 Rajiv Gandhi, Eran Halperin, Samir Khuller, Guy Kortsarz, and Srinivasan Aravind. An improved approximation algorithm for vertex cover with hard capacities. *J. Comput. Syst. Sci.*, 72(1):16–33, 2006.

18 Taha Ghasemi and Mohammadreza Razzazi. A PTAS for the cardinality constrained covering with unit balls. *Theor. Comput. Sci.*, 527:50–60, 2014.

19 Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *JoCG*, 3(1):65–85, 2012.

20 Mong-Jen Kao. Iterative partial rounding for vertex cover with hard capacities. In *SODA*, pages 2638–2653, 2017.

21 Samir Khuller and Yoram J. Sussmann. The capacitated K-center problem. *SIAM J. Discrete Math.*, 13(3):403–418, 2000.

22 Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *J. Algorithms*, 37(1):146–188, 2000.

23 Shi Li. On uniform capacitated k-median beyond the natural LP relaxation. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 696–707, 2015.

24 Shi Li. On uniform capacitated k-median beyond the natural LP relaxation. *ACM Trans. Algorithms*, 13(2):22:1–22:18, 2017.

25 Robert Lupton, F. Miller Maley, and Neal E. Young. Data collection for the sloan digital sky survey - A network-flow heuristic. *J. Algorithms*, 27(2):339–356, 1998. doi:10.1006/jagm.1997.0922.

26 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

27 Martin Pál, Éva Tardos, and Tom Wexler. Facility location with nonuniform hard capacities. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 329–338. IEEE Computer Society, 2001.

28 Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

29 Sam Chiu-wai Wong. Tight algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *SODA*, pages 2626–2637, 2017.

Minimum Neighboring Degree Realization in Graphs and Trees

Amotz Bar-Noy
City University of New York (CUNY), NY, USA
amotz@sci.brooklyn.cuny.edu

Keerti Choudhary
Tel Aviv University, Israel
keerti.choudhary@cs.tau.ac.il

Avi Cohen
Weizmann Institute of Science, Rehovot, Israel
avi.cohen@weizmann.ac.il

David Peleg
Weizmann Institute of Science, Rehovot, Israel
david.peleg@weizmann.ac.il

Dror Rawitz
Bar-Ilan University, Ramat-Gan, Israel
dror.rawitz@biu.ac.il

Abstract

We study a graph realization problem that pertains to degrees in vertex neighborhoods. The classical problem of *degree sequence realizability* asks whether or not a given sequence of n positive integers is equal to the degree sequence of some n-vertex undirected simple graph. While the realizability problem of degree sequences has been well studied for different classes of graphs, there has been relatively little work concerning the realizability of other types of information *profiles*, such as the vertex neighborhood profiles.

In this paper we introduce and explore the *minimum degrees in vertex neighborhood profile* as it is one of the most natural extensions of the classical degree profile to *vertex neighboring degree profiles*. Given a graph $G = (V, E)$, the min-degree of a vertex $v \in V$, namely $\textsc{MinND}(v)$, is given by $\min\{deg(w) \mid w \in N[v]\}$. Our input is a sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$, where $d_{i+1} > d_i$ and each n_i is a positive integer. We provide some necessary and sufficient conditions for σ to be realizable. Furthermore, under the restriction that the realization is acyclic, i.e., a tree or a forest, we provide a full characterization of realizable sequences, along with a corresponding constructive algorithm.

We believe our results are a crucial step towards understanding extremal neighborhood degree relations in graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Graph realization, neighborhood profile, graph algorithms, degree sequences

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.10

Funding US-Israel BSF grant 2018043; ARL Network Science CTA W911NF-09-2-0053.

1 Introduction

Background and Motivation. Vertex degrees occur as a central and natural parameter in many network applications, and provide information on the significance, centrality, connectedness and influence of each vertex in the network, contributing to our understanding of the network structure and properties. The *m degree sequence* of an n-vertex graph G consists of its vertex degrees, $\textsc{Deg}(G) = (d_1, \ldots, d_n)$. It is a straightforwad task to extract the degree sequence of a given graph G from its adjacency matrix or adjacency lists. A more

interesting and challenging task, known as the *realization* problem, concerns the opposite situation where, given a sequence of non-negative integers D, it is necessary to decide whether there exists a graph whose degree sequence conforms to D. A sequence that admits such a realization is called *graphic*. A necessary and sufficient condition for a given sequence of integers to be graphic (also implying an $O(n)$ decision algorithm) was presented by Erdös and Gallai in [10]. Havel and Hakimi [12, 14] described an algorithm that given a sequence of integers computes in $O(m)$ time an m-edge graph realizing it, or proves that the given sequence is not graphic. Over the years, a number of extensions of the degree realization problem were studied as well, e.g., [1, 3, 23].

The current work is motivated by the fact that similar realization questions arise naturally in a variety of *other* contexts. Typically, some type of information profile, specifying some desired vertex property (related to degrees, distances, centrality, connectedness, etc), is given to us, and we are asked to find a graph conforming to the specified profile. Questions of this type span a wide research area, which was so far studied only sparsely. The current paper makes a step towards studying one specific information profile, from the family of *neighborhood degree* profiles. Such profiles arise in the context of social networks, where it is common to look at vertex degrees as representing influence or centrality, and neighboring degrees as representing proximity to power. Neighborhood degrees were considered before in [6], but there each vertex i is associated with the *list* of degrees of all vertices in its neighborhood. Our profiles are leaner, and provide a single parameter per vertex. In [5], we studied maximum-neighborhood-degree (MaxND) profiles, in which each vertex i is associated with the *maximum* degree of the vertices in its (closed) neighborhood.

A natural problem in this direction concerns the *minimum* degrees in the vertex neighborhoods. For each vertex i, let d_i denote the minimum vertex degree in i's closed neighborhood (i.e., including the vertex i itself). Then $\text{MinND}(G) = (d_1, \ldots, d_n)$ is the minimum-neighborhood-degree profile of G.

The same realizability questions asked above for degree sequences can be posed for neighborhood degree profiles as well. This brings us to the following central question of our work:

> Minimum Neighborhood Degree Realization
>
> **Input:** A sequence $D = (d_1, \ldots, d_n)$ of non-negative integers.
>
> **Question:** Is there a graph G of size n such that the *minimum* degree in the closed neighborhood of the *i-th* vertex in G is exactly equal to d_i ?

Our Contributions. We now discuss our contributions in detail. For simplicity, we represent the input vector D alternatively in a more compact format as $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$, where n_i's are positive integers with $n(\sigma) = \sum_{i=1}^{\ell} n_i = n$; here the specification requires that G contains exactly n_i vertices whose minimum degree in neighborhood is d_i. We may assume that $d_\ell > d_{\ell-1} > \cdots > d_1 \geq 1$ (noting that vertices with minimum-neighborhood-degree zero are necessarily singletons and can be handled separately).

Conditions. We show the following necessary and sufficient conditions for $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ to be MinND realizable. The necessary condition is that

$$d_i \leq n_1 + n_2 + \ldots + n_i - 1 \ , \text{ for } i \in [1, \ell], \quad \text{and} \tag{NC1}$$

$$d_\ell \leq \left\lfloor \frac{n_1 d_1}{d_1 + 1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2 + 1} \right\rfloor + \ldots + \left\lfloor \frac{n_\ell d_\ell}{d_\ell + 1} \right\rfloor . \tag{NC2}$$

The sufficient condition is that

$$d_i \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor + \ldots + \left\lfloor \frac{n_i d_i}{d_i+1} \right\rfloor \text{ , for } i \in [1,\ell] \text{ .} \tag{SC}$$

We remark that these conditions can be computed in polynomial time, and the realizing graphs, when any exist, can be constructed in polynomial time.

Approximation bound. For any sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ satisfying the first necessary condition (NC1), the sequence $\sigma^\gamma = (d_\ell^{\lceil \gamma n_\ell \rceil}, \ldots, d_1^{\lceil \gamma n_1 \rceil})$, where $\gamma = (d_1+1)/d_1$ satisfies[1] the sufficient condition (SC), thus our necessary and sufficient conditions differ by a factor of at most 2 in the n_i's.

We leave it as an open question to resolve the problem exactly over general graphs.

▶ **Open Question.** *Does there exist a closed-form characterization for realizing* MinND *profiles for general graphs?*

For the special case of ℓ bounded by 3, we show that $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ is MinND-realizable if and only if along with (NC1) and (NC2) the following condition is satisfied:

$$d_2 \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor, \text{ or } d_3 + 1 \leq n_1 + n_2 + n_3 - \left(1 + \left\lceil \frac{d_2 - n_2}{d_1} \right\rceil \right) \tag{NC3}$$

Acyclic Realization. When the required graph G is acyclic (that is, G is a tree or a forest), we give tight bounds for realizability (in the form of a constructive algorithm as well as a matching lower bound). For a sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$, let

$$\phi(\sigma) = d_\ell^2 + 1 + \sum_{i=1}^{\ell} (n_i - 1)(d_i - 1)^2 + \sum_{i=1}^{\ell-1} d_i(d_i - 1) \text{ .}$$

We show that σ is MinND-realizable by a tree if and only if the following conditions are met:

$$d_1 = 1 \qquad \text{and} \qquad \phi(\sigma) \leq n(\sigma) \text{ .} \tag{NC-Tree}$$

Recall that $n(\sigma) = \sum_{i=1}^{\ell} n_i$. Observe that when the profile is (1^n), condition (NC-Tree) is equivalent to claiming that (1^n) is realizable for any $n \geq 2$. Indeed, the star graph provides such a realization. Next, note that d_1 and n_1 do not appear in $\phi(\sigma)$ when $\ell > 1$, because of the terms $d_i - 1$. However, n_1 is part of $n(\sigma)$, and it must be large enough to satisfy the condition. Therefore, condition (NC-Tree) can be rewritten as $\phi(\sigma) - \sum_{i=2}^{\ell} n_i \leq n_1$, where the left hand side is effectively independent of d_1 and n_1. That is, any sub-profile $\sigma' = (d_\ell^{n_\ell}, \cdots, d_2^{n_2})$ of σ can be realized if it is expanded into a full profile $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ for which n_1 is large enough. Hence in a sense, these n_1 vertices, which are leaves or neighbors of leaves, "control" the realizability of the profile.

[1] For further explanation, see the text just before Corollary 10.

The MaxND profile. We remark that in the companion paper [5] we studied the dual MaxND realization problem, which turns out to exhibit radically different behavior from the MinND realization problem, and requires different techniques. In the MaxND profile, d_i specifies the *maximum* degree in the neighborhood of the ith vertex in G. [5] gives tight bounds for realizations by an arbitrary graph and by a connected graph. However, the question of realizations with trees is left open.

It is interesting to contrast the behavior of the MinND and MaxND profiles. For general graphs, MinND appears to be more difficult, since it is nonmonotone when edges are added or deleted, while the MaxND profile is monotone. For trees, on the other hand, the realizability of the MinND profile depends only on the leaves and their parents, which simplifies the analysis; no analogous simplifying property was found for the MaxND profile.

Applicability. Realization questions may potentially be applicable in two general settings. The first involves scientific contexts, where the information profile may consist of measurement results obtained by observing some natural network of unknown structure and our goal is to build a model (possibly explaining the measurements). The second involves engineering contexts, where the profile is derived from a given specification and the goal is to implement a network abiding by the specification.

One of the concrete uses for degree realization techniques is within the framework of generating random graphs with specific given properties. In particular, given the ability to efficiently generate a graph with a given degree sequence, one can design methods for generating a random graph with a specific degree distribution based on first generating a random degree sequence from the given distribution. As happened with degree realization, one may expect that efficient solutions for the problem of realizing certain neighborhood degree profiles may lead to improved techniques for generating and simulating social networks with prescribed neighborhood degree profiles.

Finally, a popular sampling technique that takes advantage of the Friendship Paradox [11] is based on sampling a *random neighbor* of a *random vertex*. While the average of the degrees in the traditional degree profile is the expected degree of a random vertex, the lower and upper bounds on the expected degree of the random neighbor are the averages of the degrees in the MinND and MaxND profiles respectively. Providing realizations and characterizing realizable profiles may be useful in exploring and analyzing the performance of this sampling technique.

Related Work. Many works have addressed related questions such as finding all the (non-isomorphic) graphs that realize a given degree sequence, counting all the (non-isomorphic) realizing graphs of a given degree sequence, sampling a random realization for a given degree sequence as uniformly as possible, or determining the conditions under which a given degree sequence defines a unique realizing graph , cf. [8, 10, 12, 13, 14, 15, 18, 19, 21, 20, 22, 24]. Other works such as [7, 9, 16] studied interesting applications in the context of social networks.

To the best of our knowledge, the MinND realization problems have not been explored so far. There are two other related problems that we are aware of. The first is the *shotgun assembly* problem [17], where the characteristic associated with the vertex i is some description of its neighborhood up to radius r. The second is the *neighborhood degree lists* problem [6], where the characteristic associated with the vertex i is the list of degrees of all vertices in i's neighborhood. We point out that in contrast to these studies, our MinND problem applies to a more restricted profile (with a single number characterizing each vertex), and the techniques involved are totally different from those of [6, 17]. Several other realization problems are surveyed in [2, 4].

2 Preliminaries

Let H be an undirected graph. We use $V(H)$ and $E(H)$ to respectively denote the vertex set and the edge set of the graph H. For a vertex $x \in V(H)$, let $deg_H(x)$ denote the degree of x in H. Let $N_H[x] = \{x\} \cup \{y \mid (x,y) \in E(H)\}$ be the (closed) neighborhood of x in H. For a set $W \subseteq V(H)$, we denote by $N_H(W)$, the set of all the vertices lying outside the set W that are adjacent to some vertex in W, that is, $N_H(W) = (\bigcup_{w \in W} N[w]) \setminus W$. Given a vertex v in H, the minimum degree in the neighborhood of v, namely $\textsc{MinND}_H(v)$, is defined to be the minimum over the degrees of all the vertices in the neighborhood of v. Given a set of vertices A in a graph H, we denote by $H[A]$ the subgraph of H induced by the vertices of A. For a set A and a vertex $x \in V(H)$, we denote by $A \cup x$ and $A \setminus x$, respectively, the sets $A \cup \{x\}$ and $A \setminus \{x\}$. When the graph is clear from context, for simplicity, we omit the subscripts H in all our notations. Finally, given two integers $i \leq j$, we define $[i,j] = \{i, i+1, \ldots, j\}$.

Consider a profile $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ satisfying $d_\ell > d_{\ell-1} > \cdots > d_1 > 0$. Denote its size by $n(\sigma) = \sum_{i=1}^{\ell} n_i$. The profile σ is said to be *MinND realizable* if there exists an $n(\sigma)$-vertex graph G such that $|\{v \in V(G) : \textsc{MinND}(v) = d_i\}| = n_i$, namely, G contains exactly n_i vertices whose MinND is d_i, for every $i \in [1,\ell]$. Figure 1 depicts a MinND realization of $(2^3, 1^2)$. (The numbers represent vertex degrees.)

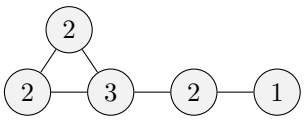


Figure 1 A MinND realization of $(2^3, 1^2)$.

3 Realizations on Acyclic graphs

In this section, we provide a complete characterisation for realizability on acyclic graphs.

3.1 Constructive Algorithm

▶ **Proposition 1.** *Any sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ satisfying $d_1 = 1$ and $\phi(\sigma) \leq n(\sigma)$ is MinND-realizable over trees.*

Proof. Initialize T to be a star with a root r and d_ℓ leaves. Let the initial set of $d_\ell + 1$ vertices be X_0. Notice $|X_0 \setminus \{r\}| = d_\ell > \ell - 1$, since $d_1 = 1$.

Partition $X_0 \setminus \{r\}$ into two sets Z_1 and Z_2, respectively of size $\ell - 1$ and $d_\ell - \ell + 1$. We label the $(i-1)^{th}$ vertex in Z_1 as $v_{i,1}$, for $i \in [2,\ell]$. Observe $|Z_2| \geq 1$.

Our algorithm (to iteratively build T) proceeds in ℓ rounds: $i = \ell, \ldots, 1$. (See Algorithm 1 for a pseudocode).

We will maintain the following invariant in our algorithm.

Invariant. *Before the beginning of round i, the vertex $v_{i,1}$ is a leaf node in the partially constructed tree T, and its neighbor r (always) has degree at least $d_\ell \geq d_i$.*

Description of round i ($i > 1$). Take the leaf node $v_{i,1} \in X_0$. Add $n_i - 1$ new vertices, namely $v_{i,2}, \ldots, v_{i,n_i}$ and connect each $v_{i,j}$ to $v_{i,j-1}$, for $2 \leq j \leq n_i$. Let V_i represent the set $\{v_{i,1}, v_{i,2}, \ldots, v_{i,n_i}\}$. Notice that V_i forms a simple path. Recall by our invariant that the neighbor of $v_{i,1}$ (other than $v_{i,2}$ in T) had degree already at least d_i. We will ensure next the following:

C1: All vertices in V_i have degree d_i.

C2: All neighbors of vertices in V_i have degree at least d_i.

To ensure condition C1, we proceed as follows: (i) Since the vertices $v_{i,1}, \ldots, v_{i,n_i-1}$ already have degree 2 in the current T_i, they are connected to $d_i - 2$ *new* vertices, and (ii) the vertex v_{i,n_i} is connected to $d_i - 1$ *new* vertices. In the process we add in total $n_i(d_i - 2) + 1$ new vertices. Let these be represented by the set A_i.

To ensure condition C2, we connect each $a \in A_i$ to an additional $d_i - 1$ *new* vertices. Let B_i be the set of new vertices added. Then, $|B_i| = |A_i| \cdot (d_i - 1)$.

We now compute the size of $V_i \cup A_i \cup B_i$.

$$\begin{aligned} |V_i \cup A_i \cup B_i| &= n_i + \big(n_i(d_i-2)+1\big) + \big(n_i(d_i-2)+1\big)\cdot(d_i-1) \\ &= n_i + d_i\big(n_i(d_i-2)+1\big) \\ &= n_i\big(d_i-1\big)^2 + d_i \end{aligned}$$

Description of round 1. Finally, in round $i = 1$, add a set Y_0 of $n(\sigma) - \phi(\sigma)$ *new* vertices to T. Observe that $n(\sigma) - \phi(\sigma) \geq 0$ due to the assumption. Connect the root node $r \in X_0$ in T to each of the vertices in Y_0.

We next show that our construction satisfies $|V(T)| = n(\sigma)$.

$$\begin{aligned} |V(T)| &= |X_0| + \sum_{i=2}^{\ell} \big(|V_i \cup A_i \cup B_i| - 1\big) + |Y_0| \\ &= d_\ell + 1 + \sum_{i=2}^{\ell} \big[n_i(d_i-1)^2 + d_i - 1\big] + n(\sigma) - \phi(\sigma) \\ &= d_\ell^2 + 1 + \sum_{i=2}^{\ell} (n_i - 1)(d_i-1)^2 + \sum_{i=2}^{\ell-1} d_i(d_i-1) + n(\sigma) - \phi(\sigma) \\ &= n(\sigma) \end{aligned}$$

Algorithm 1 Computing a tree MinND-realization for a given realizable σ.

Input: A sequence $\sigma = (d_\ell^{n_\ell} \cdots d_1^{n_1})$ satisfying $d_1 = 1$ and $n(\sigma) \geq \phi(\sigma)$.

```
Initialize T to be a star with a root r and d_ℓ leaves.
Label the i^th leaf in T as v_{i,1}, for i ∈ [2, ℓ].
for i = ℓ to 2 do
    Add n_i − 1 new vertices to T, namely v_{i,2}, ..., v_{i,n_i}.
    Connect each v_{i,j} to v_{i,j−1}, for 2 ≤ j ≤ n_i.
    Add to T a set A_i of n_i(d_i − 2) + 1 new vertices.
    Connect each v_{i,j}, for 1 ≤ j ≤ n_i − 1, to d_i − 2 isolated vertices in A_i.
    Connect v_{i,n_i} to d_i − 1 isolated vertices in A_i.
    Add to T a set B_i of |A_i| · (d_i − 1) new vertices.
    Connect each a ∈ A_i to d_i − 1 isolated vertices in B_i.
Add n(σ) − φ(σ) new vertices to T as children of the root r.
Output T.
```

Correctness Analysis

Let V_1 denote the set $V(T) \setminus \bigcup_{i=2}^{\ell} V_i$. Clearly, $|V_i| = n_i$ for $i \in [2, \ell]$, and since $|V(T)| = n(\sigma)$ it follows that $|V_1| = n(\sigma) - \sum_{i=2}^{\ell} n_i = n_1$. Therefore, if we show that for every $u \in V_i$, $\text{MinND}(u) = d_i$, for $i \in [1, \ell]$, then we are done.

Observe that the degrees of vertices in $V_i \cup A_i$ do not alter after round i, so C1 and C2 continue to hold for each V_i, $i \in [2, \ell]$. This shows that for every $u \in V_i$, $\text{MinND}(u) = d_i$, for $i \in [2, \ell]$. We are left to analyse set V_1. We have:

$$\begin{aligned} V_1 &= \left(X_0 \setminus \cup_{i=2}^{\ell}\{v_{i,1}\}\right) \cup Y_0 \cup \left(\cup_{i=2}^{\ell}(A_i \cup B_i)\right) \\ &= \{r\} \cup Z_2 \cup Y_0 \cup \left(\cup_{i=2}^{\ell}(A_i \cup B_i)\right) \end{aligned}$$

For $2 \le i \le \ell$, the set B_i contains only leaves, and each node in A_i must have a neighbor in B_i. Thus, vertices in $\cup_{i=2}^{\ell}(A_i \cup B_i)$ have MinND exactly 1.

So it is left to consider the vertices of $\{r\} \cup Z_2 \cup Y_0$, of which the vertices in $Z_2 \cup Y_0$ have already degree 1. Now recall $Z_2 \neq \emptyset$, and r is adjacent to degree-1 vertices in Z_2, thus MinND of r is 1 as well.

This completes the correctness analysis. ◀

3.2 Tightness Criterion

We next show that our construction is tight, i.e., a sequence is MinND-realizable over trees if and only if it is realizable by the procedure of Proposition 1.

▶ **Proposition 2.** *For a sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ satisfying $d_1 = 1$, a necessary condition of MinND-realizability over trees is $\phi(\sigma) \le n(\sigma)$.*

Proof. Consider a profile $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$, and let T be a MinND tree-realization of σ on V. Let $r \in V(T)$ be a vertex that satisfies $\text{MinND}(u) = d_\ell$. Root T at node r. For $i = 1, \ldots, \ell$, let $V_i = \{v \in V(T) \mid \text{MinND}(v) = d_i\}$. Observe that for each $i < \ell$, there exists (at least) one edge, denoted $(y_i, x_i) \in E(T)$, where y_i is the parent of x_i, satisfying the condition that (i) $x_i \in V_i$, and (ii) none of the vertices in the tree-path $(r \rightsquigarrow_T y_i)$ lie in V_i. These edges play a crucial role in our tight bound on $\phi(\sigma)$.

Let

$$\begin{aligned} A &= \{x_i \mid \text{MinND}(y_i) < d_i, \text{ for } i < \ell\}, \\ B &= \{x_i \mid \text{MinND}(y_i) > d_i, \text{ for } i < \ell\}. \end{aligned}$$

For each $w \in V(T)$, let $\mathcal{C}_w$ and $\mathcal{GC}_w$, respectively, be the set consisting of the children and grand-children of w in T. Also let $\mathcal{C}_A = \cup_{w \in A} \mathcal{C}_w$.

Now we define a function $\Gamma : V \mapsto 2^V$ as follows (see example in Figure 2):

$$\Gamma(w) = \begin{cases} \{r\} \cup \mathcal{C}_r \cup \mathcal{GC}_r, & \text{if } w = r, \\ \mathcal{C}_w \cup (\mathcal{GC}_w \setminus \mathcal{C}_A), & \text{if } w \in A, \\ \mathcal{GC}_w \setminus \mathcal{C}_A, & \text{otherwise.} \end{cases}$$

Figure 3 illustrates the subtree induced over $\{w\} \cup \mathcal{C}_w \cup \mathcal{GC}_w$, for some node w.

▷ Claim. $\Gamma(w) \cap \Gamma(v) = \emptyset$ for every $v, w \in V(T)$ such that $v \neq w$.

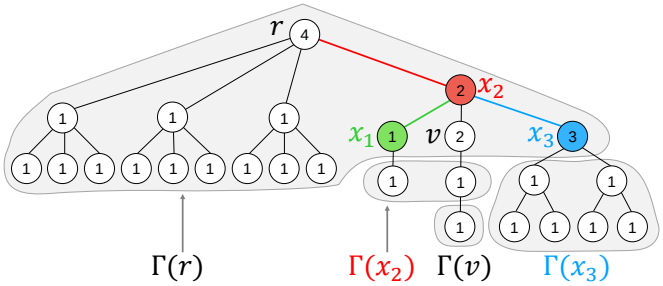


Figure 2 Tree MinND-realization of $\sigma = (4^1 3^1 2^2 1^{22})$. The number in each vertex denotes its MinND. The edges (y_1, x_1), (y_2, x_2) and (y_3, x_3) are colored green, red and blue, respectively. Further, $A = \{x_3\}$ and $B = \{x_1, x_2\}$.

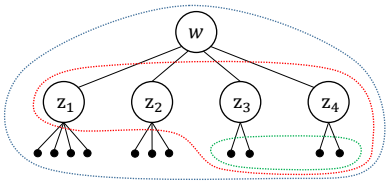


Figure 3 Illustration of a subtree induced over $\{w\} \cup \mathcal{C}_w \cup \mathcal{GC}_w$, for some node w. $\Gamma(w)$ is the set of vertices in the blue dotted line if $w = r$, in the red if $w \in A$, and in the green otherwise (assuming $z_1, z_2 \in A$).

Proof. Let us first consider the case $v = r$. The result is obviously true if $w \notin \mathcal{C}_r$, or $w \notin A$. Now if $w \in \mathcal{C}_r$, then $\text{MinND}(r) \geq \text{MinND}(w)$, thereby implying $w \notin A$.

Next consider any two vertices $v \neq w \in V(T) \setminus \{r\}$. Assume towards contradiction $\Gamma(v) \cap \Gamma(w)$ contains a node z. Then z must be a child of exactly one of the nodes v or w, and the corresponding node must lie in A. Assume $z \in \mathcal{C}_w$, and $w \in A$. Since $z \in \mathcal{C}_w \subseteq \mathcal{C}_A$, we have $z \notin \mathcal{GC}_v \setminus \mathcal{C}_A$, and also z cannot be a child of v, thereby implying $z \notin \Gamma(v)$. Hence, $\Gamma(v) \cap \Gamma(w)$ must be empty. ◁

▷ Claim. For every $v \in V_i$, $1 \leq i \leq \ell$, we have

$$|\Gamma(v)| \geq \begin{cases} d_i(d_i - 1), & \text{if } v \in A \cup B, \\ (d_i - 1)^2, & \text{if } v \notin \{r\} \cup A \cup B. \end{cases}$$

Proof. Consider a node $v \in V_i$, for some $i \leq \ell$. Observe that each $u \in \mathcal{C}_v$ must have degree at least d_i, and thus satisfy $|\mathcal{C}_u| \geq d_i - 1$. Let z_0 be v's parent and $z_1, \ldots, z_t$ be the nodes in $\mathcal{C}_v \cap A$. Since z_0 is an ancestor of $z_1, \ldots, z_t$, by definition of $A \cup B$, the MinND of all the vertices $z_0, z_1, \ldots, z_t$ must be distinct. Without loss of generality, we can assume $\text{MinND}(z_t) > \cdots > \text{MinND}(z_1)$. By definition of A, $\text{MinND}(z_1) > d_i$. Let $\Delta = \max_{j=0}^{t} \text{MinND}(z_j)$. Then $deg(v) \geq \Delta$. Consequently we have $\Delta \geq d_i + t$, since $\Delta \geq \text{MinND}(z_t) > \cdots > \text{MinND}(z_1) > d_i$. So

$$|\mathcal{C}_v \setminus A| = deg(v) - t - 1 \geq \Delta - t - 1 \geq (d_i - 1) \ . \tag{1}$$

We now consider three cases, according to whether v lies in A, B, or $V \setminus (\{r\} \cup A \cup B)$.

1. $v \in A$: By Eq. (1), $|\mathcal{GC}_v \setminus \mathcal{C}_A| \geq (d_i - 1)^2$, and also $|\mathcal{C}_v| \geq d_i - 1$. Combined, we get that $|\Gamma(v)| = |\mathcal{GC}_v \setminus \mathcal{C}_A| + |\mathcal{C}_v| \geq d_i(d_i - 1)$.

2. $v \in B$: By definition of B, MinND$(z_0) > d_i$. Thus, MinND$(z_j) > d_i$ for $j \in [0, t]$. Also, MinND of $z_0, \ldots, z_t$ are distinct. Hence, $\Delta \geq d_i + (t+1)$. So $|\mathcal{C}_v \setminus A| = deg(v) - t \geq \Delta - t \geq d_i$. This implies $|\Gamma(v)| = |\mathcal{GC}_v \setminus \mathcal{C}_A| \geq d_i(d_i - 1)$.
3. $v \notin \{r\} \cup A \cup B$: By Eq. (1), $|\mathcal{GC}_v \setminus \mathcal{C}_A| \geq (d_i - 1)^2$, implying $|\Gamma(v)| \geq (d_i - 1)^2$.

The claim follows. ◁

Note that $\Gamma(r)$ contains at least $d_\ell^2 + 1$ nodes, since the degrees of r and of its children are at least d_ℓ. Now, we are ready to prove the bound over $\phi(\sigma)$. In our calculations we use x_ℓ to denote the node r.

$$\begin{aligned} n(\sigma) = |V(T)| &\geq |\Gamma(r)| + \sum_{i=1}^{\ell} \sum_{v \in V_i \setminus \{x_i\}} |\Gamma(v)| + \sum_{i=1}^{\ell-1} |\Gamma(x_i)| \\ &\geq d_\ell^2 + 1 + \sum_{i=1}^{\ell} (n_i - 1)(d_i - 1)^2 + \sum_{i=1}^{\ell-1} d_i(d_i - 1) \\ &= \phi(\sigma) \ . \end{aligned}$$

This completes our proof of $n(\sigma) \geq \phi(\sigma)$. ◀

▶ **Corollary 3.** *For a sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ satisfying $d_1 = 1$, a necessary condition of MinND-realizability over forests is $\phi(\sigma) \leq n(\sigma)$.*

Proof. Given a sequence σ that is MinND-realizable as a forest, it can be partitioned into subsequences $\sigma_1, \ldots, \sigma_k$ corresponding to each of its connected components. By Proposition 2, $n(\sigma_i) \geq \phi(\sigma_i)$ for $i \in [1, k]$. Therefore, $n(\sigma) = \sum_{i=1}^{k} n(\sigma_i) \geq \sum_{i=1}^{k} \phi(\sigma_i) \geq \phi(\sigma)$, where the last inequality follows immediately from the definition of ϕ. ◀

By Proposition 1 and Corollary 3, and the fact that a tree always contains vertices of degree one (and hence also MinND one), the following is immediate.

▶ **Theorem 4.** *The sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ is MinND-realizable over acyclic graphs if and only if $d_1 = 1$, and $\phi(\sigma) \leq n(\sigma)$.*

4 Realizations in General graphs

We first define the notion of *leader* and *follower* crucial to our construction. Let $G = (V, E)$ be any graph. For any vertex $v \in V$, we define *leader*(v) to be a vertex in $N[v]$ of minimum degree, if there is more than one choice we pick the leader arbitrarily (these arbitrarily chosen leaders do not have to be consistent between neighbors, e.g., it is possible that two vertices u and v are the leaders of each other). In other words, *leader*$(v) \in \arg\min\{deg(w) \mid w \in N[v]\}$. Next let $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ be the min-degree sequence of G. We define V_i to be the set of those vertices in G whose minimum-degree in the closed neighborhood is exactly d_i, so $|V_i| = n_i$. Also, let L_i be the set of those vertices in G who are leaders of at least one vertex in V_i, equivalently, $L_i = \{leader(v) \mid v \in V_i\}$, and denote by $L = \cup_{i=1}^{\ell} L_i$ the set of all the leaders in G. Observe that the sets $V_1, \ldots, V_\ell$ form a partition of the vertex-set of G.

A vertex v in G is said to be a *follower*, if *leader*$(v) \neq v$. Let $F_i = \{v \in V_i \mid v \neq leader(v)\}$ be the set of all the followers in V_i. Finally we define $R = V \setminus L$ to be the set of all the non-leaders, and $F = \cup_{i=1}^{\ell} F_i$ to be the set of all the followers.

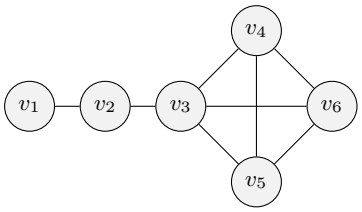


Figure 4 MinND-realization of sequence $\sigma = (3^3 2^1 1^2)$. Here $\text{MinND}(v_1) = \text{MinND}(v_2) = deg(v_1) = 1$, $\text{MinND}(v_3) = deg(v_2) = 2$, and $\text{MinND}(v_i) = 3$, for $i \in \{4, 5, 6\}$. Since $leader(v_2) = v_1$ and $leader(v_3) = v_2$, thus, v_2 is a leader as well as a follower.

We point here that there exist realizable sequences σ for which any graph G realizing σ and any leader function over G, the sets L and F have non-empty intersection. For example, consider the sequence $\sigma = (3^3 2^1 1^2)$ in Figure 4. It can be easily checked that σ has only one realizing graph, and in it, the leader-set and follower-set are non-disjoint.

We classify the sequences that admit disjoint leader and follower sets as follows.

▶ **Definition 5.** *A sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ is said to admit a Disjoint Leader-Follower (DLF) MinND-realization if there exists a graph G realizing σ and a leader function under which the sets L and F are mutually disjoint, that is, $L \cap F = \emptyset$.*[2]

▶ **Theorem 6.** *For any $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ that is MinND-realizable by a graph, say G, the following conditions must be satisfied.*
(NC1) *$d_i \leq \left(\sum_{j=1}^{i} n_j\right) - 1$, for $i \in [1, \ell]$;*
(NC2) *$d_\ell \leq \sum_{j=1}^{\ell} \left\lfloor \frac{n_j d_j}{d_j+1} \right\rfloor$.*
Further, for any leader function defined over G, and $i < \ell$, if $L_i \cap V_i \neq \emptyset$ then $d_i \leq \sum_{j=1}^{i} \left\lfloor \frac{n_j d_j}{d_j+1} \right\rfloor$.

Proof. We provide first a lower bound on the size of the leader set L_i. We show for each $i \in [1, \ell]$, we have $|L_i| \geq \left\lceil \frac{n_i}{d_i+1} \right\rceil$. Consider a vertex $a \in L_i$. Since $|N[a]| = d_i + 1$, vertex a can serve as leader for at most $d_i + 1$ vertices. This shows that $|L_i| \geq \frac{n_i}{d_i+1}$. The claim follows from the fact that $|L_i|$ is integer.

Proof of (NC1). Let w be any vertex in G such that $deg(w) = d_i$. Then w as well as all the neighbors of w must be contained in $\cup_{j=1}^{i} V_j$, therefore, we have: $d_i + 1 = |N[w]| \leq |\cup_{j=1}^{i} V_j| = \sum_{j=1}^{i} n_j$, thereby proving condition (NC1).

Proof of (NC2). Now suppose w is a vertex in G such that $\text{MinND}(w) = d_\ell$. Then $N[w]$ cannot contain vertices of degree less than d_ℓ, so $N[w] \cap L_i = \emptyset$, for each $i < \ell$. Therefore, $|N[w]| \leq n - \sum_{i=1}^{\ell-1} |L_i|$. Also $deg(w)$ must be at least d_ℓ. We thus get,

$$d_\ell + 1 \leq |N[w]| \leq n - \sum_{i=1}^{\ell-1} |L_i| = n_\ell + \sum_{i=1}^{\ell-1} (n_i - |L_i|) \leq n_\ell + \sum_{i=1}^{\ell} \left\lfloor \frac{n_i d_i}{d_i + 1} \right\rfloor .$$

Now if $n_\ell \leq d_\ell$, then $n_\ell - 1 = \left\lfloor \frac{n_\ell d_\ell}{d_\ell+1} \right\rfloor$, and so $d_\ell \leq \sum_{i=1}^{\ell} \left\lfloor \frac{n_i d_i}{d_i+1} \right\rfloor$. If $n_\ell \geq d_\ell + 1$, then $\frac{n_\ell d_\ell}{d_\ell+1} \geq d_\ell$ which implies $d_\ell \leq \left\lfloor \frac{n_\ell d_\ell}{d_\ell+1} \right\rfloor$ since d_ℓ is integral.

[2] In Section 4.1, we show that our construction realizes a DLF MinND-realization whenever one exists. In other words, the sufficient condition (SC) is also necessary for sequences that admit a DLF MinND-realization. Nevertheless, there exist MinND-realizable sequences that do not admit a DLF MinND-realization. These sequences may violate the sufficient condition (SC) despite being MinND-realizable.

Proof of last claim. Let w be any vertex lying in $L_i \cap V_i$, so $\text{MinND}(w) = deg(w) = d_i$. Recall for each $j < i$, vertices in the set L_j have degree strictly less than d_i. Since $N[w]$ cannot contain vertices of degree less than d_i, thus for each $j < i$, $N[w] \cap L_j = \emptyset$. Also vertices in $V_{i+1} \cup \ldots \cup V_\ell$ cannot be adjacent to any vertex in $\{w\} \cup \left(\cup_{j=1}^{i-1} L_j\right)$, therefore, $N[w]$ as well as $\cup_{j=1}^{i-1} L_j$ are contained in union $\cup_{j=1}^{i} V_j$. We thus get,

$$d_i + 1 = |N[w]| \leq \left|\bigcup_{j=1}^{i} V_j\right| - \left|\bigcup_{j=1}^{i-1} L_j\right| = n_i + \sum_{j=1}^{i-1}(n_i - |L_j|) \leq n_i + \sum_{j=1}^{i-1} \left\lfloor \frac{n_j d_j}{d_j + 1} \right\rfloor .$$

If $n_i \leq d_i$, then $n_i - 1 = n_i - \left\lceil \frac{n_i}{d_i+1} \right\rceil = \left\lfloor \frac{n_i d_i}{d_i+1} \right\rfloor$, and so $d_i \leq \sum_{j=1}^{i} \left\lfloor \frac{n_j d_j}{d_j+1} \right\rfloor$. If $n_i \geq d_i + 1$, then the bound trivially holds since $\frac{n_i d_i}{d_i+1} \geq d_i$ which from the fact that d_i is integral implies $d_i \leq \left\lfloor \frac{n_i d_i}{d_i+1} \right\rfloor$. ◀

We next prove the following theorem.

▶ **Theorem 7** (Sufficient condition SC). *Any sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ satisfying*

$$d_i \leq \sum_{j=1}^{i} \left\lfloor \frac{n_j d_j}{d_j + 1} \right\rfloor, \text{ for } i \in [1, \ell] ,$$

is MinND-realizable. Further, we can always compute a realizing graph, say G, and a leader function defined over G that satisfies $L \cap F = \emptyset$.

Proof. We begin with the simple case of realizing uniform sequences, and then consider the scenario of general sequences.

Uniform Sequences. Consider for simplicity first the sequence $\sigma = (d^n)$. We provide a realization for σ if $n \geq d+1$. Let $q \geq 1$ and $r \in [0, d]$ be integers satisfying $n = (q)(d+1) - r$. Take a set A of q vertices, namely a_i ($i \in [1, q]$), and another set B of dq vertices, namely b_{ij} ($i \in [1, q], j \in [1, d]$). Connect each a_i to the vertices $b_{i1}, \ldots, b_{id}$. So vertices in A have degree exactly d and vertices in B have in their neighborhood a vertex of degree d. Next if $r > 0$, then we merge b_{1j} with b_{2j}, for $j \in [1, r]$, thereby reducing r vertices in B. (Notice that b_{1j} and b_{2j} exists because $r > 0$ only when $q \geq 2$.) Thus $|A| + |B| = n$ and each vertex in A still has degree exactly d. So $|A| = \frac{n+r}{d+1} = \left\lceil \frac{n}{d+1} \right\rceil$ and $|B| = n - |A| = \left\lfloor \frac{nd}{d+1} \right\rfloor \geq d$. Finally, we add edges between each pair of vertices in B to make it a clique of size at least d; this will imply that the vertices in set B have degree at least d. It is easy to check that $\text{MinND}(v)$ for each $v \in A \cup B$ in our constructed graph is d. In our construction A forms the leader set, and B forms the follower set.

In the rest of proof, we use $\text{GRAPH}(n, d, A, B)$ to denote a function that returns the edges of the graph as constructed above (over A and B) whenever provided with four parameters n, d, A, B satisfying $n \geq d + 1$, $|A| = \left\lceil \frac{n}{d+1} \right\rceil$, and $|B| = \left\lfloor \frac{nd}{d+1} \right\rfloor$.

General Sequences. We now consider the case $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$. Initialize G to be an empty graph. Our algorithm proceeds in ℓ rounds. (See Algorithm 2 for a pseudocode.) In each round, we first add to G a set V_i of n_i new vertices and partition V_i into two sets L_i and R_i of sizes respectively $\left\lceil \frac{n_i}{d_i+1} \right\rceil$ and $\left\lfloor \frac{n_i d_i}{d_i+1} \right\rfloor$. Now if $n_i > d_i + 1$, then we solve this round independently by adding to G all the edges returned by $\text{GRAPH}(n_i, d_i, L_i, R_i)$. Notice that if $n_i \leq d_i + 1$, then L_i will contain only one vertex, say a_i. In such a case, we add edges between a_i and all the vertices in the set R_i. Also, we add edges between a_i and any arbitrarily chosen $d_i + 1 - n_i$ vertices in $\cup_{j<i} R_j$. This is possible since $d_i + 1 - n_i = d_i - \left\lfloor \frac{n_i d_i}{d_i+1} \right\rfloor \leq \sum_{j=1}^{i-1} \left\lfloor \frac{n_j d_j}{d_j+1} \right\rfloor = \sum_{j=1}^{i-1} |R_j|$. Finally, after the ℓ rounds are completed, we add edges between each pair of vertices in set $R = \cup_{i=1}^{\ell} R_i$ to make it a clique.

Let us now show bounds on the degree of vertices in sets L_i and R_i.

1. Each vertex in L_i has degree exactly d_i : Recall we add edges to vertices in L_i only in the i^{th} iteration of the for loop. If $n_i > d_i + 1$, then the degree of each vertex in L_i is exactly d_i. If $|L_i| = 1$ or, equivalently, $n_i \leq d_i + 1$, then $|R_i| = n_i - |L_i| = n_i - 1$, and so the degree of the vertex $a_i \in L_i$ is $(n_i - 1) + (d_i + 1 - n_i) = d_i$.
2. Vertices in R have degree at least d_ℓ : For any $i \in [1, \ell]$, if $n_i > d_i + 1$, then $|R_i| = \left\lceil \frac{n_i d_i}{d_i+1} \right\rceil$, and even in the case $n_i \leq d_i + 1$, we have $|R_i| = n_i - |L_i| = n_i - \left\lceil \frac{n_i}{d_i+1} \right\rceil = \left\lceil \frac{n_i d_i}{d_i+1} \right\rceil$. Thus $|R| = \sum_{i=1}^{\ell} |R_i| = \sum_{i=1}^{\ell} \left\lceil \frac{n_i d_i}{d_i+1} \right\rceil$ which is bounded below by d_i. Since $|R| \geq d_\ell$, and each vertex in R is adjacent to at least one vertex in $\cup_i L_i$, the degree of vertices in R is at least d_ℓ.

Algorithm 2 Computing a MinND-realization for a given special σ.

Input: A sequence $\sigma = (d_\ell^{n_\ell} \cdots d_1^{n_1})$ satisfying $d_i \leq \sum_{j=1}^{i} \left\lfloor \frac{n_j d_j}{d_j+1} \right\rfloor$, for $1 \leq i \leq \ell$.

```
Initialize G to be an empty graph.
for i = 1 to ℓ do
    Add to G a set V_i of n_i new vertices.
    Partition V_i in two sets L_i, R_i such that |L_i| = ⌈n_i/(d_i+1)⌉ and |R_i| = ⌊n_i d_i/(d_i+1)⌋.
    if (n_i > d_i + 1, or equivalently, |L_i| > 1) then
        Add to G all the edges returned by GRAPH(n_i, d_i, L_i, R_i).
    else if (|L_i| = 1) then
        Let a_i be the only vertex in L_i.
        Connect a_i to all vertices in R_i, and any arbitrary d_i + 1 − n_i vertices in
         ∪_{j<i} R_j.
Add edges between each pair of vertices in R = ∪_{i=1}^{ℓ} R_i to make it a clique.
Output G.
```

We next show that for any vertex $v \in V_i$, MinND$(v) = d_i$, where $i \in [1, \ell]$. If $v \in L_i$, then MinND$(v) = d_i$, since each vertex in L_i has degree d_i, and is adjacent to only vertices in R which have degree at least $d_\ell \geq d_i$. If $v \in R_i$, then also MinND$(v) = d_i$, since each vertex in R_i is adjacent to at least one vertex in L_i, and $N[v]$ is contained in the set $R \cup (\cup_{j \geq i} L_j)$, whose vertices have degree at least d_i.

The leader function over V is as follows. For each $v \in \cup_{i=1}^{\ell} L_i$, we set $leader(v) = v$, and for each $v \in R_i$, we set $leader(v)$ to any arbitrary neighbor of v in L_i. Since each vertex in $L = \cup_{i=1}^{\ell} L_i = \{leader(v) \mid v \in V\}$ is a leader of itself, the set L of leaders and the set F of followers must be mutually disjoint. ◀

As a corollary of the above results, the following is immediate.

▶ **Theorem 8.** *The sequence $\sigma = (d_2^{n_2}, d_1^{n_1})$ is MinND-realizable if and only if $d_1 \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor$ and $d_2 \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor$.*

Proof. Suppose $\sigma = (d_2^{n_2}, d_1^{n_1})$ is realizable. Then Theorem 6 implies (i) $n_1 \geq d_1 + 1$ which implies $d_1 \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor$, and (ii) $d_\ell = d_2 \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor$. The converse follows from Theorem 7. ◀

▶ Remark 9. In Appendix A, we additionally solve the more involved case of sequences of length three. That is, we provide a complete characterization of the realizability of sequences of the form $\sigma = (d_3^{n_3}, d_2^{n_2}, d_1^{n_1})$ over general graphs.

For a sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$, let $\gamma = (d_1+1)/d_1$. As $\lfloor \frac{\gamma n_1 d_1}{d_1+1} \rfloor + \ldots + \lfloor \frac{\gamma n_i d_i}{d_i+1} \rfloor \geq n_1 + \cdots + n_i \geq d_i$, we also have the following result providing a γ (≤ 2) approximation.

▶ **Corollary 10.** *For any sequence $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ satisfying the first necessary condition* (NC1)*, the sequence $\sigma^\gamma = (d_\ell^{\lceil \gamma n_\ell \rceil}, \ldots, d_1^{\lceil \gamma n_1 \rceil})$ satisfies the sufficient condition* (SC)*.*

4.1 Sequences admitting Disjoint-Leader-Follower Sets

Finally, we state our results on sequences admitting disjoint Leader-Follower sets.

▶ **Theorem 11.** *A sequence $\sigma = (n_\ell^{d_\ell} \ldots d_1^{n_1})$ is* MinND*-realizable by a graph G having disjoint leader-set (L) and follower-set (F) with respect to some leader function, if and only if, for each $i \in [1, \ell]$, $d_i \leq \sum_{j=1}^{i} \lfloor \frac{n_j d_j}{d_j+1} \rfloor$.*

Proof. Let us suppose there exists a leader function over G for which $L \cap F = \emptyset$, then for each $i \in [1, \ell]$, $L_i \subseteq V_i$. This is because if for some i, there exists $w \in L_i \setminus V_i$, then $deg(w) = d_i \neq \text{MinND}(d_i)$, which implies that w is a leader as well as a follower. Since $L_i \subseteq V_i$, by Theorem 6, $d_i \leq \sum_{j=1}^{i} \lfloor \frac{n_j d_j}{d_j+1} \rfloor$, for each $i \in [1, \ell]$. The converse claim follows from Theorem 7. ◀

References

1 Martin Aigner and Eberhard Triesch. Realizability and uniqueness in graphs. *Discrete Mathematics*, 136:3–20, 1994.
2 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Realizability of graph specifications: Characterizations and algorithms. In *25th SIROCCO*, LNCS, pages 3–13, 2018.
3 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Efficiently realizing interval sequences. In *30th ISAAC*, pages 47:1–47:15, 2019.
4 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Graph profile realizations and applications to social networks. In *13th WALCOM*, LNCS, pages 1–12, 2019.
5 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Graph realizations: Maximum degree in vertex neighborhoods. In *Proc. SWAT*, 2020.
6 Michael D. Barrus and Elizabeth A. Donovan. Neighborhood degree lists of graphs. *Discrete Mathematics*, 341(1):175–183, 2018.
7 Joseph K. Blitzstein and Persi Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics*, 6(4):489–522, 2011.
8 Sheshayya A. Choudum. A simple proof of the Erdös-Gallai theorem on graph sequences. *Bull. Austral. Math. Soc.*, 33(1):67–70, 1991.
9 Brian Cloteaux. Fast sequential creation of random realizations of degree sequences. *Internet Mathematics*, 12(3):205–219, 2016.
10 Paul Erdös and Tibor Gallai. Graphs with prescribed degrees of vertices [hungarian]. *Matematikai Lapok*, 11:264–274, 1960.
11 S.L. Feld. Why your friends have more friends than you do. *Amer. J. Sociology*, 96:1464–1477, 1991.
12 S. Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph – I. *SIAM J. Appl. Math.*, 10(3):496–506, 1962.
13 Peter L. Hammer, Toshihide Ibaraki, and Bruno Simeone. Threshold sequences. *SIAM J. Algebraic Discrete Methods*, 2(1):39–49, 1981.
14 V. Havel. A remark on the existence of finite graphs. *Casopis Pest. Mat.*, 80:477–480, 1955.
15 P.J. Kelly. A congruence theorem for trees. *Pacific J. Math.*, 7:961–968, 1957.
16 Milena Mihail and Nisheeth Vishnoi. On generating graphs with prescribed degree sequences for complex network modeling applications. *3rd ARACNE*, 2002.

17 Elchanan Mossel and Nathan Ross. Shotgun assembly of labeled graphs. *CoRR*, abs/1504.07682, 2015. `arXiv:1504.07682`.

18 Peter V. O'Neil. Ulam's conjecture and graph reconstructions. *Amer. Math. Monthly*, 77:35–43, 1970.

19 Gerard Sierksma and Han Hoogeveen. Seven criteria for integer sequences being graphic. *J. Graph Theory*, 15(2):223–231, 1991.

20 Amitabha Tripathi and Himanshu Tyagi. A simple criterion on degree sequences of graphs. *Discrete Applied Mathematics*, 156(18):3513–3517, 2008.

21 Regina Tyshkevich. Decomposition of graphical sequences and unigraphs. *Discrete Mathematics*, 220(1-3):201–238, 2000.

22 S.M. Ulam. *A collection of mathematical problems*. Wiley, 1960.

23 D.L. Wang and D.J. Kleitman. On the existence of n-connected graphs with prescribed degrees ($n > 2$). *Networks*, 3:225–239, 1973.

24 N.C. Wormald. Models of random regular graphs. *Surveys in Combin.*, 267:239–298, 1999.

A MinND realization of tri-sequences in General Graphs

Here we consider the scenario when a sequence has only three distinct degrees.

We now provide a complete characterization of sequence $\sigma = (d_3^{n_3}, d_2^{n_2}, d_1^{n_1})$.

▶ **Theorem 12.** *The necessary and sufficient conditions for* MinND*-realizability of the sequence* $\sigma = (d_\ell^{n_\ell}, \cdots, d_1^{n_1})$ *when* $\ell = 3$ *is*

1. $d_1 + 1 \leq n_1$,
2. $d_2 + 1 \leq n_1 + n_2$,
3. $d_3 \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor + \left\lfloor \frac{n_3 d_3}{d_3+1} \right\rfloor$, *and*
4. *either* $d_2 \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor$, *or* $d_3 + 1 \leq n_1 + n_2 + n_3 - \left(1 + \left\lceil \frac{d_2 - n_2}{d_1} \right\rceil\right)$.

Proof. Suppose $\sigma = (d_3^{n_3}, d_2^{n_2}, d_1^{n_1})$ is realizable, then by Theorem 6, it follows that the first three conditions stated above are necessary.

To prove that all four conditions are necessary, we are left to show that if $d_2 > \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor$, then $d_3 + 1 \leq n_1 + n_2 + n_3 - \left(1 + \left\lceil \frac{d_2 - n_2}{d_1} \right\rceil\right)$. We consider a graph G that realizes σ. Let V_1, V_2, V_3 be the partition of $V(G)$ as defined in Section 4. Consider a vertex $w \in V_2$. Observe that $leader(w)$ must lie in V_1, because if $L_2 \cap V_2$ is non-empty, then Theorem 6 implies $d_2 \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor$. We first show that $|L_1| \geq \left\lceil \frac{d_2 - n_2}{d_1} \right\rceil$. The set $N(w) \cap V_1$ has size at least $d_2 - n_2$. Each vertex $x \in L_1$ can serve as a leader of at most d_1 vertices in open-neighborhood of w. Indeed, if $x \in N(w)$ then it can not count w (lying outside $N(w)$), and if $x \notin N(w)$ then it can not count itself (again lying outside $N(w)$). Thus to cover the set $N(w) \cap V_1$ at least $\left\lceil \frac{d_2 - n_2}{d_1} \right\rceil$ leaders are required, thereby showing $|L_1| \geq \left\lceil \frac{d_2 - n_2}{d_1} \right\rceil$. Now consider a vertex $y \in V_3$, note that $N[y]$ excludes w (as degree of w is d_2), as well as L_1 (as vertices in L_1 have degree d_1). Therefore, we obtain the following relation.

$$d_3 + 1 = |\ N[y]\ | \leq |V_1 \setminus L_1| + |V_2 \setminus w| + |V_3| \leq n_1 + n_2 + n_3 - \left(1 + \left\lceil \frac{d_2 - n_2}{d_1} \right\rceil\right)$$

We now prove the sufficiency claims. If $d_2 \leq \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor$, then the conditions 1-4 are sufficient by Theorem 7. So let us focus on the scenario when $d_2 > \left\lfloor \frac{n_1 d_1}{d_1+1} \right\rfloor + \left\lfloor \frac{n_2 d_2}{d_2+1} \right\rfloor$. Let $N = n_1 + n_2 + n_3 - \left(1 + \left\lceil \frac{d_2 - n_2}{d_1} \right\rceil\right)$. The vertex-set of our realized graph $G = (V, E)$ will be a union of three disjoint sets $L_1, L_2 = \{w\}$, and Z of size respectively $\left\lceil \frac{d_2 - n_2}{d_1} \right\rceil$, 1, and N. Initially, the edge-set E is an empty-set. Between vertex pairs in Z, we add edges

so that the induced graph $G[Z]$ is identical to $\textsc{graph}(N, d_3, \left\lceil \frac{N}{d_3+1} \right\rceil, \left\lfloor \frac{Nd_3}{d_3+1} \right\rfloor)$. This step is possible since $d_3 + 1 \leq N$, and ensures that $\textsc{MinND}_{G[Z]}(z) = d_3$, for $z \in Z$. Let L_3 denote the set of those vertices in Z whose degree is equal to d_3. We connect w to arbitrary $N - n_3 = n_2 + (n_1 - |L_1 \cup L_2|)$ vertices in $Z \setminus L_3$, and any arbitrary $\alpha := d_2 - (n_1 + n_2 - |L_1 \cup L_2|)$ vertices in L_1. Since $deg_G(w) = d_2$, this step ensures that MinND of exactly n_2 vertices in Z decreases to d_2. Let Y be a subset of arbitrary $(n_1 - |L_1 \cup L_2|)$ neighbors of w in Z. Finally, we connect each $x \in L_1 \cap N[w]$ to arbitrary $d_1 - 1$ vertices in Y, and each $x' \in L_1 \setminus N[w]$ to arbitrary d_1 vertices in Y, so as to ensure each vertex in Y is adjacent to at least one leader in L_1. Since vertices in L_1 have degree d_1, this ensures $\textsc{MinND}_G(x) = d_1$, for each $x \in \{w\} \cup Y \cup L_1$. This completes the construction of G. ◀

Tight Approximation Algorithms for p-Mean Welfare Under Subadditive Valuations

Siddharth Barman
Indian Institute of Science, Bangalore, India
barman@iisc.ac.in

Umang Bhaskar
Tata Institute of Fundamental Research, Mumbai, India
umang@tifr.res.in

Anand Krishna
Indian Institute of Science, Bangalore, India
anandkrishna@iisc.ac.in

Ranjani G. Sundaram
Chennai Mathematical Institute, India
ranjanigs@cmi.ac.in

Abstract

We develop polynomial-time algorithms for the fair and efficient allocation of indivisible goods among n agents that have subadditive valuations over the goods. We first consider the Nash social welfare as our objective and design a polynomial-time algorithm that, in the value oracle model, finds an $8n$-approximation to the Nash optimal allocation. Subadditive valuations include XOS (fractionally subadditive) and submodular valuations as special cases. Our result, even for the special case of submodular valuations, improves upon the previously best known $O(n \log n)$-approximation ratio of Garg et al. (2020).

More generally, we study maximization of p-mean welfare. The p-mean welfare is parameterized by an exponent term $p \in (-\infty, 1]$ and encompasses a range of welfare functions, such as social welfare ($p = 1$), Nash social welfare ($p \to 0$), and egalitarian welfare ($p \to -\infty$). We give an algorithm that, for subadditive valuations and any given $p \in (-\infty, 1]$, computes (in the value oracle model and in polynomial time) an allocation with p-mean welfare at least $1/8n$ times the optimal.

Further, we show that our approximation guarantees are essentially tight for XOS and, hence, subadditive valuations. We adapt a result of Dobzinski et al. (2010) to show that, under XOS valuations, an $O\left(n^{1-\varepsilon}\right)$ approximation for the p-mean welfare for any $p \in (-\infty, 1]$ (including the Nash social welfare) requires exponentially many value queries; here, $\varepsilon > 0$ is any fixed constant.

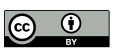
2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Discrete Fair Division, Nash Social Welfare, Subadditive Valuations, Submodular Valuations

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.11

Related Version A full version of the paper is available at `https://arxiv.org/abs/2005.07370`.

Funding *Siddharth Barman*: Supported by a Ramanujan Fellowship (SERB – SB/S2/RJN128/2015) and a Pratiksha Trust Young Investigator Award.
Umang Bhaskar: Supported by the Department of Atomic Energy, Government of India (project RTI4001), a Ramanujan Fellowship (SERB – SB/S2/RJN-055/2015), and an Early Career Research Award (SERB – ECR/2018/002766).



1 Introduction

In discrete fair division, given a set of m goods and n agents, the problem is to integrally allocate the set of goods to the agents in a fair and (economically) efficient manner [8, 17, 3]. In this thread of work, the Nash social welfare – defined as the geometric mean of the agents' valuations for their assigned bundles – has emerged as a fundamental and prominent measure of the quality of an allocation. It provides a balance between two central objectives: the social welfare (the sum of the agents' valuations) and the egalitarian welfare (the minimum valuation across the agents). Note that social welfare is a standard measure of (economic) efficiency, whereas egalitarian welfare is a fairness objective.

A Nash optimal allocation (i.e., an allocation that maximizes Nash social welfare) satisfies other fairness and efficiency criteria as well. Such an allocation is clearly Pareto optimal. Furthermore, if agents have additive valuations, then a Nash optimal allocation is known to be fair in the sense that it is guaranteed to be *envy-free up to one good* (EF1) [9] and *proportional up to one good* (Prop1) [15].[1]

As an objective, Nash social welfare is scale invariant: multiplicatively scaling any agent's valuation function by a nonnegative factor does not change the Nash optimal allocation. Furthermore, interesting connections have been established between market models and this welfare function; see, e.g., [13, 5]. As a practical application, the website `spliddit.org` uses the Nash social welfare as the optimization objective when partitioning indivisible goods [22, 9].

However, computing a Nash optimal allocation is APX-hard, even when the agents have additive valuations [24]. In terms of approximation algorithms, the problem of maximizing Nash social welfare has received considerable attention in recent years [14, 13, 2, 6, 1, 5, 19]. In particular, a polynomial-time $e^{1/e}$-approximation algorithm is known for additive valuations [5]. This algorithm preserves EF1, up to a factor of $(1+\varepsilon)$, and Pareto optimality. The approximation guarantee of $e^{1/e} \approx 1.45$ also holds for budget-additive valuations [11]. The work of Garg et al. [20] extends this line of work by considering Nash social welfare maximization under submodular valuations.

Submodular valuations capture the diminishing marginal returns property. They constitute a subclass of subadditive valuations, which, in turn, model complement-freeness. Formally, a set function v (defined over a set of indivisible goods) is subadditive if it satisfies $v(A \cup B) \leq v(A) + v(B)$, for all subsets of goods A and B. Complement-freeness is a very common assumption on valuation functions. Hence, fair division with subadditive valuations is an encompassing and important problem.

For Nash social welfare maximization under submodular valuations, Garg et al. [20] obtain an $O(n \log n)$-approximation algorithm. Prior to their work, the best known approximation ratio for submodular valuations was $(m - n + 1)$, which also extends to subadditive valuations [26]; here, m denotes the number of goods and n the number of agents. For a constant number of agents with submodular valuations, Garg et al. [20] provide an $e/(e-1)$-approximation algorithm and show that, even in this setting, improving upon $e/(e-1)$ is NP-hard.

[1] An allocation is said to be EF1 if for any pair of agents i and j, there exists a good g in j's bundle, such that i prefers her bundle to the one obtained after removing g from j's bundle. An allocation is said to be Prop1 if for each agent i there exists a good g with the property that including g into i's bundle ensures that i achieves a proportional share, i.e., her valuation ends up being at least $1/n$ times her value for all the goods.

In the context of allocating indivisible goods, two other well-studied welfare objectives are the social welfare and the egalitarian welfare. These represent, respectively, for an allocation, the average valuation of the agents and the minimum valuation of any agent. For the social welfare objective, a tight approximation factor of $e/(e-1)$ is known under submodular valuations [29]. For subadditive valuations, Feige [18] shows that social welfare maximization admits a polynomial-time 2-approximation, assuming oracle access to *demand queries*.[2]

For maximizing egalitarian welfare under additive valuations, Chakrabarty et al. [10] provide an $\widetilde{O}(n^{\varepsilon})$-approximation algorithm that runs in time $n^{O(1/\varepsilon)}$, for any $\varepsilon > 0$. Under submodular valuations, egalitarian welfare maximization admits an $\widetilde{O}(n^{1/4}m^{1/2})$-approximation algorithm [21]. Khot and Ponnuswami [23] provide a $2n$-approximation algorithm for maximizing egalitarian welfare under subadditive valuations. As a lower bound, with submodular valuations, egalitarian welfare cannot be approximated within a factor of 2, unless P = NP [7].

In this work we develop a unified treatment of fairness and efficiency objectives, including the welfare functions mentioned above. In particular, we develop an approximation algorithm for computing allocations that maximize the *generalized mean* of the agents' valuations. Formally, for exponent parameter $p \in \mathbb{R}$, the p^{th} *generalized mean* of a set of n positive reals $v_1, v_2, \ldots, v_n$ is defined as $\left(\frac{1}{n}\sum_{i=1}^{n} v_i^p\right)^{1/p}$. For an allocation (partition) $\mathcal{A} = (A_1, \ldots, A_n)$ of the indivisible goods among the n agents, we define the *p-mean welfare* of $\mathcal{A}$ as the generalized mean of the values $(v_i(A_i))_{i\in[n]}$; here $v_i(A_i)$ is the value that agent i has for the bundle A_i assigned to it. Indeed, with different values of p, the p-mean welfare encompasses a range of objectives: it corresponds to the social welfare (arithmetic mean) for $p = 1$, the Nash social welfare (geometric mean) for $p \to 0$, and the egalitarian welfare for $p \to -\infty$. In fact, p-mean welfare functions with $p \in (-\infty, 1]$ exactly correspond to the collection of functions characterized by a set of natural axioms, including the Pigou-Dalton transfer principle [25]. Hence, p-mean welfare functions, with $p \in (-\infty, 1]$, constitute an important and axiomatically-supported family of objectives.

Our Contributions. We develop a polynomial-time algorithm that, given a fair division instance with subadditive valuations and parameter $p \in (-\infty, 1]$, finds an allocation with p-mean welfare at least $1/8n$ times the optimal p-mean welfare (Theorem 9). Our algorithm uses the standard value oracle model which, when queried with any subset of goods and an agent i, returns the value that i has for the subset. For different values of p, our algorithm changes minimally, differing only in the weights of edges for a computed matching. We thus present a unified analysis for this broad class of welfare functions, suggesting further connections between these objectives than the previously mentioned axiomatization. Our result matches the best known $O(n)$-approximation for egalitarian welfare [23] and improves upon the $O(n \log n)$-approximation guarantee of Garg et al. [20] for Nash social welfare with submodular valuations. Arguably, our algorithm (and the analysis) is simpler than the one developed in [20] and simultaneously more robust, since it obtains an improved approximation ratio for subadditive valuations and a notably broader class of welfare objectives.

For clarity of exposition, we first present an $8n$-approximation algorithm for maximizing Nash social welfare under subadditive valuations (Theorem 2). We then generalize the algorithm to the class of p-mean welfare objectives.

[2] A demand-query oracle, when queried with prices $p_1, \ldots, p_m \in \mathbb{R}$ associated with the m goods, returns $\max_{S\subseteq[m]} \left(v(S) - \sum_{j\in S} p_j\right)$, for an underlying valuation function v. The current paper works with more basic value oracle, which when queried with a subset of goods returns the value this subset. Any value query can be simulated via a polynomial number of demand queries. However, the converse is not true [27].

We complement these algorithmic results by adapting a result of Dobzinski et al. [16] to show that for XOS valuations, any $O(n^{1-\varepsilon})$-approximation for p-mean welfare requires an exponential number of value queries (Section 5). Hence, in the value oracle model, our approximation guarantee is essentially tight for XOS and, hence, for subadditive valuations. We note that these are the first polynomial lower bounds on approximating either the Nash social welfare or the egalitarian welfare.

Nguyen and Rothe [26] obtain an $(m - n + 1)$-approximation guarantee for maximizing Nash social welfare with subadditive valuations. We establish two extensions of this result. First, we show that, under subadditive valuations, an $(m - n + 1)$-approximation for the p-mean welfare can be obtained for all $p \leq 0$. However, for $0 < p < 1$, we establish that it is NP-hard to obtain an $(m - n + 1)$-approximation, even under additive valuations. An analogous hardness result holds for $p = 1$ with submodular valuations.

Independent Work. In work independent of ours, Chaudhury et al. [12] also obtain an $O(n)$-approximation algorithm for maximizing generalized p-means under subadditive valuations. Their approach varies significantly from the current paper and, in particular, builds upon results on finding allocations that are approximately envy-free up to any good (EFX). Notably their algorithm computes allocations that satisfy additional fairness properties, including EF1 and either of two approximate versions of EFX.

Section 3 presents our approximation algorithm for maximizing Nash social welfare. Then, Section 4 shows that we can extend the algorithm for Nash social welfare to obtain the stated approximation bound for p-mean welfare. The tightness of these results is established in Section 5. Section 6 presents the results for the $(m - n + 1)$-approximation guarantees. All the missing proofs appear in the full version of this paper [4].

2 Notation and Preliminaries

An instance of a fair division problem is a tuple $\langle [m], [n], \{v_i\}_{i=1}^n \rangle$, where $[m] = \{1, 2, \ldots, m\}$ denotes the set of $m \in \mathbb{N}$ indivisible goods that have to be allocated (partitioned) among the set of $n \in \mathbb{N}$ agents, $[n] = \{1, 2, \ldots, n\}$. Here, $v_i : 2^{[m]} \mapsto \mathbb{R}_+$ represents the valuation function of agent $i \in [n]$. Specifically, $v_i(S) \in \mathbb{R}_+$ is the value that agent i has for a subset of goods $S \subseteq [m]$. For $g \in [m]$ and $i \in [n]$, write $v_i(g)$ to denote agent i's value for the good g, i.e., it denotes $v_i(\{g\})$.

We will assume throughout that the valuation function v_i for each agent $i \in [n]$ is (i) nonnegative: $v_i(S) \geq 0$ for all $S \subseteq [m]$, (ii) normalized: $v_i(\emptyset) = 0$, (iii) monotone: $v_i(A) \leq v_i(B)$ for all $A \subseteq B \subseteq [m]$, and (iv) subadditive: $v_i(A \cup B) \leq v_i(A) + v_i(B)$ for all subsets $A, B \subseteq [m]$.

Submodular and XOS (fractionally subadditive) valuations constitute subclasses of subadditive valuations. Formally, a set function $v : 2^{[m]} \mapsto \mathbb{R}_+$ is said to be submodular if it satisfies the *diminishing marginal returns* property: $v(A \cup \{g\}) - v(A) \geq v(B \cup \{g\}) - v(B)$, for all subsets $A \subseteq B \subsetneq [m]$ and $g \in [m] \setminus B$. A set function, $v : 2^{[m]} \mapsto \mathbb{R}_+$, is said to be XOS if it is obtained by evaluating the maximum over a collection of additive functions $\{f_r\}_{r \in [L]}$, i.e., $v(S) \coloneqq \max_{1 \leq j \leq L} \{f_r(S)\}$, for each subset $S \subseteq [m]$.[3]

We use $\Pi_n([m])$ to denote the collection of all n partitions of the indivisible goods $[m]$. An *allocation* is an n-partition $\mathcal{A} = (A_1, \ldots, A_n) \in \Pi_n([m])$ of the m goods. Here, A_i denotes the subset of goods allocated to agent $i \in [n]$ and will be referred to as a *bundle*.

[3] Here, L can be exponentially large in m.

Given a fair division instance $\mathcal{I} = \langle [m], [n], \{v_i\}_i \rangle$, the *Nash social welfare* of allocation $\mathcal{A}$ is defined as the geometric mean of the agents' valuations under $\mathcal{A}$: $\text{NSW}(\mathcal{A}) := (\prod_{i=1}^{n} v_i(A_i))^{\frac{1}{n}}$.

We will throughout use $\mathcal{N}^* = (N_1^*, \ldots, N_n^*)$ to denote an allocation that maximizes the Nash social welfare for a given fair division instance. We refer to $\mathcal{N}^*$ as a Nash optimal allocation. An allocation $\mathcal{P} = (P_1, \ldots, P_n)$ is an α-approximate solution (with $\alpha \geq 1$) of the Nash social welfare maximization problem if $\text{NSW}(\mathcal{P}) \geq \frac{1}{\alpha}\text{NSW}(\mathcal{N}^*)$.

Besides the Nash social welfare, we address a family of objectives defined by considering the *generalized means* of agents' valuations. In particular, for parameter $p \in \mathbb{R}$, the the p^{th} generalized (Hölder) mean $\text{M}_p(\cdot)$ of n nonnegative numbers $x_1, \ldots, x_n \in \mathbb{R}_+$ is defined as $\text{M}_p(x_1, \ldots, x_n) := \left(\frac{1}{n}\sum_{i=1}^{n} x_i^p\right)^{\frac{1}{p}}$.

Parameterized by p, this family of functions captures multiple fairness and efficiency measures. In particular, when $p = 1$, M_p reduces to the arithmetic mean. In the limit, M_p is equal to the geometric mean as p tends to zero. In addition, $\lim_{p\to-\infty} \text{M}_p(x_1, \ldots, x_n) = \min\{x_1, x_2, \ldots, x_n\}$.

We define the *p-mean welfare*, $\text{M}_p(\mathcal{A})$, of an allocation $\mathcal{A} = (A_1, A_2, \ldots, A_n)$ as

$$\text{M}_p(\mathcal{A}) := \text{M}_p(v_1(A_1), \ldots, v_n(A_n)) = \left(\frac{1}{n}\sum_{i=1}^{n} v_i(A_i)^p\right)^{1/p}.$$

With p equal to one, zero, and $-\infty$, the p-mean welfare corresponds to the (average) social welfare, Nash social welfare, and egalitarian welfare, respectively.

The following proposition implies that for any $p \leq -n\log n$, if instead of the p-mean welfare, we maximize the egalitarian welfare, then the resulting allocation loses a negligible factor in the approximation ratio.

▶ **Proposition 1.** *For any n nonnegative numbers $x_1, \ldots, x_n \in \mathbb{R}_+$ and $p \leq -n\log n$, we have*

$$\text{M}_{-\infty}(x_1, \ldots, x_n) \;\leq\; \text{M}_p(x_1, \ldots, x_n) \;\leq\; 2^{1/n}\,\text{M}_{-\infty}(x_1, \ldots, x_n).$$

3 An $8n$-Approximation for Nash Social Welfare

This section presents an efficient $8n$-approximation algorithm for the Nash social welfare maximization problem, under subadditive valuations. Our algorithm, Algorithm 1 (Alg), requires access to the valuation functions through basic value queries, i.e., it only requires an oracle which, when queried with a subset of goods $S \subseteq [m]$ and an agent $i \in [n]$, returns $v_i(S) \in \mathbb{R}_+$.

We first describe the ideas behind our algorithm. Write $\mathcal{N}^* = \{N_1^*, \ldots, N_n^*\}$ denote a Nash optimal allocation in the given instance and let us, for now, assume that the agents have additive valuations, i.e., for all agents $i \in [n]$ and subset of goods $S \subseteq [m]$, we have $v_i(S) = \sum_{g\in S} v_i(g)$. In the following two cases, we can readily obtain an $O(n)$ approximation. In the first case, each agent has a few "high-value" goods, i.e., each agent i has a good $g_i' \in N_i^*$ with the property that $v_i(g_i') \geq v_i(N_i^*)/n$. In such a setting, we can construct a complete bipartite graph with agents $[n]$ on one side and all the goods $[m]$ on the other. Here, the weight of edge $(i, g) \in [n] \times [m]$ is set to be $\log(v_i(g))$. In this bipartite graph, the matching $(i, g_i')_{i\in[n]}$ has Nash social welfare at least $1/n$ times the optimal and, hence, this also holds for a left-perfect maximum-weight matching in this graph.

Algorithm 1 ALG.

Input: Instance $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ with value oracle access to the valuation functions v_is.

Output: An allocation $\mathcal{B} = (B_1, B_2, \ldots, B_n)$

Initialize iteration count $t = 0$ and define $\text{SAT}_t = \emptyset$ and $\text{UNSAT}_t = [n]$
for $i \in [n]$ **do**
  Sort the goods in $[m]$ in descending order of value such that $v_i(g_1) \geq \cdots \geq v_i(g_m)$
  if $v_i([m] \setminus \{g_1, \ldots, g_{2n}\}) = 0$ **then**
    Set $\gamma_i^t = 0$
  else
    $\gamma_i^t = v_i([m])$
  end if
end for
while $\text{UNSAT}_t \neq \emptyset$ **do**
  Consider the bipartite graph $\left([n] \cup [m], [n] \times [m], \{w(i,g)\}_{i \in [n], g \in [m]}\right)$ with weight of edge $(i, g) \in [n] \times [m]$ set as $w(i, g) = \log\left(v_i(g) + \gamma_i^t\right)$
  Compute a left-perfect maximum-weight matching, π^t, in this bipartite graph
  Set $G = [m] \setminus \{\pi^t(i)\}_{i \in [n]}$ and $A = [n]$
  while there exists $a' \in A$ and $g' \in G$ such that $v_{a'}(g') \geq \frac{1}{2n} v_{a'}(G)$ **do**
    Set $B_{a'}^t = \{g'\}$ and update $G \leftarrow G \setminus \{g'\}$ along with $A \leftarrow A \setminus \{a'\}$
  end while
  Set $(B_i^t)_{i \in A} = \text{MOVINGKNIFE}\left(G, A, \{v_i\}_{i \in A}\right)$
  Define $\text{SAT}_{t+1} = \{i \in [n] \mid v_i(B_i^t) \geq \gamma_i^t\}$ and set $\gamma_i^{t+1} = \gamma_i^t$ for each $i \in \text{SAT}_{t+1}$
  Define $\text{UNSAT}_{t+1} = \{i \in [n] \mid v_i(B_i^t) < \gamma_i^t\}$ and set $\gamma_i^{t+1} = \left(1 - \frac{1}{m}\right) \gamma_i^t$ for each $i \in \text{UNSAT}_{t+1}$
  Update $t \leftarrow t + 1$
end while
return allocation $\left(B_1^{t-1} \cup \{\pi^{t-1}(1)\}, B_2^{t-1} \cup \{\pi^{t-1}(2)\}, \ldots, B_n^{t-1} \cup \{\pi^{t-1}(n)\}\right)$

In the second case, all goods are of "low-value", i.e., for all $i \in [n]$ and $g \in [m]$ we have $v_i(g) \leq v_i(N_i^*)/(2n)$. Here again an $O(n)$ approximation can be obtained via a simple round-robin algorithm, wherein the agents (in an arbitrary order) repeatedly pick their highest valued good from those remaining. At a high level, our algorithm stitches together these two extreme cases by first matching high-value goods and then allocating the low-value ones.

We connect the two cases by considering the following quantity for each agent $i \in [n]$

$$\ell_i \coloneqq \min_{S \subseteq [m] : |S| \leq 2n} \frac{1}{2n} \, v_i\left([m] \setminus S\right) . \tag{1}$$

That is, ℓ_i is the value that each agent is guaranteed to achieve in a (near) proportional allocation, even after the removal of any $2n$-size subset of goods. Our algorithm leverages the following existential guarantee (Lemma 3): there necessarily exists a good $\widehat{g}_i \in N_i^*$ with the property that

$$v_i(\widehat{g}_i) + \ell_i \geq \frac{1}{4n} v_i(N_i^*) . \tag{2}$$

This result ensures that, a single high-value good (in particular, $\widehat{g}_i$) coupled with a $2n$-approximation to all the low-value goods (i.e., ℓ_i), is sufficient to ensure a $4n$-approximation *for each agent*. At this point, if we could (i) explicitly compute ℓ_i for each agent i and (ii)

for any size-n subset of goods S, assign the remaining goods $[m] \setminus S$ such that each agent gets a bundle of value at least ℓ_i, then we would be done. This follows from the observation that in the complete bipartite graph $([n] \cup [m], [n] \times [m])$ with weight of edge (i, g) set to $\log(v_i(g) + \ell_i)$, the weight of the matching $(i, \widehat{g}_i)_i$ is a $4n$ approximation to the optimal Nash social welfare by equation (2) and, hence, the same guarantee holds for a maximum-weight matching in the graph. Condition (ii) ensures that each agent also receives at least ℓ_i after the initial assignment of the n matched goods.

For additive valuations, both conditions (i) and (ii) can be satisfied. This template was employed in the SMatch algorithm (for additive valuations) of Garg et al. [20]. However, for submodular (and subadditive) valuations, the quantity ℓ_i is hard to approximate within a sub-linear factor [28].

Therefore, instead of satisfying condition (i) explicitly, we maintain an upper bound $\gamma_i \geq \ell_i$ for each agent i. Our algorithm first obtains a maximum weight matching in the bipartite graph between agents and goods with the weight of edge $(i, g) \in [n] \times [m]$ set to $\log(v_i(g) + \gamma_i)$. It assigns all the matched goods to the respective agents, removes these goods from further consideration in this iteration, and then carries out a procedure (described below) to ensure condition (ii). If, for agent i, the bundle obtained in this procedure (i.e., the bundle obtained for i after removing the matched goods) has value less than γ_i, then we multiplicatively reduce the (over) estimate γ_i for i and repeat the algorithm.

The procedure towards satisfying condition (ii) consists of two steps. Let G be the set of goods that remain once we remove the matched n goods from $[m]$. In the first step, if there exists an agent i and a good $g \in G$ such that $v_i(g) \geq v_i(G)/(2n)$, we assign g to i and remove both from further consideration. An agent thus removed has value ℓ_i from the assigned good; note that, by definition, $\ell_i \leq v_i(G)/(2n)$. After this step, we observe that $v_i(g) \leq v_i(G)/(2n)$ for each remaining agent i and good g. In the second step, we run a *moving knife* subroutine (Algorithm 2) on the goods that are still unassigned. In this subroutine, the goods are initially ordered in an arbitrary fashion. A hypothetical knife is then moved across the goods from one side until an agent i (who has yet to receive a bundle) calls out that the goods covered so far have a collective value of at least $v_i(G)/(2n)$ for her. These covered goods are then allocated to said agent i and both the agent as well as this bundle is removed from further consideration. We show that this allocation satisfies condition (ii), i.e., the bundle assigned to each agent in this procedure has value at least ℓ_i (but it may be lower than the overestimate γ_i).

Since we can guarantee ℓ_i for each agent i, irrespective of which goods are removed in the matching step, γ_i never goes below ℓ_i, for any agent. Hence, at some point, every agent i receives a bundle of value at least γ_i in the above two steps. We show that these bundles, with the goods matched with each agent, provide an $8n$ approximation to the optimal Nash social welfare.

It is relevant to note we use ℓ_i solely for the purposes of analysis. Our algorithm executes with the overestimate γ_i and keeps reducing this value till it is realized (in the two-step procedure) for all the agents.

As mentioned previously, the SMatch algorithm (developed for additive valuations) of Garg et al. [20] relies of conditions (i) and (ii). However, for submodular valuations their work diverges considerably from the current approach. In particular, the RepReMatch algorithm (developed for submodular valuations) in [20] first finds a set of goods $\mathcal{G}$ with the property that in the bipartite graph between all the agents and $\mathcal{G}$, there is a matching wherein every agent is matched to a good with value at least as much as her highest valued good in N_i^*. To ensure this property the cardinality of $\mathcal{G}$ needs to be $n \log n$. Intuitively,

Algorithm 2 MovingKnife.

Input: Instance $\mathcal{J} = \langle G, A, \{v_i\}_{i \in A} \rangle$ with value oracle access to the valuation functions v_is
Output: An allocation $\mathcal{P} = (P_1, P_2, \ldots, P_{|A|})$

Initialize $S = \emptyset$, $\widehat{G} = G$, $\widehat{A} = A$, and bundle $P_i = \emptyset$ for all $i \in A$.
while $\widehat{G} \neq \emptyset$ and $\widehat{A} \neq \emptyset$ **do**
  Select any arbitrary good $g \in \widehat{G}$ and update $S \leftarrow S \cup \{g\}$ along with $\widehat{G} \leftarrow \widehat{G} \setminus \{g\}$.
  if for some agent $\widehat{a} \in \widehat{A}$ we have $v_{\widehat{a}}(S) \geq \frac{1}{2n} v_{\widehat{a}}(G)$ **then**
    Set $P_{\widehat{a}} = S$ and update $\widehat{A} \leftarrow \widehat{A} \setminus \{\widehat{a}\}$ along with $S \leftarrow \emptyset$.
  end if
end while
if $\widehat{G} \neq \emptyset$ **then**
  $P_{|A|} \leftarrow P_{|A|} \cup \widehat{G}$
end if
return allocation $\mathcal{P} = (P_1, \ldots, P_{|A|})$.

this requirement leads to a lower bound of $\Omega(n \log n)$ on the approximation ratio obtained in [20]. Furthermore, the steps in their algorithm to ensure condition (ii) do not extend to subadditive valuations either. Specifically, Garg et al. [20] note that their algorithm gives an approximation ratio of $\Omega(m)$ for the case of subadditive valuations. The $2n$-approximation of Khot and Ponnuswami for egalitarian welfare [23] first guesses the optimal egalitarian welfare b, and uses this to partition the goods into "large" ones (those with value higher than b/n) and "small" ones, for each agent. It then tries to ensure every agent receives a bundle with valuation at least b/n. For Nash social welfare, guessing just a single value does not appear to help, since the Nash social welfare depends on the valuation of each agent.

The following theorem constitutes our main result for Nash social welfare.

▶ **Theorem 2.** *Let $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ be a fair division instance in which the valuation function v_i, of each agent $i \in [n]$, is nonnegative, monotone, and subadditive. Given value oracle access to v_is, the algorithm Alg computes an $8n$ approximation to the Nash optimal allocation in polynomial time.*

The following lemma proves inequality (2). We state and prove it for an arbitrary allocation $\mathcal{A}^* = (A_1^*, \ldots, A_n^*)$, rather than just for the Nash optimal allocation.

▶ **Lemma 3.** *Let $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ be a fair division instance with monotone, subadditive valuations and let $\mathcal{A}^* = (A_1^*, \ldots, A_n^*)$ be any allocation in $\mathcal{I}$. Let $\widehat{g}_i$ be the most valued (by i) good in A_i^* (i.e., $\widehat{g}_i \coloneqq \arg\max_{g \in A_i^*} v_i(g)$) and ℓ_i be as defined in (1). Then, for each agent $i \in [n]$*

$$v_i(\widehat{g}_i) + \ell_i \geq \frac{1}{4n} v_i(A_i^*).$$

Proof. Consider any agent $i \in [n]$ and note that $\ell_i \geq 0$. We will establish the lemma by considering two complementary cases.

Case 1. There exists a good $g_i \in A_i^*$ with the property that $v_i(g_i) \geq \frac{1}{4n} v_i(A_i^*)$. Since $\widehat{g}_i$ is the most valued good in A_i^*, we have $v_i(\widehat{g}_i) \geq v_i(g_i)$ and the desired inequality follows.

Case 2. For all goods $g \in A_i^*$, $v_i(g) < \frac{1}{4n} v_i(A_i^*)$. Recall that $\ell_i \coloneqq \min_{S \subseteq [m], |S| \leq 2n} \frac{1}{2n} v_i([m] \setminus S)$. Let S^* be the set S that induces ℓ_i, i.e., $\ell_i = \frac{1}{2n} v_i([m] \setminus S^*)$. Monotonicity of v_i ensures that $|S^*| = 2n$ and

$$\ell_i = \frac{1}{2n} v_i([m] \setminus S^*) \geq \frac{1}{2n} v_i(A_i^* \setminus S^*)\,. \tag{3}$$

Furthermore, given that in the current case $v_i(g) < \frac{1}{4n} v_i(A_i^*)$ for all $g \in A_i^*$, we have

$$v_i(A_i^* \cap S^*) \leq \sum_{g \in A_i^* \cap S^*} v_i(g) < \sum_{g \in A_i^* \cap S^*} \frac{1}{4n} v_i(A_i^*) \leq \frac{|S^*|}{4n} v_i(A_i^*) = \frac{1}{2} v_i(A_i^*)\,. \tag{4}$$

Here, the first inequality follows from the fact that v_i is subadditive and the last since $|S^*| = 2n$.

Therefore, we obtain the desired bound in terms of ℓ_i:

$$\begin{aligned}
\ell_i &\geq \frac{1}{2n} v_i(A_i^* \setminus S^*) && \text{(via inequality (3))} \\
&\geq \frac{1}{2n} \left(v_i(A_i^*) - v_i(A_i^* \cap S^*) \right) && (v_i \text{ is subadditive}) \\
&\geq \frac{1}{4n} v_i(A_i^*) && \text{(via inequality (4))}
\end{aligned}$$

Thus, the the stated inequality $v_i(\widehat{g}_i) + \ell_i \geq \frac{1}{4n} v_i(A_i^*)$ holds even in this case. ◀

The next lemma establishes the key property of Algorithm 2 (MovingKnife): if all the goods have low value for every agent, then MovingKnife returns a near-proportional allocation.

▶ **Lemma 4.** *Consider a fair division instance $\langle G, A, \{v_i\}_{i \in A} \rangle$ wherein the agents have monotone, subadditive valuations. In addition, suppose for each agent $i \in A$ and good $g \in G$ we have $v_i(g) < \frac{1}{2n} v_i(G)$, where $n \geq |A|$. Then the allocation $(P_1, \ldots, P_{|A|})$ returned by Algorithm 2 (MovingKnife) satisfies $v_i(P_i) \geq \frac{1}{2n} v_i(G)$ for all $i \in A$.*

Proof. Given instance $\langle G, A, \{v_i\}_{i \in A} \rangle$, the MovingKnife algorithm (Algorithm 2) considers the goods in an arbitrary order and adds these goods one by one into a bundle S until an agent $\widehat{a}$ calls out that its value for S is at least $\frac{1}{2n} v_{\widehat{a}}(G)$. We assign these goods to agent $\widehat{a}$ and remove them – along with $\widehat{a}$ – from consideration. The algorithm iterates over the remaining set of agents and goods. We will show that the while loop in the MovingKnife algorithm terminates with $\widehat{A} = \emptyset$ and, hence, assigns to each agent a bundle of desired value.

Consider an integer (count) $k \in \mathbb{N}$. Let $\widehat{G}$ and $\widehat{A}$ denote the set of goods and agents, respectively, that are left unassigned after k agents are assigned bundles in MovingKnife; note that $|\widehat{A}| = |A| - k$. The arguments below establish that for each remaining agent $i \in \widehat{A}$,

$$v_i(\widehat{G}) \geq \left(1 - \frac{k}{n}\right) v_i(G) \tag{5}$$

Therefore, for any $k < |A| \leq n$, the set of unassigned goods $\widehat{G}$ is nonempty and even the last agent (i.e., with $k = |A| - 1$) receives a bundle of sufficiently high value.

To prove (5), consider any agent $i \in \widehat{A}$. Indeed, agent i has not received any goods yet, but the k agents in $A \setminus \widehat{A}$ have been assigned bundles. Let S be a bundle assigned to some agent in $A \setminus \widehat{A}$ (i.e., $P_j = S$ for some $j \in A \setminus \widehat{A}$) and g' be the last good included in S. Step 4 of the algorithm ensures that $v_i(S \setminus \{g'\}) < \frac{1}{2n} v_i(G)$; otherwise, $S \setminus \{g'\}$ would have

been assigned to agent i. Furthermore, the assumption (in the Lemma statement) gives us $v_i(g') \leq \frac{1}{2n} v_i(G)$. Hence, using these inequalities and the subadditivity of v_i, we get $v_i(S) \leq v_i(S \setminus \{g'\}) + v_i(g') \leq \frac{1}{n} v_i(G)$.

This inequality provides an upper bound on $v_i(G \setminus \widehat{G}) = v_i\left(\cup_{j \in A \setminus \widehat{A}} P_j\right)$, the total value of the set of goods assigned among the k agents in $A \setminus \widehat{A}$. Specifically, by the subadditivity of v_i, $v_i(G \setminus \widehat{G}) \leq \frac{k}{n} v_i(G)$. Therefore, $v_i(\widehat{G}) \geq v_i(G) - v_i(G \setminus \widehat{G}) \geq \left(1 - \frac{k}{n}\right) v_i(G)$.

Overall, every agent $i \in A$ is eventually assigned a bundle of value at least $\frac{1}{2n} v_i(G)$ in the while loop. ◀

Next we show that in each iteration of the while loop in ALG (Algorithm 1), the value of the assigned bundle B_i^t is at least as large as ℓ_i.

▶ **Lemma 5.** *Given a fair division instance $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ with subadditive valuations, let B_i^t be the bundle assigned to agent $i \in [n]$ in the t^{th} iteration (for $t \in \mathbb{N}$) of the outer while loop (Step 10) in ALG. Then, for all agents $i \in [n]$ and each iteration count t, we have $v_i(B_i^t) \geq \ell_i$.*

Proof. During any iteration t of the outer while loop (Step 10) in ALG and for any agent $i \in [n]$, the bundle B_i^t either consists of a single good of high value (Step 15), or of the set of goods assigned to agent i obtained after executing the MOVINGKNIFE subroutine (Step 17). We will show that in both cases the stated inequality holds.

Recall that $\ell_i \coloneqq \min_{S \subseteq [m] : |S| \leq 2n} \frac{1}{2n} v_i([m] \setminus S)$. Equivalently, $\ell_i = \min_{T \subseteq [m] : |T| \geq m - 2n} \frac{1}{2n} v_i(T)$. Therefore, we have

$$\frac{1}{2n} v_i(T) \geq \ell_i \qquad \text{for any subset } T \subseteq [m] \text{ of size at least } (m - 2n) \tag{6}$$

The relevant observation here is that, in any iteration t, the set of goods G from which the bundles B_i^ts are populated satisfies $|G| \geq m - 2n$. Specifically, in the t^{th} iteration, we start with $|G| = m - n$ (Step 13). Subsequently, the inner while loop (Step 14) assigns at most n goods and, hence, the number of goods passed on to the MOVINGKNIFE subroutine satisfies $|G| \geq m - 2n$.

First, we note that the lemma holds for any agent a' that receive a singleton bundle $B_{a'}^t = \{g'\}$ in Step 15: $v_{a'}(g') \geq \frac{1}{2n} v_{a'}(G) \geq \ell_{a'}$. Here, the first inequality follows from the selection criterion applied to g' and the second inequality from equation (6) and the fact that $|G| \geq m - 2n$.

Finally, we note that the bound also holds for the remaining agents i that receive a bundle B_i^t through the MOVINGKNIFE subroutine. As mentioned previously, at least $m - 2n$ goods are passed on as input to the subroutine, i.e., if MOVINGKNIFE is executed on instance $\mathcal{J} = \langle G, A, \{v_i\}_{i \in A} \rangle$, then we have $|G| \geq m - 2n$. Inequality (6) ensures that $\frac{1}{2n} v_i(G) \geq \ell_i$ for all $i \in A$. Finally, using Lemma 4, we get that the bundle assigned to agent $i \in A$ satisfies the stated inequality: $v_i(B_i^t) = v_i(P_i) \geq \frac{1}{2n} v_i(G) \geq \ell_i$.

Hence, the stated claim follows. ◀

We now show that the estimates γ_i^ts used in ALG also satisfy a lower bound similar to that in Lemma 5.

▶ **Lemma 6.** *Given a fair division instance $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ with subadditive valuations, let $\gamma_i^t \in \mathbb{R}_+$ be the estimate associated with agent $i \in [n]$ in the t^{th} iteration (for $t \in \mathbb{N}$) of the outer while loop (Step 10) in ALG. Then, for all agents $i \in [n]$ and each iteration count t, we have $\gamma_i^t \geq \left(1 - \frac{1}{m}\right) \ell_i$.*

Proof. Note that for any agent $i \in [n]$, the quantity $\ell_i = 0$ iff i has positive value for at most $2n$ goods. This observation implies that the initial for loop in ALG correctly identifies agents i that have $\ell_i = 0$, and sets $\gamma_i^0 = 0$. For such agents $\gamma_i^t = 0$ for all t. Hence, the lemma holds for any agent i with $\ell_i = 0$.

We now consider agents $i \in [n]$ with $\ell_i > 0$. For such an agent i, the algorithm initially sets $\gamma_i^0 = v_i([m])$. Hence, for $t = 0$ we have $\gamma_i^t \geq \left(1 - \frac{1}{m}\right) \ell_i$. An inductive argument shows that this inequality continues to hold as the algorithm progresses. In particular, if in the t^{th} iteration the algorithm does not decrement the estimate (i.e., if $i \in \text{SAT}_{t+1}$), then $\gamma_i^{t+1} = \gamma_i^t \geq \left(1 - \frac{1}{m}\right) \ell_i$.

Even otherwise, if the algorithm multiplicatively decrements the estimate (in particular, sets $\gamma_i^{t+1} = (1 - 1/m)\ \gamma_i^t$), then it must be the case that $\gamma_i^t > v_i(B_i^t)$ (i.e., $i \in \text{UNSAT}_{t+1}$). That is, after the decrement we have $\gamma_i^{t+1} \geq \left(1 - \frac{1}{m}\right) v_i(B_i^t) \geq \left(1 - \frac{1}{m}\right) \ell_i$; the last inequality follows from Lemma 5. This completes the proof. ◀

3.1 Proof of Theorem 2

In this section we prove Theorem 2 by showing that ALG runs in polynomial time (Lemma 7) and the computed allocation achieves the stated approximation ratio of $8n$ (Lemma 8).

▶ **Lemma 7** (Runtime Analysis). *Given any fair division instance $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ in which the agents have monotone, subadditive valuations, ALG (Algorithm 1) terminates after $T = O\left(nm \log\left(nmV\right)\right)$ iterations of its outer while loop (Step 10); here,*

$$V = \max_{i \in [n]} \left(\frac{\max\limits_{g \in [m]} v_i(g)}{\min\limits_{g \in [m] : v_i(g) > 0} v_i(g)} \right) .$$

Proof. By design, ALG iterates as long as $\text{UNSAT}_t \neq \emptyset$. We will bound the number of times (i.e., the distinct values of t for which) any agent $i \in [n]$ is contained in UNSAT_t and, hence, establish the stated runtime bound.

Recall that for any agent $i \in [n]$, the quantity $\ell_i = 0$ iff i has positive value for at most $2n$ goods. For such agents ALG sets $\gamma_i^0 = 0$. Therefore, these agents are contained in SAT_t, for all iterations $t \geq 1$, and do not contribute to the repetitions of the outer while loop.

For the remaining agents, with $\ell_i > 0$, the algorithm initially sets $\gamma_i^0 = v_i([m])$ and we have

$$\ell_i \geq \frac{1}{2n} \min_{g \in [m] : v_i(g) > 0} v_i(g) \,. \tag{7}$$

Using Lemma 6 and the fact that the algorithm decrements γ_i^t by a multiplicative factor of $(1 - 1/m)$ whenever $i \in \text{UNSAT}_t$, we get that the number of times agent i can be in the UNSAT_t is at most

$$\begin{aligned}
m \log\left(\frac{v_i([m])}{\ell_i}\right) &\leq m \log\left(\frac{m\ \max_{g \in [m]} v_i(g)}{\ell_i}\right) \\
&\qquad \text{(since } v_i \text{ is subadditive, } v_i([m]) \leq m \max_{g \in [m]} v_i(g)\text{)} \\
&\leq m \log\left(\frac{2nm\ \max_{g \in [m]} v_i(g)}{\min\limits_{g \in [m] : v_i(g) > 0} v_i(g)}\right) \qquad \text{(via inequality (7))} \\
&\leq m \log\left(2nmV\right) .
\end{aligned}$$

Summing over all agents, we get that the number of times $\text{UNSAT}_t \neq \emptyset$ is at most $T = O\left(nm \log\left(nmV\right)\right)$. Hence, the stated lemma follows. ◀

We now show that the allocation computed by ALG achieves the required approximation guarantee.

▶ **Lemma 8** (Approximation Guarantee). *For any fair division instance $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ with subadditive valuations, let $\mathcal{B} = (B_1, \ldots, B_n)$ denote the allocation computed by ALG. Then,* $\mathrm{NSW}(\mathcal{B}) \geq \frac{1}{8n}\mathrm{NSW}(\mathcal{N}^*)$*; here, $\mathcal{N}^*$ denotes the Nash optimal allocation in $\mathcal{I}$.*

Proof. For the given instance $\mathcal{I}$, say ALG terminates after $T+1$ iterations of the outer while loop. That is, we have $\mathrm{UNSAT}_{T+1} = \emptyset$ and, for each agent $i \in [n]$, the returned bundle $B_i = B_i^T \cup \{\pi^T(i)\}$. Here, $\pi^T(i)$ is the good assigned to agent i under the maximum weight matching π^T (considered in the last iteration) and B_i^T is the bundle populated for i (either in Step 15 or in Step 17).

The fact that $\mathrm{UNSAT}_{T+1} = \emptyset$ (i.e., $\mathrm{SAT}_{T+1} = [n]$) gives us

$$v_i(B_i^T) \geq \gamma_i^T \qquad \text{for all } i \in [n]\,. \tag{8}$$

Lemma 3 (instantiated with $\mathcal{A}^* = \mathcal{N}^*$) implies that there exists a matching – $\sigma(i) \coloneqq \widehat{g}_i \in N_i^*$, for all $i \in [n]$ – with the property that $v_i(\sigma(i)) + \ell_i \geq \frac{1}{4n} v_i(N_i^*)$. Using this inequality and Lemma 6 we get, for all $i \in [n]$:

$$v_i(\sigma(i)) + \gamma_i^T \geq \left(1 - \frac{1}{m}\right) \frac{1}{4n} v_i(N_i^*)\,. \tag{9}$$

Recall that π^T is a maximum weight matching in the bipartite graph (considered in Step 11 of ALG) with edge weights $\log\left(v_i(g) + \gamma_i^T\right)$. Given that $\sigma(\cdot)$ is some matching in the graph and π^T is a maximum weight matching, we get $\sum_{i=1}^n \log\left(v_i(\pi^T(i)) + \gamma_i^T\right) \geq \sum_{i=1}^n \log\left(v_i(\sigma(i)) + \gamma_i^T\right)$. That is,

$$\left(\prod_{i=1}^n \left(v_i(\pi^T(i)) + \gamma_i^T\right)\right)^{\frac{1}{n}} \geq \left(\prod_{i=1}^n \left(v_i(\sigma(i)) + \gamma_i^T\right)\right)^{\frac{1}{n}} \geq \left(1 - \frac{1}{m}\right) \frac{1}{4n} \mathrm{NSW}(\mathcal{N}^*)\,. \tag{10}$$

The last inequality follows from equation (9). Also, as defined previously, the optimal Nash social welfare $\mathrm{NSW}(\mathcal{N}^*) = \left(\prod_i^n v_i\left(N_i^*\right)\right)^{1/n}$.

The monotonicity of the valuation function v_i implies $v_i\{\pi^T(i)\} \cup B_i^T) \geq 1/2(v_i(\pi^T(i)) + v_i(B_i^T))$ for each $i \in [n]$. Using these observations we can lower bound the Nash social welfare of the computed allocation $\left(B_i = \{\pi^T(i)\} \cup B_i^T\right)_i$ as follows

$$\begin{aligned}
\left(\prod_{i=1}^n v_i(B_i)\right)^{\frac{1}{n}} &\geq \frac{1}{2}\left(\prod_{i=1}^n \left(v_i(\pi^T(i)) + v_i(B_i^T)\right)\right)^{\frac{1}{n}} \\
&\geq \frac{1}{2}\left(\prod_{i=1}^n \left(v_i(\pi^T(i)) + \gamma_i^T\right)\right)^{\frac{1}{n}} && \text{(via inequality (8))} \\
&\geq \left(1 - \frac{1}{m}\right) \frac{1}{8n} \mathrm{NSW}(\mathcal{N}^*)\,. && \text{(via inequality (10))}
\end{aligned}$$

This establishes the stated approximation guarantee and completes the proof of the lemma. ◀

▶ Remark. The result of Garg et al. [20] also holds for an asymmetric version of Nash social welfare maximization, in which each agent i has an associated weight $\eta_i \geq 0$ and the goal is to find an allocation $(A_1, \ldots, A_n)$ that maximizes $\left(\prod_{i \in [n]} \left(v_i(A_i)\right)^{\eta_i}\right)^{\frac{1}{\sum_{i \in [n]} \eta_i}}$. Our

approximation guarantee extends to this formulation. In particular, in Step 11 of ALG we can set the edges weights to be $\eta_i \log(v_i(g)+\gamma_i)$ (instead of $\log(v_i(g)+\gamma_i)$) and note that the subsequent arguments follow through to provide an $8n$-approximation ratio for maximizing Nash social welfare with asymmetric agents and subadditive valuations.

Also, one can use Theorem 2, in conjunction with the m/n approximation guarantee of Nguyen and Rothe [26],[4] to obtain an $O(\sqrt{m})$-approximation algorithm for maximizing Nash social welfare under subadditive valuations: for instances in which $m \geq n^2$, the $8n$ approximation suffices. Otherwise, if $m < n^2$ (i.e., $m/n < \sqrt{m}$), then we can invoke the result of Nguyen and Rothe [26].

4 An $8n$-Approximation for p-Mean Welfare

This section shows that we can extend Algorithm 1 and obtain an $8n$ approximation for maximizing the p-mean welfare as well.

For maximizing p-mean welfare, ALG (Algorithm 1) is modified as follows: In Step 11, the weight $w(i,g)$ of edge $(i,g) \in [n] \times [m]$ is set as $(v_i(g)+\gamma_i^t)^p$ (instead of $\log(v_i(g)+\gamma_i^t)$).[5] Furthermore,

- **(i)** For $p \in (0,1]$, in Step 12 we compute a left-perfect *maximum*-weight matching, π^t, otherwise
- **(ii)** For finite $p < 0$, we compute a left-perfect *minimum*-weight matching, π^t, in Step 12
- **(iii)** For maximizing egalitarian welfare (the $p = -\infty$ case), we set edge weights to be $(v_i(g)+\gamma_i^t)$ and compute a max-min matching[6] π^t with respect to these weights.

Theorem 9 below establishes that, with these changes in ALG (Algorithm 1), we can efficiently compute an allocation with p-mean welfare at least $\frac{1}{8n}$ times the optimal (p-mean welfare). Note that by Proposition 1, for $p \leq -n \log n$, we can maximize the egalitarian welfare, instead of the p-mean welfare, and the allocation thus obtained is an $8n$-approximation to the optimal p-mean welfare allocation.

▶ **Theorem 9.** *Let $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ be a fair division instance in which the valuation function v_i, of each agent $i \in [n]$, is nonnegative, monotone, and subadditive. Then, given value oracle access to v_is , one can efficiently compute an $8n$ approximation to the optimal p-mean welfare for any $p \in (-\infty, 1]$.*

Proof. We first note that Lemmas 3, 4, 5, 6, and 7 hold as is for p-mean welfare. In particular, using Lemma 7 we get that, even with the above-mentioned changes, the algorithm runs in polynomial time.

To complete the proof of the theorem, we will next show that the computed allocation $\mathcal{B} = (B_1, \ldots, B_n)$ satisfies $\mathrm{M}_p(\mathcal{B}) \geq \frac{1}{8n}\mathrm{M}_p(\mathcal{A}^*(p))$, where $\mathcal{A}^*(p)$ is a p-mean welfare maximizing allocation.

For the given instance $\mathcal{I}$, say the modified algorithm terminates after $T+1$ iterations of the outer while loop. That is, we have $\mathrm{UNSAT}_{T+1} = \emptyset$ and, for each agent $i \in [n]$, the returned bundle $B_i = B_i^T \cup \{\pi^T(i)\}$. Here, $\pi^T(i)$ is the good assigned to agent i under the matching π^T (considered in the last iteration) and B_i^T is the bundle populated for i in the final iteration.

[4] While Theorem 4 in [26] provides the above-mentioned approximation guarantee of $(m-n+1)$, its proof can in fact be easily modified to obtain an approximation ratio of m/n.

[5] Recall that the $p = 0$ case corresponds to Nash social welfare. Since we already have the desired approximation guarantee for this case, it is not explicitly addressed in this section.

[6] In particular, via binary search (over edge weights), we find a matching wherein the minimum edge weight (across agents) is as high as possible.

The fact that $\text{UNSAT}_{T+1} = \emptyset$ (i.e., $\text{SAT}_{T+1} = [n]$) gives us

$$v_i(B_i^T) \geq \gamma_i^T \qquad \text{for all } i \in [n] \tag{11}$$

Lemma 3 implies that there exists a matching – $\sigma(i) \coloneqq \widehat{g}_i \in A_i^*(p)$, for all $i \in [n]$ – with the property that $v_i(\sigma(i)) + \ell_i \geq \frac{1}{4n} v_i(A_i^*(p))$. Using this inequality and Lemma 6 we get, for all $i \in [n]$:

$$v_i(\sigma(i)) + \gamma_i^T \geq \left(1 - \frac{1}{m}\right) \frac{1}{4n} v_i(A_i^*(p)) \tag{12}$$

Recall that, given p, the modified algorithm computes π^T based on the sign of p. Hence, we split the proof of Theorem 9 into three cases depending on whether $p > 0$, $p < 0$, or $p = -\infty$.

Case (i). $p > 0$. In this case, π^T is a left-perfect maximum-weight matching in the bipartite graph $([n] \cup [m], [n] \times [m])$ with edge weights $\left(v_i(g) + \gamma_i^T\right)^p$. Given that $\sigma(\cdot)$ is some (left-perfect) matching in the graph and π^T is a maximum-weight matching, we get $\sum_{i=1}^n \left(v_i(\pi^T(i)) + \gamma_i^T\right)^p \geq \sum_{i=1}^n \left(v_i(\sigma(i)) + \gamma_i^T\right)^p$. Therefore, with $p > 0$, the following inequality holds

$$\left(\frac{1}{n}\sum_{i=1}^n \left(v_i(\pi^T(i)) + \gamma_i^T\right)^p\right)^{\frac{1}{p}} \geq \left(\frac{1}{n}\sum_{i=1}^n \left(v_i(\sigma(i)) + \gamma_i^T\right)^p\right)^{\frac{1}{p}} \geq \left(1 - \frac{1}{m}\right) \frac{1}{4n} \mathrm{M}_p(\mathcal{A}^*(p)) \,. \tag{13}$$

The last inequality follows from (12).

Case (ii). Finite $p < 0$. By design, in this case, π^T is a left-perfect minimum-weight matching in the bipartite graph $([n] \cup [m], [n] \times [m])$ with edge weights $\left(v_i(g) + \gamma_i^T\right)^p$. Given that $\sigma(\cdot)$ is some left-perfect matching in the graph and π^T is a minimum-weight matching, we get $\sum_{i=1}^n \left(v_i(\pi^T(i)) + \gamma_i^T\right)^p \leq \sum_{i=1}^n \left(v_i(\sigma(i)) + \gamma_i^T\right)^p$. The fact that p is negative gives us

$$\left(\frac{1}{n}\sum_{i=1}^n \left(v_i(\pi^T(i)) + \gamma_i^T\right)^p\right)^{\frac{1}{p}} \geq \left(\frac{1}{n}\sum_{i=1}^n \left(v_i(\sigma(i)) + \gamma_i^T\right)^p\right)^{\frac{1}{p}} \geq \left(1 - \frac{1}{m}\right) \frac{1}{4n} \mathrm{M}_p(\mathcal{A}^*(p)) \tag{14}$$

The last inequality follows from (12).

Case (iii). $p = -\infty$. In this case, π^T is a max-min matching computed with edge weights $\left(v_i(\pi^T(i)) + \gamma_i^T\right)$. Given that $\sigma(\cdot)$ is some matching in the graph and matching π^T maximizes the value of the minimum matched edge, we get $\min_{i\in[n]}(v_i(\pi^T(i)) + \gamma_i^T) \geq \min_{i\in[n]}(v_i(\sigma^T(i)) + \gamma_i^T)$. Therefore,

$$\min_{i\in[n]}(v_i(\pi^T(i)) + \gamma_i^T) \geq \min_{i\in[n]}(v_i(\sigma^T(i)) + \gamma_i^T) \geq \left(1 - \frac{1}{m}\right) \frac{1}{4n} \mathrm{M}_p(\mathcal{A}^*(p)) \tag{15}$$

The last inequality follows from (12).

The monotonicity of the valuation function v_i implies $v_i(\{\pi^T(i)\} \cup B_i^T) \geq 1/2(v_i(\pi^T(i)) + v_i(B_i^T))$ for each $i \in [n]$. Using these observations we can lower bound the p-mean welfare of the computed allocation $\left(B_i = \{\pi^T(i)\} \cup B_i^T\right)_i$ as follows

$$\left(\sum_{i=1}^{n}(v_i(B_i))^p\right)^{\frac{1}{p}} \geq \frac{1}{2}\left(\sum_{i=1}^{n}\left(v_i(\pi^T(i))+v_i(B_i^T)\right)^p\right)^{\frac{1}{p}}$$
$$\geq \frac{1}{2}\left(\sum_{i=1}^{n}\left(v_i(\pi^T(i))+\gamma_i^T\right)^p\right)^{\frac{1}{p}} \qquad \text{(via inequality (11))}$$
$$\geq \left(1-\frac{1}{m}\right)\frac{1}{8n}\mathrm{M}_p(\mathcal{A}^*(p)) \qquad \text{(via inequality (13), (14), or (15))}$$

This establishes the stated approximation guarantee and completes the proof of the theorem. ◀

5 Lower Bound on Approximating p-Mean Welfare

This section shows that, under XOS valuations, maximizing the p-mean welfare for $p \in (-\infty, 1]$ within a sub-linear (in n) approximation factor necessarily requires an exponential number of value queries (Theorem 10). This result directly implies that the approximation ratio obtained in Theorems 2 and 9 (via polynomially many value queries) is essentially tight. We note that this query lower bound is unconditional, i.e., it does not depend on any complexity theoretic assumption.

We establish Theorem 10 by directly adapting a result of Dobzinski et al. [16], which provides a similar lower bound for social welfare. The impossibility result here holds under XOS valuations;[7] recall that XOS valuations constitute a special class of subadditive functions.

▶ **Theorem 10.** *For fair division instances $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ with XOS valuations and $p \in (-\infty, 1]$, finding an allocation with p-mean welfare at least $1/n^{1-\varepsilon}$ times the optimal requires exponentially many value queries; here $\varepsilon > 0$ is any fixed constant.*

Here, we briefly explain the salient points of the proof of this lower bound and provide the details in the full version of the current paper [4]. Dobzinski et al. [16] construct two (families of) instances, both with n agents, $m = n^2$ goods, and XOS valuations for the agents. In the first instance, each agent has the same valuation function $f : 2^{[m]} \mapsto \mathbb{R}_+$ and maximum average social welfare (1-mean welfare) is $n^{4\delta}$, for a fixed constant $\delta > 0$. In the second instance, each agent has her own (non-identical) valuation function $v_i : 2^{[m]} \mapsto \mathbb{R}_+$ and there exists an allocation in which each agent has value n for her bundle. For any $p \leq 1$, it follows that in the first instance the optimal p-mean welfare is at most $n^{4\delta}$ (via the generalized mean inequality), while for the second instance, the optimal p-mean welfare is at least n (since there exists an allocation where every agent achieves value n). The proof of Dobzinski et al. [16] goes on to show that it takes an exponential number of value queries to distinguish between the two instances. However, given an $O(n^{1-\varepsilon})$-approximation algorithm for the p-mean welfare, one can readily distinguish between the two instances (by choosing $\delta < \varepsilon/4$). Hence such an algorithm must make an exponential number of value queries.

[7] Our results work under the value oracle model and do not require an explicit description of the underlying additive functions that define the XOS function at hand.

6 $(m - n + 1)$-Approximation Guarantees

This section provides two extensions of the result of Nguyen and Rothe [26], which shows the Nash social welfare maximization problem (under subadditive valuations) admits an $(m - n + 1)$-approximation algorithm. First, we show that an $(m - n + 1)$-approximation for the p-mean welfare can be obtained for all $p \leq 0$ and with subadditive valuations. Then, we establish that it is NP-hard to extend this positive result to any $p \in (0, 1)$, even under additive valuations, i.e., it is NP-hard to obtain an $(m - n + 1)$-approximation for $0 < p < 1$. The proofs of these two results are delegated to the full version of this paper [4].

▶ **Theorem 11.** *Let $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ be a fair division instance in which the valuation function v_i, of each agent $i \in [n]$, is nonnegative, monotone, and subadditive. Then, given value oracle access to v_is, one can efficiently compute an $(m - n + 1)$ approximation to the p-mean welfare maximization problem for any $p \in (-\infty, 0]$.*

The next theorem asserts that it is unlikely that Theorem 11 extends to $p \in (0, 1)$.

▶ **Theorem 12.** *For fair division instances $\mathcal{I} = \langle [m], [n], \{v_i\}_{i=1}^n \rangle$ with additive valuations and for any fixed $p \in (0, 1)$, computing an allocation with p-mean welfare at least $1/(m-n+1)$-times the optimal (for all m and n) is* NP*-hard.*

Note that this hardness result (in light of Theorem 9) is relevant for instances in which $m < 2n$.

References

1 Nima Anari, Shayan Oveis Gharan, Tung Mai, and Vijay V Vazirani. Nash Social Welfare for Indivisible Items under Separable, Piecewise-Linear Concave Utilities. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2274–2290, 2018.

2 Nima Anari, Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. Nash Social Welfare, Matrix Permanent, and Stable Polynomials. In *Proceedings of the 8th Conference on Innovations in Theoretical Computer Science (ITCS)*, 2017.

3 Haris Aziz. Developments in multi-agent fair allocation. In *AAAI*, pages 13563–13568, 2020.

4 Siddharth Barman, Umang Bhaskar, Anand Krishna, and Ranjani G. Sundaram. Tight approximation algorithms for p-mean welfare under subadditive valuations. *CoRR*, abs/2005.07370, 2020. `arXiv:2005.07370`.

5 Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Finding fair and efficient allocations. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, pages 557–574. ACM, 2018.

6 Xiaohui Bei, Jugal Garg, Martin Hoefer, and Kurt Mehlhorn. Earning Limits in Fisher Markets with Spending-Constraint Utilities. In *Proceedings of the International Symposium on Algorithmic Game Theory (SAGT)*, pages 67–79, 2017.

7 Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3):11–18, 2005.

8 Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia. *Handbook of computational social choice*. Cambridge University Press, 2016.

9 Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum nash welfare. *ACM Trans. Economics and Comput.*, 7(3):12:1–12:32, 2019.

10 Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 107–116. IEEE Computer Society, 2009.

11 Bhaskar Ray Chaudhury, Yun Kuen Cheung, Jugal Garg, Naveen Garg, Martin Hoefer, and Kurt Mehlhorn. On fair division for indivisible items. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

12 Bhaskar Ray Chaudhury, Jugal Garg, and Ruta Mehta. Fair and efficient allocations under subadditive valuations, 2020. `arXiv:2005.06511`.

13 Richard Cole, Nikhil R. Devanur, Vasilis Gkatzelis, Kamal Jain, Tung Mai, Vijay V. Vazirani, and Sadra Yazdanbod. Convex program duality, fisher markets, and nash social welfare. In *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, pages 459–460. ACM, 2017.

14 Richard Cole and Vasilis Gkatzelis. Approximating the Nash Social Welfare with Indivisible Items. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC)*, pages 371–380, 2015.

15 Vincent Conitzer, Rupert Freeman, and Nisarg Shah. Fair public decision making. In *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, pages 629–646. ACM, 2017.

16 Shahar Dobzinski, Noam Nisan, and Michael Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. *Math. Oper. Res.*, 35(1):1–13, 2010.

17 Ulle Endriss. *Trends in Computational Social Choice*. Lulu. com, 2017.

18 Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM J. Comput.*, 39(1):122–142, 2009.

19 Jugal Garg, Martin Hoefer, and Kurt Mehlhorn. Approximating the Nash Social Welfare with Budget-Additive Valuations. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2326–2340, 2018.

20 Jugal Garg, Pooja Kulkarni, and Rucha Kulkarni. Approximating nash social welfare under submodular valuations through (un)matchings. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2673–2687. SIAM, 2020.

21 Michel X. Goemans, Nicholas J. A. Harvey, Satoru Iwata, and Vahab S. Mirrokni. Approximating submodular functions everywhere. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 535–544. SIAM, 2009.

22 Jonathan Goldman and Ariel D Procaccia. Spliddit: Unleashing fair division algorithms. *ACM SIGecom Exchanges*, 13(2):41–46, 2015.

23 Subhash Khot and Ashok Kumar Ponnuswami. Approximation algorithms for the max-min allocation problem. In *APPROX-RANDOM*, 2007.

24 Euiwoong Lee. Apx-hardness of maximizing nash social welfare with indivisible items. *Inf. Process. Lett.*, 122:17–20, 2017.

25 Hervé Moulin. *Fair division and collective welfare*. MIT Press, 2003.

26 Trung Thanh Nguyen and Jörg Rothe. Minimizing envy and maximizing average nash social welfare in the allocation of indivisible goods. *Discret. Appl. Math.*, 179:54–68, 2014.

27 Noam Nisan, T Roughgarden, E Tardos, and Vijay V Vazirani. Algorithmic game theory - Cambridge University Press, 2007.

28 Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal on Computing*, 40(6):1715–1737, 2011.

29 Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 67–74. ACM, 2008.

Mincut Sensitivity Data Structures for the Insertion of an Edge

Surender Baswana
Department of Computer Science & Engineering, IIT Kanpur, India
sbaswana@cse.iitk.ac.in

Shiv Gupta
Department of Computer Science & Engineering, IIT Kanpur, India
shivguptamails@gmail.com

Till Knollmann
Heinz Nixdorf Institute, Paderborn University, Germany
tillk@mail.upb.de

Abstract

Let $G = (V, E)$ be an undirected graph on n vertices with non-negative capacities on its edges. The *mincut sensitivity problem* for the insertion of an edge is defined as follows.

Build a compact data structure for G and a given set $S \subseteq V$ of vertices that, on receiving any edge $(x, y) \in S \times S$ of positive capacity as query input, can efficiently report the set of all pairs from $S \times S$ whose mincut value increases upon insertion of the edge (x, y) to G.

The only result that exists for this problem is for a single pair of vertices (Picard and Queyranne, Mathematical Programming Study, 13 (1980), 8-16). We present the following results for the single source and the all-pairs versions of this problem.

1. *Single source*: Given any designated source vertex s, there exists a data structure of size $\mathcal{O}(|S|)$[1] that can output all those vertices from S whose mincut value to s increases upon insertion of any given edge. The time taken by the data structure to answer any query is $\mathcal{O}(|S|)$.
2. *All-pairs*: There exists an $\mathcal{O}(|S|^2)$ size data structure that can output all those pairs of vertices from $S \times S$ whose mincut value gets increased upon insertion of any given edge. The time taken by the data structure to answer any query is $\mathcal{O}(k)$, where k is the number of pairs of vertices whose mincut increases.

For both these versions, we also address the problem of reporting the values of the mincuts upon insertion of any given edge. To derive our results, we use interesting insights into the *nearest* and the *farthest* mincuts for a pair of vertices. In addition, a crucial result, that we establish and use in our data structures, is that there exists a directed acyclic graph of $\mathcal{O}(n)$ size that compactly stores the farthest mincuts from all vertices of V to a designated vertex s in the graph. We believe that this result is of independent interest, especially, because it also complements a previously existing result by Hariharan et al. (STOC 2007) that the nearest mincuts from all vertices of V to s is a laminar family, and hence, can be stored compactly in a tree of $\mathcal{O}(n)$ size.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Mathematics of computing → Network flows; Mathematics of computing → Graph algorithms

Keywords and phrases Mincut, Sensitivity, Data Structure

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.12

Related Version A full version including all omitted proofs can be found online here: `http://www.cse.iitk.ac.in/users/sbaswana/Papers-published/esa-2020-fv.pdf`.

Funding *Surender Baswana*: The research work was carried out while the author was at Heinz Nixdorf Institute, on leave from IIT Kanpur. The research was supported by Alexander von Humboldt Foundation.

[1] Data structure sizes are in *words* unless specified otherwise, where a word occupies $\Theta(\log n)$ bits.

Till Knollmann: This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (GZ: SFB 901/3) under the project number 160364472.

Acknowledgements We would like to convey special thanks to Jannik Castenow from the Heinz Nixdorf Institute and the Paderborn University for many valuable discussions. Additionally, we would like to thank Rajesh Chitnis and Robert Krauthgamer for promptly answering a few of our queries related to their paper [4].

1 Introduction

Let $G = (V, E)$ be a graph on $n = |V|$ vertices and $m = |E|$ edges with a non-negative capacity on each edge. A *mincut* for a pair of vertices u and v is a set of edges with least capacity whose removal disconnects v from u. It is a fundamental concept in graph theory. Moreover, the area of designing algorithms for the mincut and its variants has been extensively researched ever since the seminal result on the maxflow-mincut duality by Ford and Fulkerson [5].

It is often the case that one is more interested in the mincuts between vertices belonging to a relatively small part of the input graph than the mincuts between all vertices. Hence, consider a subset of vertices $S \subseteq V$ and any subset of pairs of vertices $Q \subseteq S \times S$ whose mincut we are interested in. The objective is to have the knowledge about how sensitive the mincuts of pairs of vertices from Q are, with respect to any change in the subgraph induced by S. This change could be a change in the capacity of an existing edge in the subgraph induced by S or insertion of a new edge between any two vertices in S. This knowledge of the impact on various mincuts due to any change in the network can make the network administrators well prepared for such changes when they indeed occur in future.

An important measure of the impact of a change in the capacity of an edge is the number of pairs of vertices whose mincut value changes. The change in the capacity of an edge could be either an increase or a decrease. We focus on the case when the capacity of an edge is allowed to increase only. For this data structure problem, the query input is a new edge with positive capacity or an existing edge whose capacity is increased. It can easily be observed that as far as the mincut between any pair of vertices is concerned, increasing the capacity of an existing edge (x, y) by amount Δ is equivalent to adding one more edge between x and y with capacity Δ. So henceforth, we only consider the insertion of an edge. On receiving any such edge, the objective is to efficiently report the pairs of vertices whose mincut increases.

Based on the discussion above, we now formally define the problem of mincut sensitivity.

▶ **Problem 1.** *Preprocess $G = (V, E)$, a set $S \subseteq V$, and a set $Q \subseteq S \times S$ to build a compact data structure that, on receiving any edge $(x, y) \in S \times S$ of positive capacity as query input, can efficiently report all those pairs from Q whose mincut value increases upon insertion of the edge (x, y) to G.*

We expect the bounds for the data structure of Problem 1 to depend on only the size of S instead of V. For simultaneously achieving efficient query time and compact space, the only previous solution that exists for this problem is for a single pair of vertices only, i.e., when $|Q| = 1$. It consists of a data structure that occupies $\mathcal{O}(|S|)$ space and achieves $\mathcal{O}(1)$ query time. This solution follows from an observation made by Picard and Queyranne in their seminal paper [11].

1.1 Our Contribution

We address the single source and the all-pairs versions of Problem 1, along with the problem of reporting the new value of the mincut between any pair of vertices after the edge insertion.

Single source all destinations: In the single source case, we are interested in the mincuts between a designated vertex $s \in S$ and all other vertices from set S, i.e., the (s,t)-mincut, for all $t \in S \setminus \{s\}$.

▶ **Theorem 2.** *For an undirected graph and any subset S of vertices with a designated vertex $s \in S$, there exists an $\mathcal{O}(|S|)$ size data structure that can report all those vertices from S whose mincut value to s increases upon insertion of any given edge from $S \times S$. The time taken by this data structure to answer any such query is $\mathcal{O}(|S|)$.*

The $\mathcal{O}(|S|)$ space and $\mathcal{O}(|S|)$ query time of our data structure for undirected graphs can be much less than $\mathcal{O}(|V|)$ for smaller S. Also, it is in sharp contrast with the following lower bound result for directed graphs that we also prove.

▶ **Theorem 3.** *Any data structure for a directed graph to answer a mincut sensitivity query from a designated source vertex to any designated subset of q vertices must use $\Omega(q^2)$ bits of space for at least one directed graph.*

The proof of Theorem 3 basically establishes that any such data structure can be used to store any balanced bipartite graph on $2|S|$ vertices implicitly. Interestingly, the same proof also establishes an $\Omega(|S|^2)$ lower bound for the single source version of two other fundamental problems for directed graphs, namely, *reachability sensitivity* as well as *distance sensitivity* for the insertion of an edge. These facts add more significance to the result stated in Theorem 2 for the single source mincut sensitivity.

All-pairs: When considering the mincuts between all pairs of vertices in S, i.e., the (u,v)-mincut, for all $u, v \in S$ such that $u \neq v$, we obtain the following result.

▶ **Theorem 4.** *For an undirected graph and any subset S of vertices, there exists an $\mathcal{O}(|S|^2)$ size data structure that can report all those pairs of vertices from $S \times S$ whose mincut value increases upon insertion of any given edge from $S \times S$. The time taken by the data structure to answer any such query is $\mathcal{O}(k)$, where k is the number of pairs whose mincut increases.*

Note that the query time of the data structure in Theorem 4 is optimal. Moreover, if the objective is to report just the number of all-pairs from the set $S \times S$ whose mincut increases upon insertion of any given edge, our data structure can accomplish this objective in $\mathcal{O}(\min(k, |S| \log |S|))$ time, which is $\mathcal{O}(|S| \log |S|)$ always.

▶ Remark 5. Our results for mincut sensitivity directly extend to maxflow sensitivity as well due to the equivalence between maxflow and mincut [5].

To achieve all our results, we use interesting insights into the *nearest* and the *farthest* mincuts for a pair of vertices – two concepts that exist since the seminal work of Ford and Fulkerson on maximum flow [5]. Additionally, a crucial result about the farthest mincuts that we establish and use in one of our data structures is the following.

▶ **Theorem 6.** *For an undirected graph on n vertices and a designated source vertex s, there exists a directed acyclic graph (DAG) of size $\mathcal{O}(n)$ that compactly stores the farthest mincuts from all vertices $v \in V \setminus \{s\}$ to s. For any v, the set of vertices defining the farthest mincut from v to s can be reported in time that is of the order of the size of the set.*

The graph theoretic result of Theorem 6 is of independent interest in addition to its applications in the mincut sensitivity problem. This is because, not only it adds to our understanding of mincuts, but it also complements an earlier result of Hariharan et al. [9] that showed that the nearest mincuts from all vertices to s form a laminar family, and hence, can be stored in a tree data structure occupying only $\mathcal{O}(n)$ space.

1.1.1 On reporting the value of mincut

In addition to reporting the pairs of vertices whose mincut increases upon insertion of a given edge, it may be important to output the new values of their mincuts. Indeed, if the edge capacities in the graph are integral and the inserted edge has unit capacity, our data structures from Theorems 2 and 4 can also report the new values of the affected mincuts upon insertion of an edge, i.e., the value of the mincut will be increased by one for the reported pairs of vertices. However, if there is no restriction on the capacity of the inserted edge, we show that even for the single source case it is not possible to accomplish this objective with any data structure of subquadratic size.

▶ **Theorem 7.** *There exists a set $\mathcal{G}$ of undirected graphs on n vertices with integer edge capacities in the range $[1, n^{2+\varepsilon}]$ (for any $\varepsilon > 0$) for which the following claim holds true. Any data structure for an undirected graph that can report the value of the mincut between a designated source vertex and any other vertex upon insertion of any edge of integer capacity polynomial in n must require $\Omega(n^2 \varepsilon \log n)$ bits of space for at least one graph from $\mathcal{G}$.*

For the all-pairs case, it turns out that any such data structure also provides a generalization of the flow-tree [6, 8]. That is, the data structure will also be able to report the mincut value between a vertex u and a pair $\{x, y\}$ of vertices for any $u, x, y \in V$. Chitnis, Kamma, and Krauthgamer [4] showed that there will be total $\mathcal{O}(n^2)$ distinct mincut values separating any vertex from any pair of vertices in an undirected graph. However, to the best of our knowledge, designing an $\mathcal{O}(n^2)$ size data structure that returns the value of any such mincut in non-trivial query time is still an open problem.

1.2 Overview of our results

We begin with the result of Picard and Queyranne [11] for mincut sensitivity for a source-destination pair (s, t). For any maximum flow f from s to t, let G_f be the corresponding residual graph. Notice that there is no path from s to t in G_f. Let R be the set of vertices which are reachable from s in G_f, and let T be the set of vertices from which t is reachable in G_f. Picard and Queyranne [11] made the following crucial observation.

▶ **Lemma 8** (Picard and Queyranne [11])**.** *The maxflow from s to t increases upon insertion of an edge (x, y) if and only if x belongs to R and y belongs to T.*

Without loss of generality, we can assume that the vertices of set S are labeled from 1 to $|S|$. Based on Lemma 8, the data structure for mincut sensitivity for a (s, t)-pair where $s, t \in S$, stores each of $R \cap S$ and $T \cap S$ in Boolean arrays indexed by the vertices of set S.

The subsets R and $V \backslash T$ in Lemma 8 turn out to be the smallest, and the largest subsets of vertices defining a mincut from s to t, respectively. In the literature on mincuts, R and $V \backslash T$ are respectively called the *nearest* and the *farthest* mincut from s to t; we define these notions more formally in the next section. Therefore, in order to design a compact and efficient data structure for the single source and the all-pairs versions of the mincut sensitivity problem, it is natural to explore if we can have a compact way to store these two types of cuts

for multiple pairs of vertices. We now provide an overview of the compact data structures for the nearest and the farthest mincuts, and the way these data structures are used to solve the mincut sensitivity problem.

Compact data structures for the nearest and the farthest mincuts

Interestingly, Hariharan et al. [9] showed that the nearest mincuts from all vertices to a designated vertex s in an undirected graph form a laminar family – If the subsets of vertices defining the nearest mincuts from u to s, and v to s intersect, then one of them must be a subset of the other. As a result, the nearest mincuts from all vertices to s can be stored compactly in a tree data structure occupying $\mathcal{O}(n)$ space only. However, the farthest mincuts do not constitute a laminar family. Let F_u and F_v be the subsets of vertices that define the farthest mincuts to s from u and v respectively. It is quite possible that F_u and F_v intersect each other but none of them is a subset of the other. In other words, two intersecting farthest mincuts to s may *cross* each other. Moreover, there may be $\Theta(n)$ vertices whose farthest mincuts to s cross the farthest mincut of a single vertex to s. This poses a challenge for designing a compact data structure for storing all the farthest mincuts to s. However, we overcome this challenge using crucial insights into the farthest mincuts.

Using the submodularity of cuts, we first establish the existence of a DAG on $\mathcal{O}(n)$ nodes that stores the farthest mincuts from all vertices to any designated vertex s. However, this DAG could have $\mathcal{O}(n^2)$ edges, and establishing the sparsity of the DAG turns out to be the main hurdle. We overcome this by proving the following interesting property of the farthest mincuts to s:

For any three vertices, either the intersection of their farthest mincuts to s is empty or the farthest mincut from at least one of them is a subset of one of the other two.

Using this property, we are able to prune away all the unnecessary edges from the DAG structure storing farthest mincuts to s. As a result, each node in the DAG turns out to have at most 2 incoming edges, so the size of the DAG is $\mathcal{O}(n)$.

For the objective of solving mincut sensitivity for a subset $S \subseteq V$, we present data structures for the nearest mincuts (likewise the farthest mincuts) that consist of vertices of S only instead of V. Their size is $\mathcal{O}(|S|)$. See Theorem 18 and Theorem 6.

Solving the mincut sensitivity problem

For solving the single source mincut sensitivity problem, we use the tree data structure storing the nearest mincuts and the DAG data structure storing the farthest mincuts, from all vertices of the set S to s. The size of the data structure is $\mathcal{O}(|S|)$. Following Lemma 8, it takes $\mathcal{O}(|S|)$ time using this data structure to determine whether the insertion of a given edge increases the (s, v)-mincut value for any vertex $v \in S$. This leads to $\mathcal{O}(|S|^2)$ time to find all vertices from S whose mincut value from s increases due to the insertion of a given edge. In order to reduce the query time to $\mathcal{O}(|S|)$, we make use of the following insight:

A vertex $x \in S$ belongs to the farthest mincut from v to s if and only if x is reachable from v in the DAG structure storing the farthest mincuts to s.

Solving the all-pairs version of the mincut sensitivity problem with optimal query time turns out to be more challenging. As the underlying graph is undirected, the subset of vertices that defines the farthest mincut from u to v is the complement of the subset of the vertices that defines the nearest mincut from v to u. As a result, keeping the nearest-mincut tree data structure for each vertex of S suffices to solve this problem. The data structure takes $\mathcal{O}(1)$ time to determine for any pair (u, v), whether insertion of an edge, say (x, y), increases

(u, v)-mincut value. This observation implies an $\mathcal{O}(|S|^2)$ time algorithm for computing all pairs of vertices from set S whose mincut value increases upon insertion of edge (x, y). But it is quite wasteful if k, the number of pairs whose mincut value increases, is much smaller than $|S|^2$. To accomplish $\mathcal{O}(k)$ query time, we make use of multiple insights into the structure of the nearest mincuts. The most crucial insight is the following:

The vertices whose mincut value to s increases upon insertion of an edge (x, y) lie contiguously on the paths from x and y to their lowest common ancestor in the tree that stores the nearest mincuts to s.

This insight leads to an $\mathcal{O}(|S| + k)$ query time. To get rid of the additive factor of $|S|$, we use another insight that helps finding the *right* pool of vertices whose nearest-mincut tree we need to query.

1.3 Related work

Our research is related to the field of dynamic graph algorithms that emphasizes on efficient data structures to handle changes in a network. For dynamic graph algorithms, the objective is to maintain the solution of a problem for an online sequence of edge insertions or deletions with worst case time complexity better than that of the best static algorithm. There do exist efficient dynamic algorithms for maintaining a global mincut – an incremental algorithm by Goranci, Henzinger, and Thorup [7], and a fully dynamic algorithm by Thorup [12]. However, there does not exist any dynamic algorithm for all-pairs mincuts whose worst case time complexity is better than the best static algorithm. Hartmann and Wagner [10] presented a fully dynamic algorithm for maintaining an all-pairs mincut tree for undirected graphs. Although it achieves a significant speedup over the best static algorithm on many real world graphs, its worst case asymptotic time complexity is not better than the best static algorithm for an all-pairs mincut tree.

1.4 Organization of the paper

Equipped with notations, definitions, and well known lemmas introduced in Section 2, we present the compact data structures for nearest and farthest mincuts in Sections 3 and Section 4, respectively. The data structures for the single source and the all-pairs versions of the mincut sensitivity problem are presented in Section 5 and 6 respectively. Due to the space constraint, the proofs of some theorems and lemmas had to be omitted from this version. So we recommend the reader to refer to the full version of this paper [2].

2 Preliminaries

Our results consider an undirected graph $G = (V, E)$ on n vertices where each edge is assigned a non-negative capacity through a function $c : E \to R^+$.

▶ **Definition 9** ((s, t)-cut). *A subset of edges whose removal disconnects t from s is called an (s, t)-cut. An (s, t)-mincut is an (s, t)-cut of smallest capacity.*

▶ **Definition 10** (set defining a cut). *A subset $A \subset V$ is said to define an (s, t)-cut if $s \in A$ and $t \notin A$. The corresponding cut is denoted by cut$(A, \bar{A})$ or more compactly cut(A).*

When there is no scope of confusion, we do not distinguish between a mincut and the set defining the mincut. We can extend the capacity function c on edges to any subset $A \subset V$ in a natural way as follows: $c(A)$ denotes the sum of the capacities of all those edges which have exactly one endpoint in A. With this generalization, we now state a well-known property of cuts, namely, the submodularity of cuts.

▶ **Lemma 11** (Submodularity of cuts). *Given an undirected graph $G = (V, E)$ with positive edge capacities, the following inequality holds true for any two subsets $A, B \subset V$.*

$$c(A) + c(B) \geq c(A \cup B) + c(A \cap B)$$

The following lemma states an important property of an (s,t)-mincut.

▶ **Lemma 12.** *Let $A \subset V$ define an (s,t)-mincut with $s \in A$. For any subset $A' \subset A$ with $s \notin A'$, if α is the number of edges incident on A' from $V \backslash A$, and β is the number of edges incident on A' from $A \backslash A'$, then $\alpha \leq \beta$.*

2.1 The nearest and the farthest mincuts

▶ **Definition 13** (Nearest and farthest mincuts from s to t). *The subset $A \subset V$ with $s \in A$ is said to define the nearest (likewise the farthest) mincut from s to t if (1) cut$(A, \bar{A})$ defines an (s,t)-mincut, and (2) For every other subset $A' \subset V$ that defines an (s,t)-mincut, $A \subset A'$ (likewise $A' \subset A$). We use s_t^N and s_t^F to denote the nearest and the farthest mincut from s to t, respectively.*

One can easily show using Lemma 11 that the nearest and the farthest mincut from s to t are unique. Additionally, t_s^N and s_t^F partition the set of vertices V as stated in the following lemma.

▶ **Lemma 14.** *For any pair of vertices $s, t \in V$, (i) $s_t^N \cap t_s^N = \emptyset$, and (ii) $s_t^F = V \backslash t_s^N$.*

In the light of Lemma 14, we can restate Lemma 8 for undirected graphs as follows.

▶ **Lemma 15** (Picard and Queyranne [11]). *The insertion of an edge (x, y) can increase the mincut between s and t if and only if $x \in s_t^N$ and $y \in t_s^N$ or vice versa.*

▶ Remark 16. In order to explore the relationship among the farthest mincuts from a set of vertices to a vertex s, we focus only on the connected component of s. This is because for each vertex outside this component, its farthest mincut to s is obvious. Therefore, without loss of generality we assume G to be connected in the rest of the paper.

3 A compact data structure for all nearest mincuts to vertex s

The following theorem plays the key role in compactly storing all nearest mincuts to s.

▶ **Theorem 17** (Hariharan et al. [9]). *For any two distinct vertices $u, v \in S$, either u_s^N and v_s^N are mutually disjoint or one of them is a subset of the other.*

For a given subset $S \subseteq V$, let $\mathcal{N} = \{x_s^N \cap S | x \in S \backslash \{s\}\}$. Using Theorem 17 we can arrange the sets of $\mathcal{N}$ in a forest of disjoint trees as follows. We refer to a vertex in this forest as *node*. For each set present in $\mathcal{N}$, we create a unique node in the forest. We assign each vertex $v \in S$ to the node ν corresponding to $v_s^N \cap S$. The parent of a node ν is defined as the unique node μ such that the set corresponding to μ is the smallest superset of the set corresponding to ν, if such a superset exists. If no such superset exists, ν will be the root of a tree. We create a dummy node and assign it as the parent of the root of every tree in this forest. Let us denote the resulting rooted tree by $\mathcal{T}(s)$. Figure 1 shows an example graph and the corresponding $\mathcal{T}(s)$ for the case $S = V$.

It can be observed that if a vertex $v \in S$ is assigned to node ν, then the subtree rooted at ν stores the set $v_s^N \cap S$. So it follows that a vertex, say x, belongs to $v_s^N \cap S$ if either x and v are assigned to the same node in $\mathcal{T}(s)$ or the node containing v is an ancestor of the

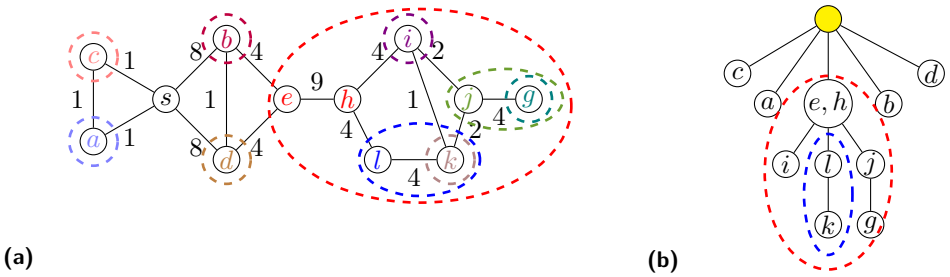


Figure 1 **(a)** The nearest mincut from a vertex to s is encircled with same color. **(b)** Tree $\mathcal{T}(s)$.

node containing x. This check can be easily done in $\mathcal{O}(1)$ time if we augment $\mathcal{T}(s)$ to answer lowest common ancestor (LCA) query for any pair of nodes (see [3]). We can thus state the following theorem.

▶ **Theorem 18.** *For an undirected graph $G = (V, E)$, a subset $S \subseteq V$, and any vertex $s \in S$, there exists an $\mathcal{O}(|S|)$ size data structure $\mathcal{T}(s)$ that can report in $\mathcal{O}(1)$ time whether $x \in v_s^N$ for any $x, v \in S$.*

4 A compact data structure for all farthest mincuts to vertex s

In this section, we present a novel data structure that compactly stores the farthest mincuts to vertex s from a subset of vertices. Our main result can be summarized as follows.

▶ **Theorem 19.** *For an undirected graph $G = (V, E)$, any subset $S \subseteq V$, and a designated vertex $s \in S$, there exists a directed acyclic graph $\mathcal{D}(s)$ having $\mathcal{O}(|S|)$ nodes and $\mathcal{O}(|S|)$ edges that can report $v_s^F \cap S$ in time of the order of the size of $v_s^F \cap S$ for any $v \in S \backslash \{s\}$.*

Lemma 14(ii) implies that $s_v^N = V \backslash v_s^F$. So we can compute $s_v^N \cap S$ in $\mathcal{O}(|S|)$ time once we have $v_s^F \cap S$. Therefore, we can state the following corollary of Theorem 19.

▶ **Corollary 20.** *For an undirected graph $G = (V, E)$, any subset $S \subseteq V$, and a designated vertex $s \in S$, there exists a data structure of $\mathcal{O}(|S|)$ size that takes just $\mathcal{O}(|S|)$ time to compute $s_v^N \cap S$ for any $v \in S \backslash \{s\}$.*

Next, we show how to compute a DAG storing all farthest mincuts to s in space $\mathcal{O}(|S|^2)$. Thereafter, we reduce its space complexity to $\mathcal{O}(|S|)$ only.

4.1 A DAG of size $\mathcal{O}(|S|^2)$

Our data structure to store all farthest mincuts to a designated vertex s uses the observation captured in Lemma 21. In its core, it tells us that certain farthest mincuts are related by a subset relation which can be exploited to store them compactly.

▶ **Lemma 21.** *Let x and v be any two vertices in G. If $x \in v_s^F$, then $x_s^F \subseteq v_s^F$.*

Proof. We use Lemma 11 on the submodularity of cuts and provide a proof by contradiction. Let A and B refer to the sets v_s^F and x_s^F, respectively. It is given that $x \in v_s^F$. Assume that $x_s^F \nsubseteq v_s^F$. This would imply that $B \backslash A \neq \emptyset$, hence A must be a proper subset of $A \cup B$.

Observe that $A \cap B$ defines a valid (x, s)-cut since x is present in both A and B, whereas $s \notin x_s^F$. This observation implies that $c(A \cap B) \geq c(B)$ since B defines an (x, s)-mincut. This inequality and Lemma 11 imply the following inequality.

$$c(A \cup B) \leq c(A) \tag{1}$$

Now observe that $A \cup B$ defines a valid (v,s)-cut since v belongs to A, whereas s belongs neither to A nor to B. Since A defines a (v,s)-mincut, so Inequality 1 implies that $c(A \cup B)$ must be equal to the capacity of an (v,s)-mincut. But A is a proper subset of $A \cup B$. This would imply that the cut defined by A is not the farthest mincut from v to s – a contradiction. ◀

Let $\mathcal{F} = \{v_s^F \cap S \mid v \in S \backslash \{s\}\}$. We now use Lemma 21 to build a directed acyclic graph $D = (\mathcal{V}, \mathcal{E})$ that stores $\mathcal{F}$ as follows. We use *node* to refer to a vertex of this DAG.

For each set present in $\mathcal{F}$, we create a unique node in D. The set of nodes thus created constitutes $\mathcal{V}$. We denote by $\mathcal{F}(\nu)$ the set in $\mathcal{F}$ corresponding to node ν. The edge set $\mathcal{E}$ of D is defined as follows.

$$\mathcal{E} = \{(\nu, \mu) \mid \mathcal{F}(\mu) \subset \mathcal{F}(\nu)\}$$

It can be observed that if $X \subset Y$ for any two sets X and Y in $\mathcal{F}$, then $|X| < |Y|$. Hence D is acyclic. To efficiently retrieve $x_s^F \cap S$ for any given $x \in S$, we can augment D as follows.

- We create an array J_s indexed by vertices of set S such that for any $v \in S \setminus \{s\}$, $J_s[v]$ stores the pointer to node μ that corresponds to $v_s^F \cap S$, that is, $\mathcal{F}(\mu) = v_s^F \cap S$.
- Each node μ of D stores a list $L(\mu)$ of all those vertices $v \in S$ such that $J_s[v] = \mu$.
- We introduce a dummy node and add an edge from it to every other node which has no incoming edge.

Lemma 22 follows immediately from Lemma 21 and the construction of D described above.

▶ **Lemma 22.** *Let x and u be any two vertices of set S. x is present in $u_s^F \cap S$ if and only if either $J_s[u] = J_s[x]$ or there is an edge from $J_s[u]$ to $J_s[x]$ in D.*

Lemma 22 implies that for each vertex $v \in S \backslash \{s\}$, $v_s^F \cap S$ is the set of vertices stored in the list $L(J_s[v])$ and the lists of all the nodes with an incoming edge from $J_s[v]$ in D.

The subset relation $\subset$ is transitive. So, if there is a path from a node ν to another node μ in D, then (ν, μ) is also an edge in D. In other words, the transitive closure of D is D itself. This observation in conjunction with Lemma 22 leads us to the following lemma which will be crucial for our data structure for the single source mincut sensitivity problem.

▶ **Lemma 23.** *Let x and u be any two vertices in set S. x is present in $u_s^F \cap S$ if and only if $J_s[x]$ is reachable from $J_s[u]$ in D.*

Notice that D has $\mathcal{O}(|S|)$ nodes, but it could have $\Theta(|S|^2)$ edges. So the total space occupied by D could be $\Theta(|S|^2)$. A natural idea to overcome this hurdle is to remove as many edges as possible from D without affecting the reachability between any pair of its vertices so that Lemma 23 continues to hold. In other words, we compute another DAG D^τ which is the transitive reduction of D. Aho, Garey, and Ullman [1] showed that computing the transitive reduction of a DAG is as easy as computing its transitive closure. While in general a transitive reduction does not always lead to a reduced number of edges, it does so in the case of D. In the following subsection we present crucial insights into crossing farthest mincuts that ensure that each node of D^τ will have at most two incoming edges. The data structure $\mathcal{D}(s)$ in Theorem 19 for storing all farthest mincuts to s is D^τ only.

4.2 Bounding the in-degree of D^τ by 2

A set of vertices $\mathcal{I} \subset V$ is said to be a set of *incomparable* vertices with respect to the mincuts to s if for each $u, v \in \mathcal{I}$ with $u \neq v$ it holds that $u \notin v_s^F$ and $v \notin u_s^F$. The following lemma highlights an important property for a pair of incomparable vertices.

▶ **Lemma 24.** *For any two incomparable vertices u and v, there does not exist any edge between the set $u_s^F \cap v_s^F$ and the set $V \backslash (u_s^F \cup v_s^F)$.*

We shall now use Lemma 12 and Lemma 24 to derive the following lemma which will play a crucial role in establishing that D^τ has indegree 2 only.

▶ **Lemma 25.** *For any three incomparable vertices $u, v, w \in V$, $u_s^F \cap v_s^F \cap w_s^F = \emptyset$.*

Proof. We give a proof by contradiction. Let B, L, and R denote the sets u_s^F, v_s^F, and w_s^F, respectively. Figure 2(i) illustrates these sets. For a clear distinction, we have used different colors for these sets in this figure, and correspondingly assigned the labels B (for blue), L (for light green), and R (for red) to these sets.

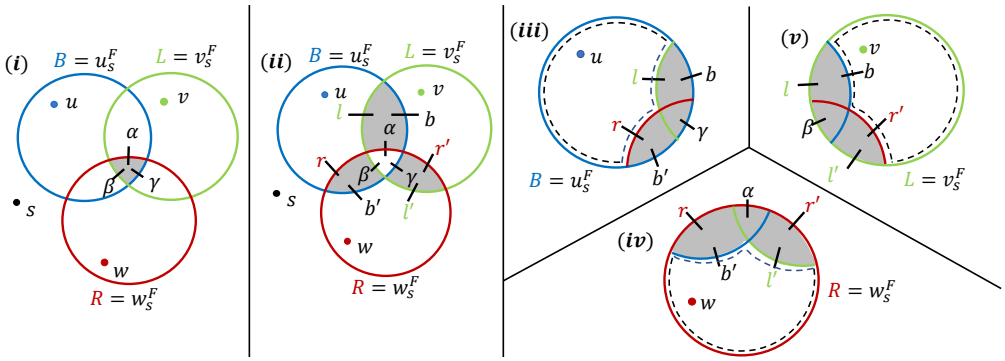


Figure 2 Intersection of the farthest mincuts to s.

Suppose the common intersection $B \cap L \cap R$ of these sets (shown shaded in Figure 2(i)) is not an empty set. By applying Lemma 24 for $B \cap L$, $L \cap R$, $R \cap B$, we can infer that each vertex in the common intersection will have edges incident only from the sets $B \cap L$, $L \cap R$, $R \cap B$. Considering the set $B \cap L \cap R$ as a single entity, let α, β, γ be the number of edges incident on it from $(B \cap L) \backslash R$, $(R \cap B) \backslash L$, $(L \cap R) \backslash B$, respectively. As the graph is connected (see Remark 16), we have:

$$\alpha + \beta + \gamma > 0. \tag{2}$$

Let us consider the set $(B \cap L) \backslash R$, that is, the set $B \cap L$ after removing the common intersection $B \cap L \cap R$. It follows from Lemma 24 that the edges incident on this set will be from $B \backslash L$ and $L \backslash B$ only, apart from the edges incident from $B \cap L \cap R$. Similar claims hold for the sets $(B \cap R) \backslash L$ and $(R \cap L) \backslash B$ as well. Figure 2(ii) shows these sets as shaded regions along with the edges incident on them. For example, l, b, α are the number of edges incident on $(B \cap L) \backslash R$ from $B \backslash L$, $L \backslash B$, and $B \cap R \cap L$, respectively.

The rest of the proof is as follows. Exploiting the fact that u, v, w are incomparable, we suitably apply Lemma 12 to arrive at inequalities that eventually leads to contradict Inequality 2.

B defines an (s, u)-mincut. Since u is incomparable with both v and w, it is not present in the set $B \cap (R \cup L)$ shown shaded in Figure 2(iii). Notice that the number of edges incident on this set from $V \backslash B$ is $b + b' + \gamma$ whereas the number of edges incident on this set from the rest of B, that is, the set $B \backslash (R \cup L)$ (enclosed by dotted boundary in Figure 2(iii)) is at most $l + r$. So we get the following inequality by substituting B and $B \cap (R \cup L)$ in place of A and A' respectively in Lemma 12:

$$b + b' + \gamma \leq l + r \tag{3}$$

In Figure 2, R defines an (s,w)-mincut and L defines an (s,v)-mincut. Hence, with similar arguments as above, analyzing the (s,w)-mincut in Figure 2(iv), and analyzing the (s,v)-mincut in Figure 2(v), we get the following inequalities, respectively:

$$r + r' + \alpha \leq b' + l', \qquad l + l' + \beta \leq b + r'$$

Adding the above inequalities with Inequality 3 and canceling identical terms on either sides we get $\alpha + \beta + \gamma \leq 0$. This contradicts Inequality 2 and completes the proof. ◀

The following is a simple corollary of Lemma 25.

▶ **Corollary 26.** *Let $A \in \mathcal{F}$. If U, V, W are any three distinct sets from $\mathcal{F}$ such that $A \subset U$, $A \subset V$, and $A \subset W$. Then at least one of the sets from $\{U, V, W\}$ must be a proper subset of one of the remaining two.*

We can use Corollary 26 to establish the following lemma.

▶ **Lemma 27.** *The indegree of any node in D^τ will be at most 2.*

Figure 3 shows farthest mincuts from a sample of vertices to s in our example graph. Notice that the farthest mincut b_s^F *crosses* the farthest mincut d_s^F. Also, the farthest mincuts from j and g to s are identical, so j and g are mapped to a single node in D^τ.

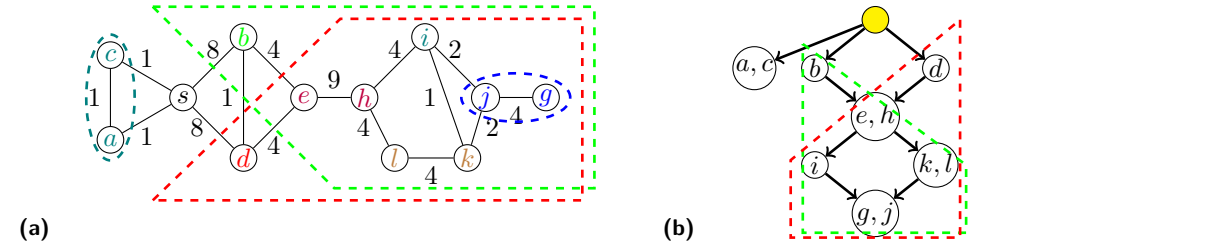


Figure 3 **(a)** A dotted boundary defines a farthest mincut to s from a vertex of the same color. **(b)** The DAG D^τ.

5 Single source mincut sensitivity for insertion of an edge

We now present an $\mathcal{O}(|S|)$ space data structure that can report all those vertices from S whose mincut value to s increases upon insertion of any given edge $(x,y) \in S \times S$. The data structure will consist of the tree structure $\mathcal{T}(s)$ from Theorem 18 and DAG structure $\mathcal{D}(s)$ from Theorem 19.

Let $A_x = \{v \in S | x \in s_v^N\}$ and $A_y = \{v \in S | y \in s_v^N\}$. It follows from Lemma 15 that if (s,v)-mincut increases, then v must belong to A_x or A_y. Furthermore, for any $v \in A_x$, (s,v)-mincut increases if $y \in v_s^N$. Using Theorem 18, it takes just $\mathcal{O}(1)$ time to do this check for any given $v \in A_x$ (likewise A_y). So, in order to report all vertices from S whose mincut from s increases upon insertion of edge (x,y) in $\mathcal{O}(|S|)$ time, all we need is an $\mathcal{O}(|S|)$ time algorithm to compute A_x and A_y. We now provide $\mathcal{O}(|S|)$ time algorithm to compute A_x; we can compute A_y in $\mathcal{O}(|S|)$ time in a similar manner.

It follows from Lemma 14(ii) that computing A_x is equivalent to computing the set $\bar{A_x} = \{v \in S | x \in v_s^F\}$. Recall that $J_s[x]$ is the node containing x in $\mathcal{D}(s)$. It follows from Lemma 23 that $x \in v_s^F$ if and only if $J_s[x]$ is reachable from $J_s[v]$ in $\mathcal{D}(s)$. Therefore, we can compute $\bar{A_x}$ by first reversing the edges of $\mathcal{D}(s)$ and then traversing all the nodes reachable

from $J_s[x]$. For each node λ reachable from $J_s[x]$ in the reversed $\mathcal{D}(s)$, x is present in v_s^F for each vertex $v \in L(\lambda)$. Since $\mathcal{D}(s)$ has $\mathcal{O}(|S|)$ edges, it takes $\mathcal{O}(|S|)$ time to reverse it and traverse it to compute $\bar{A}_x$. This establishes the proof of Theorem 2 for the single source mincut sensitivity problem.

6 All-pairs mincut sensitivity data structure for insertion of an edge

Our data structure consists of the nearest-mincut tree $\mathcal{T}(z)$ from Theorem 18 for each $z \in S$. Each of these trees occupies $\mathcal{O}(|S|)$ space, so the space occupied by the data structure is $\mathcal{O}(|S|^2)$. For the rest of this section, x, y, z are any arbitrary vertices from S. Upon insertion of edge (x, y), let k be the number of pairs of vertices from $S \times S$ whose mincut value increases. We present an $\mathcal{O}(k)$ time algorithm to output all these pairs using four insights into the nearest-mincut trees. Our first insight is stated in Lemma 28. It implies that for all vertices belonging to a node μ in $\mathcal{T}(z)$, it suffices to determine for any single vertex, say $u \in \mu$, whether the (z, u)-mincut value increases upon insertion of any given edge.

▶ **Lemma 28.** *Let $u, v \in S$ be any two vertices belonging to the same node in $\mathcal{T}(z)$. Upon insertion of any given edge, (u, z)-mincut value increases iff (v, z)-mincut value increases.*

Let μ and ν be the nodes in $\mathcal{T}(z)$ containing x and y respectively. Our second insight, stated in the following lemma, specifies the location of vertices in $\mathcal{T}(z)$ whose mincut value to s increases upon insertion of edge (x, y).

▶ **Lemma 29.** *Let $\omega = LCA(\mu, \nu)$ in $\mathcal{T}(z)$. Upon insertion of edge (x, y), the mincut value from z to only those vertices may increase that belong to the nodes of (1) the path from μ to ω but excluding ω, and (2) the path from ν to ω but excluding ω.*

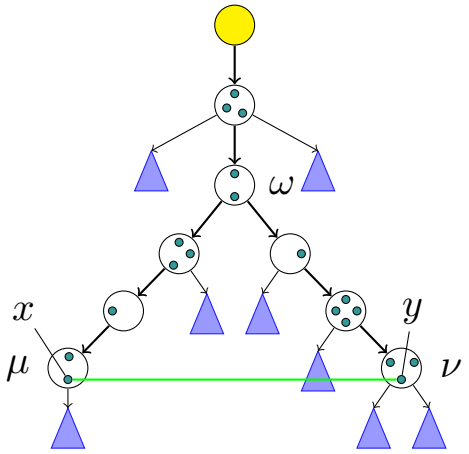


Figure 4 The tree $\mathcal{T}(z)$ from the perspective of μ and ν. If v is a vertex whose node in $\mathcal{T}(z)$ does not belong to either of ω-ν and ω-μ paths, then v must be present in one of the subtrees hanging from these paths (shown in blue). Now consider any node, say γ, lying on the path from ω to the root of $\mathcal{T}(z)$. Both x and y belong to the subtree rooted at γ.

Proof. Let us view $\mathcal{T}(z)$ from the perspective of the paths from μ and ν to the root of $\mathcal{T}(z)$. The reader is advised to refer to Figure 4 for a better understanding. If v is a vertex whose node in $\mathcal{T}(z)$ does not belong to these paths, then v must be present in one of the subtrees (shown in blue in Figure 4) hanging from these paths. Notice that neither x nor y belongs to

the subtree containing v. So it follows from Lemma 15 that the mincut from z to v is not affected by the insertion of edge (x, y). Now consider any node, say γ, lying on the path from ω to the root of $\mathcal{T}(z)$. Both x and y belong to the subtree rooted at γ. So using Lemma 15 again, the mincut from z to any vertex of γ remains unaffected by the insertion of edge (x, y). ◀

Our third and most crucial insight is that the vertices whose mincut value to z increases upon insertion of edge (x, y) belong to a *contiguous* sequence of nodes on the paths from the node containing y and the node containing x to their LCA in $\mathcal{T}(z)$. The following lemma states this insight for the node containing y.

▶ **Lemma 30.** *Let ω be the LCA of the nodes containing x and y in $\mathcal{T}(z)$. Let u and v be any two vertices lying on the path from the node containing y to ω in $\mathcal{T}(z)$ such that the node containing u is an ancestor of the node containing v. If (z, u)-mincut value increases upon insertion of edge (x, y), then (z, v)-mincut value also increases upon insertion of (x, y).*

Proof. It follows from the construction of $\mathcal{T}(z)$ that $y \in v_z^N$ and $v_z^N \subseteq u_z^N$. It is given that the insertion of edge (x, y) increases (z, u)-mincut value and u is an ancestor of y in $\mathcal{T}(z)$. So Lemma 15 implies:

$$x \in z_u^N \tag{4}$$

It follows from Lemma 14(i) that $z_u^N \cap u_z^N = \emptyset$. So $v \notin z_u^N$ since $v \in u_z^N$. Applying Lemma 14(ii), we get $v \in u_z^F$. So it follows from Lemma 21 that $v_z^F \subseteq u_z^F$. Applying Lemma 14(ii) again, it follows that $z_u^N \subseteq z_v^N$. Using this fact and Equation 4, we can infer that $x \in z_v^N$. Since we have already established that $y \in v_z^N$, so using Lemma 15 we can conclude that (z, v)-mincut value will also increase upon insertion of edge (x, y). ◀

It is a simple corollary of Lemma 30 that if (y, z)-mincut value does not increase upon insertion of edge (x, y), then for any vertex v present in any ancestor of the node containing y in $\mathcal{T}(z)$, (v, z)-mincut value will also not increase. So, to compute all-pairs of vertices whose mincut increases, we need to explore the nearest-mincut tree of only those vertices z whose mincut value to y (and likewise to x) increases.

We now describe how to process $\mathcal{T}(z)$ for a vertex z given that (y, z)-mincut value increases upon insertion of (x, y). For each such z, first we enumerate all vertices present in the node, say ν, to which y belongs. We then begin an upward traversal of $\mathcal{T}(z)$ starting from the parent of ν. For any node, say λ, that we traverse, we pick any arbitrary vertex from it, say v, and determine whether $x \in z_v^N$ by querying $\mathcal{T}(v)$. It takes $\mathcal{O}(1)$ time to answer this query (see Theorem 18). If $x \in z_v^N$, it follows from Lemma 28 that each vertex present in λ has its mincut value to z increased. So we enumerate all vertices from λ, and continue processing the parent of λ in a similar manner. If $x \notin z_v^N$, we stop the traversal. It follows from Lemma 30 that the vertices enumerated in this way are precisely the vertices whose mincut value to z increases. To efficiently identify each vertex z, such that the (y, z)-mincut value increases upon insertion of edge (x, y), we exploit the fourth insight into the nearest-mincut trees which is stated in the following lemma. This lemma can be seen as a corollary of Lemma 15.

▶ **Lemma 31.** *(y, z)-mincut value increases upon insertion of edge (x, y) iff $x \in z_y^N$.*

It follows from Lemma 31 that the vertices present in the node containing x and its ancestors in $\mathcal{T}(y)$ are precisely the vertices whose mincut value to y increases. We can identify all these vertices in optimal time by traversing $\mathcal{T}(y)$ upward from the node containing x.

We have described above the processing of each z such that the (y, z)-mincut value increases due to the insertion of edge (x, y). A similar processing must be carried out for all vertices z, such that the (x, z)-mincut value increases due to the insertion of edge (x, y).

It follows from the description given above that we can compute all those pairs of vertices from $S \times S$ whose mincut value increases upon the insertion of any given edge in $\mathcal{O}(k)$ time, where k is the number of these pairs. If our goal is to just report the value of k, we can accomplish it in $\mathcal{O}(\min(k, |S| \log |S|))$ time by suitably augmenting the nearest-mincut trees. We can thus conclude with Theorem 32 which extends Theorem 4.

▶ **Theorem 32.** *For an undirected graph $G = (V, E)$, and a subset S of vertices, there exists an $\mathcal{O}(|S|^2)$ size data structure that can report all pairs of vertices whose mincut increases upon insertion of a query edge. The guaranteed query time is $\mathcal{O}(k)$, where k is the number of pairs whose mincut increases. We can also report k in $\mathcal{O}(\min(k, |S| \log |S|))$ time.*

References

1. Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972. doi:10.1137/0201008.
2. Surender Baswana, Shiv Gupta, and Till Knollmann. Mincut Sensitivity Data Structures for the Insertion of an Edge, 2020. URL: http://www.cse.iitk.ac.in/users/sbaswana/Papers-published/esa-2020-fv.pdf.
3. Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.
4. Rajesh Chitnis, Lior Kamma, and Robert Krauthgamer. Tight bounds for gomory-hu-like cut counting. In *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, pages 133–144, 2016. doi:10.1007/978-3-662-53536-3_12.
5. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.
6. R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. URL: http://www.jstor.org/stable/2098881.
7. Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. Incremental exact min-cut in polylogarithmic amortized update time. *ACM Trans. Algorithms*, 14(2):17:1–17:21, 2018. doi:10.1145/3174803.
8. Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM J. Comput.*, 19(1):143–155, February 1990. doi:10.1137/0219009.
9. Ramesh Hariharan, Telikepalli Kavitha, Debmalya Panigrahi, and Anand Bhalgat. An $\tilde{O}(mn)$ gomory-hu tree construction algorithm for unweighted graphs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 605–614, 2007. See also the extended version at http://hariharan-ramesh.com/papers/gohu.pdf. doi:10.1145/1250790.1250879.
10. Tanja Hartmann and Dorothea Wagner. Fast and simple fully-dynamic cut tree construction. In *Algorithms and Computation - 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, pages 95–105, 2012. doi:10.1007/978-3-642-35261-4_13.
11. Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *In Rayward-Smith V.J. (eds) Combinatorial Optimization II. Mathematical Programming Studies*, 13(1):8–16, 1980. doi:10.1007/BFb0120902.
12. Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007. doi:10.1007/s00493-007-0045-2.

Linear Time LexDFS on Chordal Graphs

Jesse Beisegel
Brandenburg University of Technology, Cottbus, Germany
jesse.beisegel@b-tu.de

Ekkehard Köhler
Brandenburg University of Technology, Cottbus, Germany
ekkehard.koehler@b-tu.de

Robert Scheffler
Brandenburg University of Technology, Cottbus, Germany
robert.scheffler@b-tu.de

Martin Strehler
Brandenburg University of Technology, Cottbus, Germany
martin.strehler@b-tu.de

Abstract

Lexicographic Depth First Search (LexDFS) is a special variant of a Depth First Search (DFS), which was introduced by Corneil and Krueger in 2008. While this search has been used in various applications, in contrast to other graph searches, no general linear time implementation is known to date. In 2014, Köhler and Mouatadid achieved linear running time to compute some special LexDFS orderings for cocomparability graphs. In this paper, we present a linear time implementation of LexDFS for chordal graphs. Our algorithm even implements the extended version LexDFS$^+$ and is, therefore, able to find any LexDFS ordering for this graph class. To the best of our knowledge this is the first unrestricted linear time implementation of LexDFS on a non-trivial graph class. In the algorithm we use a search tree computed by Lexicographic Breadth First Search (LexBFS).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Trees; Theory of computation → Graph algorithms analysis

Keywords and phrases LexDFS, chordal graphs, linear time implementation, search trees, LexBFS

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.13

Acknowledgements The authors would like to thank one of the anonymous referees for his or her many helpful comments.

1 Introduction

Graph searches are among the most basic algorithms in computer science. Nevertheless, they are very powerful tools and can be used to compute many important graph properties. For example, *Breadth First Search* (BFS) is the standard procedure for testing bipartiteness or computing shortest paths with respect to the number of edges. Similarly, *Depth First Search* (DFS) can be used in algorithms to find strongly connected components in directed graphs [24] or to test for planarity [14].

In 1976, Rose, Tarjan, and Lueker [21] proposed a modified variant of BFS to compute perfect vertex elimination orderings of chordal graphs. This search, since named *Lexicographic Breadth First Search* (LexBFS), uses the ordering of the already visited vertices and visits the vertex with lexicographically largest neighborhood next. Rose, Tarjan, and Lueker also gave a linear time implementation of LexBFS using partition refinement, which, for example, also provides a linear time greedy algorithm for finding minimum colorings of chordal graphs.

It was only in 2008 that a corresponding lexicographical variant for DFS was introduced by Corneil and Krueger [8]. Similar to LexBFS, this search computes perfect elimination orderings on chordal graphs. Therefore, it can be used to find minimum colorings as well as all minimal separators and all maximal cliques on this graph class [25]. Besides this, LexDFS was used in the field of data mining to design an efficient hierarchical clustering algorithm [10]. However, no general linear time implementation of LexDFS is known to date. An implementation with running time in $\mathcal{O}(\min\{n^2, n + m \log n\})$ is given in [17]. Spinrad announced an $\mathcal{O}(m \log \log n)$-implementation [23] which has not been published as of yet. In [15], Köhler and Mouatadid present the first linear time algorithm to compute a LexDFS cocomparability ordering, that is, a special class of LexDFS orderings can be computed in linear time on cocomparability graphs using modular decomposition. However, there are LexDFS orderings of cocomparability graphs that cannot be computed by this approach. Even more restricting, it is not possible to choose an arbitrary start vertex for the search. Nevertheless, this result can be used to design linear time algorithms which find minimum path covers [5], maximum matchings [20] as well as maximum independent sets, minimum clique covers and minimum vertex covers [6] on cocomparability graphs.

Search trees are an important concept in the theory of graph searches. Already in 1972, Tarjan [24] gave a complete characterization of DFS-trees as so-called palm trees. However, no algorithm that determines whether a given spanning tree of a graph G is a DFS-tree of G was specified in that work. Using the concept of palm trees, Hopcroft and Tarjan developed a linear time algorithm for testing planarity of a graph [14]. In 1985, Hagerup [12] formulated the problem of checking whether a given spanning tree of G can be obtained by a DFS and presented a linear time algorithm for this problem. In the same year, Hagerup and Novak [13] presented a linear time algorithm for the recognition of BFS-trees. Similar results were obtained by Korach and Ostfeld [16] for DFS-trees and Manber [19] for BFS-trees. Recently, Beisegel et al. [1, 2] studied the search tree recognition problem for LexBFS, LexDFS and other searches.

Our Contribution

In this paper, we give the first linear time implementation of LexDFS on chordal graphs. We show for all graphs that the computation of a LexDFS ordering is linear time equivalent to the construction of a LexDFS search tree, i.e., there are linear time reductions between both problems. The combination of this result with some properties of search trees of LexBFS on chordal graphs yields a linear time implementation of LexDFS$^+$, an extended version of LexDFS, which uses vertex orderings to break ties during the search. This implementation is able to compute any LexDFS ordering of a given chordal graph. To the best of our knowledge this is the first unrestricted linear time implementation of LexDFS on a non-trivial graph class. Furthermore, we show that testing whether a given ordering is in fact a LexDFS ordering is linear time equivalent to the recognition of LexDFS search trees.

2 Preliminaries

Throughout this paper, we consider finite, simple, undirected and connected graphs $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges. An edge between u and v is simply denoted by uv. For a vertex $v \in V$, the *neighborhood* of v is denoted by $N(v)$, i.e., $N(v) = \{u \in V \mid uv \in E\}$. For a subset $S \subseteq V$, we define the neighborhood as $N(S) = \{v \in V \setminus S \mid \exists u \in S : uv \in E\}$.

Given a subset S of vertices in G, the *subgraph of G induced by S* is denoted by $G[S]$, where $V(G[S]) = S$ and $E(G[S]) = \{uv \in E(G) \mid u \in S,\ v \in S\}$. The subgraph induced by $V(G) \setminus S$ is denoted by $G - S$ and, in the case where S contains just one element, we simply write $G - v$ instead of $G - \{v\}$.

A graph G that contains no induced cycle of length larger than 3 is called *chordal*. Other equivalent definitions of chordal graphs can be found in [4]. A *tree* is an acyclic connected graph and a *spanning tree* of a graph G is an acyclic connected subgraph of G which contains all vertices of G. A tree together with a distinguished *root vertex* s is said to be *rooted*. In such a rooted tree a vertex v is an *ancestor* of vertex w if v is an element of the unique path from w to the root s. In particular, if v is adjacent to w, it is called the *parent* of w. A vertex w is called a *descendant (child)* of v if v is an ancestor (the parent) of w.

A (connected) graph search is, in the most general sense, a mechanism for systematically visiting all vertices of a graph. Starting at a vertex $s \in V$, we expand the set of vertices S beginning with $S = \{s\}$ by moving a vertex from $N(S)$ to S, which may also add new neighbors to $N(S)$ in consequence. The result of this procedure is a *search ordering* $\sigma = (v_1 = s, v_2, \ldots, v_n)$ of the vertices of the graph listing the vertices in order of occurrence. For any linear vertex ordering σ we write $u \prec_\sigma v$ if u appears before v in the ordering and say that u is *to the left* of v and that v is *to the right* of u. Furthermore, σ^- denotes the reverse ordering of σ, that is, $\sigma^- = (v_n, v_{n-1}, \ldots, v_1)$.

There are many graph search protocols which differ in the way in which a vertex from $N(S)$ is chosen next. The two most common graph searches are *Breadth First Search* and *Depth First Search* which can be simply described as using a queue and a stack to store the vertices in $N(S)$, respectively. Given a graph search protocol $\mathcal{P}$ and a vertex ordering σ, we say that σ is a *$\mathcal{P}$-ordering* if there exists a valid $\mathcal{P}$ search on G that returns σ.

In [8], Corneil and Krueger present a characterizing *four point property* of DFS orderings.

▶ **Lemma 1** ([8]). *A vertex ordering σ is a DFS ordering of a graph $G = (V, E)$ if and only if for every triple $a \prec_\sigma b \prec_\sigma c$ where $ac \in E$ and $ab \notin E$ there is a vertex d with $a \prec_\sigma d \prec_\sigma b$ such that $db \in E$.*

In the same paper, the authors introduced *Lexicographic Depth First Search* (LexDFS, see Algorithm 1), a variant of DFS which uses labels and their lexicographic order to break ties during the search.

Algorithm 1 Lexicographic Depth First Search.

Input: Connected graph $G = (V, E)$ and a distinguished vertex $s \in V$
Output: Ordering σ of V starting at s

```
1 begin
2     label(s) ← (0);
3     foreach vertex v ∈ V − s do assign to v the empty label;
4     for i ← 1 to n do
5         pick an unnumbered vertex v with lexicographically largest label;
6         σ(i) ← v;
7         foreach unnumbered vertex w ∈ N(v) do prepend i to label(w);
```

The idea of using such a lexicographic order of labels originates from an algorithm for the calculation of perfect elimination orderings of chordal graphs, given by Rose, Lueker, and Tarjan [21], since named *Lexicographic Breadth First Search* (LexBFS, see Algorithm 2).

Both LexDFS and LexBFS are special variants of the standard searches and, thus, every LexDFS ordering is also a DFS ordering and every LexBFS ordering is also a BFS ordering. In both algorithms, the vertices are labeled by their already visited neighbors (see line 7

Algorithm 2 Lexicographic Breadth First Search.

Input: Connected graph $G = (V, E)$ and a distinguished vertex $s \in V$
Output: Ordering σ of V starting at s

```
1 begin
2     label(s) ← (n);
3     foreach vertex v ∈ V − s do assign to v the empty label;
4     for i ← 1 to n do
5         pick an unnumbered vertex v with lexicographically largest label;
6         σ(i) ← v;
7         foreach unnumbered vertex w ∈ N(v) do append (n − i) to label(w);
```

in Algorithm 1 and 2). While in LexDFS vertices visited later in the search have a larger significance for the lexicographic order of the label, in LexBFS it is the opposite, i.e., vertices visited earlier have a larger impact.

Corneil and Krueger [8] also present a four point property of LexDFS orderings.

▶ **Lemma 2** ([8]). *A vertex ordering σ is a LexDFS ordering of a graph $G = (V, E)$ if and only if for every triple $a \prec_\sigma b \prec_\sigma c$ where $ac \in E$ and $ab \notin E$ there is a vertex d with $a \prec_\sigma d \prec_\sigma b$ such that $db \in E$ and $dc \notin E$.*

A variant of LexDFS and LexBFS is the technique of "multisweeping". This describes the multiple application of some graph search, where each run of the search uses the ordering given by the previous application as a so-called *"tie-break" rule*, that is, a priority list which decides which vertex can be visited next in those cases where the given search paradigm allows several different options. It was first used by Simon [22] in an algorithm for the recognition of interval graphs which is flawed as was shown by Ma [18]. Nevertheless, "multisweeping" has proven to be very fruitful in recent years [5, 9]. In the case of LexDFS, this technique implies a new search scheme known as LexDFS$^+$: Given a ordering ρ of the vertices, LexDFS$^+(\rho)$ is computed by executing a regular LexDFS with the modification that in line 5 of Algorithm 1 the rightmost element with regard to ρ is chosen among all vertices with lexicographically largest label. The searches DFS$^+$, BFS$^+$ and LexBFS$^+$ are defined analogously. Note that all these searches yield unique orderings, as there are no more ties to break in the algorithms.

It is not difficult to see that, given the reverse of a search ordering as tie break, such a procedure yields that same ordering again. For a more general result see Corneil et al. [7].

▶ **Observation 3.** *Let $G = (V, E)$ be a graph and let σ be a vertex ordering of G. The ordering σ is a LexDFS ordering of G if and only if LexDFS$^+(\sigma^-)$ is equal to σ. This also holds for LexBFS and LexBFS$^+(\sigma^-)$.*

Using a technique called *partition refinement*, LexBFS can be implemented in linear time [11, 21]. Given a set S, we call $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_k)$ a partition of S if $S = \bigcup_{i=1}^{k} Q_i$ with non-empty, pairwise disjoint sets Q_i ($Q_i \cap Q_j = \emptyset$ for $i \neq j$). Note that a partition is an *ordered* list of subsets. We say that a subset $S' \subseteq S$ refines $\mathcal{Q}$ if Q_i is replaced by a subpartition (A_i, B_i) where $A_i = Q_i \cap S'$ and $B_i = Q_i \setminus A_i$ whenever both sets are non-empty. In particular, for LexBFS we start with $\mathcal{Q} = (V)$ and starting vertex s. Now, we refine $\mathcal{Q}$ with $\{s\}$ which separates s in a single set. Afterwards, we refine with $N(s)$. In the first iteration, this yields the partition $(\{s\}, N(s), V \setminus (N(s) \cup \{s\}))$. The vertex whose neighborhood is used to refine the partition classes is called a *pivot*. Choose the next pivot v from $N(s)$ and

refine with $\{v\}$ and then $N(v)$. For $Q_i = \{v\}$, repeat refining using an element from the set Q_{i+1} as the next pivot, maintaining the order of the partition classes created so far. As shown in [11], this final partition can be computed in linear time and it is actually a LexBFS ordering. The pivot is just the vertex visited by the search and it *pulls* its neighbors to the front of each set. Unfortunately, no linear time implementation of partition refinement for LexDFS is known to date.

Usually, a graph search is associated with a search tree which is a spanning tree of the graph. Given a BFS ordering $\sigma = (v_1, \ldots, v_n)$, a vertex v_i is typically connected to the leftmost neighbor in $(v_1, \ldots, v_{i-1})$, i.e., it is connected to the vertex that was current at the point at which v_i was added to $N(S)$. On the contrary, given a DFS ordering $\sigma = (v_1, \ldots, v_n)$, a vertex v_i is connected to the rightmost neighbor in $(v_1, \ldots, v_{i-1})$, i.e., it is connected to the neighbor which occurred last before v_i itself was visited. These two different approaches of constructing a search tree give rise to the following definition.

▶ **Definition 4** ([2]). *Given a search ordering $\sigma = (v_1, \ldots, v_n)$ of a given search on a connected graph $G = (V, E)$, the* first-in tree *(or $\mathcal{F}$-tree) of σ is the tree consisting of the vertex set V and an edge from each vertex different from v_1 to its leftmost neighbor in σ. The* last-in tree *(or $\mathcal{L}$-tree) of σ is the tree consisting of the vertex set V and an edge from each vertex v_i different from v_1 to its rightmost neighbor v_j in σ with $j < i$. In both cases, v_1 is the* root *of the search tree.*

The notation of $\mathcal{F}$-trees and $\mathcal{L}$-trees was introduced in [2], where the recognition problem of these search trees was studied. In contrast to the original definition, we always assume that a search tree has a designated root. In particular, two search trees on a graph G are equal if they use the same edge set and if they have the same root. Given a search protocol $\mathcal{P}$ and a spanning tree T of G rooted in s, we say that T is an *$\mathcal{L}$-tree* (*$\mathcal{F}$-tree*) of $\mathcal{P}$ on G if there is a $\mathcal{P}$-ordering of G starting at s with $\mathcal{L}$-tree ($\mathcal{F}$-tree) T.

Although, it would be most natural to consider the $\mathcal{F}$-tree for LexBFS, the $\mathcal{L}$-tree of LexBFS is a key ingredient in our procedure on chordal graphs.

3 Search Orderings and Trees of LexDFS

Given a search ordering, it is easy to construct the corresponding search tree in linear time by simply using Definition 4. However, if we are only given a search tree, then it is not immediately clear how to find a search ordering that results in this tree. In this section, we present a linear time algorithm that computes a LexDFS ordering for a given $\mathcal{L}$-tree of LexDFS. Using this result we prove that both recognition and creation of search orderings and $\mathcal{L}$-trees are linear time equivalent in the case of LexDFS.

The main idea of this algorithm is to use a special tie-break rule in form of a ordering of the vertices τ such that a simple run of DFS$^+(\tau)$ on the tree is a LexDFS ordering of G with tree T. This tie-break rule τ is computed by using a form of partition refinement which moves from the leaves of the tree towards its root. The pseudo code of this procedure is given in Algorithm 3.

Before we begin with the analysis of this algorithm we present some general results on $\mathcal{L}$-trees of DFS. The first is a lemma by Tarjan [24] which characterizes $\mathcal{L}$-trees of DFS.

▶ **Lemma 5** ([24]). *Let $G = (V, E)$ be a graph and let T be a spanning tree of G. Then T is an $\mathcal{L}$-tree of G generated by DFS if and only if for each edge $uv \in E$ it holds that either u is an ancestor of v in T or v is an ancestor of u in T.*

Algorithm 3 ORDERING(G, T, s, ρ).

Input: Connected graph $G = (V, E)$, an $\mathcal{L}$-tree T of DFS on G rooted in $s \in V$, ordering ρ of V ending with vertex s
Output: ordering σ of V starting at s

```
1  begin
2      β ← reverse of a BFS ordering of T starting at s;
3      Q ← (V);
4      for i ← 1 to n do
5          v ← β(i);
6          refine Q with {w ∈ N(v) | w ≺_β v};
7      order every set in Q with respect to ρ⁻ and move {s} to the leftmost position;
8      τ ← reverse of the final order of vertices in Q;
9      σ ← DFS⁺(τ) on T;
10     return σ;
```

In order to make sure that Algorithm 3 returns a DFS ordering of G with $\mathcal{L}$-tree T, we prove the following statement.

▶ **Lemma 6.** *Let T be an $\mathcal{L}$-tree of some DFS on G rooted in s and let σ be a DFS ordering of T starting at s. Then σ is a DFS ordering of G with $\mathcal{L}$-tree T.*

Proof. We show that σ is a DFS ordering of G by proving that it fulfills the characterization given in Lemma 1. Let a, b and c be three vertices in G with $a \prec_\sigma b \prec_\sigma c$, $ac \in E(G)$ and $ab \notin E(G)$. We have to show that there is a vertex d with $a \prec_\sigma d \prec_\sigma b$ such that $db \in E(G)$. Assume that $ac \in E(T)$. As σ is a DFS ordering of the tree T, there is a vertex d with $a \prec_\sigma d \prec_\sigma b$ and $db \in E(T) \subseteq E(G)$ due to Lemma 1. Therefore, we can assume that ac is not contained in $E(T)$. By Lemma 5 this implies that a is an ancestor of c in T. Let $P = (a = w_1, \ldots, w_k = c)$ be the unique path between a and c in T. As σ is a DFS ordering of T, it holds that $a \prec_\sigma w_2 \prec_\sigma \ldots \prec_\sigma c$. If there is an $1 < i < k$ such that $w_i = b$, then w_{i-1} is a vertex between a and b in σ with $w_{i-1}b \in E(G)$. Otherwise, there exists an $1 \leq i < k$ with $w_i \prec_\sigma b \prec_\sigma w_{i+1}$ and $w_iw_{i+1} \in E(T)$. As in the first case there exists a vertex d with $a \preceq_\sigma w_i \prec_\sigma d \prec_\sigma b$ and $db \in E(T) \subseteq E(G)$. By Lemma 1, this proves that σ is a DFS ordering.

Let T' be the $\mathcal{L}$-tree of σ with regard to G and assume for contradiction that there is an edge uv in T' that is not part of T. Due to Lemma 5, we can assume without loss of generality that u is an ancestor of v in T. Since uv is not part of T, vertex u is not the parent of v in T. Let w be the parent of v in T. Note that this means that w is a descendant of u in T. Since σ is a DFS ordering on T, vertex w must be to the left of v and to the right of u in σ. Since $vw \in E(G)$, edge uv cannot be part of T', as T' is the $\mathcal{L}$-tree of σ; a contradiction. ◀

With these results on DFS we can proceed to the analysis of Algorithm 3. First we will prove correctness.

▶ **Theorem 7.** *Let T be an $\mathcal{L}$-tree of some DFS on G rooted in s and let ρ be an arbitrary ordering of V ending in s. Let σ be the ordering produced by Algorithm 3 with input (G, T, s, ρ). Then T is an $\mathcal{L}$-tree of LexDFS rooted in s if and only if σ is a LexDFS ordering of G.*

Proof. Assume σ is a LexDFS ordering of G. Due to Lemma 6, the $\mathcal{L}$-tree of σ is T and, therefore, it is an $\mathcal{L}$-tree of LexDFS.

For the other direction, assume that T is an $\mathcal{L}$-tree of LexDFS. Let σ^* be a LexDFS ordering of G such that the $\mathcal{L}$-tree of σ^* is T and the common prefix of σ and σ^* is maximal among all LexDFS orderings with $\mathcal{L}$-tree T. If σ and σ^* are equal, then we are done. Otherwise let $i \in \{1, \dots, n\}$ be the first index for which $v = \sigma(i) \neq \sigma^*(i) = v^*$. By Lemma 6, both σ and σ^* are DFS orderings with $\mathcal{L}$-tree T and v and v^* have the same parent p in T. If v and v^* have the same neighborhood in the set $S = \{\sigma(j) \mid j < i\}$, then v could have been taken by LexDFS instead of v^* and this choice would not have had an impact on the $\mathcal{L}$-tree of the ordering, due to Lemma 5. Hence, there must be a vertex $w \in S$ with $wv^* \in E(G)$ and $wv \notin E(G)$ and for all vertices x with $w \prec_\sigma x \prec_\sigma v$ it holds that both v and v^* are adjacent to x or both are not adjacent to x. Note that w is an ancestor of both v and v^* in T and therefore, it is to the right of both vertices in β.

However, this means that before the iteration of the `for`-loop in lines 4–6, where we consider vertex w, both v and v^* are in the same set of $\mathcal{Q}$. After this iteration, vertex v^* is in a set of $\mathcal{Q}$ to the left of the set containing v. Therefore, v^* is to the right of v in τ, as τ uses the reverse ordering of $\mathcal{Q}$. Thus, the search $\mathrm{DFS}^+(\tau)$ visits v^* before v, as both are children of p; a contradiction to v being to the left of v^* in σ. ◀

As seen in the proof, it is not necessary to use the reverse of a BFS ordering for β. Any ordering will suffice, where for every vertex w all ancestors in T are to the right of w in the ordering. Having shown that Algorithm 3 returns a correct LexDFS ordering for any $\mathcal{L}$-tree of LexDFS, we will now evaluate its running time.

▶ **Lemma 8.** *Algorithm 3 has running time in $\mathcal{O}(n + m)$.*

Proof. Algorithm 3 begins with an execution of BFS which can be done in linear time. In the `for`-loop we iterate through the neighborhood of every vertex exactly once. Thus, the overall costs are in $\mathcal{O}(n + m)$. To sort the sets of $\mathcal{Q}$ with respect to ρ^- we iterate through ρ^- and move the considered vertex to the end of its set. The final $\mathrm{DFS}^+(\tau)$ can be executed in linear time by first sorting the neighborhoods of all vertices with respect to τ. ◀

The last results imply that the construction of an $\mathcal{L}$-tree of LexDFS is linear time equivalent to the computation of a LexDFS ordering.

▶ **Theorem 9.** *For a given graph family $\mathcal{G}$ and $\mathcal{O}(M) \supseteq \mathcal{O}(n + m)$ the following two statements are equivalent:*

1. *There is an algorithm with running time in $\mathcal{O}(M)$ that computes a LexDFS ordering for any graph in $\mathcal{G}$ and any starting vertex s.*
2. *There is an algorithm with running time in $\mathcal{O}(M)$ that creates an $\mathcal{L}$-tree of LexDFS for any graph in $\mathcal{G}$ and any root s.*

Proof. It is easy to see that the $\mathcal{L}$-tree of an arbitrary vertex ordering can be constructed in $\mathcal{O}(n + m)$. Therefore, an $\mathcal{O}(M)$-algorithm for the computation of a LexDFS ordering starting at s directly implies an $\mathcal{O}(M)$-algorithm for the creation of an $\mathcal{L}$-tree of LexDFS rooted in s.

If, on the other hand, we can compute an $\mathcal{L}$-tree of LexDFS rooted in s in time $\mathcal{O}(M)$, then we can use Algorithm 3 to create a corresponding LexDFS ordering in linear time, due to Theorem 7 and Lemma 8. ◀

This linear time equivalence does not only hold for the computation but also for the recognition of $\mathcal{L}$-trees and orderings of LexDFS. To prove this we need the following two technical lemmas.

▶ **Lemma 10.** *Let T be an $\mathcal{L}$-tree of a DFS on G rooted in s, let ρ be an arbitrary ordering of V ending with s and let σ be the ordering produced by Algorithm 3 with input (G, T, s, ρ). Furthermore, let v and w be two vertices in G with $v \prec_\sigma w$ which have the same parent in T and the same neighborhood in the set $Y = \{x \mid x \prec_\sigma v\}$. Then v is to the right of w in ρ.*

Proof. As v is to the left of w in σ by assumption, vertex v was taken before w in $DFS^+(\tau)$. Since v and w have the same parent in T, it holds that v is to the right of w in τ. If the vertex v is pulled by a vertex x in the `for`-loop of Algorithm 3, then x is adjacent to v and has a smaller distance to the root s in T. By Lemma 5, vertex x is an ancestor of v in T and x is to the left of v in σ. As v and w have the same neighborhood in Y, the vertex x is also adjacent to w and pulls it, too. This implies that v and w are in the same set of $\mathcal{Q}$ after the `for`-loop. Therefore, v has to be to the right of w in ρ, as it is to the right of w in τ. ◀

▶ **Lemma 11.** *Let T be an $\mathcal{L}$-tree of LexDFS on G rooted in s and let σ be a vertex ordering of a graph G starting at s whose corresponding $\mathcal{L}$-tree is T. Algorithm 3 returns σ for input (G, T, s, σ^-) if and only if σ is a LexDFS ordering of G.*

Proof. If Algorithm 3 returns σ for input (G, T, s, σ^-), then, by Theorem 7, σ is a LexDFS ordering of G.

Therefore, we assume that σ is a LexDFS ordering and Algorithm 3 returns the LexDFS ordering σ^* for input (G, T, s, σ^-) with $\sigma \neq \sigma^*$. Let $i \in \{1, \ldots, n\}$ be the first index where $v = \sigma(i) \neq \sigma^*(i) = v^*$ and let σ_i be the prefix of the first $i-1$ elements of σ (and σ^*). It follows that v and v^* have the same neighborhood in σ_i and, thus, the same parent in T. Since v is to the right of v^* in σ^- it follows from Lemma 10 that v must be to the left of v^* in σ^*; a contradiction. ◀

Now we can prove the linear time equivalence of tree recognition and ordering verification for LexDFS.

▶ **Theorem 12.** *For a given graph family $\mathcal{G}$ and $\mathcal{O}(M) \supseteq \mathcal{O}(n+m)$ the following two statements are equivalent:*

1. *There is an algorithm with running time in $\mathcal{O}(M)$ that checks for any graph G in $\mathcal{G}$ and any vertex s whether a given ordering beginning in s is a LexDFS ordering of G.*
2. *There is an algorithm with running time in $\mathcal{O}(M)$ for any graph G in $\mathcal{G}$ and any vertex s that checks whether a given spanning tree rooted in s is an $\mathcal{L}$-tree of LexDFS on G.*

Proof. Assume we have an algorithm $\mathcal{A}$ for the recognition of LexDFS orderings with running time in $\mathcal{O}(M)$. For a given spanning tree T of G rooted in s we first decide in linear time whether T is an $\mathcal{L}$-tree of DFS (see [12, 16]). If not, then it is not an $\mathcal{L}$-tree of LexDFS. Otherwise, we execute Algorithm 3 with the input (G, T, s, ρ), where ρ is an arbitrary ordering of the vertices of G, and get the vertex ordering σ as result in linear time. Due to Theorem 7, T is an $\mathcal{L}$-tree of LexDFS if and only if σ is an LexDFS ordering of G. We use $\mathcal{A}$ to decide this in time $\mathcal{O}(M)$.

Now assume we have an algorithm $\mathcal{A}$ for the recognition of $\mathcal{L}$-trees of LexDFS with running time in $\mathcal{O}(M)$ and get a vertex ordering σ starting at s. We first create the $\mathcal{L}$-tree T of σ in linear time and check whether T is an $\mathcal{L}$-tree of LexDFS in time $\mathcal{O}(M)$. If not, σ is not an LexDFS ordering of G. Otherwise, we call Algorithm 3 with input (G, T, s, σ^-). Due to Lemma 11, the resulting vertex ordering is equal to σ if and only if σ is a LexDFS ordering of G. ◀

Note that this result does not hold for search orderings and their corresponding trees in general (if $\mathcal{P} \neq \mathcal{NP}$). Beisegel et al. [1, 2] show for example that the recognition problem of $\mathcal{F}$-trees of both LexBFS and LexDFS is $\mathcal{NP}$-complete, whereas it is easy to recognize the corresponding orderings.

4 LexDFS on Chordal Graphs

We will now use the results of the last section to derive a linear time implementation of LexDFS for chordal graphs. We first show that LexBFS and LexDFS have the same set of $\mathcal{L}$-trees on chordal graphs. This fact is also implied by a more general result in [1]. Since this work has not been published yet and we only need a special case here, we give an alternative proof for the sake of completeness.

In [3], Berry et al. show that a whole range of different graph search schemes share the same set of search orderings on chordal graphs. Among these searches are variants of both LexDFS and LexBFS, called *CompLexDFS* and *CompLexBFS*, respectively. For these algorithms we replace line 5 in both Algorithm 1 and 2 by "choose a component C of the graph induced by the unnumbered vertices and take a vertex in C with lexicographically largest label".

▶ **Lemma 13** ([3]). *For any chordal graph G a linear vertex ordering is a CompLexDFS ordering if and only if it is a CompLexBFS ordering.*

We now show that both LexDFS and LexBFS compute the same $\mathcal{L}$-trees as their respective Comp-variants for any graph.

▶ **Lemma 14.** *A spanning tree T of a graph G rooted in s is an $\mathcal{L}$-tree of LexDFS (LexBFS) on G if and only if T is an $\mathcal{L}$-tree of CompLexDFS (CompLexBFS) on G.*

Proof. Since every ordering of LexDFS is also an ordering of CompLexDFS, every $\mathcal{L}$-tree of LexDFS on G is also an $\mathcal{L}$-tree of CompLexDFS.

For the reverse we first introduce some technical definitions. Let $\tau = (w_1, \dots, w_n)$ be some ordering of the vertices of G. We define $C_\tau(w_i)$ to be the connected component of $G - \{w_1, \dots, w_{i-1}\}$ containing w_i. Now, consider a CompLexDFS ordering σ of G with $\mathcal{L}$-tree T. Let σ^* be the LexDFS$^+(\sigma^-)$ ordering of G. We claim that T is the $\mathcal{L}$-tree of σ^*. To this end, we show that $C_\sigma(v) = C_{\sigma^*}(v)$ for every vertex $v \in V$. Furthermore, we show that for every vertex $w \in C_\sigma(v) = C_{\sigma^*}(v)$ it holds that the label of w at point where v is chosen in σ is the same as the label of w when v is chosen in σ^*.

Assume for contradiction that v is the leftmost vertex in σ which does not fulfill both of these properties. Let w be the rightmost vertex in σ with $w \prec_\sigma v$ such that w has a neighbor in $C_\sigma(v)$. Due to choice of v, it holds that $C_\sigma(w) = C_{\sigma^*}(w)$ and both components are labeled the same at the moment w is chosen in the respective search. As the labels cannot be changed from outside of the component, there must be a vertex in $C_\sigma(v)$ that is between w and v in σ^*. Let x be the leftmost vertex in σ^* with this property. At the point where x is chosen by σ^* the labels of both x and v are the same as in σ at the point when v was chosen. Therefore, the labels of x and v must be the same at the point where x was chosen in σ^*. However, v is to the right of x in σ^- and it has to be chosen before x in σ^*; a contradiction.

Now, let T^* be the $\mathcal{L}$-tree of σ^* and assume that the parent of vertex y in T is p and the parent of y in T^* is $p^* \neq p$. Due to the observation above, both p and p^* must be to the left of y in both σ and σ^*. Therefore, it holds that $p \prec_{\sigma^*} p^*$ and $p^* \prec_\sigma p$. However, this is a contradiction to the observation above since p would be in $C_\sigma(p^*)$ but not in $C_{\sigma^*}(p^*)$.

Since no special property of LexDFS and CompLexDFS is used in the proof above, the claim also holds for LexBFS and CompLexBFS. ◀

Combining Lemmas 13 and 14 yields the following corollary.

▶ **Corollary 15.** *Let $G = (V, E)$ be a chordal graph and T be a spanning tree of G rooted in $s \in V$. The tree T is an $\mathcal{L}$-tree of LexDFS of G if and only if T is an $\mathcal{L}$-tree of LexBFS of G.*

Using this corollary, we can compute an $\mathcal{L}$-tree of LexDFS rooted in vertex s for any chordal graph G by using LexBFS. This tree can then be used as the input for Algorithm 3 to return a LexDFS ordering for G. Furthermore, it is possible to implement LexDFS$^+$ in linear time for chordal graphs using the same approach (see Algorithm 4).

Algorithm 4 LexDFS$^+$ on chordal graphs.

Input: Chordal graph $G = (V, E)$, vertex $s \in V$, ordering ρ of V ending with s
Output: The LexDFS$^+(\rho)$ ordering σ of G

```
1 begin
2     π ← LexBFS+(ρ) ordering of G;
3     T ← L-tree of π;
4     σ ← Ordering(G, T, s, ρ);
5     return σ;
```

▶ **Theorem 16.** *Let $G = (V, E)$ be a chordal graph, s be a vertex in V and ρ be an arbitrary ordering of V ending in s. Then for input (G, s, ρ) Algorithm 4 produces the LexDFS$^+(\rho)$ ordering of G in time $\mathcal{O}(n + m)$.*

Proof. Due to Corollary 15, the tree T is an $\mathcal{L}$-tree of LexDFS. By Theorem 7, Ordering(G, T, s, ρ) produces a LexDFS ordering of G starting at s.

It remains to show that σ is also the LexDFS$^+(\rho)$ ordering. Let σ^* be the LexDFS$^+(\rho)$ ordering of G and assume that $\sigma \neq \sigma^*$. Let $i \in \{1, \ldots, n\}$ be the first index where $v = \sigma(i) \neq \sigma^*(i) = v^*$ and let σ_i be the prefix of the first $i - 1$ elements of σ (and σ^*). It follows that v and v^* have the same neighborhood in σ_i and v^* must be to the right of v in ρ.

Assume that v is to the left of v^* in the LexBFS$^+(\rho)$ ordering π. Since $v \prec_\rho v^*$, vertex v had a larger label than v^* at the point where it was chosen in π. This implies that there is a vertex w with $w \prec_\pi v \prec_\pi v^*$ such that $wv \in E(G)$ but $wv^* \notin E(G)$. Due to Lemma 5, vertex w has to be an ancestor of v in T and, therefore, w is in σ_i. This is a contradiction as v and v^* have the same neighbors in σ_i.

Therefore, we can assume that v^* is to the left of v in π. If v and v^* have the same parent in T, then vertex v has to be to the right of v^* in ρ, due to Lemma 10; a contradiction. Thus, assume that p is the parent of v but not the parent of v^* in T. However since p is in σ_i, vertex v^* is adjacent to p in G and, therefore, is a descendant of p in T, due to Lemma 5. Let x be the unique child of p in T, which is an ancestor of v^*. Note that $x \neq v$ since otherwise $v \prec_\pi v^*$. Furthermore, it holds that both $x \prec_\pi v^*$ and $v \prec_\sigma x$ and every neighbor of x which is to the left of x in π is an element of σ_i, due to the choice of i. If x has the same neighborhood in σ_i as v and v^*, then $v^* \prec_\rho x$ and, due to Lemma 10, it holds that $x \prec_\sigma v$; a contradiction. Thus, x has a neighbor in σ_i which is neither a neighbor of v nor of v^*. Let y be the rightmost vertex in σ_i with this property. Since σ is a LexDFS ordering there must be a vertex z with $y \prec_\sigma z \prec_\sigma v$ such that $vz \in E(G)$ and $xz \notin E(G)$, due to Lemma 2. Since z is also adjacent to v^* we can use Lemma 2 again leading to a vertex u with $z \prec_\sigma u \prec_\sigma x$ that is adjacent to x but not to v^*. Due to the choice of y, vertex u must be to the right of v in σ. This is contradiction to p being the parent of x in the $\mathcal{L}$-tree T of σ.

Since all three steps can be executed in linear time (see Lemma 8), the algorithm has linear running time in total. ◀

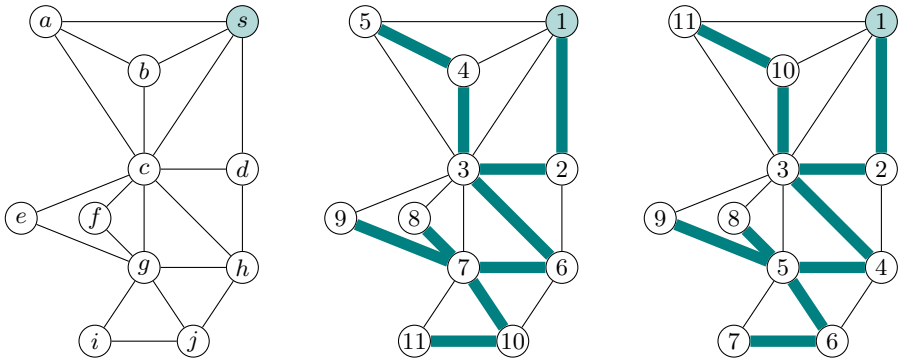


Figure 1 The graph $G = (V, E)$ on the left side is chordal. In the middle, the $\mathcal{L}$-tree of a LexBFS starting at s is shown. Vertex labels correspond to the index in the search ordering. On the right side, the same tree is shown, but now the labeling fits to a LexDFS starting at s.

It follows from Observation 3 that Algorithm 4 is able to compute any LexDFS ordering of a chordal graph G.

▶ **Corollary 17.** *Algorithm 4 can compute any LexDFS ordering of a chordal graph G.*

This result does not hold for efficient implementations of graph searches in general. One example is *Maximal Neighborhood Search (MNS)* introduced by Corneil and Krueger in 2008 [8] as a generalization of both LexBFS and LexDFS. This search can be implemented with linear running time by implementing LexBFS. However, not every MNS ordering is a LexBFS ordering, so this approach can only compute a subset of the MNS orderings of a graph. Observation 3 also leads to an easy recognition algorithm of LexDFS orderings.

▶ **Corollary 18.** *LexDFS orderings can be recognized in linear time on chordal graphs.*

Note that this result can also be achieved using Theorem 12 and the fact that $\mathcal{L}$-trees of LexDFS on chordal graphs can be recognized in linear time (see [1]).

To illustrate the final procedure of Algorithm 4, we give the following example.

▶ **Example 19.** Given the chordal graph in Figure 1, start vertex s and $\rho = (a, b, \ldots, j, s)$, we begin by computing a LexBFS$^+(\rho)$ ordering using partition refinement. The first pivot is s with neighborhood $N(s) = \{d, c, b, a\}$, which yields the partition $(s)(d, c, b, a)(j, i, h, g, f, e)$. With d as the next pivot, we obtain $(s)(d)(c)(b, a)(h)(j, i, g, f, e)$. After a few more steps, we have $\pi = (s, d, c, b, a, h, g, f, e, j, i)$ which is the LexBFS$^+(\rho)$ ordering. Now, we consider the $\mathcal{L}$-tree T induced by this ordering, which is shown in Figure 1.

By Corollary 15, we see that T is also an $\mathcal{L}$-tree of LexDFS rooted in s. We first compute the ordering β and, as seen before, we can use any ordering where the children of a vertex are always to the left of their parent. Thus, we can use $\beta = \pi^-$, although it is not a BFS ordering of T, since T is an $\mathcal{L}$-tree and not a standard $\mathcal{F}$-tree.

Now, we iterate through β and use partition refinement, beginning with $\mathcal{Q} = \beta$, to compute a final tie-breaking rule τ (see Algorithm 3). Vertex i has an empty neighborhood to the left so nothing has to be done. Vertex j as neighbor i to the left in β, so the first refinement of $\mathcal{Q}$ occurs. After processing vertices e and f with empty left neighborhoods, we refine for g with $\{e, f, i, j\}$. This yields the intermediate partition $(i)(j, e, f)(g, h, a, b, c, d, s)$. Finally, we obtain $\mathcal{Q} = (i)(j)(e, f)(a)(g)(h)(b)(c)(d)(s)$.

$\mathcal{Q}$ is post-processed to compute τ. Here, we sort (e, f) with respect to ρ^-, which is the only part of $\mathcal{Q}$ with more than one vertex. Furthermore, we move s to the leftmost position and reverse the whole ordering. This yields $\tau = (d, c, b, h, g, a, e, f, j, i, s)$. Now, we perform

a final DFS$^+(\tau)$ on the tree T. We start in s and follow the unique path to c. Vertex c has two children in T and τ forces us to take h, since h is to the right of b in τ. Similarly, j is chosen after g due to τ. The final LexDFS$^+(\rho)$ ordering is $(s, d, c, h, g, j, i, f, e, b, a)$. It is shown on the right side of Figure 1.

5 Conclusion

In this paper, we have presented the first linear time implementation of LexDFS on chordal graphs. This is already the second important subclass of perfect graphs, the other being cocomparability graphs, that admits a linear time implementation of LexDFS. In contrast to the algorithm for cocomparability graphs [15], however, our approach can compute any LexDFS ordering with arbitrary start vertices. Thus, it also yields the first unrestricted linear time implementation of LexDFS on interval graphs, which are the intersection of cocomparability graphs and chordal graphs. It remains an open question whether this result can be algorithmically exploited to efficiently solve problems on other subclasses of chordal graphs besides interval graphs in linear time.

In the light of these results, the question of whether LexDFS can be executed in linear time in general is even more interesting. There are several open questions. Can the search tree approach be extended? Are there other graph classes where rooted $\mathcal{L}$-trees of LexBFS and LexDFS coincide or is this a characteristic property of chordal graphs? It is also possible that there are other graph classes and other modifications of graph searches that produce a tree equal to an $\mathcal{L}$-tree of LexDFS in linear time on graphs of this particular class.

However, if the answer was "no", that is, we cannot find a general linear time algorithm for LexDFS, it is an interesting question whether recognizing LexDFS orderings can be done faster than actually generating one. In particular, is it possible to check in linear time whether a given vertex ordering or a search tree belongs to LexDFS?

References

1 Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler, and Martin Strehler. The recognition problem of graph search trees. Submitted.

2 Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler, and Martin Strehler. Recognizing graph search trees. In *Proceedings of Lagos 2019, the tenth Latin and American Algorithms, Graphs and Optimization Symposium*, volume 346 of *ENTCS*, pages 99–110. Elsevier, 2019. doi:10.1016/j.entcs.2019.08.010.

3 Anne Berry, Richard Krueger, and Geneviève Simonet. Maximal label search algorithms to compute perfect and minimal elimination orderings. *SIAM Journal on Discrete Mathematics*, 23(1):428–446, 2009. doi:10.1137/070684355.

4 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999. doi:10.1137/1.9780898719796.

5 Derek G. Corneil, Barnaby Dalton, and Michel Habib. LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM Journal on Computing*, 42(3):792–807, 2013. doi:10.1137/11083856X.

6 Derek G. Corneil, Jérémie Dusart, Michel Habib, and Ekkehard Köhler. On the power of graph searching for cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 30(1):569–591, 2016. doi:10.1137/15M1012396.

7 Derek G. Corneil, Jérémie Dusart, Michel Habib, Antoine Mamcarz, and Fabien De Montgolfier. A tie-break model for graph search. *Discrete Applied Mathematics*, 199:89–100, 2016. doi:10.1016/j.dam.2015.06.011.

8 Derek G. Corneil and Richard M. Krueger. A unified view of graph searching. *SIAM Journal on Discrete Mathematics*, 22(4):1259–1276, 2008. doi:10.1137/050623498.

9 Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS structure and recognition of interval graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1905–1953, 2009. doi:10.1137/S0895480100373455.

10 Jean Creusefond, Thomas Largillier, and Sylvain Peyronnet. A LexDFS-based approach on finding compact communities. In Mehmet Kaya, Özcan Erdoğan, and Jon Rokne, editors, *From Social Data Mining and Analysis to Prediction and Community Detection*, pages 141–177. Springer, Cham, 2017. doi:10.1007/978-3-319-51367-6_7.

11 Michel Habib, Ross McConnell, Christophe Paul, and Laurent Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2):59–84, 2000. doi:10.1016/S0304-3975(97)00241-7.

12 Torben Hagerup. Biconnected graph assembly and recognition of DFS trees. Technical Report A 85/03, Universität des Saarlandes, 1985. doi:10.22028/D291-26437.

13 Torben Hagerup and Manfred Nowak. Recognition of spanning trees defined by graph searches. Technical Report A 85/08, Universität des Saarlandes, 1985.

14 John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM*, 21:549–568, 1974. doi:10.1145/321850.321852.

15 Ekkehard Köhler and Lalla Mouatadid. Linear time LexDFS on cocomparability graphs. In R. Ravi and Inge Li Gørtz, editors, *Algorithm Theory – SWAT 2014*, volume 8503 of *LNCS*, pages 319–330, Cham, 2014. Springer. doi:10.1007/978-3-319-08404-6_28.

16 Ephraim Korach and Zvi Ostfeld. DFS tree construction: Algorithms and characterizations. In Jan van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science*, volume 344 of *LNCS*, pages 87–106, Berlin, Heidelberg, 1989. Springer. doi:10.1007/3-540-50728-0_37.

17 Richard M. Krueger. *Graph Searching*. PhD thesis, University of Toronto, 2005. URL: http://www.cs.toronto.edu/~krueger/papers/thesis.ps.

18 Tze-Heng Ma. Unpublished manuscript.

19 Udi Manber. Recognizing breadth-first search trees in linear time. *Information Processing Letters*, 34(4):167–171, 1990. doi:10.1016/0020-0190(90)90155-Q.

20 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. A linear-time algorithm for maximum-cardinality matching on cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2820–2835, 2018. doi:10.1137/17M1120920.

21 Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. doi:10.1137/0205021.

22 Klaus Simon. A new simple linear algorithm to recognize interval graphs. In Hanspeter Bieri and Hartmut Noltemeier, editors, *Computational Geometry – Methods, Algorithms and Applications*, volume 553 of *LNCS*, pages 289–308, Berlin, Heidelberg, 1991. Springer. doi:10.1007/3-540-54891-2_22.

23 Jeremy P. Spinrad. Efficient implementation of lexicographic depth first search. Submitted.

24 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

25 Shou-Jun Xu, Xianyue Li, and Ronghua Liang. Moplex orderings generated by the LexDFS algorithm. *Discrete Applied Mathematics*, 161(13-14):2189–2195, 2013. doi:10.1016/j.dam.2013.02.028.

Grundy Distinguishes Treewidth from Pathwidth

Rémy Belmonte
University of Electro-Communications, Chofu, Tokyo, Japan
https://remybelmonte.wordpress.com/
remybelmonte@gmail.com

Eun Jung Kim
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, Paris, France
https://www.lamsade.dauphine.fr/~kim/
eun-jung.kim@dauphine.fr

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, Paris, France
https://www.lamsade.dauphine.fr/~mlampis/
michail.lampis@lamsade.dauphine.fr

Valia Mitsou
Université de Paris, IRIF, CNRS, France
https://www.irif.fr/~vmitsou/
vmitsou@irif.fr

Yota Otachi
Nagoya University, Nagoya, 464-8601, Japan
https://www.math.mi.i.nagoya-u.ac.jp/~otachi/cv.html
otachi@nagoya-u.jp

Abstract

Structural graph parameters, such as treewidth, pathwidth, and clique-width, are a central topic of study in parameterized complexity. A main aim of research in this area is to understand the "price of generality" of these widths: as we transition from more restrictive to more general notions, which are the problems that see their complexity status deteriorate from fixed-parameter tractable to intractable? This type of question is by now very well-studied, but, somewhat strikingly, the algorithmic frontier between the two (arguably) most central width notions, treewidth and pathwidth, is still not understood: currently, no natural graph problem is known to be W-hard for one but FPT for the other. Indeed, a surprising development of the last few years has been the observation that for many of the most paradigmatic problems, their complexities for the two parameters actually coincide exactly, despite the fact that treewidth is a much more general parameter. It would thus appear that the extra generality of treewidth over pathwidth often comes "for free".

Our main contribution in this paper is to uncover the first natural example where this generality comes with a high price. We consider Grundy Coloring, a variation of coloring where one seeks to calculate the worst possible coloring that could be assigned to a graph by a greedy First-Fit algorithm. We show that this well-studied problem is FPT parameterized by pathwidth; however, it becomes significantly harder (W[1]-hard) when parameterized by treewidth. Furthermore, we show that Grundy Coloring makes a second complexity jump for more general widths, as it becomes para-NP-hard for clique-width. Hence, Grundy Coloring nicely captures the complexity trade-offs between the three most well-studied parameters. Completing the picture, we show that Grundy Coloring is FPT parameterized by modular-width.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Treewidth, Pathwidth, Clique-width, Grundy Coloring

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.14

Related Version https://arxiv.org/abs/2008.07425

Funding Supported under the PRC CNRS JSPS 2019-2020 program, project PARAGA (Parameterized Approximation Graph Algorithms).
Rémy Belmonte: The author was partially supported by JSPS KAKENHI Grant Number JP18K11157.
Eun Jung Kim: The author was partially supported by ANR JCJC Grant Number 18-CE40-0025-01
Yota Otachi: The author was partially supported by JSPS KAKENHI Grant Numbers JP18K11168, JP18K11169, JP18H04091.

1 Introduction

The study of the algorithmic properties of *structural graph parameters* has been one of the most vibrant research areas of parameterized complexity in the last few years. In this area we consider graph complexity measures ("graph width parameters"), such as treewidth, and attempt to characterize the class of problems which become tractable for each notion of width. The most important graph widths are often comparable to each other in terms of their generality. Hence, one of the main goals of this area is to understand which problems separate two comparable parameters, that is, which problems transition from being FPT for a more restrictive parameter to W-hard for a more general one[1]. This endeavor is sometimes referred to as determining the "price of generality" of the more general parameter.

The two most widely studied graph widths are probably treewidth and pathwidth, which have an obvious containment relationship to each other. Despite this, to the best of our knowledge, no natural problem is currently known to delineate their complexity border in the sense we just described. Our main contribution is exactly to uncover a natural, well-known problem which fills this gap. Specifically, we show that Grundy Coloring, the problem of ordering the vertices of a graph to maximize the number of colors used by the First-Fit coloring algorithm, is FPT parameterized by pathwidth, but W[1]-hard parameterized by treewidth. We then show that Grundy Coloring makes a further complexity jump if one considers clique-width, as in this case the problem is para-NP-complete. Hence, Grundy Coloring turns out to be an interesting specimen, nicely demonstrating the algorithmic trade-offs involved among the three most central graph widths.

Graph widths and the price of generality. Much of modern parameterized complexity theory is centered around studying graph widths, especially treewidth and its variants. In this paper we focus on the parameters summarized in Figure 1, and especially the parameters that form a linear hierarchy, from vertex cover, to tree-depth, pathwidth, treewidth, and clique-width. Each of these parameters is a strict generalization of the previous ones in this list. On the algorithmic level we would expect this relation to manifest itself by the appearance of more and more problems which become *intractable* as we move towards the more general parameters. Indeed, a search through the literature reveals that for each step in this list of parameters, several *natural* problems have been discovered which distinguish the two consecutive parameters (we give more details below). The one glaring exception to this rule seems to be the relation between treewidth and pathwidth.

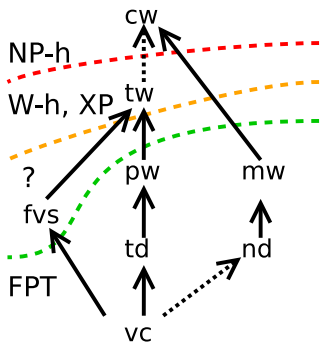
Treewidth is a parameter of central importance to parameterized algorithmics, in part because wide classes of problems (notably all MSO_2-expressible problems [18]) are FPT for this parameter. Treewidth is usually defined in terms of tree decompositions of graphs, which naturally leads to the equally well-known notion of pathwidth, defined by forcing the decomposition to be a path. On a graph-theoretic level, the difference between the two

[1] We assume the reader is familiar with the basics of parameterized complexity theory, such as the classes FPT and W[1], as given in standard textbooks [21].

notions is well-understood and treewidth is known to describe a much richer class of graphs. In particular, while all graphs of pathwidth k have treewidth at most k, there exist graphs of constant treewidth (in fact, even trees) of unbounded pathwidth. Naturally, one would expect this added richness of treewidth to come with some negative algorithmic consequences in the form of problems which are FPT for pathwidth but W-hard for treewidth. Furthermore, since treewidth and pathwidth are probably the most studied parameters in our list, one might expect the problems that distinguish the two to be the first ones to be discovered.

Nevertheless, so far this (surprisingly) does not seem to have been the case: on the one hand, FPT algorithms for pathwidth are DPs which also extend to treewidth; on the other hand, we give (in Section 1.1) a semi-exhaustive list of dozens of natural problems which are W[1]-hard for treewidth and turn out without exception to also be hard for pathwidth. In fact, even when this is sometimes not explicitly stated in the literature, the same reduction that establishes W-hardness by treewidth also does so for pathwidth. Intuitively, an explanation for this phenomenon is that the basic structure of such reductions typically resembles a $k \times n$ (or smaller) grid, which has both treewidth and pathwidth bounded by k.

Our main motivation in this paper is to take a closer look at the algorithmic barrier between pathwidth and treewidth and try to locate a natural (that is, not artificially contrived) problem whose complexity transitions from FPT to W-hard at this barrier. Our main result is the proof that Grundy Coloring is such a problem. This puts in the picture the last missing piece of the puzzle, as we now have natural problems that distinguish the parameterized complexity of any two consecutive parameters in our main hierarchy.



Parameter	Result	Ref
Clique-width	para-NP-hard	Theorem 25
Treewidth	W[1]-hard	Theorem 16
Pathwidth	FPT	Theorem 20
Modular-width	FPT	Theorem 26

In the figure, clique-width, treewidth, pathwidth, tree-depth, vertex cover, feedback vertex set, neighborhood diversity, and modular-width are indicated as cw, tw, pw, td, vc, fvs, nd, and mw respectively. Arrows indicate more general parameters. Dotted arrows indicate that the parameter may increase exponentially, (e.g. graphs of vc k have nd at most $2^k + k$).

Figure 1 Summary of considered graph parameters and results.

Grundy Coloring. In the Grundy Coloring problem we are given a graph $G = (V, E)$ and are asked to order V in a way that maximizes the number of colors used by the greedy (First-Fit) coloring algorithm. The notion of Grundy coloring was first introduced by Grundy in the 1930s, and later formalized in [17]. Since then, the complexity of Grundy Coloring has been very well-studied (see [1, 3, 14, 30, 44, 46, 52, 55, 73, 74, 76, 77, 78] and the references therein). For the natural parameter, namely the number of colors to be used, Grundy coloring was recently proved to be W[1]-hard in [1]. An XP algorithm for Grundy Coloring parameterized by treewidth was given in [74], using the fact that the Grundy number of any graph is at most $\log n$ times its treewidth. In [13] Bonnet et al. explicitly asked whether this can be improved to an FPT algorithm. They also observed that the problem is FPT parameterized by vertex cover. It appears that the complexity of Grundy Coloring parameterized by pathwidth was never explicitly posed as a question and it was

not suspected that it may differ from that for treewidth. We note that, since the problem (as given in Definition 1) is easily seen to be MSO_1 expressible for a fixed Grundy number, it is FPT for all considered parameters if the Grundy number is also a parameter [19], so we intuitively want to concentrate on cases where the Grundy number is large.

Our results. Our results illuminate the complexity of Grundy Coloring parameterized by pathwidth and treewidth, as well as clique-width and modular-width. More specifically:

1. We show that Grundy Coloring is W[1]-hard parameterized by treewidth via a reduction from k-Multi-Colored Clique. The main building block of our reduction is the structure of binomial trees, which have treewidth one but unbounded pathwidth, which explains the complexity jump between the two parameters. As mentioned, an XP algorithm is known in this case [74], so this result is in a sense tight.
2. We show that Grundy Coloring is FPT parameterized by pathwidth. Our main tool here is a combinatorial lemma, which draws heavily from known combinatorial bounds on the performance of First-Fit coloring on intervals graphs [53, 65]. We use this lemma to show that on any graph the Grundy number is at most a linear function of the pathwidth.
3. We show that Grundy Coloring is para-NP-complete parameterized by clique-width, that is, NP-complete for graphs of constant clique-width (specifically, clique-width 6).
4. We show that Grundy Coloring is FPT parameterized by neighborhood diversity (which is defined in [56]) and leverage this result to obtain an FPT algorithm parameterized by modular-width (which is defined in [38]).

Our main interest is concentrated in the first two results, which achieve our goal of finding a natural problem distinguishing pathwidth from treewidth. The result for clique-width nicely fills out the picture by giving an intuitive view of the evolution of the complexity of the problem and showing that in a case where no non-trivial bound can be shown on the optimal value, the problem becomes hopelessly hard from the parameterized point of view.

Other related work. Let us now give a brief survey of "price of generality" results involving our considered parameters, that is, results showing that a problem is efficient for one parameter but hard for a more general one. In this area, the results of Fomin et al. [35], introducing the term "price of generality", have been particularly impactful. This work and its follow-ups [36, 37], were the first to show that four natural graph problems (Coloring, Edge Dominating Set, Max Cut, Hamiltonicity) which are FPT for treewidth, become W[1]-hard for clique-width. In this sense, these problems, as well as problems discovered later such as counting perfect matchings [20], SAT [68, 23], $\exists\forall$-SAT [59], Orientable Deletion [45], and d-Regular Induced Subgraph [16], form part of the "price" we have to pay for considering a more general parameter. This line of research has thus helped to illuminate the complexity border between the two most important sparse and dense parameters (treewidth and clique-width), by giving a list of *natural* problems distinguishing the two. (An artificial MSO_2-expressible such problem was already known much earlier [19, 58]).

Let us now focus in the area below treewidth in Figure 1 by considering problems which are in XP but W[1]-hard parameterized by treewidth. By now, there is a small number of problems in this category which are known to be W[1]-hard even for vertex cover: List Coloring [31] was the first such problem, followed by CSP (for the vertex cover of the dual graph) [70], and more recently by (k, r)-Center, d-Scattered Set, and Min Power Steiner Tree [49, 48, 50] on weighted graphs. Intuitively, it is not surprising that problems W[1]-hard by vertex cover are few and far between, since this is a very restricted parameter.

Indeed, for most problems in the literature which are W[1]-hard by treewidth, vertex cover is the only parameter (among the ones considered here) for which the problem becomes FPT.

A second interesting category are problems which are FPT for tree-depth ([66]) but W[1]-hard for pathwidth. Mixed Chinese Postman Problem was the first discovered problem of this type [43], followed by Min Bounded-Length Cut [25, 10], ILP [40], Geodetic Set [51] and unweighted (k, r)-Center and d-Scattered Set [49, 48].

To the best of our knowledge, for all remaining problems which are known to be W[1]-hard by treewidth, the reductions that exist in the literature also establish W[1]-hardness for pathwidth. Below we give a (semi-exhaustive) list of problems which are known to be W[1]-hard by treewidth. After reviewing the relevant works we have verified that all of the following problems are in fact shown to be W[1]-hard parameterized by pathwidth (and in many case by feedback vertex set and tree-depth), even if this is not explicitly claimed.

1.1 Known problems which are W-hard for treewidth and for pathwidth

- Precoloring Extension and Equitable Coloring are shown to be W[1]-hard for both tree-depth and feedback vertex set in [31] (though the result is claimed only for treewidth). This is important, because Equitable Coloring often serves as a starting point for reductions to other problems. A second hardness proof for this problem was recently given in [22]. These two problems are FPT by vertex cover [33].
- Capacitated Dominating Set and Capacitated Vertex Cover are W[1]-hard for both tree-depth and feedback vertex set [24] (though again the result is claimed for treewidth).
- Min Maximum Out-degree on weighted graphs is W[1]-hard by tree-depth and feedback vertex set [72].
- General Factors is W[1]-hard by tree-depth and feedback vertex set [71].
- Target Set Selection is W[1]-hard by tree-depth and feedback vertex set [9] but FPT for vertex cover [67].
- Bounded Degree Deletion is W[1]-hard by tree-depth and feedback vertex set, but FPT for vertex cover [11, 39].
- Fair Vertex Cover is W[1]-hard by tree-depth and feedback vertex set [54].
- Fixing Corrupted Colorings is W[1]-hard by tree-depth and feedback vertex set [12] (reduction from Precoloring Extension).
- Max Node Disjoint Paths is W[1]-hard by tree-depth and feedback vertex set [29, 34].
- Defective Coloring is W[1]-hard by tree-depth and feedback vertex set [8].
- Power Vertex Cover is W[1]-hard by tree-depth but open for feedback vertex set [2].
- Majority CSP is W[1]-hard parameterized by the tree-depth of the incidence graph [23].
- List Hamiltonian Path is W[1]-hard for pathwidth [62].
- L(1,1)-Coloring is W[1]-hard for pathwidth, FPT for vertex cover [33].
- Counting Linear Extensions of a poset is W[1]-hard (under Turing reductions) for pathwidth [26].
- Equitable Connected Partition is W[1]-hard by pathwidth and feedback vertex set, FPT by vertex cover [28].
- Safe Set is W[1]-hard parameterized by pathwidth, FPT by vertex cover [7].
- Matching with Lower Quotas is W[1]-hard parameterized by pathwidth [4].
- Subgraph Isomorphism is W[1]-hard parameterized by the pathwidth of G, even when G, H are connected planar graphs of maximum degree 3 and H is a tree [61].
- Metric Dimension is W[1]-hard by pathwidth [15].
- Simple Comprehensive Activity Selection is W[1]-hard by pathwidth [27].

- Defensive Stackelberg Game for IGL is W[1]-hard by pathwidth (reduction from Equitable Coloring) [5].
- Directed (p, q)-Edge Dominating Set is W[1]-hard parameterized by pathwidth [6].
- Maximum Path Coloring is W[1]-hard for pathwidth [57].
- Unweighted k-Sparsest Cut is W[1]-hard parameterized by the three combined parameters tree-depth, feedback vertex set, and k [47].
- Graph Modularity is W[1]-hard parameterized by pathwidth plus feedback vertex set [63].

Let us also mention in passing that the algorithmic differences of pathwidth and treewidth may also be studied in the context of problems which are hard for constant treewidth. Such problems also generally remain hard for constant pathwidth (examples are Steiner Forest [42], Bandwidth [64], Minimum mcut [41]). One could also potentially try to distinguish between pathwidth and treewidth by considering the parameter dependence of a problem that is FPT for both. Indeed, for a long time the best-known algorithm for Dominating Set had complexity 3^k for pathwidth, but 4^k for treewidth. Nevertheless, the advent of fast subset convolution techniques [75], together with tight SETH-based lower bounds [60] has, for most problems, shown that the complexities on the two parameters coincide exactly.

Finally, let us mention a case where pathwidth and treewidth have been shown to be quite different in a sense similar to our framework. In [69] Razgon showed that a CNF can be compiled into an OBDD (Ordered Binary Decision Diagram) of size FPT in the pathwidth of its incidence graphs, but there exist formulas that always need OBDDs of size XP in the treewidth. Although this result does separate the two parameters, it is somewhat adjacent to what we are looking for, as it does not speak about the complexity of a decision problem, but rather shows that an OBDD-producing algorithm parameterized by treewidth would need XP time simply because it would have to produce a huge output in some cases.

2 Definitions and Preliminaries

For non-negative integers i, j, we use $[i, j]$ to denote the set $\{k \mid i \leq k \leq j\}$. Note that if $j < i$, then the set $[i, j]$ is empty. We will also write simply $[i]$ to denote the set $[1, i]$.

We give two equivalent definitions of our main problem.

▶ **Definition 1.** *A k-Grundy Coloring of a graph $G = (V, E)$ is a partition of V into k non-empty sets $V_1, \ldots, V_k$ such that: (i) for each $i \in [k]$ the set V_i induces an independent set; (ii) for each $i \in [k-1]$ the set V_i dominates the set $\bigcup_{i<j\leq k} V_j$.*

▶ **Definition 2.** *A k-Grundy Coloring of a graph $G = (V, E)$ is a proper k-coloring $c : V \to [k]$ that results by applying the First-Fit algorithm on an ordering of V; the First-Fit algorithm colors one by one the vertices in the given ordering, assigning to a vertex the minimum color that is not already assigned to one of its preceding neighbors.*

The Grundy number of a graph G, denoted by $\Gamma(G)$, is the maximum k such that G admits a k-Grundy Coloring. In a given Grundy Coloring, if $u \in V_i$ (equiv. if $c(u) = i$) we will say that u was given color i. The Grundy Coloring problem is the problem of determining the maximum k for which a graph G admits a k-Grundy Coloring. It is not hard to see that a proper coloring is a Grundy coloring if and only if every vertex assigned color i has at least one neighbor assigned color j, for each $j < i$.

3 W[1]-Hardness for Treewidth

In this section we prove that Grundy Coloring parameterized by treewidth is W[1]-hard (Theorem 16). Our proof relies on a reduction from k-Multi-Colored Clique and initially establishes W[1]-hardness for a more general problem where we are given a target color for a set of vertices (Lemma 8); we then reduce this to Grundy Coloring. Interestingly, this intermediate problem turns out to be W[1]-hard even for pathwidth (Lemma 12), since our reduction uses the standard strategy of constructing a grid-like structure of dimensions $k \times n$. The reason this reduction fails to prove that Grundy Coloring is W[1]-hard by pathwidth is that we use some gadgets to implement the targets and a support operation (which "pre-colors" some vertices) and for these gadgets we use trees of unbounded pathwidth. The results of Section 4 show that this is essential: our reduction *needs* some part that causes it to have high pathwidth, otherwise the Grundy number of the constructed graph would be bounded by the parameter, resulting in an instance that can be solved in FPT time.

Let us now present the different parts of our construction. We will make use of the structure of binomial trees T_i.

▶ **Definition 3.** *The binomial tree T_i with root r_i is a rooted tree defined recursively in the following way: T_1 consists simply of its root r_1; in order to construct T_i for $i > 1$, we construct one copy of T_j for all $j < i$ and a special vertex r_i, then we connect r_j with r_i. An alternative equivalent definition of the binomial tree T_i, $i \geq 2$ is that we construct two trees T_{i-1} , T'_{i-1}, we connect their roots r_{i-1}, r'_{i-1} and select one of them as the new root r_i.*

▶ **Proposition 4.** *Let $i \geq 2$, T_i be a binomial tree and $1 \leq t < i$. There exist 2^{i-t-1} binomial trees T_t which are vertex-disjoint and non-adjacent subtrees in T_i, where no T_t contains the root r_i of T_i.*

▶ **Proposition 5.** *$\Gamma(T_i) \leq i$. Furthermore, for all $j \leq i$ there exists a Grundy coloring which assigns color j to the root of T_i.*

The proofs of Propositions 4 and 5 can be found in the full version of this paper.

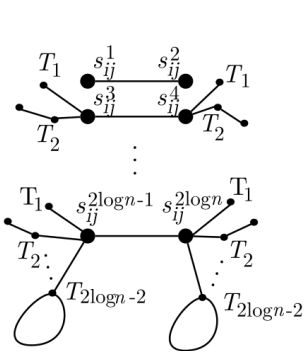
A Grundy coloring of T_i that assigns color i to r_i is called *optimal*. If r_i is assigned color $j < i$ then we call the Grundy coloring *sub-optimal*.

We now define a generalization of the Grundy coloring problem with target colors and show that it is W[1]-hard parameterized by treewidth. We later describe how to reduce this problem to Grundy Coloring such that the treewidth does not increase by a lot.

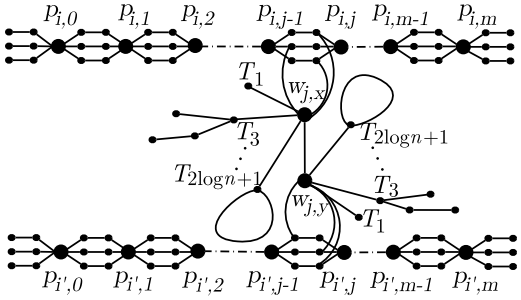
▶ **Definition 6** (Grundy Coloring with Targets)**.** *We are given a graph $G(V, E)$, an integer $t \in \mathbb{N}$ called the target and a subset $S \subset V$. (For simplicity we will say that vertices of S have target t.) If G admits a Grundy Coloring which assigns color t to some vertex $s \in S$ we say that, for this coloring, vertex s* achieves its target*. If there exists a Grundy Coloring of G which assigns to all vertices of S color t, then we say that G admits a* Target-achieving Grundy Coloring*. Grundy Coloring with Targets is the decision problem associated to the question "given G, S, t as defined above, does G admit a Target-achieving Grundy Coloring ?".*

We will also make use of the following operation:

▶ **Definition 7** (Tree-support)**.** *Given a graph $G = (V, E)$, a vertex $u \in V$ and a set N of positive integers, we define the* tree-support *operation as follows: (a) for all $i \in N$ we add a copy of T_i in the graph; (b) we connect u to the root r_i of each of the T_i. We say that we* add supports *N on u. The trees T_i will be called the* supporting trees *or* supports *of u. Slightly abusing notation, we also call* supports *the numbers $i \in N$.*



(a) Vertex Selection gadget $S_{i,j}$.



(b) Propagators $p_{i,j}$ and Edge Selection gadget W_j. The edge selection checkers and the supports of the $p_{i,j}$ and $s^l_{i,j}$ are not depicted. In the example $B_x = 010$ and $B_y = 100$.

Figure 2 The gadgets. Figure 2a is an enlargment of Figure 2b between $p_{i,j-1}$ and $p_{i,j}$.

Intuitively, the tree-support operation ensures that vertex u may have at least one neighbor of color i for each $i \in N$ in a Grundy coloring, and thus increase the color u can take. Observe that adding supporting trees to a vertex does not increase the treewidth, but does increase the pathwidth (binomial trees have unbounded pathwidth).

Our reduction is from k-Multi-Colored Clique, proven to be W[1]-hard in [32]: given a k-multipartite graph $G = (V_1, V_2, \ldots, V_k, E)$, decide if for every $i \in [k]$ we can pick $u_i \in V_i$ forming a clique, where k is the parameter. We can also assume that $\forall i \in [k], |V_i| = n$, that n is a power of 2, and that $V_i = \{v_{i,0}, v_{i,1}, \ldots, v_{i,n-1}\}$. Furthermore, let $|E| = m$. We construct an instance of Grundy Coloring with Targets $G' = (V', E')$ and $t = 2\log n + 4$ (where all logarithms are base two) using the following gadgets:

Vertex selection $S_{i,j}$. See Figure 2a. This gadget consists of $2\log n$ vertices $S^1_{i,j} \cup S^2_{i,j} = \bigcup_{l\in[\log n]}\{s^{2l-1}_{i,j}\} \cup \bigcup_{l\in[\log n]}\{s^{2l}_{i,j}\}$, where for each $l \in [\log n]$ we connect vertex $s^{2l-1}_{i,j}$ to $s^{2l}_{i,j}$ thus forming a matching. Furthermore, for each $l \in [2, \log n]$, we add supports $[2l-2]$ to vertices $s^{2l-1}_{i,j}$ and $s^{2l}_{i,j}$. Observe that the vertices $s^{2l-1}_{i,j}$ and $s^{2l}_{i,j}$ together with their supports form a binomial tree T_{2l} with either of these vertices as the root. We construct $k(m+2)$ gadgets $S_{i,j}$, one for each $i \in [k], j \in [0, m+1]$.

The vertex selection gadget $S_{i,1}$ encodes in binary the vertex that is selected in the clique from V_i. In particular, for each pair $s^{2l-1}_{i,1}, s^{2l}_{i,1}$, $l \in [\log n]$ either of these vertices can take the maximum color in an optimal grundy coloring of the binomial tree T_{2l} (that is, a coloring that gives the root of the binomial tree T_{2l} color $2l$). A selection corresponds to bit 0 or 1 for the l^{th} binary position. In order to ensure that for each $j \in [m]$ all (middle) $S_{i,j}$ encode the same vertex, we use propagators.

Propagators $p_{i,j}$. See Figure 2b. For $i \in [k]$ and $j \in [0, m]$, a propagator $p_{i,j}$ is a single vertex connected to all vertices of $S^2_{i,j} \cup S^1_{i,j+1}$. To each $p_{i,j}$, we also add supports $\{2\log n + 1, 2\log n + 2, 2\log n + 3\}$. The propagators have target $t = 2\log n + 4$.

Edge selection W_j. See Figure 2b. Let $j = (v_{i,x}, v_{i',y}) \in E$, where $v_{i,x} \in V_i$ and $v_{i',y} \in V_{i'}$. The gadget W_j consists of four vertices $w_{j,x}, w_{j,y}, w'_{j,x}, w'_{j,y}$. We call $w'_{j,x}, w'_{j,y}$ the *edge selection checkers*. We have the edges $(w_{j,x}, w_{j,y}), (w'_{j,x}, w_{j,x}), (w'_{j,y}, w_{j,y})$. Let us now describe the connections of these vertices with the rest of the graph. Let $B_x = b_1 b_2 \ldots b_{\log n}$ be the binary representation of x. We connect $w_{j,x}$ to each vertex $s^{2l-b_l}_{ij}$, $l \in [\log n]$ (we do similarly for $w_{j,y}, S_{i',j}$, and B_y). We add to each of $w_{j,x}, w_{j,y}$ supports $\bigcup_{l\in[\log n+1]}\{2l-1\}$. We add to each of $w'_{j,x}, w'_{j,y}$ supports $[2\log n + 3] \setminus \{2\log n + 1\}$ and set the target $t = 2\log n + 4$ for these two vertices. We construct m such gadgets, one for each edge. We say that W_j *is activated* if at least one of $w_{j,x}, w_{j,y}$ receives color $2\log n + 3$.

Edge validators $q_{i,i'}$. We construct $\binom{k}{2}$ of them, one for each pair $(i, i'), i < i' \in [k]$. The edge validator is a single vertex that is connected to all vertices $w_{j,x}$ for which j is an edge between V_i and $V_{i'}$. We add supports $[2\log n + 1]$ and a target of $t = 2\log n + 4$.

The edge validator plays the role of an "or" gadget: in order for it to achieve its target, at least one of its neighboring edge selection gadgets should be activated.

▶ **Lemma 8.** *G has a clique of size k if and only if G' has a target-achieving Grundy coloring.*

Proof. $\Rightarrow$) Suppose that G has a clique. We color the vertices of G' in the following order: First, we color the vertex selection gadget $S_{i,j}$. We start from the supports which we color optimally. We then color the matchings as follows: let $v_{i,x}$ be the vertex that was selected in the clique from V_i and $b_1 b_2 \dots b_{\log n}$ be the binary representation of x; we color vertices $s_{i,j}^{2l-(1-b_l)}$, $l \in [\log n]$ with color $2l-1$ and vertices $s_{i,j}^{2l-b_l}$, $l \in [\log n]$ will receive color $2l$. For the propagators, we color their supports optimally. Propagators have $2\log n + 3$ neighbors each, all with different colors, so they receive color $2\log n + 4$, thus achieving the targets.

Then, we color the edge validators $q_{i,i'}$ and the edge selection gadgets W_j that correspond to edges of the clique (that is, $j = (v_{i,x}, v_{i',y}) \in E$ and $v_{i,x} \in V_i$, $v_{i',y} \in V_{i'}$ are selected in the clique). We first color the supports of $q_{i,i'}, w_{j,x}, w_{j,y}$ optimally. From the construction, vertex $w_{j,x}$ is connected with vertices $s_{i,j}^{2l-b_l}$ which have already been colored $2l$, $l \in [\log n]$ and with supports $\bigcup_{l\in[\log n+1]}\{2l-1\}$, thus $w_{j,x}$ will receive color $2\log n + 2$. Similarly $w_{j,y}$ already has neighbors which are colored $[2\log n + 1]$, but also $w_{j,x}$, thus it will receive color $2\log n + 3$. These W_j will be activated. Since both $w_{j,x}, w_{j,y}$ connect to $q_{i,i'}$, the latter will be assigned color $2\log n + 4$, thus achieving its target. As for $w'_{j,x}$ and $w'_{j,y}$, these vertices have one neighbor colored c, where $c = 2\log n + 2$ or $c = 2\log n + 3$. We color their support T_c sub-optimally so that the root receives color $2\log n + 1$; we color their remaining supports optimally. This way, vertices $w'_{j,x}, w'_{j,y}$ can be assigned color $t = 2\log n + 4$, achieving the target.

Finally, for the remaining W_j, we claim that we can assign to both $w_{j,x}, w_{j,y}$ a color that is at least as high as $2\log n + 1$. Indeed, we assign to each supporting tree T_r of $w_{j,x}$ a coloring that gives its root the maximum color that is $\leq r$ and does not appear in any neighbor of $w_{j,x}$ in the vertex selection gadget. We claim that in this case $w_{j,x}$ will have neighbors with all colors in $[2\log n]$, because in every interval $[2l-1, 2l]$ for $l \in [\log n]$, $w_{j,x}$ has a neighbor with a color in that interval and a support tree T_{2l+1}. If $w_{j,x}$ has color $2\log n + 1$ then we color the supports of $w'_{j,x}$ optimally and achieve its target, while if $w_{j,x}$ has color higher than $2\log n + 1$, we achieve the target of $w'_{j,x}$ as in the previous paragraph.

$\Leftarrow$) Suppose that G' admits a coloring that achieves the target for all propagators, edge selection checkers, and edge validators. We will prove the following three claims:

▷ Claim 9. The coloring of the vertex selection gadgets is consistent throughout. This corresponds to a selection of k vertices of G.

▷ Claim 10. $\binom{k}{2}$ edge selection gadgets have been activated. That correspond to $\binom{k}{2}$ edges of G being selected.

▷ Claim 11. If an edge selection gadget $W_j = \{w_{j,x}, w_{j,y}\}$ with $j = (v_{i,x}, v_{i',y})$ has been activated then the coloring of the vertex selection gadgets $S_{i,j}$ and $S_{i',j}$ corresponds to the selection of vertices $v_{i,x}$ and $v_{i',y}$. In other words, selected vertices and edges form indeed a clique of size k in G.

Proof of Claim 9. Suppose that an edge selection checker $w'_{j,x}$ achieved its target. We claim that this implies that $w_{j,x}$ has color at least $2\log n+1$. Indeed, $w'_{j,x}$ has degree $2\log n+3$, so its neighbors must have all distinct colors in $[2\log n+3]$, but among the supports there are only 2 neighbors which may have colors in $[2\log n+1, 2\log n+3]$. Therefore, the missing color must come from $w_{j,x}$. We now observe that vertices from the vertex selection gadgets have color at most $2\log n$, because if we exclude from their neighbors the vertices $w_{j,x}$ (which we argued have color at least $2\log n+1$) and the propagators (which have target $2\log n+4$), these vertices have degree at most $2\log n-1$.

Suppose that a propagator $p_{i,j}$ achieves its target of $2\log n+4$. Since this vertex has a degree of $2\log n+3$, that means that all of its neighbors should receive all the colors in $[2\log n+3]$. As argued, colors $[2\log n+1, 2\log n+3]$ must come from the supports. Therefore, the colors $[2\log n]$ come from the neighbors of $p_{i,j}$ in the vertex selection gadgets.

We now note that, because of the degrees of vertices in vertex selection gadgets, only vertices $s_{i,j}^{2\log n}, s_{i,j+1}^{2\log n-1}$ can receive colors $2\log n, 2\log n-1$; from the rest, only $s_{i,j}^{2\log n-2}, s_{i,j+1}^{2\log n-3}$ can receive colors $2\log n-2, 2\log n-3$ etc. Thus, for each $l\in[\log n]$, if $s_{i,j}^{2l}$ receives color $2l-1$ then $s_{i,j+1}^{2l-1}$ should receive color $2l$ and vice versa. With similar reasoning, in all vertex selection gadgets we have that $s_{i,j}^{2l-1}, s_{i,j}^{2l}$ received the two colors $\{2l-1, 2l\}$ since they are neighbors. As a result, the colors of $s_{i,j+1}^{2l-1}$, $s_{i,j}^{2l-1}$ (and thus the colors of $s_{i,j+1}^{2l}$, $s_{i,j}^{2l}$) are the same, therefore, the coloring is consistent, for all values of $j\in[m]$. ◁

Proof of Claim 10. If an edge validator achieves its target of $2\log n+4$, then at least one of its neighbors from an edge selection gadget has received color $2\log n+3$. We know that each edge selection gadget only connects to a unique edge validator, so there should be $\binom{k}{2}$ edge selection gadgets which have been activated in order for all edge validators to achieve the target. ◁

Proof of Claim 11. Suppose that an edge validator $q_{i,i'}$ achieves its target. That means that there exists an edge selection gadget $W_j=\{w_{j,x}, w_{j,y}, w'_{j,x}, w'_{j,y}\}$ for which at least one of its vertices $\{w_{j,x}, w_{j,y}\}$, say vertex $w_{j,x}$, has received color $2\log n+3$. Let j be an edge connecting $v_{i,x}\in V_i$ to $v_{i',y}\in V_{i'}$. Since the degree of $w_{j,x}$ is $2\log n+4$ and we have already assumed that two of its neighbors ($q_{i,i'}$ and $w'_{j,x}$) have color $2\log n+4$, in order for it to receive color $2\log n+3$ all its other neighbors should receive all colors in $[2\log n+2]$. The only possible assignment is to give colors $2l$, $l\in[\log n]$ to its neighbors from $S_{i,j}$ and color $2\log n+2$ to $w_{j,y}$. The latter is, in turn, only possible if the neighbors of $w_{j,y}$ from $S_{i',j}$ receive all colors $2l$, $l\in[\log n]$. The above corresponds to selecting vertex $v_{i,x}$ from V_i and $v_{i',y}$ from $V_{i'}$. ◁

◀

▶ **Lemma 12.** *Let G'' be the graph that results from G' if we remove all the tree-supports. Then G'' has pathwidth at most $\binom{k}{2}+2k+3$.*

The proof of Lemma 12 can be found in the full version of the paper.

We will now show how to implement the targets using the tree-filling operation below.

▶ **Definition 13** (Tree-filling). *Let $G=(V,E)$ be a graph and $S=\{s_1,s_2,\ldots,s_j\}\subset V$ a set of vertices with target t. The* tree-filling *operation is the following. First, we add in G a binomial tree T_i, where $i=\lceil\log j\rceil+t+1$. Observe that, by Proposition 4, there exist $2^{i-t-1}>j$ vertex-disjoint and non-adjacent sub-trees T_t in T_i. For each $s\in S$, we find such a copy of T_t in T_i, identify s with its root r_t, and delete all other vertices of the sub-tree T_t.*

The tree-filling operation might in general increase treewidth, but we will do it in a way that it only increases by a constant factor in regards to the pathwidth of G.

▶ **Lemma 14.** *Let $G = (V, E)$ be a graph of pathwidth w and $S = \{s_1, \ldots, s_j\} \subset V$ a subset of vertices having target t. Then there is a way to apply the tree-filling operation such that the resulting graph H has $tw(H) \leq 4w + 5$.*

Proof. Construction of H. Let $(\mathcal{P}, \mathcal{B})$ be a path-decomposition of G whose largest bag has size $w + 1$ and $B_1, B_2, \ldots, B_j \in \mathcal{B}$ distinct bags where $\forall a, s_a \in B_a$ (assigning a distinct bag to each s_a is always possible, as we can duplicate bags if necessary). We call those bags *important*. We define an ordering $o : S \to \mathbb{N}$ of the vertices of S that follows the order of the important bags from left to right, that is $o(s_a) < o(s_b)$ if B_a is on the left of B_b in $\mathcal{P}$. For simplicity, let us assume that $o(s_a) = a$ and that B_a is to the left of B_b if $a < b$.

We describe a recursive way to do the substitution of the trees in the tree-filling operation. Crucially, when $j > 2$ we will have to select an appropriate mapping between the vertices of S and the disjoint subtrees T_t in the added binomial tree T_i, so that we will be able to keep the treewidth of the new graph bounded.

- If $j = 1$ then $i = t + 1$. We add to the graph a copy of T_i, arbitrarily select the root of a copy of T_t contained in T_i, and perform the tree-filling operation as described.
- Suppose that we know how to perform the substitution for sets of size at most $\lceil j/2 \rceil$, we will describe the substitution process for a set of size j. We have $i = \lceil \log j \rceil + t + 1$ and for all j we have $\lceil \log \lceil j/2 \rceil \rceil = \lceil \log j \rceil - 1$. Split the set S into two (almost) equal disjoint sets S^L and S^R of size at most $\lceil j/2 \rceil$, where for all $s_a \in S^L$ and for all $s_b \in S^R$, $a < b$. We perform the tree-filling on each of these sets by constructing two binomial trees T^L_{i-1}, T^R_{i-1} and doing the substitution; then, we connect their roots and set the root of the left tree as the root r_i of T_i, thus creating the substitution of a tree T_i.

Small treewidth. We now prove that the new graph H that results from applying the tree-filling operation on G and S as described above has a tree decomposition $(\mathcal{T}, \mathcal{B}')$ of width $4w + 5$; in fact we prove by induction on j a stronger statement: if $A, Z \in \mathcal{B}$ are the left-most and right-most bags of $\mathcal{P}$, then there exists a tree decomposition $(\mathcal{T}, \mathcal{B}')$ of H of width $4w + 5$ with the added property that there exists $R \in \mathcal{B}'$ such that $A \cup Z \cup \{r_i\} \subset R$, where r_i is the root of the tree T_i.

For the base case, if $j = 1$ we have added to our graph a T_i of which we have selected an arbitrary sub-tree T_t, and identified the root r_t of T_t with the unique vertex of S that has a target. Take the path decomposition $(\mathcal{P}, \mathcal{B})$ of the initial graph and add all vertices of A (its first bag) and the vertex r_i (the root of T_i) to all bags. Take an optimal tree decomposition of T_i of width 1 and add r_i to each bag, obtaining a decomposition of width 2. We add an edge between the bag of $\mathcal{P}$ that contains the unique vertex of S, and a bag of the decomposition of T_i that contains the selected r_t. We now have a tree decomposition of the new graph of width $2w + 2 < 4w + 5$. Observe that the last bag of $\mathcal{P}$ now contains all of A, Z and r_i.

For the inductive step, suppose we applied the tree-filling operation for a set S of size $j > 1$. Furthermore, suppose we know how to construct a tree decomposition with the desired properties (width $4w + 5$, one bag contains the first and last bags of the path decomposition $\mathcal{P}$ and r_i), if we apply the tree-filling operation on a target set of size at most $j - 1$. We show how to obtain a tree decompostition with the desired properties if the target set has size j.

By construction, we have split the set S into two sets S^L, S^R and have applied the tree-filling operation to each set separately. Then, we connected the roots of the two added trees to obtain a larger binomial tree. Observe that for $|S| = j > 1$ we have $|S^L|, |S^R| < j$.

Let us first cut $\mathcal{P}$ in two parts, in such a way that the important bags of S^L are on the left and the important bags of S^R are on the right. We call $A^L = A$ and Z^L the leftmost and rightmost bags of the left part and A^R, $Z^R = Z$ the leftmost and rightmost bags of the right part. We define as G^L (respectively G^R) the graph that contains all the vertices of the left (respectively right) part. Let r_i be the root of T_i and r_{i-1} the root of its subtree T_{i-1}. From the inductive hypothesis, we can construct tree decompositions $(\mathcal{T}^{\mathcal{L}}, \mathcal{B}^{\mathcal{L}})$, $(\mathcal{T}^{\mathcal{R}}, \mathcal{B}^{\mathcal{R}})$ of width $4w + 5$ for the graphs H^L, H^R that occur after applying tree-filling on G^L, S^L and G^R, S^R; furthermore, there exist $R^L \in \mathcal{B}^{\mathcal{L}}, R^R \in \mathcal{B}^{\mathcal{R}}$ such that $R^L \supseteq A \cup Z^L \cup \{r_i\}$ and $R^R \supseteq A^R \cup Z \cup \{r_{i-1}\}$.

We construct a new bag $R' = A \cup A^R \cup Z^L \cup Z \cup \{r_{i-1}, r_i\}$, and we connect R' to both R^L and R^R, thus combining the two tree-decompositions into one. Last we create a bag $R = A \cup Z \cup \{r_i\}$ and attach it to R'. This completes the construction of $(\mathcal{T}, \mathcal{B}')$.

Observe that $(\mathcal{T}, \mathcal{B}')$ is a valid tree-decomposition for H:

- $V(H) = V(H^L) \cup V(H^R)$, thus $\forall v \in V(H), v \in \mathcal{B}^{\mathcal{L}} \cup \mathcal{B}^{\mathcal{R}} \subset \mathcal{B}$.
- $E(H) = E(H^L) \cup E(H^R) \cup \{(r_{i-1}, r_i)\}$. We have that $r_{i-1}, r_i \in R' \in \mathcal{B}$. All other edges were dealt with in $\mathcal{T}^{\mathcal{L}}, \mathcal{T}^{\mathcal{R}}$.
- Each vertex $v \in V(H)$ that belongs in exactly one of H^L, H^R trivially satisfied the connectivity requirement: bags that contain v are either fully contained in $\mathcal{T}^{\mathcal{L}}$ or $\mathcal{T}^{\mathcal{R}}$. A vertex v that is in both H^L and H^R is also in $Z^L \cap A^R$ due to the properties of path-decompositions, hence in R'. Therefore, the sub-trees of bags that contain v in $\mathcal{T}^{\mathcal{L}}, \mathcal{T}^{\mathcal{R}}$, form a connected sub-tree in $\mathcal{T}$.

The width of $\mathcal{T}$ is $\max\{tw(H^L), tw(H^R), |R'| - 1\} = 4w + 5$. ◀

The last thing that remains to do in order to complete the proof is to show the equivalence between achieving the targets and finding a Grundy coloring.

▶ **Lemma 15.** *Let G and G' be two graphs as described in Lemma 8 and let H be constructed from G' by using the tree-filling operation. Then G has a clique of size k iff $\Gamma(H) \geq \lceil \log(k(m+1) + \binom{k}{2} + 2m) \rceil + 2\log n + 5$. Furthermore, $tw(H) \leq 4\binom{k}{2} + 8k + 17$.*

The proof of Lemma 15 can be found in the full version of the paper.

▶ **Theorem 16.** *Grundy Coloring parameterized by treewidth is W[1]-hard.*

4 FPT for pathwidth

In this section, we show that, in contrast to treewidth, Grundy Coloring is FPT parameterized by pathwidth. We achieve this by providing an upper bound on the Grundy number of any graph as a function of its pathwidth. Pipelining this with the algorithm of [74], we obtain a dependency on pathwidth alone. In order to obtain our bound, we rely on the following result on the performance ratio of the first-fit coloring algorithm on interval graphs.

▶ **Theorem 17** ([65]). *First-Fit is 8-competitive for online coloring interval graphs.*

In other words, interval graphs satisfy $\Gamma(G) \leq 8 \cdot \chi(G)$. Since for any interval graph G we have $\chi(G) = pw(G) + 1$, we immediately obtain the following:

▶ **Corollary 18.** *For every interval graph G, $\Gamma(G) \leq 8 \cdot (pw(G) + 1)$.*

▶ **Lemma 19.** *For every graph G, $\Gamma(G) \leq 8 \cdot (pw(G) + 1)$.*

Proof. For a contradiction, suppose there exists G such that $\Gamma(G) > 8 \cdot (pw(G)+1)$, and let $c : V(G) \to \{1, \ldots, \Gamma(G)\}$ be a Grundy coloring using $\Gamma(G)$ colors. In addition, let G have the smallest possible number of vertices, i.e., there is no G' satisfying those conditions with $|V(G')| < |V(G)|$. This implies that, for every optimal path decomposition of G, there is no bag B and vertices $u, v \in B$ such that $c(u) = c(v)$.

Indeed, if such vertices exist, adding the edge uv to G and contracting uv yields a new graph G' such that $pw(G') \leq pw(G)$ (edge contraction does not increase the pathwidth), $\Gamma(G') \geq \Gamma(G)$ (since c when limited to V(G') is a valid Grundy coloring of G') and $|V(G')| < |V(G)|$, contradicting the assumption that G is smallest possible.

In addition, for any u, v such that $c(u) \neq c(v)$ and $v \notin N(u)$, adding edge uv to G does not decrease the Grundy number of G since c remains a valid Grundy coloring of the new graph. In particular, since, as previously observed, vertices in any bag of an optimal path decomposition of G all have pairwise different colors, turning every bag of such a decomposition into a clique does not decrease the Grundy number of G. More precisely, this yields a graph G' such that $pw(G') = pw(G)$ and $\Gamma(G') \geq \Gamma(G)$, where G' is an interval graph. Applying Corollary 18 we obtain $\Gamma(G) \leq \Gamma(G') \leq 8 \cdot (pw(G')+1)$, contradiction. ◀

Combining Lemma 19 with the $O^*(2^{O(tw(G)\cdot\Gamma(G))})$ algorithm of [74], we have:

▶ **Theorem 20.** *Grundy Coloring can be solved in time* $O^*(2^{O(pw(G)^2)})$.

Finally, note that there exist interval graphs that satisfy $\Gamma(G) \geq r \cdot pw(G)$, for any $r < 5$ [53], therefore, the constant in Lemma 19 cannot be improved below 5.

5 NP-hardness for Constant Clique-width

In this section we prove that Grundy Coloring is NP-hard even for constant clique-width via a reduction from 3-SAT. We use a similar idea of adding supports as in Section 3, but supports now will be cliques instead of binomial trees. The support operation is defined as:

▶ **Definition 21.** *Given a graph $G = (V, E)$, a vertex $u \in V$ and a set of positive integers S, we define the **support** operation as follows: for each $i \in S$, we add to G a clique of size i (using new vertices) and we connect one arbitrary vertex of each such clique to u.*

When applying the support operation we will say that we support vertex u with set S and we will call the vertices introduced supporting vertices. Intuitively, the support operation ensures that the vertex u may have at least one neighbor with color i for each $i \in S$.

We are now ready to describe our construction. Suppose we are given a 3CNF formula ϕ with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. We assume without loss of generality that each clause contains exactly three variables. We construct a graph $G(\phi)$ as follows:

1. For each $i \in [n]$ we construct two vertices x_i^P, x_i^N and the edge (x_i^P, x_i^N).
2. For each $i \in [n]$ we support the vertices x_i^P, x_i^N with the set $[2i-2]$. (Note that x_1^P, x_1^N have empty support).
3. For each $i \in [n], j \in [m]$, if variable x_i appears in clause c_j then we construct a vertex $x_{i,j}$. Furthermore, if x_i appears positive in c_j, we connect $x_{i,j}$ to $x_{i'}^P$ for all $i' \in [n]$; otherwise we connect $x_{i,j}$ to $x_{i'}^N$ for all $i' \in [n]$.
4. For each $i \in [n], j \in [m]$ for which we constructed a vertex $x_{i,j}$ in the previous step, we support that vertex with the set $(\{2k \mid k \in [n]\} \cup \{2i-1, 2n+1, 2n+2\}) \setminus \{2i\}$.
5. For each $j \in [m]$ we construct a vertex c_j and connect to all (three) vertices $x_{i,j}$ already constructed. We support the vertex c_j with the set $[2n]$.

6. For each $j \in [m]$ we construct a vertex d_j and connect it to c_j. We support d_j with the set $[2n+3] \cup [2n+5, 2n+3+j]$.
7. We construct a vertex u and connect it to d_j for all $j \in [m]$. We support u with the set $[2n+4] \cup [2n+5+m, 10n+10m]$.

This completes the construction. Before we proceed, let us give some intuition. Observe that we have constructed two vertices x_i^P, x_i^N for each variable. The support of these vertices and the fact that they are adjacent, allow us to give them colors $\{2i-1, 2i\}$. The choice of which gets the higher color encodes an assignment to variable x_i. The vertices $x_{i,j}$ are now supported in such a way that they can "ignore" the values of all variables except x_i; for x_i, however, $x_{i,j}$ "prefers" to be connected to a vertex with color $2i$ (since $2i-1$ appears in the support of $x_{i,j}$, but $2i$ does not). Now, the idea is that c_j will be able to get color $2n+4$ if and only if one of its literal vertices $x_{i,j}$ was "satisfied" (has a neighbor with color $2i$). The rest of the construction checks if all clause vertices are satisfied in this way.

We now state the lemmata that certify the correctness of our reduction. Their proofs appear in the full version of the paper.

▶ **Lemma 22.** *If ϕ is satisfiable then $G(\phi)$ has a Grundy coloring with $10n+10m+1$ colors.*

▶ **Lemma 23.** *If $G(\phi)$ has a Grundy coloring with $10n+10m+1$ colors, then ϕ is satisfiable.*

▶ **Lemma 24.** *The graph $G(\phi)$ has constant clique-width.*

▶ **Theorem 25.** *Given graph $G=(V,E)$, k-Grundy Coloring is NP-hard even when the clique-width of the graph $cw(G)$ is a constant.*

6 FPT for modular-width

In this section we show that Grundy Coloring is FPT parameterized by modular-width. Recall that $G=(V,E)$ has modular-width w if V can be partitioned into at most w modules, such that each module is a singleton or induces a graph of modular-width w. Neighborhood diversity is the restricted version of this measure where modules are required to be cliques or independent sets. We sketch the main ideas of the algorithm (a full proof is in the full version of the paper).

The first step is to show that Grundy Coloring is FPT parameterized by neighborhood diversity. Similarly to the standard Coloring algorithm for this parameter [56], we observe that, without loss of generality, all modules can be assumed to be cliques, and hence any color class has one of 2^w possible types. We would like to use this to reduce the problem to an ILP with 2^w variables, but unlike Coloring, the ordering of color classes matters. We thus prove that the optimal solution can be assumed to have a "canonical" structure where each color type only appears in consecutive colors. We then extend the neighborhood diversity algorithm to modular-width using the idea that we can calculate the Grundy number of each module separately, and then replace it with an appropriately-sized clique.

▶ **Theorem 26.** *Let $G=(V,E)$ be a graph of modular-width w. The Grundy number of G can be computed in time $2^{O(w2^w)}n^{O(1)}$.*

References

1 Pierre Aboulker, Édouard Bonnet, Eun Jung Kim, and Florian Sikora. Grundy coloring & friends, half-graphs, bicliques. In *37th Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2020.

2 Eric Angel, Evripidis Bampis, Bruno Escoffier, and Michael Lampis. Parameterized power vertex cover. *Discrete Mathematics & Theoretical Computer Science*, 20(2), 2018. URL: http://dmtcs.episciences.org/4873.

3 Júlio Araújo and Cláudia Linhares Sales. On the grundy number of graphs with few p_4's. *Discrete Applied Mathematics*, 160(18):2514–2522, 2012. doi:10.1016/j.dam.2011.08.016.

4 Ashwin Arulselvan, Ágnes Cseh, Martin Groß, David F. Manlove, and Jannik Matuschke. Matchings with lower quotas: Algorithms and complexity. *Algorithmica*, 80(1):185–208, 2018. doi:10.1007/s00453-016-0252-6.

5 Haris Aziz, Serge Gaspers, Edward J. Lee, and Kamran Najeebullah. Defender stackelberg game with inverse geodesic length as utility metric. In Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar, editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 694–702. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018. URL: http://dl.acm.org/citation.cfm?id=3237486.

6 Rémy Belmonte, Tesshu Hanaka, Ioannis Katsikarelis, Eun Jung Kim, and Michael Lampis. New results on directed edge dominating set. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 67:1–67:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.67.

7 Rémy Belmonte, Tesshu Hanaka, Ioannis Katsikarelis, Michael Lampis, Hirotaka Ono, and Yota Otachi. Parameterized complexity of safe set. In Pinar Heggernes, editor, *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*, volume 11485 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2019. doi:10.1007/978-3-030-17402-6_4.

8 Rémy Belmonte, Michael Lampis, and Valia Mitsou. Parameterized (approximate) defective coloring. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 10:1–10:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.10.

9 Oren Ben-Zwi, Danny Hermelin, Daniel Lokshtanov, and Ilan Newman. Treewidth governs the complexity of target set selection. *Discrete Optimization*, 8(1):87–96, 2011. doi:10.1016/j.disopt.2010.09.007.

10 Matthias Bentert, Klaus Heeger, and Dušan Knop. Length-bounded cuts: Proper interval graphs and structural parameters, 2019. arXiv:1910.03409.

11 Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1-2):53–60, 2012. doi:10.1016/j.dam.2011.08.013.

12 Marzio De Biasi and Juho Lauri. On the complexity of restoring corrupted colorings. *J. Comb. Optim.*, 37(4):1150–1169, 2019. doi:10.1007/s10878-018-0342-2.

13 Édouard Bonnet, Florent Foucaud, Eun Jung Kim, and Florian Sikora. Complexity of grundy coloring and its variants. *Discrete Applied Mathematics*, 243:99–114, 2018. doi:10.1016/j.dam.2017.12.022.

14 Édouard Bonnet, Michael Lampis, and Vangelis Th. Paschos. Time-approximation trade-offs for inapproximable problems. *J. Comput. Syst. Sci.*, 92:171–180, 2018. doi:10.1016/j.jcss.2017.09.009.

15 Édouard Bonnet and Nidhi Purohit. Metric dimension parameterized by treewidth. *CoRR*, abs/1907.08093, 2019. arXiv:1907.08093.

16 Hajo Broersma, Petr A. Golovach, and Viresh Patel. Tight complexity bounds for FPT subgraph problems parameterized by the clique-width. *Theor. Comput. Sci.*, 485:69–84, 2013. doi:10.1016/j.tcs.2013.03.008.

17 Claude A. Christen and Stanley M. Selkow. Some perfect coloring properties of graphs. *J. Comb. Theory, Ser. B*, 27(1):49–59, 1979. doi:10.1016/0095-8956(79)90067-4.

18 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

19 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

20 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

21 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

22 Guilherme de C. M. Gomes, Carlos V. G. C. Lima, and Vinícius Fernandes dos Santos. Parameterized complexity of equitable coloring. *Discrete Mathematics & Theoretical Computer Science*, 21(1), 2019. URL: http://dmtcs.episciences.org/5464.

23 Holger Dell, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Tobias Mömke. Complexity and approximability of parameterized max-csps. *Algorithmica*, 79(1):230–250, 2017. doi:10.1007/s00453-017-0310-8.

24 Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Capacitated domination and covering: A parameterized perspective. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 78–90. Springer, 2008. doi:10.1007/978-3-540-79723-4_9.

25 Pavel Dvorák and Dusan Knop. Parameterized complexity of length-bounded cuts and multicuts. *Algorithmica*, 80(12):3597–3617, 2018. doi:10.1007/s00453-018-0408-7.

26 Eduard Eiben, Robert Ganian, K. Kangas, and Sebastian Ordyniak. Counting linear extensions: Parameterizations by treewidth. *Algorithmica*, 81(4):1657–1683, 2019. doi:10.1007/s00453-018-0496-4.

27 Eduard Eiben, Robert Ganian, and Sebastian Ordyniak. A structural approach to activity selection. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 203–209. ijcai.org, 2018. doi:10.24963/ijcai.2018/28.

28 Rosa Enciso, Michael R. Fellows, Jiong Guo, Iyad A. Kanj, Frances A. Rosamond, and Ondrej Suchý. What makes equitable connected partition easy. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2009. doi:10.1007/978-3-642-11269-0_10.

29 Alina Ene, Matthias Mnich, Marcin Pilipczuk, and Andrej Risteski. On routing disjoint paths in bounded treewidth graphs. In *SWAT*, volume 53 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

30 Paul Erdös, Stephen T. Hedetniemi, Renu C. Laskar, and Geert C. E. Prins. On the equality of the partial grundy and upper ochromatic numbers of graphs. *Discrete Mathematics*, 272(1):53–64, 2003. doi:10.1016/S0012-365X(03)00184-5.

31 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011. doi:10.1016/j.ic.2010.11.026.

32 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

33 Jirí Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theor. Comput. Sci.*, 412(23):2513–2523, 2011. doi:10.1016/j.tcs.2010.10.043.

34 Krzysztof Fleszar, Matthias Mnich, and Joachim Spoerhase. New algorithms for maximum disjoint paths based on tree-likeness. In *ESA*, volume 57 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

35 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Algorithmic lower bounds for problems parameterized with clique-width. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 493–502. SIAM, 2010. doi:10.1137/1.9781611973075.42.

36 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

37 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019. doi:10.1145/3280824.

38 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

39 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.33.

40 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artif. Intell.*, 257:61–71, 2018. doi:10.1016/j.artint.2017.12.006.

41 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997. doi:10.1007/BF02523685.

42 Elisabeth Gassner. The steiner forest problem revisited. *J. Discrete Algorithms*, 8(2):154–163, 2010. doi:10.1016/j.jda.2009.05.002.

43 Gregory Z. Gutin, Mark Jones, and Magnus Wahlström. The mixed chinese postman problem parameterized by pathwidth and treedepth. *SIAM J. Discrete Math.*, 30(4):2177–2205, 2016. doi:10.1137/15M1034337.

44 András Gyárfás and Jenö Lehel. On-line and first fit colorings of graphs. *Journal of Graph Theory*, 12(2):217–227, 1988. doi:10.1002/jgt.3190120212.

45 Tesshu Hanaka, Ioannis Katsikarelis, Michael Lampis, Yota Otachi, and Florian Sikora. Parameterized orientable deletion. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018, June 18-20, 2018, Malmö, Sweden*, volume 101 of *LIPIcs*, pages 24:1–24:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.SWAT.2018.24.

46 Frédéric Havet and Leonardo Sampaio. On the grundy and b-chromatic numbers of a graph. *Algorithmica*, 65(4):885–899, 2013. doi:10.1007/s00453-011-9604-4.

47 Ramin Javadi and Amir Nikabadi. On the parameterized complexity of sparsest cut and small-set expansion problems, 2019. arXiv:1910.12353.

48 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159 of *Lecture Notes in Computer Science*, pages 292–305. Springer, 2018. doi:10.1007/978-3-030-00256-5_24.

49 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discrete Applied Mathematics*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

50 Chamanvir Kaur and Neeldhara Misra. On the parameterized complexity of spanning trees with small vertex covers. In *CALDAM*, volume 12016 of *Lecture Notes in Computer Science*, pages 427–438. Springer, 2020.

51 Leon Kellerhals and Tomohiro Koana. Parameterized complexity of geodetic set, 2020. arXiv:2001.03098.

52 Hal A. Kierstead and Karin Rebecca Saoub. First-fit coloring of bounded tolerance graphs. *Discrete Applied Mathematics*, 159(7):605–611, 2011. doi:10.1016/j.dam.2010.05.002.

53 Hal A. Kierstead, David A. Smith, and William T. Trotter. First-fit coloring on interval graphs has performance ratio at least 5. *Eur. J. Comb.*, 51:236–254, 2016. doi:10.1016/j.ejc.2015.05.015.

54 Dusan Knop, Tomás Masarík, and Tomás Toufar. Parameterized complexity of fair vertex evaluation problems. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany.*, volume 138 of *LIPIcs*, pages 33:1–33:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.33.

55 Guy Kortsarz. A lower bound for approximating grundy numbering. *Discrete Mathematics & Theoretical Computer Science*, 9(1), 2007. URL: http://dmtcs.episciences.org/391.

56 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.

57 Michael Lampis. Parameterized maximum path coloring. *Theor. Comput. Sci.*, 511:42–53, 2013. doi:10.1016/j.tcs.2013.01.012.

58 Michael Lampis. Model checking lower bounds for simple graphs. *Logical Methods in Computer Science*, 10(1), 2014. doi:10.2168/LMCS-10(1:18)2014.

59 Michael Lampis and Valia Mitsou. Treewidth with a quantifier alternation revisited. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.26.

60 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

61 Dániel Marx and Michal Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 542–553. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.542.

62 Kitty Meeks and Alexander Scott. The parameterised complexity of list problems on graphs of bounded treewidth. *Inf. Comput.*, 251:91–103, 2016. doi:10.1016/j.ic.2016.08.001.

63 Kitty Meeks and Fiona Skerman. The parameterised complexity of computing the maximum modularity of a graph. In *IPEC*, volume 115 of *LIPIcs*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

64 Burkhard Monien. The bandwidth minimization problem for caterpillars with hair length 3 is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 7(4):505–512, 1986.

65 N. S. Narayanaswamy and R. Subhash Babu. A note on first-fit coloring of interval graphs. *Order*, 25(1):49–53, 2008. doi:10.1007/s11083-008-9076-6.

66 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

67 André Nichterlein, Rolf Niedermeier, Johannes Uhlmann, and Mathias Weller. On tractable cases of target set selection. *Social Netw. Analys. Mining*, 3(2):233–256, 2013. doi:10.1007/s13278-012-0067-7.

68 Sebastian Ordyniak, Daniël Paulusma, and Stefan Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theor. Comput. Sci.*, 481:85–99, 2013. doi:10.1016/j.tcs.2012.12.039.

69 Igor Razgon. On obdds for cnfs of bounded treewidth. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7982.

70 Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010. doi:10.1016/j.jcss.2009.04.003.

71 Marko Samer and Stefan Szeider. Tractable cases of the extended global cardinality constraint. *Constraints*, 16(1):1–24, 2011. doi:10.1007/s10601-009-9079-y.

72 Stefan Szeider. Not so easy problems for tree decomposable graphs. *CoRR*, abs/1107.1177, 2011. arXiv:1107.1177.

73 Zixing Tang, Baoyindureng Wu, Lin Hu, and Manouchehr Zaker. More bounds for the grundy number of graphs. *J. Comb. Optim.*, 33(2):580–589, 2017. doi:10.1007/s10878-015-9981-8.

74 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math.*, 10(4):529–550, 1997. doi:10.1137/S0895480194275825.

75 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

76 Manouchehr Zaker. Grundy chromatic number of the complement of bipartite graphs. *Australasian J. Combinatorics*, 31:325–330, 2005. URL: http://ajc.maths.uq.edu.au/pdf/31/ajc_v31_p325.pdf.

77 Manouchehr Zaker. Results on the grundy chromatic number of graphs. *Discrete Mathematics*, 306(23):3166–3173, 2006. doi:10.1016/j.disc.2005.06.044.

78 Manouchehr Zaker. Inequalities for the grundy chromatic number of graphs. *Discrete Applied Mathematics*, 155(18):2567–2572, 2007. doi:10.1016/j.dam.2007.07.002.

On the Complexity of BWT-Runs Minimization via Alphabet Reordering

Jason W. Bentley
Department of Mathematics, University of Central Florida, Orlando, FL, USA
jason.bentley@ucf.edu

Daniel Gibney
Department of Computer Science, University of Central Florida, Orlando, FL, USA
https://www.cs.ucf.edu/~dgibney/
daniel.j.gibney@gmail.com

Sharma V. Thankachan
Department of Computer Science, University of Central Florida, Orlando, FL, USA
http://www.cs.ucf.edu/~sharma/
sharma.thankachan@ucf.edu

Abstract

The Burrows-Wheeler Transform (BWT) has been an essential tool in text compression and indexing. First introduced in 1994, it went on to provide the backbone for the first encoding of the classic suffix tree data structure in space close to entropy-based lower bound. Within the last decade, it has seen its role further enhanced with the development of compact suffix trees in space proportional to "r", the number of runs in the BWT. While r would superficially appear to be only a measure of space complexity, it is actually appearing increasingly often in the time complexity of new algorithms as well. This makes having the smallest value of r of growing importance. Interestingly, unlike other popular measures of compression, the parameter r is sensitive to the lexicographic ordering given to the text's alphabet. Despite several past attempts to exploit this fact, a provably efficient algorithm for finding, or approximating, an alphabet ordering which minimizes r has been open for years.

We help to explain this lack of progress by presenting the first set of results on the computational complexity of minimizing BWT-runs via alphabet reordering. We prove that the decision version of this problem is NP-complete and cannot be solved in time $poly(n) \cdot 2^{o(\sigma)}$ unless the Exponential Time Hypothesis fails, where σ is the size of the alphabet and n is the length of the text. Moreover, we show that the optimization variant is APX-hard. In doing so, we relate two previously disparate topics: the optimal traveling salesperson path of a graph and the number of runs in the BWT of a text. In addition, by relating recent results in the field of dictionary compression, we illustrate that an arbitrary alphabet ordering provides an $O(\log^2 n)$-approximation. Lastly, we provide an optimal linear-time algorithm for a more restricted problem of finding an optimal ordering on a subset of symbols (occurring only once) under ordering constraints.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

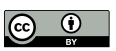
Keywords and phrases BWT, NP-hardness, APX-hardness

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.15

Related Version A full version of the paper is available at https://arxiv.org/abs/1911.03035.

Funding Supported in part by the U.S. National Science Foundation (NSF) under CCF-1703489.

Acknowledgements We would like to thank the reviewers for their valuable feedback and Chandra Chekuri for his helpful correspondence.



1 Introduction and Related Work

The Burrows-Wheeler Transform (BWT) is an essential building block in the fields of text compression and indexing with a myriad of applications in bioinformatics and information retrieval [24, 25, 26, 30]. Since it first arose in 1994 [6], it has been utilized to provide the popular compression algorithm bzip2 and has been adapted to provide powerful compressed text indexing data structures, such as the FM-index [12]. Hence, improvements to the algorithmic aspects of this transformation and related data structures can have a significant impact on the research community.

The BWT of a text $T[1, n]$, denoted by $BWT(T)$ is a reversible transformation which can be defined as follows: sort the circular shifts of T in lexicographical order and place the sorted circular shifts in a matrix. By reading the last column of this matrix from top to bottom we obtain $BWT(T)$. To make the transformation invertible a new symbol \$ (lexicographically smaller than others) is appended to T prior to sorting the circular shifts. See Figure 1 for an example. Historically, the BWT was introduced for the purpose of text compression [6], where its effectiveness is based on symbols with shared preceding context forming long *runs* (maximal unary substrings).

F		L
\$	mississipp	i
i	\$mississip	p
i	ppi\$missis	s
i	ssippi\$mis	s
i	ssissippi\$	m
m	ississippi	\$
p	i\$mississi	p
p	pi\$mississ	i
s	ippi\$missi	s
s	issippi\$mi	s
s	sippi\$miss	i
s	sissippi\$m	i

Figure 1 Column L shows the BWT of *mississippi*. The number of runs $r = 9$.

Recently, the number of runs "r" in the BWT has become of increasing interest. This can be attributed to the fact that many modern text collections are highly repetitive, which makes their compression effective via the BWT followed by Run-Length encoding (i.e., in space proportional to r). This raised an interesting question: can we also index the text in space propositional to r? Note that the FM-index needs space proportional to n (i.e., $\approx n \log \sigma$ bits, where σ is the alphabet size). The data-structure community has made great strides in answering this question [3, 5, 14, 21, 23, 31]. The first such index was developed by Mäkinen and Navarro in 2005 [28]. However, it lacked the ability to efficiently locate the occurrences of a pattern within space $\tilde{O}(r)$. After a decade of related research [29, 14], we now have fully functional suffix trees in space proportional to r, developed by Gagie et al. [15]. Also note that the recent optimal BWT construction algorithm for highly repetitive texts is parameterized by r [19]. A technique reducing the value of this parameter r would have a significant impact on a large body of work.

A natural way to minimize r is to change the lexicographic ordering assigned to symbols of the alphabet. To demonstrate that this can have an impact on r, consider as an example the text *mississippi* with the usual ordering $\$ < i < m < p < s$ where $r = 9$, but with the ordering $\$ < s < i < p < m$ we have $r = 8$. In fact, there exist string families in which r differs by a factor of $\Omega(\log n)$ for different orderings. This problem of reordering the alphabet is clearly fixed-parameter tractable in alphabet size σ and has a trivial $O(\sigma!\ n)$ time solution. This may be adequate when σ is small as in DNA sequences. However, this is far from satisfactory from a theoretical point of view, or even from a practical point when the alphabet is slightly larger, such as in protein sequences, natural language texts, ascii texts, etc.

1.1 Related Work

The work in 2018 on block sorting based transformations by Giancarlo et al. gives a theoretical treatment of alphabet ordering in the context of the Generalized BWT [16]. It was shown that for any alphabet ordering, r is at most twice the number of runs in the original text, a result which then holds for the standard BWT as well. Note however that this gives no lower bound on r, and thus gives no results on the approximability of the run minimization problem. There have been multiple previous attempts to develop other approaches to alphabet ordering. In bioinformatics, the role of ordering on proteins was considered in [34] with approaches evaluated experimentally. Similar heuristic approaches evaluated through experiments were done in [1]. Researchers have also considered more restricted versions of this problem. For example, one can try to order a restricted subset of the alphabet, or limit wherein the ordering symbols can be placed. On this problem, heuristics have been utilized. Software tools like BEETL utilize these techniques to handle collections of billions of reads [8]. Another related work in [7] shows, how to permute a given set of strings in linear time, such that the number of runs in the BWT of the (long) string obtained by concatenating the input strings, separated by the same delimiter symbol is minimized.

Even more recently, a work by Giancarlo *et al.,* considered the case where ordering is assigned to the nodes of a string's suffix tree, to minimize the number of runs in the BWT [17]. Interestingly, this problem can be solved in polynomial time. Although their technique can potentially minimize the number of runs in the BWT to an even greater extent than modifying the ordering on the alphabet, it also requires storing the order for each of these nodes, which can require more space. We leave open the problem of finding a trade-off between the strategy of ordering the alphabet and ordering the nodes of the suffix tree.

Given the lack of success with attacking the main problem from the upper bound side, perhaps it is best to approach the problem from the perspective of lower bounds and hardness. To this end, we show why a provably efficient algorithm has been evasive.

2 Problem Definitions and Our Results

Let Σ denotes the alphabet and $\sigma = |\Sigma|$. A run in a string T is a maximal unary sub-string. Let $\rho(T)$ be the number of runs in T.

▶ **Problem 1** (Alphabet Ordering (AO)). *Given a string $T[1, n]$ and an integer t, decide whether there exists an ordering of the symbols in its alphabet such that $\rho(BWT(T)) \leq t$.*

▶ **Theorem 2.** *The alphabet ordering problem is NP-complete and its corresponding minimization problem is APX-hard.*

The problem can be solved in $n \cdot \sigma! = n \cdot 2^{O(\sigma \log \sigma)}$ time naively. However, any significant improvement seems unlikely as per the Exponential Time Hypothesis (ETH) [27].

▶ **Corollary 3.** *Under ETH, AO cannot be solved in time* $poly(n) \cdot 2^{o(\sigma)}$.

It is known that $\rho(BWT(T))$ can be lower bounded by the size of string attractor γ, a recently proposed compressibility measure [22]. Kempa and Kociumaka showed that $\rho(BWT(T))$ can be upper bounded by $O(\gamma \log^2 n)$ [20]. However, γ is independent of the alphabet ordering and the following result is immediate.

▶ **Corollary 4.** *Any alphabet ordering is an* $O(\log^2 n)$*-approximation for AO.*

We also introduce a specialization of AO, one where we impose more constraints on the ordering given to alphabet symbols.

▶ **Problem 5** (Constrained Alphabet Ordering (CAO))**.** *Given a set of* d *strings* $T_1, \ldots, T_d$ *of total length* N*, find an ordering* π *on the symbols* $\$_i$ $(1 \leq i \leq d)$ *such that* $\$_{\pi(1)} \prec \$_{\pi(2)} \ldots \prec \$_{\pi(d)} \prec 0 \ldots \prec \sigma - 1$ *and* $\rho(BWT(T_1\$_1T_2\$_2 \ldots T_d\$_d))$ *is minimized.*

We call $\$_1, \$_2, \ldots, \$_d$ *special symbols*. In Section 5.3, we provide an example where an optimal ordering of special symbols removes a factor of $\Omega(\log_\sigma d)$ in the number of runs, demonstrating that this can be a worthwhile preprocessing step. We refer to [8] for an immediate use case in bioinformatics, where the input is a large collection of DNA reads.

▶ **Theorem 6.** *The constrained alphabet ordering problem can be solved in linear time.*

In the full version [4] of this paper, we extend these hardness results to the related problem of ordering source vertex on Wheeler graphs. Wheeler graphs are a recently introduced class of graphs which allow for BWT based indexing [2, 13, 18].

3 Preliminaries: L-reductions

Our inapproximability results use L-reductions [9]. We will be reducing a problem A, with some known inapproximability results, to a new problem B. We will use the following notation:

- $\mathrm{OPT}_A(x)$ denotes the cost of an optimal solution to the instance x of Problem A.
- $c_A(y)$ denotes the cost of a solution y to an instance x of Problem A (suppressing the x in the notation $c_A(x, y)$).
- Since all problems presented here are minimization problems the approximation ratio can be written as $R_A(x, y) = \frac{c_A(y)}{\mathrm{OPT}_A(x)}$, which is ≥ 1.
- Let $f_A(x) = x'$ be a mapping of an instance x of Problem A to instance x' of Problem B.
- Let y' be a solution to instance $x' = f_A(x)$ and $g_B(y') = y$ be the mapping of a solution y' to a solution y for instance x.

Taking x, y, x' y' as above, an L-reduction is defined by the pair of functions (f_A, g_B), computable in polynomial time, such that there exist constants $\alpha, \beta > 0$, where for all x and y the following two conditions hold:

$$\mathrm{OPT}_B(f_A(x)) \leq \alpha\, \mathrm{OPT}_A(x) \quad \text{and} \quad c_A(g_B(y')) - \mathrm{OPT}_A(x) \leq \beta\Big(c_B(y') - \mathrm{OPT}_B(f_A(x))\Big).$$

As a result, $R_B(x', y') = 1 + \varepsilon$ implies $R_A(x, y) \leq 1 + \alpha\beta\varepsilon = 1 + O(\varepsilon)$. L-reductions preserve APX-hardness [32].

4 Hardness of Alphabet Ordering

We will demonstrate a sequence of L-reductions from the $(1,2)$-TSP Cycle problem, where the aim is to find a Hamiltonian cycle of minimum weight through an undirected *complete graph* on n vertices where all edges have weights either 1 or 2. The $(1,2)$-TSP Cycle problem is APX-hard, even with only $\Theta(n)$ edges of weight 1 [33]. The first reduction is to $(1,2)$-TSP Path, where the goal is to find a Hamiltonian path of minimum weight, rather than a cycle.

▶ **Lemma 7.** *(1,2)-TSP Path is APX-hard, even with only $\Theta(n)$ edges of weight 1.*

Proof. We will give an approximation preserving reduction from $(1,2)$-TSP to $(1,2)$-TSP Path. By the APX-hardness of $(1,2)$-TSP Cycle, we obtain Lemma 7.

Let x be the input graph G for $(1,2)$-TSP Cycle and let f_A map the graph G to an identical graph G'. Let g_B map the $(1,2)$-TSP Path y' given to G' to the cycle in G obtained by connecting the end points of the path with an edge of weight at most 2. Hence the cost $c_B(y')$ is always at most the cost $c_A(g_B(y'))$. At the same time, the weight $\mathrm{OPT}_A(x)$ of an optimal cycle in G is bound above by the weight $\mathrm{OPT}_B(f_A(x))$ of an optimal path in G' plus 2. Thus, $c_B(y') \leq c_A(g_B(y'))$ and $\mathrm{OPT}_A(x) \leq \mathrm{OPT}_B(f_A(x)) + 2$. Therefore,

$$\frac{\mathrm{OPT}_B(f_A(x))}{c_B(y')} \leq 1+\varepsilon \implies \frac{\mathrm{OPT}_A(x)}{c_A(g_B(y'))} \leq \frac{\mathrm{OPT}_B(f_A(x))+2}{c_B(y')} \leq 1+\varepsilon+\frac{2}{n} \leq 1+O(\varepsilon). \quad \blacktriangleleft$$

We proceed to present our reduction which consists of two phases.

4.1 Reduction Phase 1

Given a complete graph on n vertices and $m = \Theta(n)$ edges of weight 1 as input to (1,2)-TSP Path, remove all edges of weight 2. We call the resulting graph G. Construct the incidence matrix for G (a row for each edge, and a column for each vertex, where the two 1's in a row indicate which two vertices are incident to the edge for that row). Then add 2ℓ rows of all 0's to bottom of the matrix, where $\ell = 4m$. Next, add two additional columns c_s and c_t where $c_s[i] = 1$ if $i \in \{m+2, m+4, \ldots, m+2\ell\}$ and 0 otherwise, and $c_t[i] = 1$ if $i \in \{m+1, m+3, \ldots, m+2\ell-1\}$ and 0 otherwise (see Figure 2). We call this matrix M.

$$\begin{array}{c} \boldsymbol{c_s} \quad\quad\quad\quad\quad \boldsymbol{c_t} \\ \left[\begin{array}{ccccc} 0 & ? & \ldots & ? & 0 \\ 0 & ? & \ldots & ? & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & ? & \ldots & ? & 0 \end{array}\right]_{m} \\ \left[\begin{array}{ccccc} 0 & 0 & \ldots & 0 & 1 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & \ldots & 0 & 1 \\ 1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & 1 \\ 1 & 0 & \ldots & 0 & 0 \end{array}\right]_{2\ell} \end{array}$$

Figure 2 The modified incidence matrix for the graph G. Each of the first m rows is for an edge. The bottom $2\ell = 8m$ rows are added as are the outer two most columns.

We now present an intermediate problem that we call Column Ordering (CO), which is: given a matrix M constructed as above, find an optimal ordering on the columns so as to *minimize the number of runs in its linearization.* We will use M_π to denote the matrix M with the ordering π applied to its columns and $L(M_\pi)$ to denote the string obtained by concatenating the rows of M_π from top to bottom. We call $L(M_\pi)$ the linearization of M_π.

Next, we describe the function which maps solutions of our instance of Column Ordering back to a solution of $(1,2)$-TSP Path. Ignoring the added columns c_s and c_t, the ordering π induces a collection of disjoint paths in G, which we call P, where two vertices form an edge if their columns are adjacent and there exists a row with 1's in both columns. Given P we create a (1,2)-TSP Path by connecting the paths in P with $|P|-1$ edges of weight 2. Note that this can be done in linear time.

▶ **Lemma 8.** *If c_s and c_t are the first and last columns of M_π respectively, then the cost of our CO solution is $\rho(L(M_\pi)) = 2m_1 + 4(m-m_1) + 2\ell + 1 = 4m - 2m_1 + 2\ell + 1$, where m_1 is the number of rows whose edges are in the collection of paths P. The corresponding cost of the solution to (1,2)-TSP Path is $m_1 + 2(n-1-m_1) = 2(n-1) - m_1$.*

Proof. Ignoring the first run of $L(M_\pi)$ for the moment, every row in M_π corresponding to an edge in P contributes two runs to $\rho(L(M_\pi))$ (e.g. $0\ldots0110\ldots0$). Any row whose edge is not in P and not in the bottom 2ℓ rows, contributes four (e.g. $0\ldots010\ldots010\ldots0$) and there are $m-m_1$ such (rows) edges. The extra 2ℓ rows in total contribute 2ℓ runs. Adding the "+1" term for the start of $L(M_\pi)$ gives the desired expression. The second statement follows from the TSP Path having m_1 edges of weight 1 and the $n-1$ edges in total needed to form a Hamiltonian path. ◀

▶ **Lemma 9.** *If c_s and c_t are not the first and last columns respectively, then the solution to CO is sub-optimal.*

Proof. If c_t is first and c_s is last, then one extra run is contributed over c_s being first and c_t last, while maintaining the rest of the ordering to be the same. In any configuration where either c_s or c_t are not ends of the matrix, the bottom rows will contribute at least 3ℓ runs. Letting m_1^* denote the optimal number of edges of P, then the optimal $\rho(L(M_{\pi^*}))$ is $4m - 2m_1^* + 2\ell + 1 < 4m + 2\ell \leq 3\ell$. Note that the first inequality is strict since we can always find at least one edge for P. ◀

It is immediate from Lemmas 8 and 9 that an optimal solution for CO is one which maximizes m_1, and this provides an optimal solution for (1,2)-TSP Path. We now must show that our reduction is also an L-reduction. Lemmas 10 and 11 consider the two possible cases.

▶ **Lemma 10.** *If c_s and c_t are the first and last columns respectively in a solution to CO, then the L-reduction conditions hold.*

Proof. By Lemmas 8 and 9, the optimal cost for the instance of CO can be expressed as $4m - 2m_1^* + 2\ell + 1$ and the optimal cost for the instance of (1,2)-TSP Path as $2(n-1) - m_1^*$. To prove Condition (i), we need to show there exists an $\alpha > 0$ such that

$$4m - 2m_1^* + 2\ell + 1 \leq \alpha(2(n-1) - m_1^*)$$

Since $m = \Theta(n)$ there exists a constant $C > 1$, such that for n large enough $m \leq Cn$. The left hand side can be bounded above by $4Cn - 2m_1^* + 8Cn + 1 = 12Cn - 2m_1^* + 1$ (recall $\ell = 4m$). Since $m_1^* \leq n-1$ it is easy to find such an α for $n \geq 2$. Below is the inequality for Condition (ii), which is true for $\beta \geq 1/2$.

$$(2(n-1) - m_1) - (2(n-1) - m_1^*) \leq \beta\Big((4m - 2m_1 + 2\ell + 1) - (4m - 2m_1^* + 2\ell + 1)\Big)$$ ◀

▶ **Lemma 11.** *If c_s and c_t are not the first and last columns respectively in a solution to CO, the L-reduction conditions still hold.*

Proof. Condition (i) holds since the optimal solution values to the overall problem have not changed. For Condition (ii), we consider the two scenarios:

- **Scenario 1**: c_s or c_t are not at the far ends of M_π. Then the cost of the solution for CO, which is at least 3ℓ, exceeds the cost for any solution considered in Lemma 10. Furthermore, any corresponding solution for (1,2)-TSP Path has already been considered in Lemma 10, where now the right-hand is larger than it was in Lemma 10.
- **Scenario 2**: c_t is the first column of M_π and c_s is the last. Then, again, we have already considered a solution in Lemma 10 which has solution cost one less for CO and yet had the same solution cost for (1,2)-TSP Path.

This completes the proof. ◀

4.2 Reduction Phase 2

Given the matrix M as constructed in Phase 1 from G, we will now construct a string T as input to the problem AO. It is easier to describe T in terms of its substrings, which are created by iterating through the matrix M as follows:

- For $1 \leq j \leq n+2$, $1 \leq i \leq m+2\ell$: if $M_{i,j} = 1$ output the substring $10^{i+1}2C_j$
- For $1 \leq j \leq n+2$: output the substring $0^{m+2\ell+2}2C_j$
- Append to each substring created above a unique $\$_i$ symbol ($1 \leq i \leq 2m+2\ell+n+2$).

The string T is the concatenation of these substrings in any order and $|T| = O(n^2)$. The alphabet set Σ is $\{0,1,2\} \cup \{C_1, C_2, \ldots, C_{n+2}\} \cup \{\$_1, \$_2, \ldots, \$_{2m+2\ell+n+2}\}$ and $\sigma = \Theta(n)$.

Given a solution π to this instance of AO we use the relative ordering given to the C_i symbols as the ordering for the columns of M_π. For the analysis of why this works, we define some properties that we would like $BWT(T)$ and π to have. For any symbol $a \in \Sigma$ we will call the maximal set of indices where the F column of the sorted circular shift matrix has only a's as the *a-block*. Our goal will be to"simulate" the linearization of $L(M_\pi)$ within the 0-block of $BWT(T)$. We let C_s and C_t denote the symbols for columns c_s and c_t respectively.

The following are the key properties that an optimal solution π^* will have:

1. For a fixed j, all C_j symbols are placed adjacently in $BWT(T)$;
2. All 2 symbols are placed adjacently in $BWT(T)$;
3. The symbol 2 is adjacent to the symbol 0 in the ordering;
4. The $\$_i$ symbols are ordered in such a way as to minimize the number of runs of 1 in the 0-block of $BWT(T)$.
5. The symbols C_s and C_t are both positioned at the beginning and end respectively of the alphabet ordering given to the C_i symbols.

The 0-block of $BWT(T)$ will consist of 0's, 1's, and $\$_i$ symbols. All $\$_i$ symbols will be adjacent within the 0-block. This is since the $\$_i$ symbols succeeded by 0, are all succeeded by the substring $0^{m+2\ell+2}2$ and every occurrence of $0^{m+2\ell+2}2$ preceded by a $\$_i$ symbol (when T is viewed as a circular string). Let r_0 denote the number of runs created in the 0-block of $BWT(T)$, minus the number of $\$_i$ symbols in the 0-block of $BWT(T)$.

▶ **Lemma 12.** *Unless all of the above properties hold, the solution to AO is suboptimal.*

Proof. If any of Properties 1–3 are violated, we can exchange our solution with one which maintains the value r_0 but reduces the runs created in other blocks. This is since the alphabet ordering can be modified to have these properties, while at the same time maintaining the relative orderings of symbols within the 0-block. In the case of Property 4, given that

Properties 1–3 hold, modifying the solution so that the property holds can only decrease r_0, while it maintains the number of runs created in other blocks. Assuming properties 1–4 hold, there are two possibilities, either C_s and C_t are extremal or they are not.

- In the case of being extremal, if $C_s < C_t$, then by Property 4, the $2\ell = 8m$ instances of 1's in the bottom 2ℓ rows of M_π shall correspond to $4m$ runs of two consecutive 1's in the 0-block of $BWT(T)$. The upper rows of M_π shall correspond to at most $2m$ runs of 1's in the 0-block of $BWT(T)$. Hence, in the 0-block there are at most $6m + 1$ runs of 1's making at most $6m + 2$ runs of zeros to surround them, so that $r_0 \leq 12m + 3$. In the case where $C_t < C_s$, one additional run of 1's is created over the same configuration where the positions of C_s and C_t are swapped.
- In the case of them not being extremal, considering only the last 2ℓ rows of M_π, there are $8m$ runs of lonely 1's in the 0-block of $BWT(T)$, and at least $8m + 1$ runs of 0's to surround them, leading to $r_0 \geq 16m + 1$.

This completes the proof. ◀

As mentioned earlier, we aim to have a substring of $BWT(T)$ within the 0-block which is the same as $L(M_\pi)$ except for the lengths of its runs, i.e., the number of runs will be the same. We will call this substring the simulation of $L(M_\pi)$.

▶ **Lemma 13.** *If all Properties 1–5 hold, then* $r_0 = \rho(L(M_\pi)) - 1$ *and* $\rho(BWT(T)) = r_0 + \sigma - 1$.

Proof. We will first show that when Properties 1–5 hold, $r_0 = \rho(L(M_\pi)) - 1$, i.e., that the simulation works. Within the 0-block of $BWT(T)$, row i is simulated by the characters preceding each substring $0^{i+1}2$. Note that they all appear consecutively in the 0-block. Within the simulation of the i^{th} row, if the value of the j^{th} column of M_π is 0, then the characters preceding substrings of the form $0^{i+1}2C_j$ are all 0. If the value of the j^{th} column of M is 1, then there exists a single substring of the form $0^{i+1}2C_j$ preceded by a 1, and the remaining substrings of the form $0^{i+1}2C_j$ are all preceded by 0. Note that all characters preceding $0^{i+1}2C_j$ are consecutive within the i^{th} row, however, the unique \$'s following each substring allow the characters following each $0^{i+1}2C_j$ to have their orders swapped. Because of Property 5, in the column ordering of M_π there will never be a run of more than two consecutive 1's in $L(M_\pi)$. Hence, when Property 4 is applied, we know that 1's which would are adjacent in $L(M_\pi)$ are adjacent in the 0-block. Combining all these observations gives us that $L(M_\pi)$ is successfully simulated within the 0-block. The "-1" term in the expression for r_0 arises due to Property 2. This is since the 0 symbol in 0-block of $BWT(T)$ that is adjacent to the 2-block does not contribute a run. We have shown $r_0 = L(M_\pi) - 1$.

Finally, the fact that $\rho(BWT(T)) = r_0 + \sigma - 1$ follows from Properties 1–3 which cause every symbol except 1 to contribute exactly one run to $\rho(BWT(T))$ outside of the simulation (1's first appearance is within the simulation). ◀

▶ **Lemma 14.** *If all Properties 1–5 hold, the L-reduction conditions are satisfied.*

Proof. By Lemma's 12 and 13 we have the optimal cost for AO being $r_0^* + \sigma - 1$ and optimal cost for CO as $r_0^* + 1$. For Condition (i) note that $\sigma = \Theta(n)$ and because there are at most 5 runs created by each row, $m + 2\ell \leq r_0^* \leq 5(m + 2\ell)$, so that $r_0^* = \Theta(n)$. Hence, we can find an α such that $r_0^* + \sigma - 1 \leq \alpha(r_0^* + 1)$. For Condition (ii), we have $(r_0 + 1) - (r_0^* + 1) \leq \beta((r_0 + \sigma - 1) - (r_0^* + \sigma - 1))$ with $\beta = 1$. ◀

▶ **Lemma 15.** *If any of Properties 1–5 are violated, the L-reduction conditions are satisfied.*

Proof. Condition (i) is satisfied since optimal values for the overall problem are unchanged. For Condition (ii), if any of the first four properties are violated, we have already shown in Lemma 14 that the inequality holds in the harder case where $\rho(L(M_\pi))$ has the same value but the overall number of runs in $BWT(T)$ is less. If the first four properties hold and the fifth property does not hold, there are two cases. In the first case, if C_t is ordered first and C_s last, then swapping C_s and C_t modifies both sides of the inequality for Condition (ii) by the same amount. In the second case, if either C_s or C_t are not ordered first or last, the left hand side of the inequality in Condition (ii), that is $\Big(\rho(L(M_\pi)) - \rho(L(M_{\pi^*}))\Big)$, will be large, as this corresponds to the columns c_s and c_t not being first or last. However, the right-hand side $\Big((r_0 + \sigma - 1) - (r_0^* + \sigma - 1)\Big)$ will be large as well, perhaps even larger as there may exist runs of three of four 1's in $L(M_\pi)$ that cannot be simulated in the 0-block of $BWT(T)$. In particular, $r_0 \geq \rho(L(M_\pi)) - 1$ and $\rho(L(M_{\pi^*})) = r_0^* + 1$, so that with $\beta = 1$

$$\rho(L(M_\pi)) - \rho(L(M_{\pi^*})) \leq (r_0 + 1) - \rho(L(M_{\pi^*})) \leq \beta\Big((r_0 + \sigma - 1) - (r_0^* + \sigma - 1)\Big). \quad \blacktriangleleft$$

We have shown an L-reduction from (1,2)-TSP Path to AO. This combined with Lemma 7 completes the proof for Theorem 2.

4.3 Proof of Corollary 3

Assuming ETH, there exists no $2^{o(n)}$ time algorithm for Hamiltonian Path Problem [10]. Our reduction allows us to determine the minimum number of paths in G needed to cover all the vertices and can hence solve Hamiltonian Path. This can be done by first constructing an incidence matrix for G and then applying the rest of the reduction as in Section 4. Since the alphabet size σ is linear in n and $|T| = \Theta(n^2)$, an $|T|^{O(1)} \cdot 2^{o(\sigma)}$ time algorithm for AO would imply an $2^{o(n)}$ time algorithm for Hamiltonian Path, a contradiction.

5 Constrained Alphabet Ordering

5.1 Reducing to a Simpler Problem

Recall that we wish to find an ordering on the special symbols $\$_1, \ldots, \$_d$ such that the number of runs in the BWT of $T = T_1\$_1 \ldots T_d\$_d$ is minimized and the \$ symbols are lexicographically before other symbols. We will consider our alphabet to be over integers that are bounded by $N^{O(1)}$, where $N = |T|$. Let s be an arbitrary substring of T without \$ symbols. The symbols in T which are followed by $s\$_i$ will form a contiguous portion of $BWT(T)$. However, their ordering within that contiguous portion is determined by the relative ordering given to $\$_i$ symbols. Hence, we can arrange the symbols within this portion of $BWT(T)$ so that identical symbols are placed adjacently.

For example, let $c_1 s\$_1$, $c_2 s\$_2$, ..., $c_t s\$_t$ be substrings of T. The symbols c_1, c_2, ... c_t will be contiguous in $BWT(T)$ in some order. Now, suppose that $c_2 = c_4 = c_7$. By rearranging the $\$_2$, $\$_4$, and $\$_7$ to be adjacent within the relative ordering of the \$ symbols, we can make c_2, c_4, and c_7 appear consecutively. Taking this one step further, we can also change the relative ordering of $\$_2$, $\$_4$, and $\$_7$, so that if the substrings $\alpha c_2 s\$_2$, $\beta c_4\$_4$, and $\alpha s\$_7$ occur in T, then the two α's will be adjacent in the contiguous portion of $BWT(T)$ corresponding to the substrings $c_2 s\$_2$, $c_4 s\$_4$, and $c_7 s\$_7$.

Hence, the set of symbols $B_s = \{x \mid xs\$_i \text{ is a substring of } T \text{ for some } i \in [1, d]\}$ can be modeled as a tuple where each symbol appears only once within the tuple. Along with each symbol x in B_s, we will maintain a set $\Delta_s^x = \{\$_i \mid xs\$_i \text{ is a substring of } T\}$. We will arrange

all non-empty tuples B_s in the lexicographic ordering of s. As such, these tuples can be constructed by first assigning any ordering to the \$ symbols (where they are lexicographically first in the alphabet) and then using the longest common prefix (LCP) between consecutive suffixes in lexicographic order. These values are obtained directly from the longest common prefix array. The suffix array and longest common prefix array can both be constructed in linear time assuming an integer alphabet of size $N^{O(1)}$ [11]. We will define the problem of ordering the symbols within these tuples as a new problem.

▶ **Problem 16** (Tuple Ordering (TO)). *Given a list of tuples $t_1, \ldots, t_q$ in a fixed order, each containing a subset of symbols from Σ, order the symbols in each tuple such that the total number of runs in the string formed by their concatenation $t_1 \cdot t_2 \cdot \ldots \cdot t_q$ is minimized (not considering '(', ')' and commas, of course).*

We will show that TO can be solved in linear time. To map solutions of TO back to solutions of CAO, a tuple for B_s needs to maintain pointers to each tuple B_{xs}, where x is a symbol. Then given a solution to TO, we start with the tuple for B_ε. The ordering given to symbols within this tuple provides us with a partial ordering on the \$ symbols. The symbols in Δ_ε^x associated with the first symbol x within the tuple are ordered before the symbols Δ_ε^y associated with the second symbol y, etc. Then for a symbol x, the tuple for B_x provides a refinement of this partial ordering. In particular, it provides a partial ordering on Δ_ε^x. To recover the total ordering on \$ symbols, we recursively refine the partial ordering at our current tuple by examining all of the tuples which the current tuple points to. Note that this works since for a given tuple for B_s, the sets Δ_s^x are disjoint. The time required to recover this solution is proportional to N.

5.2 Solving the Tuple Ordering Problem in Linear Time

We show how to reduce the TO problem to the single-source shortest path problem on a DAG G, which is constructed as follows. For each tuple t_i, create two sets of vertices L_i and R_i, both of size $|t_i|$, such that for each symbol $c \in t_i$, there exists a vertex with label c in L_i as well as in R_i. Between each pair of vertices $u \in L_i$ and $v \in R_i$, where the label of u is not equal to the label of v, create a directed edge of weight 1 from u to v. If $|t_i| = 1$, then create a directed edge of weight 1 from the unique vertex in L_i to the unique vertex in R_i. For each R_i and L_{i+1} $(1 \leq i \leq q-1)$, and each pair $u \in R_i$ and $v \in L_{i+1}$, create a directed edge from u to v, with weight 1 if they have the same label, and weight 2 otherwise. Finally, create a start vertex s and directed edges of weight 1 from s to each vertex in L_1, and an end vertex e with directed edges of weight 1 from each vertex in R_q to e. See Figure 3 for an illustration.

Clearly, the shortest path from s to e is the one with the fewest edges of weight 2, and this path gives us a tuple ordering which minimizes the number of runs created by the tuples. To obtain this ordering, for a tuple t_i, place as the left-most symbol the label of the vertex used in L_i within the shortest path, and the right-most symbol the label of the vertex used in R_i within the shortest path. The other symbols can be ordered arbitrarily. Because G a DAG, this shortest path can be found in time proportional to the number of edges, which is $O(\sigma^2 q)$. Next, we show how to solve this in time proportional to the number of vertices of G.

Rather than constructing the edges in G, we can work from left-to-right maintaining the shortest path from s to the vertices in our current level of G, either L_i or R_i. Suppose our current level is L_i and we wish to extend the solution to the level R_i. Assuming $|t_i| \geq 2$, we identify the vertices v_1 and v_2 in L_i with the first and second shortest paths (they may have the same length) from s, respectively. For each vertex u in R_i, if the label of u is not the same as the label for v_1, we make the shortest path to u the path from s to v_1, then the edge

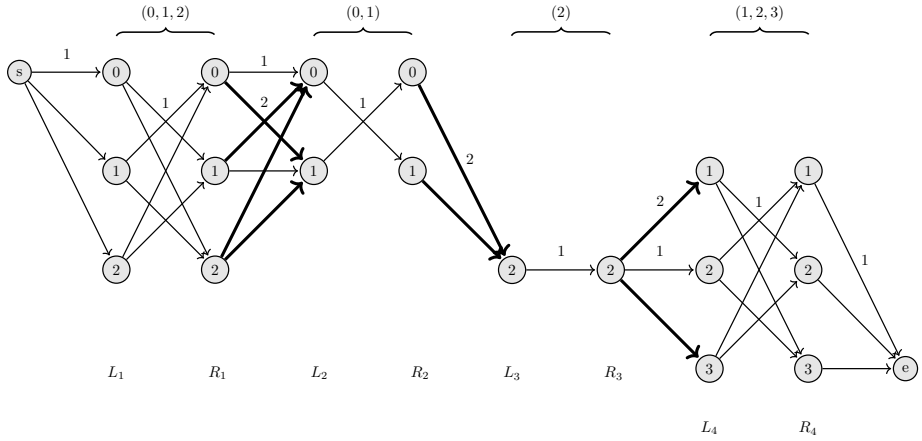


Figure 3 The graph G constructed for the tuple ordering instance $(0, 1, 2), (0, 1), (2), (1, 2, 3)$.

from v_1 to u, otherwise we make it the path from s to v_2, then the edge from v_2 to u. If $|t_i| = 1$, we make the shortest path from s to u the path from s to the unique vertex v in L_i, then the edge from v to the unique vertex u. To extend a solution from R_i to L_{i+1}, we first identify the vertex v_1 in R_i with the shortest path from s. For each vertex u in L_{i+1}, if a vertex with matching label v_u exists in R_i, we take as the shortest path to u the shorter of the following two paths: (i) the path from s to v_1, then from v_1 to u, or (ii) the path from s to v_u, then from v_u to u. If no such vertex with matching label exists in R_i, take as the shortest path from s to u the path from s to v_1, then from v_1 to u.

5.3 An Example of the Effectiveness of CAO

Lastly, we provide an example where the \$ symbol ordering greatly reduces the number of runs in the BWT. Let d be the number of strings and n the length of the strings. It is possible for a set of special symbols to be ordered such that the number of runs is $\Omega(nd)$. Let $\sigma = 2$ and $d = \sigma^n$. Consider the d distinct binary strings concatenated with special symbols in lexicographic order. Under the ordering $\$_1 < \$_2 \ldots < \$_d$, the string $BWT(T)$ alternates between the \$'s, 0's, and 1's, yielding $\Omega(nd)$ runs. On the other hand, for this same case, arranging the \$'s in the optimal ordering will give $O(d)$ runs in total. This is since for any substring s of T, the contiguous section of $BWT(T)$ containing the characters preceding $s\$_i$ for $i \in [1, d]$ contains at most the start of two runs. For example, with $n = 3$, we would have $T = 000\$_1001\$_2010\$_3011\$_4100\$_5101\$_6110\$_7111\$_8$. The number of runs in $BWT(T)$ under the naive ordering $\$_1 < \$_2 < \ldots < \$_8$, is 32 with $BWT(T) = 010101010101\$_8\$_101\$_2\$_3010101\$_4\$_501\$_6\$_7$. The number of runs using an optimal ordering $\$_3 < \$_5 < \$_2 < \$_7 < \$_4 < \$_6 < \$_1 < \$_8$ is 19 with $BWT(T) = 000011111100 01\$_8\$_101\$_2\$_3001110\$_4\$_501\$_6\$_7$.

References

1 Jürgen Abel. Post BWT stages of the burrows-wheeler compression algorithm. *Softw., Pract. Exper.*, 40(9):751–777, 2010. doi:10.1002/spe.982.

2 Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 911–930, 2020. doi:10.1137/1.9781611975994.55.

3 Hideo Bannai, Travis Gagie, et al. Online lz77 parsing and matching statistics with rlbwts. In *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

4 Jason Bentley, Daniel Gibney, and Sharma V. Thankachan. On the complexity of bwt-runs minimization via alphabet reordering. *CoRR*, abs/1911.03035, 2019. arXiv:1911.03035.

5 Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big bwts. *Algorithms for Molecular Biology*, 14(1):13, 2019.

6 Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. *SRC Research Report*, 124, 1994.

7 Bastien Cazaux and Eric Rivals. Linking BWT and XBW via aho-corasick automaton: Applications to run-length encoding. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 24:1–24:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.24.

8 Anthony J. Cox, Markus J. Bauer, Tobias Jakobi, and Giovanna Rosone. Large-scale compression of genomic sequence databases with the Burrows–Wheeler transform. *Bioinformatics*, 28(11):1415–1419, May 2012. doi:10.1093/bioinformatics/bts173.

9 Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 262–273. IEEE, 1997.

10 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Lower bounds based on the exponential-time hypothesis. In *Parameterized Algorithms*, pages 467–521. Springer, 2015.

11 Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000. doi:10.1145/355541.355547.

12 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 390–398, 2000. doi:10.1109/SFCS.2000.892127.

13 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for bwt-based data structures. *Theor. Comput. Sci.*, 698:67–78, 2017. doi:10.1016/j.tcs.2017.06.016.

14 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in bwt-runs bounded space. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1459–1477, 2018. doi:10.1137/1.9781611975031.96.

15 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in bwt-runs bounded space. *J. ACM*, 67(1), January 2020. doi:10.1145/3375890.

16 Raffaele Giancarlo, Giovanni Manzini, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Block sorting-based transformations on words: Beyond the magic BWT. In *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, pages 1–17, 2018. doi:10.1007/978-3-319-98654-8_1.

17 Raffaele Giancarlo, Giovanni Manzini, Giovanna Rosone, and Marinella Sciortino. A new class of searchable and provably highly compressible string transformations. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 12:1–12:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.12.

18 Daniel Gibney and Sharma V. Thankachan. On the hardness and inapproximability of recognizing wheeler graphs. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.*, pages 51:1–51:16, 2019. doi:10.4230/LIPIcs.ESA.2019.51.

19 Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1344–1357, 2019. doi:10.1137/1.9781611975482.82.

20 Dominik Kempa and Tomasz Kociumaka. Resolution of the burrows-wheeler transform conjecture. *CoRR*, abs/1910.10631, 2019. arXiv:1910.10631.

21 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 827–840, 2018. doi:10.1145/3188745.3188814.

22 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

23 Alan Kuhnle, Taher Mun, Christina Boucher, Travis Gagie, Ben Langmead, and Giovanni Manzini. Efficient construction of a complete index for pan-genomics read alignment. In *Research in Computational Molecular Biology - 23rd Annual International Conference, RECOMB 2019, Washington, DC, USA, May 5-8, 2019, Proceedings*, pages 158–173, 2019. doi:10.1007/978-3-030-17083-7_10.

24 Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome biology*, 10(3):R25, 2009.

25 Heng Li and Richard Durbin. Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

26 Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

27 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

28 Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. In *Combinatorial Pattern Matching, 16th Annual Symposium, CPM 2005, Jeju Island, Korea, June 19-22, 2005, Proceedings*, pages 45–56, 2005. doi:10.1007/11496656_5.

29 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of individual genomes. In *Research in Computational Molecular Biology, 13th Annual International Conference, RECOMB 2009, Tucson, AZ, USA, May 18-21, 2009. Proceedings*, pages 121–137, 2009. doi:10.1007/978-3-642-02008-7_9.

30 Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

31 Tatsuya Ohno, Kensuke Sakai, Yoshimasa Takabatake, I Tomohiro, and Hiroshi Sakamoto. A faster implementation of online rlbwt and its application to lz77 parsing. *Journal of Discrete Algorithms*, 52:18–28, 2018.

32 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

33 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993. doi:10.1287/moor.18.1.1.

34 Lianping Yang, Guisong Chang, Xiangde Zhang, and Tianming Wang. Use of the burrows–wheeler similarity distribution to the comparison of the proteins. *Amino acids*, 39(3):887–898, 2010.

Simulating Population Protocols in Sub-Constant Time per Interaction

Petra Berenbrink
Universität Hamburg, Germany
petra.berenbrink@uni-hamburg.de

David Hammer
University of Southern Denmark, Odense, Denmark
Goethe University Frankfurt, Germany
hammer@imada.sdu.dk

Dominik Kaaser
Universität Hamburg, Germany
dominik.kaaser@uni-hamburg.de

Ulrich Meyer
Goethe University Frankfurt, Germany
umeyer@ae.cs.uni-frankfurt.de

Manuel Penschuck
Goethe University Frankfurt, Germany
mpenschuck@ae.cs.uni-frankfurt.de

Hung Tran
Goethe University Frankfurt, Germany
hung@ae.cs.uni-frankfurt.de

Abstract

We consider the efficient simulation of population protocols. In the population model, we are given a system of n agents modeled as identical finite-state machines. In each step, two agents are selected uniformly at random to interact by updating their states according to a common transition function. We empirically and analytically analyze two classes of simulators for this model. First, we consider sequential simulators executing one interaction after the other. Key to the performance of these simulators is the data structure storing the agents' states. For our analysis, we consider plain arrays, binary search trees, and a novel Dynamic Alias Table data structure. Secondly, we consider batch processing to efficiently update the states of multiple independent agents in one step. For many protocols considered in literature, our simulator requires amortized sub-constant time per interaction and is fast in practice: given a fixed time budget, the implementation of our batched simulator is able to simulate population protocols several orders of magnitude larger compared to the sequential competitors, and can carry out 2^{50} interactions among the same number of agents in less than 400 s.

2012 ACM Subject Classification Computing methodologies → Agent / discrete models

Keywords and phrases Population Protocols, Simulation, Random Sampling, Dynamic Alias Table

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.16

Related Version A full version [17] of the paper is available at `http://arxiv.org/abs/2005.03584`.

Supplementary Material Implementations and data: `https://ae.cs.uni-frankfurt.de/r/p/pps`.

Funding This work was partially supported by the Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2, ME 2088/4-2, and ME 2088/5-1.

Acknowledgements This project was initiated on a workshop of the DFG FOR 2975/1. We thank the anonymous reviewers for their insightful comments and pointers, as well as the Center for Scientific Computing, University of Frankfurt, for making their HPC facilities available.

1 Introduction

We consider the *population model*, introduced by Angluin et al. [5] to model systems of resource-limited mobile agents that interact to solve a common task. Agents are modeled as finite-state machines. The computation of a *population protocol* is a sequence of pairwise interactions of agents. In each interaction, the two participating agents observe each other's states and update their own state according to a transition function common to all agents.

Typical applications of population protocols are networks of passively mobile sensors [5]. As an example, consider a flock of birds, where each bird is equipped with a simple sensor. Two sensors communicate whenever their birds are sufficiently close. An application could be a distributed disease monitoring system raising an alarm if the number of birds with high temperature rises above some threshold. Further processes which resemble properties of population protocols include chemical reaction networks [38], programmable chemical controllers at the level of DNA [22], or biochemical regulatory processes in living cells [21].

While the computational power of population protocols with constantly many states per agent is well understood by now (see below), less is known about the power of protocols with state spaces growing with the population size. In this setting, much interest has been on analyzing the runtime and state space requirements for *probabilistic* population protocols, where the two interacting agents are sampled in each time step independently and uniformly at random from the population. This notion of a probabilistic scheduler allows the definition of a *runtime* of a population protocol. The runtime and the number of states are the main performance measures used in the theoretical analysis of population protocols.

For the theoretical analysis of population protocols, a large toolkit is available in the literature. Consequently, the remaining gaps between upper and lower bounds for many quantities of interest have been narrowed down: for many protocols, the required number of states has become sub-logarithmic, while the runtime approaches more and more the (trivial) lower bounds for any meaningful protocol. So far, when designing new protocols, simulations have always proven a versatile tool in getting an intuition for these stochastic processes. However, once observables are of order $\log\log n$ and below, naive population protocol simulators fail to deliver the necessary insights (e.g., $\log\log n \leq 5$ for typical input sizes of $n \leq 2^{32}$). Our main contribution in this paper is a new simulation approach allowing to execute a large number of interactions even if the population size exceeds 2^{40}. In the remainder of this section, we first give a formal model definition in Section 1.1 and then describe our main contributions and related work in Sections 1.2 and 1.3.

1.1 Formal Model Definition

In the *population model*, we are given a distributed system of n agents modeled as finite-state machines. A *population protocol* is specified by a state space $Q = \{q_1, \ldots, q_{|Q|}\}$, an output domain Y, a transition function $\delta\colon Q \times Q \to Q \times Q$, and an output function $\gamma\colon Q \to Y$. At time t, each agent i has a state $s_i(t) \in Q$, which is updated during the execution of the protocol. The current output of agent i in state $s_i(t)$ is $\gamma(s_i(t))$. The *configuration* $C(t) = \{s_1(t), \ldots, s_n(t)\}$ of the system at time t contains the states of the agents after t interactions. For the sake of readability, we omit the parameter t in $C(t)$ and $s_i(t)$ when it is clear from the context. The *initial configuration* is denoted $C(0) = C_0$.

The *computation* of a population protocol runs in a sequence of discrete time steps. In each time step, a *probabilistic scheduler* selects an ordered pair of agents (u, v) independently and uniformly at random to *interact*. Agent u is called the *initiator* and agent v is the *responder*. During this *interaction*, both agents u and v observe each other's state and update their states according to the transition function δ such that $(s_u(t{+}1),\ s_v(t{+}1)) \leftarrow \delta(s_u(t),\ s_v(t))$.

Table 1 Simulating N interactions among n agents in $|Q|$ states. For MULTIBATCHED, we restrict $|Q| = \omega(\sqrt{\log n})$. Values indicated by † hold in expectation.

	Simulator	Section	Time Complexity	Space Complexity (bits)
Sequential	$\text{SEQ}_{\text{Array}}$	Section 2	$\Theta(N)$	$\Theta(n \log \lvert Q\rvert)$
	$\text{SEQ}_{\text{Linear}}$	Section 2	$\mathrm{O}(N\lvert Q\rvert)$	$\Theta(\lvert Q\rvert \log n)$
	SEQ_{BST}	Section 2	$\Theta(N \log \lvert Q\rvert)$	$\Theta(\lvert Q\rvert \log n)$
	$\text{SEQ}_{\text{Alias}}$	Section 2	$\Theta(N)$ w.h.p.	$\Theta(\lvert Q\rvert \log n)$
Batch	BATCHED	Section 4	$\mathrm{O}(N(\log n + \lvert Q\rvert^2)/\sqrt{n})^\dagger$	$\Theta(\lvert Q\rvert \log n)$
	MULTIBATCHED	Section 5	$\mathrm{O}(N\lvert Q\rvert\sqrt{\log n}/\sqrt{n})^\dagger$	$\Theta(\lvert Q\rvert \log n)$

A given problem for the population model specifies the agents' initial states, the output domain O, and (a set of) *desired (output) configurations* for a given input. As an example, consider the MAJORITY problem. Each agent is initially in one of two states q_A and q_B corresponding to two opinions A and B. Assuming that A is the initially dominant opinion, the protocol concludes once all agents u give $\gamma(s_u) = A$ as their output. Any configuration in which all agents output the initially dominant opinion is a desired configuration.

This notion of a desired configuration allows to formally define two notions of a *runtime* of a population protocol. The *convergence time* T_C is the number of interactions until the system enters a desired configuration and never leaves the desired configurations in a given run. The *stabilization time* T_S is the number of interactions until the system enters a desired *stable* configuration for which there does not exist any sequence of interactions due to which the system leaves the desired configurations. A population protocol is *stable*, if it always eventually reaches a desired output configuration.

A number of variants of this model are commonly used. For *symmetric protocols*, the order of the interacting agents is irrelevant for the transition. In particular, this means that if $\delta(q_u, q_v) = (q'_u, q'_v)$, then $\delta(q_v, q_u) = (q'_v, q'_u)$. In *protocols with probabilistic transition functions*, the outcome of an interaction may be a random variable. In *one-way protocols*, only the initiator updates its states such that $\delta(q_u, q_v) = (q'_u, q_v)$ for any interaction.

Model Assumptions. We assume a *meaningful* protocol which converges after at most $N = \text{poly}(n)$ interactions, has an $\mathrm{O}(1)$ time transition function δ, and uses $|Q| < \sqrt{n}$ states (observe that many relevant protocols only use $|Q| = \mathrm{O}(\text{polylog}\, n)$ states; see Section 1.3).

1.2 Our Contributions

In this paper, we present a new approach for simulating population protocols. Our simulator allows us to efficiently simulate a large number, N, of interactions for large populations of size n. Our findings are summarized in Table 1.

Sequential Simulators. As a baseline, we directly translate the population model into a sequential algorithm framework SEQ: SEQ selects for each interaction two agents uniformly at random, updates their states, and repeats. We analyze the runtime and memory consumption of various variants in Section 2.

Batch Processing. To speed up the simulation, we introduce and exploit *collision-free runs*, a sequence of interactions where no agent participates more than once. Our algorithms BATCHED and MULTIBATCHED coalesce these independent interactions into batches for

improved efficiency. BATCHED first samples the length ℓ of a collision-free run. It then randomly pairs ℓ independent agents, adds one more interaction – the collision – reusing one of the run's agents, and finally repeats. BATCHED is presented in Section 4 and extended into MULTIBATCHED in Section 5. We discuss practical details and heuristics in Section 6.

Dynamic Alias Tables. The simulation of population protocols often needs an *urn-like* data structure to efficiently sample random agents (marbles) and update their states (colors). The *alias method* [41, 40] enables random sampling from arbitrary discrete distributions in $O(1)$ time. However, it is static in that the distribution may not change over time. Thus, we extend it and analyze a *Dynamic Alias Table* in Section 2. It supports sampling with and without replacement uniformly at random (u.a.r.) and addition of elements (if the urn is sufficiently full). Due to its practical performance and simplicity compared to more general solutions [30, 33], we believe this data structure might be of independent interest and show:

▶ **Theorem 1.** *Let U be a Dynamic Alias Table that stores an urn of n marbles, where each marble has one of k possible colors. U requires $\Theta(k \log n)$ bits of storage. If $n \geq k^2$, we can*

- *select a marble u.a.r. from U with replacement in expected constant time,*
- *select a marble u.a.r. from U without replacement in expected amortized constant time,*
- *and add a marble of a given color to U in amortized constant time.*

1.3 Related Work

Population Protocols. The population model was introduced in [5], assuming a constant number of states per agent. Together with [6, 9], they show that all semilinear predicates are stably computable in this model. In the following, we focus on two prominent problems, MAJORITY and LEADER ELECTION. For a broad overview, we refer to surveys [12] and [26].

In [8] a MAJORITY protocol with three states is presented where the agents agree on the majority after $O(n \log n)$ interactions w.h.p. (*with high probability* $1 - n^{-\Omega(1)}$), if the initial numbers of agents holding each opinion differ by at least $\omega(\sqrt{n} \log n)$. In [34, 25], four-state protocols are analyzed that stabilize in expectation in $O(n^2 \log n)$ interactions. In a recent series of papers [35, 1, 2, 4, 19, 15, 14, 13], bounds for the MAJORITY problem have been gradually improved. The currently best known protocol [13] solves MAJORITY w.h.p. in $O(n \log^{3/2} n)$ interactions using $O(\log n)$ states. Regarding lower bounds, [1] shows that protocols with less than $(\log \log n)/2$ states require in expectation $\Omega(n^2/\operatorname{polylog}(n))$ interactions to stabilize. In [2] it is shown that any MAJORITY protocol that stabilizes in $n^{2-\Omega(1)}$ expected interactions requires $\Omega(\log n)$ states under some natural assumptions.

The goal for LEADER ELECTION protocols is that exactly one agent is in a designated leader state. Doty and Soloveichik [24] show that any population protocol with a constant number of states that stably elects a leader requires $\Omega(n^2)$ expected interactions, a bound matched by a natural two-state protocol. Upper bounds for protocols with a non-constant number of states per agent were presented in [3, 1, 19, 2, 18, 28, 29, 16]. In [28] a LEADER ELECTION protocol that stabilizes w.h.p. in $O(n \log^2 n)$ interactions, using $O(\log \log n)$ states (matching a corresponding lower bound [1]) is presented. The core idea is to synchronize the agents using a *phase-clock*. The currently best known protocol for LEADER ELECTION is due to [16], stabilizing in expected $O(n \log n)$ interactions using $O(\log \log n)$ states per agent.

As a tool for self-synchronization, so-called *phase-clocks* have been explored in a wide range of related areas, see, e.g., the seminal paper [10]. In the population model, the concept of phase-clocks was first introduced in [7] under the assumption that a leader is present. These clocks were generalized in [28] to a *junta* of n^ε agents. In Section 7 we empirically analyze a variant of this phase-clock process.

Simulations have proven a versatile tool to get an intuitive understanding of population protocols. This is also reflected in the related work: See, e.g., [7, 8, 4, 3] for some examples of papers that also present empirical data. However, to the best of our knowledge, our paper is the first systematic analysis of simulators for population protocols.

Sampling from Discrete Distributions. The methods to sample non-uniform variates heavily depend on the modeling and properties of the required probability distributions.

If the distribution is governed by a closed-form density function $f(x)$, "numerical tricks" can yield efficient algorithms with small memory footprints; this is the case for most well-known distributions [23, 20]. A standard approach is the *inverse sampling technique* [23]. It needs to compute the inverse F^{-1} of f's cumulative density function $F(x) = \int_{-\infty}^{x} f(y)\mathrm{d}y$ (cf. Section 6.1). Another concept is *rejection sampling* [20, Sec. 5.2.5/6]. It obtains a sample x from a suitable simpler distribution g. In order to generate the distribution f, the sample is accepted only with probability proportional to $f(x)/g(x)$. The process repeats until a sample is obtained (cf. Section 3). A special case is the *ratio-of-uniforms* method, commonly used to obtain hypergeometric random variates [39].

Dedicated data structures support sampling from arbitrary discrete distributions. Given a finite universe $U = \{1, \ldots, u\}$ with probabilities $(p_1, \ldots, p_u)$ with $\sum_i p_i = 1$, Walker's *alias tables* [41] allow sampling in constant expected time, and can be constructed in $\mathrm{O}(u)$ time [40]; recently Hübschle-Schneider et al. [31] discussed engineering aspects and parallel construction.

In this paper, we model an urn with n balls with $\mathrm{O}(\sqrt{n})$ colors (typically much less) as a distribution over k colors where p_i is proportional to the number of balls with color i. While alias tables work in the static case, they do neither support removal nor insertion of balls without rebuilding the data structure.

A suited binary tree can be constructed in $\mathrm{O}(k)$ time, and supports updates and sampling in $\mathrm{O}(\log k)$ time. Two independent but similar data structures by [30] and [33] support updates and sampling in expected constant time. They partition the input into groups such that the probabilities of elements within a group differ by at most a factor of two. This allows rejection sampling *within* a group with an acceptance rate of at least 1/2. Since updates to the distribution can affect the partition sizes, over-provisioning and table doubling involving garbage collection [33, App. B] is used.

The approaches differ in the way they select a group to sample from. They are however both recursive in the sense that the group selection is carried out with another instance of the data structure itself. To achieve constant access times, the recursion is stopped after constantly many layers, and the remaining very small problem is treated as a special case. As a result, the data structures are quite complex, and were excluded in preliminary experiments due to performance considerations. By constraining the supported distributions, simpler schemes can be obtained [36, 27]. However, in our use case, they incur impractically high rejection rates, as we cannot give non-trivial bounds on the number of balls per color.

It is note-worthy that Hagerup et al. [30] operate on integer weights (modeled as *generalized distributions* lacking the normalization constraint), and require integer arithmetic only.

2 Sequential Simulation

As a baseline, we first consider variants of Seq, a sequential approach defined in Algorithm 1. It is a direct translation of the machine model discussed in Section 1.1. Seq carries out N steps in a fully serialized manner. For each interaction, it selects two agents uniformly at random, computes their new states based on their current ones, and updates the configuration.

Algorithm 1 SEQ: The algorithmic framework for sequential simulation.

input: configuration C, transition function δ, number of steps N
for $t \leftarrow 1$ **to** N **do**
  sample and remove agents i and j without replacement from C
  add agents in states $\delta(s_i, s_j)$ to C

Under the realistic assumption that the transition function δ can be evaluated in constant time, SEQ's runtime and memory footprint is dominated by storing, sampling from, and updating the configuration C. We therefore consider appropriate data structures. In the population model, agents typically are anonymous, i.e., we cannot distinguish two agents in the same state. Hence, we can store a configuration C as an unordered multiset $\hat{C}$ and maintain multiplicities rather than individual states. To this end, SEQ requires an *urn-like* data structure which efficiently supports (i) weighted sampling (with and without replacement) and (ii) adding of single agents. In the following, we consider various data structures and their impact on the complexity of the sequential approach.

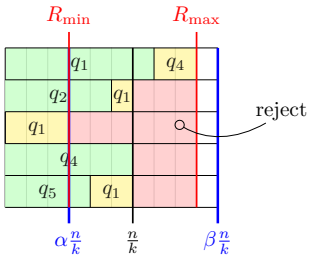
Array. $\text{SEQ}_{\mathsf{Array}}$ maintains the configuration C in an array $A[1 \dots n]$ where $A[i]$ holds s_i, the state of the i-th agent. Sampling with replacement is trivial, as we only draw a uniform variate $X \in [n]$ and return $A[X]$. Sampling without replacement works analogously: we overwrite $A[i]$ with $A[n]$ and remove the array's last element $A[n]$. Adding new elements is possible by appending. (Note that we do not grow the memory since we always store at most n agents in the array.) This leads to an $\mathrm{O}(N)$ time algorithm and a memory footprint of $\mathrm{O}(n \log |Q|)$ bits, which can be prohibitively large if simulating large populations in parallel.

Linear Search. $\text{SEQ}_{\mathsf{Linear}}$ maintains the multiset $\hat{C}$ in an array A such that $A[i]$ holds the number of agents in state q_i. Sampling requires a linear search on A in $\mathrm{O}(|Q|)$ per sample. This results in a worst-case simulation time of $\Theta(N|Q|)$. Nevertheless, in practice $\text{SEQ}_{\mathsf{Linear}}$ is among the fastest sequential variants for small $|Q|$ (see Section 7). Compared to $\text{SEQ}_{\mathsf{Array}}$, it has a significantly smaller memory footprint of $\mathrm{O}(|Q| \log n)$ bits.

Binary Search Tree. $\text{SEQ}_{\mathsf{BST}}$ maintains the multiset $\hat{C}$ using a balanced binary search tree. The i-th leaf (from left to right) encodes $\hat{C}_i$, the number of agents in state i. Each inner node v stores the number ℓ_v of agents in its left subtree. To randomly sample an agent, we draw an integer X from $\{0, \dots, n-1\}$ uniformly at random and compare it to the root's value ℓ_r. If $X < \ell_r$, the sample is in the interval covered by the left sub-tree, and we descend accordingly. Otherwise, we update $X \leftarrow X - \ell_r$ and descend into the right subtree. We recurse until some leaf i is reached, where we emit an agent of state i.

Each operation on the tree involves a simple path from the root to a leaf of length $\Theta(\log |Q|)$. Since the work per level is constant, all operations take $\Theta(\log |Q|)$ time. Thus, $\text{SEQ}_{\mathsf{BST}}$ requires $\Theta(N \log |Q|)$ total time and $\mathrm{O}(|Q| \log n)$ bits of memory.

$\text{SEQ}_{\mathsf{Alias}}$ combines the linear runtime of $\text{SEQ}_{\mathsf{Array}}$ (w.h.p.) with the small memory footprint of $\text{SEQ}_{\mathsf{BST}}$, provided $|Q| < \sqrt{n}$. At the heart of $\text{SEQ}_{\mathsf{Alias}}$ lies a *Dynamic Alias Table* introduced in the following section.



$F[i]$	$S[i]$	$A[i]$	rejection probability
7	2	q_4	0
5	1	q_1	$3/9$
0	3	q_1	$6/9$
6	0	q_1	$3/9$
4	2	q_1	$3/9$

Figure 1 Dynamic Alias Table storing $\hat{C} = (q_1 : 13,\ q_2 : 5,\ q_3 : 0,\ q_4 : 8,\ q_5 : 4)$, i.e., $n = 30$ and $k = 5$. This imbalanced configuration will soon need rebuilding, e.g., after the next decrease of q_1 in row 3, or after adding two more agents in state q_1 in row 1.

3 Dynamic Alias Tables

Dynamic Alias Tables. In this section we introduce our Dynamic Alias Table data structure. Our goal is to model an urn which initially contains n marbles, each of which has one of k possible colors. We assume that the colors are identified by numbers in $\{1, \ldots, k\}$. The urn defines a probability distribution D for the color of a marble drawn uniformly at random: let p_i be the probability that we sample a marble of color i.

Original Alias Method. The alias method [41] uses a table with two columns and k rows, one row for each element (color) in the distribution D. Each row i has two entries corresponding to two elements. Each element has a weight in $[0, 1]$, and the two weights sum up to 1 in each row. The first element of row i is always element i. It has weight $F[i]$. The second element of row i is stored in $A[i]$. It has a weight of $1 - F[i]$. This means that the original alias method uses only the two arrays, F and A, to store the distribution $p_1, \ldots, p_k$.

To sample from D in the original alias method, we first sample a row i uniformly at random from $\{1, \ldots, k\}$. Then we draw a random real $X \in [0, 1)$. If $X < F[i]$, we return the left element, i. Otherwise, we return the right element, $A[i]$. In the following, we modify the alias method and call the resulting data structure Dynamic Alias Table.

Dynamic Alias Tables. Recall that we assume that our distribution corresponds to an urn storing n marbles of k different colors. First, we explicitly add a second weight array $S[i]$ which stores the weight of the alias. Now instead of storing just one real value $F[i]$ for each row i, we store the exact numbers of marbles as integers for the first and the second entry of row i in $F[i]$ and $S[i]$, respectively (cf. generalized distributions of [30]). As before, the first entry of row i corresponds to color i and the second entry of row i corresponds to color $A[i]$. As illustrated in Figure 1, the rows are constructed in such a way that the total weight of each row no longer adds up to the real value 1, but to the integer value $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$ such that all rows in total add up to n.

▶ **Observation 2.** *Let U be a Dynamic Alias Table encoding an urn with n marbles and k colors. The data structure U can be constructed in $O(k)$ time.*

Proof. The algorithm by Vose [40] can generate an (original) alias table representation of such a discrete probability distribution D in $O(k)$ time. It is straightforward to define a mapping between the weights of the original alias method as computed in [40] and the two integer values used in our Dynamic Alias Table. It follows that the Dynamic Alias Table (using integer weights) can be constructed in $O(k)$ time. ◀

Updating and Sampling from Dynamic Alias Tables. Our modified data structure now allows us to sample elements with and without replacement. Let $R_i = F[i] + S[i]$ denote the weight of row i and define $R_{\min}$ and $R_{\max}$ as smallest and largest row weights, respectively. Observe that in general $R_{\min} \neq R_{\max}$.

In order to sample from U, we first select row i uniformly at random from $\{1, \ldots, k\}$. Then we draw a uniform variate X from $\{0, \ldots, R_{\max} - 1\}$.[1] There are three possible events: If $X < F[i]$, we emit the first element i. If $F[i] \leq X < R_i$, we emit the second element $A[i]$. Otherwise, we reject the trial and restart the sampling process.

If we sample from U without replacement, we decrement the weight of the element just sampled. This is always possible, since only elements with strictly positive weights can be sampled in the first place. If we add a new element with color i to U, we increment the weight of the first element of row i.

In order to guarantee expected constant sampling time, we ensure that the fraction between $R_{\min}$ and $R_{\max}$ does not exceed a certain value. Let $0 < \alpha < 1$ and $\beta > 1$ be two parameters chosen such that $\beta/\alpha = \mathrm{O}(1)$. After each update to U we require

$$\alpha\lfloor n/k \rfloor \leq R_{\min} \leq R_{max} \leq \beta\lceil n/k \rceil. \tag{1}$$

Otherwise, we rebuild the data structure in $\mathrm{O}(k)$ time.

We are now ready to show Theorem 1.

Proof of Theorem 1. We start with the memory complexity. The Dynamic Alias Table U stores the values of k, n, and $R_{\max}$ as well as three arrays. Array $F[1 \ldots k]$ stores the weight of the first column, array $S[1 \ldots k]$ stores the weight of the second column, and array $A[1 \ldots k]$ stores the alias, i.e., the element of the second column. All entries are integers from $\{0, \ldots, n\}$ (recall that we assume $k \leq \sqrt{n}$). Thus, the Dynamic Alias Table requires $\Theta(k \log n)$ bits of memory.

Let us now consider the sampling procedure. First, we consider the rejection probability. Recall that we first sample a row i and then draw a uniform variate X from $\{0, \ldots, R_{\max} - 1\}$. As before, we denote the total weight of row i as R_i with $R_i = F[i] + S[i]$. A sampling trial in row i is rejected if $X \geq R_i$, i.e., with probability $R_i/R_{\max}$. Therefore, the probability to reject a sample from any row is at most $R_{\min}/R_{\max}$. From the conditions in Equation (1) we get that the rejection probability is at most $\alpha/\beta = \mathrm{O}(1)$ and, conversely, we have at least a constant success probability of $(\beta - \alpha)/\beta$. The number of trials until we emit an element is therefore geometrically distributed and has an expected value of at most $\beta/(\beta - \alpha) = \mathrm{O}(1)$.

It remains to show that we emit an element of color i with probability $p_i = \hat{C}_i/n$, where $\hat{C}_i$ is the number of marbles of color i in the Dynamic Alias Table U. We consider a single sampling trial. Observe that in each trial we are given a uniform probability space $\mathbf{\Omega} = \{(i, x) \ : \ 1 \leq i \leq k \text{ and } 0 \leq x < R_{\max}\}$. From this probability space we draw the row i and the value X uniformly at random. Fix a color c and let $\mathcal{S}_c$ be the set of all events (i, x) which lead to emission of an element of color c for this probability space $\mathbf{\Omega}$. An event (i, x) is in $\mathcal{S}_c$ if and only if (i) $i = c$ and $x < F[i]$ or (ii) $A[i] = c$ and $F[i] \leq x < F[i] + S[i]$.

The Dynamic Alias Table U is constructed such that the total weight for each color c always equals $\hat{C}_c$. Therefore, counting all elementary events gives us $|\mathcal{S}_c| = \hat{C}_c$. Observe that $\mathbf{\Omega}$ is a uniform probability space since the row i and the value X are drawn uniformly. It has size $|\mathbf{\Omega}| = kR_{\max}$. Hence, all events in $\mathcal{S}_c$ have equal probability $1/(kR_{\max})$, and we get $\Pr[\mathcal{S}_c] = |\mathcal{S}_c|/(kR_{\max}) = \hat{C}_c/(kR_{\max})$.

[1] Observe that in a practical implementation one can draw a single uniform variate U' from $\{1, \ldots, k \cdot R_{\max} - 1\}$, and derive U and X from it.

Let $\mathcal{R}$ be the event that a trial is rejected. Analogously to before, we enumerate over all elementary events and obtain $|\mathcal{R}| = \sum_i (R_{\max} - R_i) = kR_{\max} - n$. For the complementary event $\overline{\mathcal{R}}$ we get $|\overline{\mathcal{R}}| = n$, which matches the intuition that the urn contains n marbles. Hence, we have $\Pr[\overline{\mathcal{R}}] = n/(kR_{\max})$. Observe that $\mathcal{S}_c$ and $\mathcal{R}$ are mutually exclusive and hence $\mathcal{S}_c \cap \overline{\mathcal{R}} = \mathcal{S}_c \setminus \mathcal{R} = \mathcal{S}_c$.

Rejected trials emit no element, but are repeated. Hence, we condition on $\overline{\mathcal{R}}$ and obtain

$$p_c = \Pr[\mathcal{S}_c \mid \overline{\mathcal{R}}] = \frac{\Pr[\mathcal{S}_c \cap \overline{\mathcal{R}}]}{\Pr[\overline{\mathcal{R}}]} = \frac{\Pr[S_c]}{\Pr[\overline{\mathcal{R}}]} = \frac{\hat{C}_c}{kR_{\max}} \cdot \frac{kR_{\max}}{n} = \frac{\hat{C}_c}{n}.$$

This means that a color c is indeed emitted with the correct probability $p_c = \hat{C}_c/n$.

Finally, we consider the amortized costs of rebuilding the Dynamic Alias Table U ever so often. Observe that U has to be rebuilt whenever the condition in Equation (1) is violated. This can happen in two possible ways.

- Case 1: $R_{\min} < \alpha\lfloor n/k \rfloor$.
  In this case there must exist a row i for which $R_i < \alpha\lfloor n/k \rfloor$. Observe that by assumption of the theorem we have $n \geq k^2$, and after rebuilding U we have $R_{\min} = \lfloor n/k \rfloor$. In order to have $R_i < \alpha\lfloor n/k \rfloor$, at least $\lfloor n/k \rfloor - \alpha\lfloor n/k \rfloor \geq k(1-\alpha)$ elements must have been deleted from row i. As $0 < \alpha < 1$, this happens only after at least $k(1-\alpha) = \Omega(k)$ sampling operations. Now according to Observation 2, rebuilding takes time $\mathrm{O}(k)$. Together this implies that rebuilding U takes amortized constant time per update of U.
- Case 2: $R_{\max} > \beta\lceil n/k \rceil$.
  The second case follows analogously to the first case. If $R_{\max} > \beta\lceil n/k \rceil$, at least $\beta\lceil n/k \rceil - \lceil n/k \rceil \geq k(\beta-1)$ elements must have been added. This takes at least $k(\beta-1) = \Omega(k)$ insertions, and hence rebuilding U takes amortized constant time per insertion.

Removal and insertion operations can be arbitrarily mixed and interact only beneficially towards the amortization arguments. This concludes the proof of the theorem. ◀

4 Batch Processing

So far, we discussed algorithms to simulate a population protocol step-by-step. These simulators can output the population's configuration $C(t)$ for each time step $1 \leq t \leq N$. With a time complexity of $\mathrm{O}(N)$, the simulators $\textsc{Seq}_{\mathsf{Array}}$ and $\textsc{Seq}_{\mathsf{Alias}}$ are optimal in this sense. In practice, however, it often suffices to obtain a configuration snapshot every $\Theta(n)$ steps. In this setting, we can achieve sub-constant work per interaction under mild assumptions.

Recall that $\textsc{Seq}_{\mathsf{BST}}$ has a small memory footprint but a sub-optimal time complexity of $\Theta(N \log |Q|)$. Observe, however, that the underlying binary search tree can update the multiplicity of any existing state in time $\mathrm{O}(\log |Q|)$ *independently* of the changed quantity. Here, we introduce the new algorithm Batched (see Algorithm 2) to exploit this observation. The algorithm uses a binary search tree to store the configuration. It updates $\Omega(\sqrt{n})$ agents in expectation with each access and therefore reduces the time complexity to $\mathrm{O}(N(\log n + |Q|^2)/\sqrt{n})$ which is $\mathrm{o}(N)$ for $|Q| = \mathrm{o}(n^{1/4})$ and $N = \Theta(\mathrm{poly}(n))$.

Batching interactions. In order to coalesce individual updates into batches, Batched uses the notion of *collision-free runs* as illustrated in Figure 2. We interpret the execution of a protocol as a sequence $i_1, i_2, \ldots$ where at time t agents i_{2t-1} and i_{2t} interact.

Let ℓ be the largest index such that all $i_1, \ldots, i_\ell$ are distinct. Then, the first $\lfloor \ell/2 \rfloor$ interactions are independent of each other and can be rearranged in any order. We refer to them as a collision-free run of length ℓ. If ℓ is odd, the first agent of the $(\lfloor \ell/2 \rfloor + 1)$-th

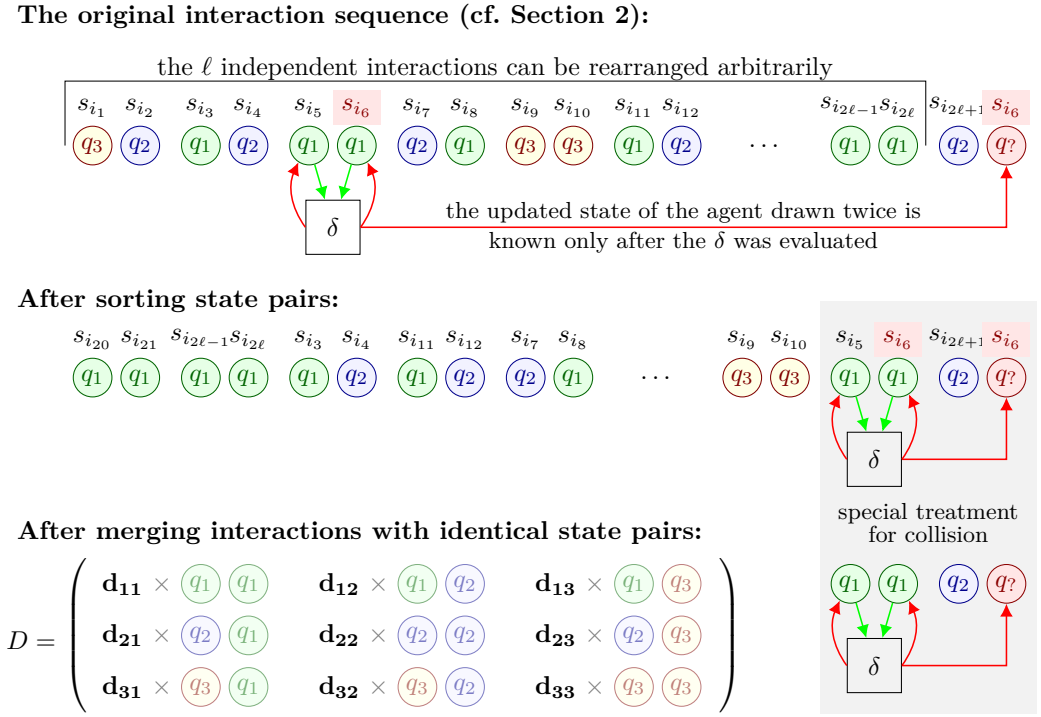


Figure 2 Batch processing uses collision-free runs, long sequences of independent interactions, which can be rearranged and grouped together.

interaction is also considered collision-free. Since we are free to reorder the interactions, we can group all interactions of states (q_i, q_j) together, evaluate $\delta(q_i, q_j)$, and update all accordingly affected states in one step.

Now instead of sampling a sequence of agents and partitioning the sequence into collision-free runs, we take the opposite direction. We first sample only the length ℓ of a collision-free run from the appropriate probability distribution (see below). Then, we randomly match ℓ agents as discussed below. Finally, we reuse one of the agents from the matching in order to plant a collision. These steps are repeated until at least N interactions are simulated.

Matching Agents. We simulate sampling ℓ agents without replacement to construct a collision-free run of length ℓ. While we cannot afford to draw the agents individually, we only need to know how many interactions n_{ij} of each state pair (q_i, q_j) we encountered. Thus, a run can be modeled by a $|Q|\times|Q|$ matrix $D = (n_{ij})$ with $\sum_{ij} n_{ij} = \lfloor \ell/2 \rfloor$. (If ℓ is odd, we remove one agent and treat it individually.)

To obtain D, we first sample the row sums $D_i = \sum_j n_{ij}$ of the matrix from a multivariate hypergeometric distribution. This simulates sampling $\lfloor \ell/2 \rfloor$ initiating agents without replacement. We then sample values within each row analogously to find the matching responding agents. Sampling D takes $\mathrm{O}(|Q|^2)$ time in total since each individual sample from a hypergeometric distribution can be computed in $\mathrm{O}(1)$ time [39].

For correctness, note that our sampling approach corresponds to first selecting $\lfloor \ell/2 \rfloor$ agents as initiators and then $\lfloor \ell/2 \rfloor$ agents as responders. That is, we first sample agents $i_1, i_3, \ldots, i_{2\lfloor \ell/2 \rfloor - 1}$ and then agents $i_2, i_4, \ldots, i_{2\lfloor \ell/2 \rfloor}$ (instead of the natural interleaved variant $i_1, i_2, \ldots, i_{2\lfloor \ell/2 \rfloor}$). Since, each draw is taken uniformly at random, the permutation does not change the distribution (see full version [17] for a formal proof).

Algorithm 2 BATCHED: The algorithmic framework for simulation in batches.

input: configuration C, transition function δ, number of steps N
$t \leftarrow 0$
while $t < N$ **do**
  $\ell \leftarrow$ sample length of a collision-free run
  let $D = (d_{ij})$ be a $|Q| \times |Q|$ matrix and sample d_{ij} as the number of interactions (q_i, q_j) among ℓ interactions ▷ `batch processing`
  let C' be an empty configuration
  foreach $(q_i, q_j) \in Q^2$ **do**
    remove from C: d_{ij} agents in states q_i, and d_{ij} agents in states q_j
    $(q_i', q_j') \leftarrow \delta(q_i, q_j)$
    add to C': d_{ij} agents in states q_i', and d_{ij} agents in states q_j'
  if ℓ *is even* **then** ▷ `plant a collision`
    sample agent c_1 without replacement from C' ▷ `collision at` c_1
    merge C' into C
    sample agent c_2 without replacement from C
  else
    sample agent c_1 without replacement from C
    sample agent c_2 without replacement from C' ▷ `collision at` c_2
    merge C' into C
  add agents $\delta(c_1, c_2)$ to C
  $t \leftarrow t + \ell + 1$

Length of a Collision-Free Run. In the following, we analyze the length ℓ of a collision-free run. Observe that the following analysis is similar to the analysis of a generalized variant of the birthday problem [32]. We consider a generalization which we also use in Section 5. We assume that r agents have already interacted and ask how many more collision-free agents can be added. Formally we define the distribution $\mathsf{coll}(n, r)$ as follows.

▶ **Definition 3.** Consider a sequence $a_1, a_2, \ldots$ of agents sampled independently and uniformly at random. Let A_0 be a set of r initially prescribed agents and let $A_i = A_{i-1} \cup \{a_i\}$ be the set of agents after i draws. We define the random variable ℓ as the smallest index s.t. $a_\ell \in A_{\ell-1}$. We say $\ell \sim \mathsf{coll}(n, r)$, where n is the total number of agents and r is the number of prescribed agents.

In Section 6.1 we discuss how we can sample from this distribution using the inverse sampling technique [23]. We find that sampling ℓ takes $\mathrm{O}(\log n)$ time. In practice, this is comparable to the time it takes to sample a hypergeometric random variate. In the following, we show basic properties of $\mathsf{coll}(n, r)$ in order to show bounds on the runtime of BATCHED.

▶ **Lemma 4.** *Let $\ell \sim \mathsf{coll}(n, r)$. Then ℓ has support $\{1, \ldots, n - r\}$ and distribution* $\Pr[\ell = k] = n^{-(k+1)}(n - r)!(r + k)/(n - r - k)!$.

Proof. Consider an urn with n marbles. Initially, r marbles are red, while the remaining $n - r$ marbles are green. We now take out one marble at a time: if it is green, we keep on going (think of a traffic light) and put a red one back in. If we take a red marble, we stop. Observe that the number of marbles we take out is exactly ℓ as above, as the green marbles represent new unconsidered agents while the red ones represent agents in $A_{\ell-1}$.
This directly leads to the acclaimed distribution:

$$\Pr[\ell = k] = \underbrace{\prod_{i=0}^{k-1} \frac{(n-r)-i}{n}}_{\text{select } k \text{ out of } n-r} \cdot \underbrace{\frac{r+k}{n}}_{(k+1)\text{-th is red}}$$

◀

▶ **Lemma 5.** *Let $\ell \sim \mathsf{coll}(n, 0)$. Then $\mathbb{E}[\ell] = \Theta(\sqrt{n})$.*

Proof. We first upper bound $\mathbb{E}[\ell] = \mathrm{O}(\sqrt{n})$ and then give a matching lower bound $\mathbb{E}[\ell] = \Omega(\sqrt{n})$. In both cases, we write $\mathbb{E}[\ell] = \sum_{i=0}^{n} \Pr[\ell \geq i]$ and split the sum at $\sqrt{n}$. Then we bound both terms appropriately. Observe that for some fixed value i we have $\Pr[\ell \geq i] = \prod_{j=0}^{i-1}(1 - j/n)$. For the upper bound on $\mathbb{E}[\ell]$ we get

$$\mathbb{E}[\ell] = \sum_{i=0}^{n} \Pr[\ell \geq i] = \sum_{i=0}^{n} \prod_{j=0}^{i-1} \left(1 - \frac{j}{n}\right) \leq \sum_{i=0}^{\sqrt{n}-1} 1 + \sum_{i=\sqrt{n}}^{\infty} \left(1 - \frac{\sqrt{n}}{n}\right)^{i} \leq 2\sqrt{n}.$$

Similarly, we get for the lower bound on $\mathbb{E}[\ell]$ that

$$\begin{aligned}\mathbb{E}[\ell] &= \sum_{i=0}^{n} \Pr[\ell \geq i] = \sum_{i=0}^{n} \prod_{j=0}^{i-1} \left(1 - \frac{j}{n}\right) \geq \sum_{i=0}^{\sqrt{n}} \prod_{j=0}^{i-1} \left(1 - \frac{\sqrt{n}}{n}\right) = \sum_{i=0}^{\sqrt{n}} \left(1 - \frac{1}{\sqrt{n}}\right)^{i} \\ &= \sqrt{n}\left(1 - \left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}+1}\right) \geq \sqrt{n}\left(1 - e^{-1}\right).\end{aligned}$$

Therefore we have $\mathbb{E}[\ell] = \Theta(\sqrt{n})$. ◀

Using Lemma 5, we are now ready to bound the runtime and space complexity of Batched.

▶ **Theorem 6.** *Let n be the number of agents and $|Q|$ the number of states. Batched simulates N interactions in $\mathrm{O}(N(|Q|^2 + \log n)/\sqrt{n})$ expected time using $\Theta(|Q| \log n)$ bits.*

Proof. According to Lemma 5, each batch simulates $\Theta(\sqrt{n})$ interactions in expectation. It takes $\mathrm{O}(\log n)$ time to sample the length of a collision-free run ℓ (see Section 6.1) and $\mathrm{O}(|Q|^2)$ time (cf. [39]) to sample the interaction numbers and process the interactions for all pairs of states. This implies the runtime complexity. The space complexity follows immediately from the binary search tree used to store the configuration. ◀

5 Merging Batches

In an empirical evaluation, we found that our implementation of algorithm Batched spends most time in the batch processing step (to sample and transition the $|Q| \times |Q|$ matrix D); this is especially true for complex protocols with non-trivial state space sizes. As the matrix sampling cost is independent of the length ℓ of the underlying collision-free run, we modify the algorithm to support more than one collision per batch processing step.

Introducing Epochs. An execution of the improved algorithm MultiBatched logically consists of several epochs. For each epoch, the algorithm samples the *lengths* $\ell_1, \ell_2, \ldots, \ell_\rho$ of multiple collision-free runs $R_1, \ldots, R_\rho$. As no agent may appear twice in the union of those collision-free sequences, later runs become shorter in expectation ($\mathbb{E}[\ell_{i+1}] < \mathbb{E}[\ell_i]$), naturally limiting the number ρ of runs per epoch. After each run R_i, we plant one collision, i.e., an interaction with an agent that was already considered in the current epoch. An epoch concludes with a single batch processing step, in which matrix D is sampled and processed analogously to algorithm Batched.

Tracking Dependencies. While algorithm Batched only reorders and groups together independent interactions, our improved algorithm MultiBatched delays most interactions until the end of the epoch. To do so, the algorithm conceptually assigns each agent one of three types, and updates these labels as it progresses through the epoch:

- untouched agents did not interact in the current epoch. Hence, all agents are labeled untouched at the beginning of an epoch.
- updated agents took part in at least one interaction that was already evaluated. Thus, updated agents are already assigned their most recent state.
- delayed agents took part in *exactly one* interaction that was not yet evaluated. Thus, delayed agents are still in the same state they had at the beginning of the epoch, but are scheduled to interact at a later point in time. We additionally require that their interaction partner is also labeled delayed.

Analogously to algorithm Batched, we maintain two urns C and C'. Urn C' contains updated agents, while urn C stores untouched and delayed agents (or in other words, all agents whose state was not updated in the current epoch). At any point in time, an agent is either in C or C' meaning that $|C| + |C'| = n$. Due to symmetry, we do not explicitly differentiate untouched from delayed agents. We rather maintain only the number T of delayed agents and lazily select them while planting collisions or during batch processing.

If a delayed agent a is selected while planting a collision, it takes part in a second interaction and – by definition – cannot be labeled delayed any more. Thus, we randomly draw a second delayed agent b, evaluate their transition, store the updated state of b in C', and directly evaluate a again in the planted collision. Finally, we decrease $T \leftarrow T - 2$ as agents a and b changed their labels from delayed to updated. Observe that we might repeat this step in the (unlikely) case that a planted collision involved two formerly delayed agents.

Length of an Epoch. We now analyze the length of an epoch. We start by extending the analysis of $\mathsf{coll}(n, r)$ to the $r = \Omega(\sqrt{n})$ regime (reached after $\mathrm{O}(1)$ runs w.h.p.). The following lemmas establish expected value and concentration.

▶ **Lemma 7.** *Let $\ell \sim \mathsf{coll}(n, r)$ and $r = \Omega(\sqrt{n})$. Then $\mathbb{E}[\ell] = \Theta(n/r)$.*

Proof. The proof follows analogously to Lemma 5. Again, we start with the upper bound.

$$\mathbb{E}[\ell] = \sum_{i=0}^{n-r} \Pr[\ell \geq i] = \sum_{i=0}^{n-r} \prod_{j=0}^{i-1} \left(1 - \frac{j+r}{n}\right) \leq \sum_{i=0}^{\infty} \left(1 - \frac{r}{n}\right)^i = \frac{n}{r}.$$

For the lower bound we derive a general result for arbitrary r.

$$\begin{aligned}\mathbb{E}[\ell] &= \sum_{i=0}^{n-r} \Pr[\ell \geq i] = \sum_{i=0}^{n-r} \prod_{j=0}^{i-1} \left(1 - \frac{j+r}{n}\right) \geq \sum_{i=0}^{r-1} \prod_{j=0}^{i-1} \left(1 - \frac{j+r}{n}\right) \geq \sum_{i=0}^{r-1} \left(1 - \frac{2r}{n}\right)^i \\ &= \frac{n}{2r}\left(1 - \left(1 - \frac{2r}{n}\right)^r\right) \geq \frac{n}{2r}(1 - e^{-2r^2/n}).\end{aligned}$$

The last inequality holds since $e^{-2r^2/n}$ constitutes an upper bound on $(1 - 2r/n)^r$ as it can be rewritten as $(1 - 2r/n)^{n \cdot r/n}$ and $(1 - 2r/n)^n \leq e^{-2r}$. For $r = \Omega(\sqrt{n})$ the second factor $(1 - \exp(-2r^2/n))$ is $\Omega(1)$ which proves the claim. ◀

▶ **Lemma 8.** *Let $\ell \sim \mathsf{coll}(n, r)$ and $r = \Omega(\sqrt{n})$. Then $\ell = \Theta(n/r)$ with probability $1 - o(1)$.*

Proof. We prove the claim by showing that $\Pr[\ell < t]$ and $\Pr[\ell > t]$ are $o(1)$ for $t = o(n/r)$ and $t = \omega(n/r)$, respectively. We have

$$\Pr[\ell < t] = 1 - \Pr[\ell \geq t] = 1 - \prod_{i=0}^{t-1}\left(1 - \frac{i+r}{n}\right) \leq 1 - \left(1 - \sum_{i=0}^{t-1}\frac{i+r}{n}\right) \leq \frac{2tr + t^2}{2n},$$

where the first inequality follows from the Weierstrass product inequality. Furthermore, with $r = \Omega(\sqrt{n})$ we have $n/r = O(\sqrt{n})$ and thus $t = o(n/r)$ such that $t^2 = o(n)$ and $tr = o(n)$. For the second part, we have

$$\Pr[\ell > t] = \prod_{i=0}^{t}\left(1 - \frac{i+r}{n}\right) \leq \left(1 - \frac{r}{n}\right)^t \leq e^{-rt/n},$$

and for $t = \omega(n/r)$ and thus $rt/n = \omega(1)$ the claim follows. ◀

Intuitively, Lemma 7 shows that for sufficiently many prescribed agents r, the probability of drawing a colliding agent remains approximately r/n throughout the run. Similar to a geometric distribution, this results in a concentrated expected length of $\Theta(n/r)$. We now estimate the number of agents sampled after ρ runs.

▶ **Lemma 9.** *Let $L_k = \sum_{i=1}^{k} \ell_i$ be the number of agents drawn in an epoch with k runs. Then, for $|Q| = \omega(\sqrt{\log n})$, $|Q| = o(\sqrt{n \log n})$ and $\rho = \mathrm{O}(|Q|^2/\log n)$ we have $\mathbb{E}[L_\rho] = \Theta(\sqrt{\rho n})$.*

Proof. The variable L_k equivalently corresponds to the number of marbles $B(k, n)$ that need to be drawn in the birthday problem s.t. k coincidences occur. The asymptotics of $\mathbb{E}[B(k, n)]$ have first been studied by Kuhn and Struik [32] for the cases that $k = o(n^{1/4})$. Their results have since been improved by Arratia et al. [11] where the asymptotic bounds on the moments of $B(k, n)$ have been calculated for more general conditions on k. By [11, Corollary 12] for k a function of n, i.e., $k = k_n$ where $k_n \to \infty$ and $k_n/n \to 0$ it holds that

$$\mathbb{E}[B(k_n, n)] \sim \sqrt{2nk_n} \quad \text{as } n \to \infty.$$

By assumption the conditions are met since $\rho = \omega(1)$ and $\rho = o(n)$, thus $\mathbb{E}[L_\rho] = \Theta(|Q|\sqrt{n/\log n}) = \Theta(\sqrt{\rho n})$. ◀

Complexity. In order to analyze MultiBatched's runtime, we first establish the time required per epoch, and then bound the total expected runtime and memory requirements.

▶ **Lemma 10.** *MultiBatched takes time $\mathrm{O}(\rho \log n + |Q|^2)$ for an epoch of ρ runs.*

Proof. Planting a collision is done by drawing the two interacting agents from the appropriate urns and setting them to be updated which requires $\mathrm{O}(1)$ time. Sampling the length of a single collision-free run takes time $\mathrm{O}(\log n)$ (see Section 6.1). For the final batch-processing step MultiBatched takes $\Theta(|Q|^2)$ time independently of the number of delayed agents. ◀

▶ **Theorem 11.** *Let n be the number of agents and $|Q|$ the number of states. MultiBatched simulates N interactions in $\mathrm{O}(N|Q|/\sqrt{n/\log n})$ expected time if $|Q| = \omega(\sqrt{\log n})$ and $|Q| = \mathrm{o}(\sqrt{n \log n})$.*

Proof. Combining Lemmas 9 and 10, we find a runtime of $\mathrm{O}(N(\rho \log n + |Q|^2)/\sqrt{\rho n})$. Setting $\rho = \Theta(|Q|^2/\log n)$ balances the cost of sampling runs and planting collisions with the cost of batch processing, and thus does not increase the asymptotic cost per epoch. Higher values of ρ only increase the expected time complexity. ◀

MultiBatched has sub-constant work per interaction for $|Q| = \mathrm{o}\left(\sqrt{\frac{n}{\log n}}\right)$ and $N = \Theta(\mathrm{poly}\, n)$.

6 Heuristics and Implementation Details

Implementations of all discussed simulators (including scripts to reproduce figures and numbers included in this paper) are freely available. In the following, we highlight important aspects necessary to obtain simulators that are both fast in practice and highly customizable.

All simulators are implemented in C++ and use compile-time specializations to implement specific protocols and experimental setups.

In contrast to pure deterministic functions, non-deterministic transition functions (possibly with side-effects) have to be informed about every interaction carried out.[2] To allow for batch processing, we cannot use the natural invocation order. Instead, we inform the protocol how often a state pair will interact within an epoch. It is then expected to assign all participating agents to the appropriate states. See the full version of this paper [17] for more details and listings.

6.1 Sampling the Length of a Collision-Free Run

Recall that Batched and MultiBatched repeatedly sample the length ℓ of a collision-free run. In the following we discuss how this sampling can be implemented using the inverse sampling technique (Ist) [23]: let $\mathrm{cdf}(x)$ be the cumulative density function of a target distribution. Then, Ist draws a uniform variate U from $[0;1]$, solves $U = \mathrm{cdf}(x)$ for x, and returns it as the sample. We denote the CDF of $\mathsf{coll}(n,k)$ as $F_{n,k}(t)$. Lemma 4 yields:

$$1 - F_{n,k}(t) = \Pr[\ell > t] = \prod_{i}^{t} \frac{n-k-i}{n} = \frac{1}{n^t} \frac{(n-k)!}{(n-k-t-1)!} \overset{(x-1)!=\Gamma(x)}{=} n^{-t} \frac{\Gamma(n-k+1)}{\Gamma(n-k-t)}.$$

Since we are not aware of an inverse that can be evaluated fast, we numerically solve $U = F_{n,k}(t)$ for t. To avoid numerical instabilities, we rewrite the expression in terms of $\log\Gamma(x)$, which is available as the C standard function `lgamma`(x):

$$U = 1 - n^{-t} \frac{\Gamma(n-k+1)}{\Gamma(n-k-t)} \quad \Leftrightarrow \quad \log(1-U) = \log\Gamma(n-k+1) - \log\Gamma(n-k-t) - t\log n$$

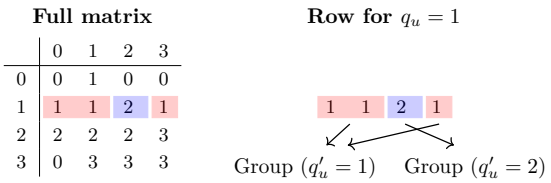
Lacking a cheap derivative of the RHS, we rely on first-order numerical inversion methods only. In this context, an ad-hoc combination of binary search and regula-falsi gave most consistent results. We jump-start the search using a small look-up table containing lower and upper bounds on t for intervals of U and k.

While the method requires $\mathrm{O}(\log n)$ evaluations of $F_{n,k}(\cdot)$, we observe less than ten calls on average for $n = 2^{50}$. The resulting sampling algorithm has a practical runtime comparable to the sampling of hypergeometric random variates. Since the latter is sampled much more frequently, further optimizations will yield limited results to the total runtimes of Batched and MultiBatched.

6.2 Heuristics

The frequent sampling of hypergeometric random variates, dominates MultiBatched's runtime. In the following, we discuss three heuristics to reduce this number.

[2] Observe that many protocols with non-deterministic transition functions have been derandomized, see, e.g., the notion of (biased) synthetic coins in [1, 18]. While this is supported by the simulator, we feel it is more convenient to offer the most expressive interface possible.



	0	1	2	3
0	0	1	0	0
1	1	1	2	1
2	2	2	2	3
3	0	3	3	3

Figure 3 Simplified transition matrix Δ' for a clock with period m=4. The initiator's phase (row) is circularly incremented only when matched with a suitable responder (column).

Renaming. In the *renaming heuristic* we exploit the observation that agents are typically not uniformly distributed over all states. Instead there are often sparsely populated states which are seldom hit when sampling an agent.

It can be beneficial to consider these states last. SEQ$_{\mathsf{Linear}}$'s linear search, for instance, stops as soon as the sampled state is found. The same is true when sampling BATCHED's and MULTIBATCHED's interaction matrices: for row i, we draw D_i agents from a multivariate hypergeometric distribution. This is implemented by obtaining $|Q|-1$ properly parametrized hypergeometric variates; the process terminates early once all D_i agents have been sampled.

In both examples, we maximize the probability of early stopping by considering highly populated states first. To this end, we maintain a permutation $\pi\colon [|Q|] \to [|Q|]$ that sorts states decreasingly by their sizes. We then process states in the order indicated by π. If this permutation is updated once every $\Omega(|Q| \log |Q|)$ interactions, the sorting step becomes asymptotically negligible for sequential simulators. For BATCHED and MULTIBATCHED, π can be updated once every $\Omega(\log q)$ batches.

Partitioning. If δ is a deterministic function, we can model it as a matrix $\Delta \in (Q \times Q)^{|Q|\times|Q|}$. The matrix of a deterministic one-way protocol can be further simplified to $\Delta' \in Q^{|Q|\times|Q|}$ since the states of the responders remain unchanged.

For many meaningful protocols, the entries of Δ' are not random but exhibit some structure. As an example, consider the simplified[3] phase-clock transition matrix illustrated in Figure 3. Here, each row contains only two different output states. The *partitioning heuristic* uses this observation during the batch steps of BATCHED and MULTIBATCHED. When sampling D_i responders for initiators in state q_i, we group together all entries in the i-th row of Δ' that assign the same new state to the initiating agents. It then suffices to draw one random hypergeometric variate per group.

Note that the heuristic does not reduce the runtime complexity of the algorithm – even if we precompute the partitioning. This is due the fact that we still need to compute the population sizes for each group which involves $\Theta(|Q|)$ additions per row. For pathological protocols, the number of hypergeometric random variates required for each row remains $\Omega(|Q|)$. A simple worst-case protocol is the transition function $\delta(q_u, q_v) = (q_v, q_v)$.

Skipping. Generalizing *partitioning heuristic* to two-way protocols tends to be ineffective in practice: since initiator and responder may both update their states, the transition matrix is often more fragmented. The partitioning overhead then easily exceeds the potential savings. For such protocols MULTIBATCHED uses a coarser partitioning, and only detects and skips transitions that preserve the configuration (i.e., $\delta(q_u, q_v) = (q_u, q_v)$ or $\delta(q_u, q_v) = (q_v, q_u)$).

[3] Formally the states of the phase-clock are pairs (x, b) where x represents the *phase*, and b marks an agent as *leader*. For the sake of simplicity we assume that all agents are followers.

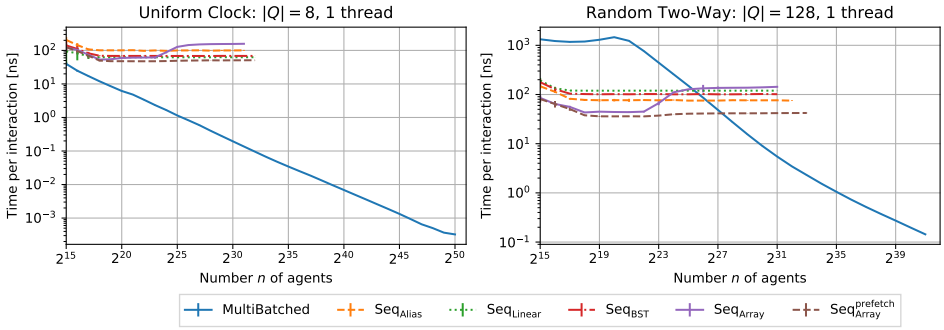


Figure 4 Processing time per interaction as function of the number of agents n. Each series ends with the largest value n for which the median of the total processing time is below 400 s.

6.3 Dynamic Epoch Lengths

Recall that MultiBatched is split into several epochs that each consist of multiple collision-free runs. In our implementation, we add runs to an epoch until the number of interactions exceeds a specified threshold T. The value of T has to be chosen as a trade-off between the cost of adding another run (i.e., sampling the run length and planting a collision) versus the diminishing return it yields (as later runs become shorter in expectation). This trade-off depends on the protocol and its configuration. For instance, the batch processing cost of a convergent protocol may become smaller compared to the initial costs (e.g., when most agents are in only a small fraction of the states).

As the trade-off is dynamic, we maximize the throughput using a control loop that dynamically optimizes the length of an epoch. Given the currently best value known for T, it increases (and later decreases) T to $1.1T$ and $0.9T$, respectively. For each of the three values, we measure the throughput, chose the T which maximizes it and repeat. Since the throughput response curve is single-peaked, the process will find a nearly optimal T.

7 Experimental Evaluation

In the following, we empirically evaluate the various simulation algorithms. The code is compiled using `g++-8.3` with flags `-O3 -march=native` and executed on the following system: 2× Intel Xeon Gold 6148 CPU @ 2.4 GHz (40 cores/80 hardware threads in total), 192 GiB DDR4 RAM @ 2666 MHz. Each data point is the median of at least five measurements (using different random seeds); error bars indicate their standard deviation.

Seq_{BST}'s search tree is implemented as an array with breath-first-indexing (i.e., the weight of node $i \geq 1$ is stored at $A[i]$; its left child is at index $2i$, its right child at $2i+1$). In order to reduce pipeline stalls, we implement a branch-free traversal similarly to [37]. $\text{Seq}_{\text{Array}}$ uses an array with 32 bit words to store states. We additionally consider $\text{Seq}_{\text{Array}}^{\text{prefetch}}$ which prefetches states for eight[4] interactions ahead of time as a latency hiding technique. If our Dynamic Alias Table implementation detects an imbalance necessitating rebuilding of the data structure, it will first attempt to quickly solve the problem by swapping the alias of the affected row with up to five randomly selected partners; only if this fails a rebuild is issued. This heuristic does not change the asymptotic time complexity of the data structure.

[4] This is the optimal value measured for this CPU type and slightly varies between machines.

The runtime of most simulators has non-trivial dependencies on the input parameters, protocol, and state distribution. Hence, we simulate a small number $N = n$ of interactions to prevent measuring artifacts caused by significant changes in the state distributions. MultiBatched typically simulates slightly more interactions due to the batching granularity. Since the runtime of all simulators is linear in N, we always report the time *per interaction* to ease extrapolation.

Three different protocols are used to highlight certain aspects of the algorithms.

- *Uniform Clock* and *Running Clock* implement the same deterministic one-way protocol phase-clock protocol inspired by [28]. In the *running* variant, all agents start in the first phase, and $\sqrt{n}$ of them are marked. Due to the choice of parameters, we expect only one out of $\Theta(\sqrt{n})$ interactions to change states. Thus, even at the end of the benchmark, the population is still highly concentrated in the lowest phase. The *uniform* variant, in contrast, evenly distributes marked and unmarked agents over all phases. This results in a constant update probability per interaction.
- The *Random Two-Way* protocol uses a deterministic transition function $\delta(q_i, q_j) = d_{ij}$ where each d_{ij} is initially drawn independently and uniformly at random from Q. Initially agents are evenly distributed over all states.

To ensure the correctness of our implementations, we rely on unit tests (e.g., using a family of protocols that count the number of interactions of each agent). Additionally, we cross-reference the results of various algorithms. Since we cannot expect two simulators to yield the same set of interactions, we compare their simulated dynamics. We, for instance, monitor the number of interactions required until a Uniform Clock starts running.

Number of Agents. We begin our experimental study by investigating the dependencies on the problem size n. To this end, we search for the largest number n of agents that a simulator can simulate within a fixed time budget of 400 s. Figure 4 reports such measurements for two different settings (see the full version of this paper [17] for the full set).

For the *Uniform Clock* protocol with $|Q| = 8$ states, the fastest Seq variant reaches $n = 2^{32}$ within the time budget. In the same time, MultiBatched simulates $N = n = 2^{50}$ interactions. For *Random Two-Way*, the ratio between the achievable population sizes is smaller but still exceeds three orders of magnitude. We attribute the different ratios mainly to the batching step. MultiBatched requires $\Theta(|Q|^2)$ hypergeometric variates per batching step for the latter protocol, since neither the partitioning nor the skipping heuristics (see Section 6) are effective on a featureless transition matrix with uniformly random states. For the *Uniform Clock* protocol, the partitioning heuristic reduces the number of hypergeometric random variates to less than $2|Q|$ per batch.

Number of States. As summarized in Table 1, the number $|Q|$ of states crucially affects the algorithms' runtimes. To quantify the practical impact, we carry out scaling experiments for $4 \leq |Q| \leq 128$ while fixing all remaining parameters. Figure 5 visualizes the results.

MultiBatched performs best in all cases supporting our previous analysis. It shows almost no scaling behavior for both clock protocols. For *Random Two-Way*, the algorithm is almost one order of magnitude faster than its competitors despite a slow-down of 40 between the smallest and largest state sizes.

$\text{Seq}^{\mathsf{prefetch}}_{\mathsf{Array}}$ is the second fastest solution in almost all settings. In the *Running Clock* campaign, it is however outperformed by $\text{Seq}_{\mathsf{Linear}}$. This can be explained by the fact that initially almost all agents are in state 0 which results in a constant time look-up despite the usage of a linear search. This behavior motivates the renaming heuristic.

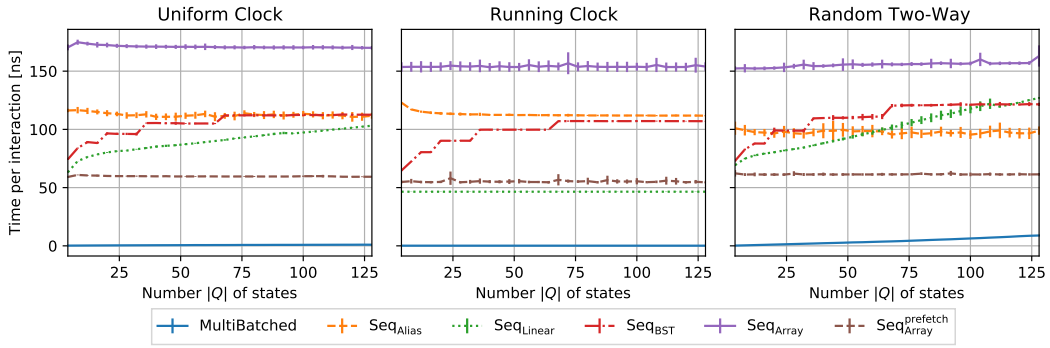


Figure 5 Processing time per interaction as function of the number of states $|Q|$ with $n = 2^{30}$.

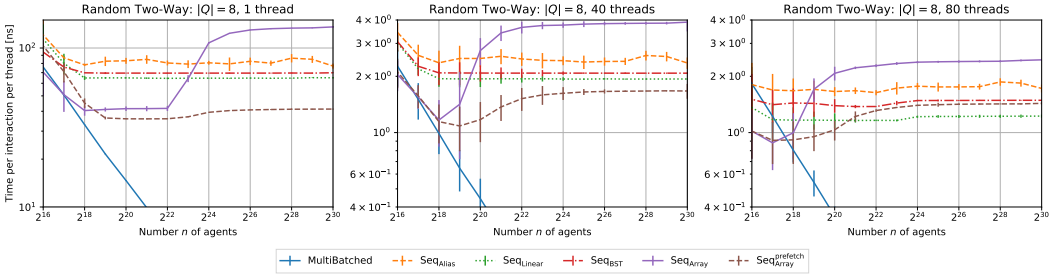


Figure 6 Effect of process parallelism on a machine with 40 CPU cores (plus HyperThreading).

We observe no systematic dependency on $|Q|$ for $\textsc{Seq}_{\textsf{Alias}}$ rendering it a good choice for very large state spaces. While it is up to a factor of 2.0 slower compared to $\textsc{Seq}^{\textsf{prefetch}}_{\textsf{Array}}$, the algorithm might be preferable in a parallel setting.

Memory Footprint and Parallelism. Due to the stochastic nature of the protocols, we expect that in almost all applications several runs of the same protocol are required to derive statistically significant results. On modern machines with many processor cores, one should be able to maximize the throughput by executing multiple independent simulations in parallel. As visualized in Figure 6, most simulators scale well with the number of threads and typically achieve a self-speedup of 40 to 50 times using 40 cores (plus HyperThreading) at $n = 2^{30}$. A notable exception is $\textsc{Seq}^{\textsf{prefetch}}_{\textsf{Array}}$, which reaches only a speedup of 30 as it saturates the memory controllers of both CPU sockets.

Another aspect of parallel execution is the memory footprint. Since $\textsc{Seq}_{\textsf{Array}}$ requires constant memory per agent, our implementations of $\textsc{Seq}_{\textsf{Array}}$ and $\textsc{Seq}^{\textsf{prefetch}}_{\textsf{Array}}$ allocate in excess of 320 GB main memory to execute 80 processes in parallel. Although, this number can be reduced by constant factors using a more efficient representation of states, it has to be contrasted to the competing algorithms with a state space of only a few kilobytes[5] for the same campaign.

[5] We report no exact numbers as the system's process overheads exceed the simulators' internal states.

8 Conclusions and Open Problems

We considered the simulation of large population protocols to allow the experimental investigation of slowly scaling observables in such systems. Two algorithm classes are discussed.

Sequential simulators carry out each interaction one after the other. We analyze the variants $\text{SEQ}_{\mathsf{Array}}$, $\text{SEQ}_{\mathsf{Linear}}$, $\text{SEQ}_{\mathsf{BST}}$, and $\text{SEQ}_{\mathsf{Alias}}$ which differ in the data structures maintaining the agents' states, and demonstrate substantial differences in their practical performances. As a by-product, we describe the Dynamic Alias Table which might be independently applicable.

Batched simulators coalesce interactions to achieve asymptotical speed-ups for protocols with a limited number of states. Our implementation then simulates more than 2^{50} interactions in 400 s which is several orders of magnitudes larger than the fastest sequential simulator.

Possible Extensions. In some variants of the population model it is assumed that interactions are limited to some communication network. Since we store the configurations in our batched simulators as a multiset, it is not clear how to directly adapt our approach. We believe this might be an interesting extension of our work.

Further variants are concerned with the way how agents interact. It is straightforward to adapt our simulator software to a setting where, e.g., a random matching of agents interacts in each time step. Furthermore, our approach could be generalized to a setting where more than two agents interact. In this case we are in need of good heuristics for partitioning the transition-tensor, since the work for updating a batch grows exponentially in the number of interacting agents. In general, sampling the interaction counters is a frequent and costly task. Therefore, any improved heuristic to sample from the $|Q| \times |Q|$ matrix using only $o(|Q|^2)$ variates would yield a measureable benefit for the total runtime of a simulation.

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *SODA*, pages 2560–2579. SIAM, 2017. doi:10.1137/1.9781611974782.169.

2 Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In *SODA*. SIAM, 2018. doi:10.1137/1.9781611975031.144.

3 Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *ICALP (2)*, volume 9135 of *LNCS*, pages 479–491. Springer, 2015. doi:10.1007/978-3-662-47666-6_38.

4 Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In *PODC*, pages 47–56. ACM, 2015. doi:10.1145/2767386.2767429.

5 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Comput.*, 18(4):235–253, 2006. doi:10.1007/s00446-005-0138-3.

6 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *PODC*, pages 292–299. ACM, 2006. doi:10.1145/1146381.1146425.

7 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Comput.*, 21(3):183–199, 2008. doi:10.1007/s00446-008-0067-z.

8 Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Comput.*, 21(2):87–102, 2008. doi:10.1007/s00446-008-0059-z.

9 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Comput.*, 20(4):279–304, 2007. doi:10.1007/s00446-007-0040-2.

10 Anish Arora, Shlomi Dolev, and Mohamed G. Gouda. Maintaining digital clocks in step. *Parallel Process. Lett.*, 1:11–18, 1991.

11 Richard Arratia, Skip Garibaldi, and Joe Kilian. Asymptotic distribution for the birthday problem with multiple coincidences, via an embedding of the collision process. *Random Struct. Algorithms*, 48(3):480–502, 2016. doi:10.1002/rsa.20591.

12 James Aspnes and Eric Ruppert. An introduction to population protocols. *Bull. EATCS*, 93:98–117, 2007.

13 Stav Ben-Nun, Tsvi Kopelowitz, Matan Kraus, and Ely Porat. An $O(\log^{3/2} n)$ parallel time population protocol for majority with $O(\log n)$ states. In *PODC*, page to appear, 2020.

14 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. Majority & stabilization in population protocols. *CoRR*, abs/1805.04586, 2018. arXiv:1805.04586.

15 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A population protocol for exact majority with o(log5/3 n) stabilization time and theta(log n) states. In *DISC*, volume 121 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl, 2018. doi:10.4230/LIPIcs.DISC.2018.10.

16 Petra Berenbrink, George Giakkoupis, and Peter Kling. Optimal time and space leader election in population protocols. In *STOC*, pages 119–129, 2020.

17 Petra Berenbrink, David Hammer, Dominik Kaaser, Ulrich Meyer, Manuel Penschuck, and Hung Tran. Simulating population protocols in sub-constant time per interaction. *CoRR*, 2020. arXiv:2005.03584.

18 Petra Berenbrink, Dominik Kaaser, Peter Kling, and Lena Otterbach. Simple and efficient leader election. In *SOSA@SODA*, volume 61 of *OASICS*, pages 9:1–9:11. Schloss Dagstuhl, 2018. doi:10.4230/OASIcs.SOSA.2018.9.

19 Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Brief announcement: Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. In *PODC*. ACM, 2017. doi:10.1145/3087801.3087858.

20 Paul Bratley, Bennett L. Fox, and Linus Schrage. *A guide to simulation, 2nd Edition*. Springer, 1987.

21 Luca Cardelli and Attila Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific reports*, 2:656, 2012. doi:10.1038/srep00656.

22 Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature nanotechnology*, 8(10):755, 2013. doi:10.1038/nnano.2013.189.

23 Luc Devroye. *Non-Uniform Random Variate Generation*. Springer, 1986. doi:10.1007/978-1-4613-8643-8.

24 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. In *DISC*, volume 9363 of *LNCS*, pages 602–616. Springer, 2015. doi:10.1007/978-3-662-48653-5_40.

25 Moez Draief and Milan Vojnovic. Convergence speed of binary interval consensus. *SIAM J. Control and Optimization*, 50(3):1087–1109, 2012. doi:10.1137/110823018.

26 Robert Elsässer and Tomasz Radzik. Recent results in population protocols for exact majority and leader election. *Bull. EATCS*, 126, 2018. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/549/546.

27 Bennett L. Fox. Generating markov-chain transitions quickly: I. *INFORMS J. Comput.*, 2(2):126–135, 1990.

28 Leszek Gasieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. In *SODA*. SIAM, 2018. doi:10.1137/1.9781611975031.169.

29 Leszek Gasieniec, Grzegorz Stachowiak, and Przemyslaw Uznanski. Almost logarithmic-time space optimal leader election in population protocols. In *SPAA*, pages 93–102. ACM, 2019. doi:10.1145/3323165.3323178.

30 Torben Hagerup, Kurt Mehlhorn, and J. Ian Munro. Maintaining discrete probability distributions optimally. In *ICALP*, volume 700 of *LNCS*, pages 253–264. Springer, 1993.

31 Lorenz Hübschle-Schneider and Peter Sanders. Parallel weighted random sampling. In *ESA*, volume 144 of *LIPIcs*, pages 59:1–59:24. Schloss Dagstuhl, 2019.

32 Fabian Kuhn and René Struik. Random walks revisited: Extensions of pollard's rho algorithm for computing multiple discrete logarithms. In *Selected Areas in Cryptography*, volume 2259 of *LNCS*, pages 212–229. Springer, 2001. doi:10.1007/3-540-45537-X_17.

33 Yossi Matias, Jeffrey Scott Vitter, and Wen-Chun Ni. Dynamic generation of discrete random variates. In *SODA*, pages 361–370. ACM/SIAM, 1993.

34 George B. Mertzios, Sotiris E. Nikoletseas, Christoforos L. Raptopoulos, and Paul G. Spirakis. Determining majority in networks with local interactions and very small local memory. In *ICALP (1)*, volume 8572 of *LNCS*, pages 871–882. Springer, 2014. doi:10.1007/978-3-662-43948-7_72.

35 Yves Mocquard, Emmanuelle Anceaume, James Aspnes, Yann Busnel, and Bruno Sericola. Counting with population protocols. In *NCA*, pages 35–42. IEEE Computer Society, 2015. doi:10.1109/NCA.2015.35.

36 Sanguthevar Rajasekaran and Keith W. Ross. Fast algorithms for generating discrete random variates with changing distributions. *ACM Trans. Mod. Comp. Sim.*, 3(1), 1993.

37 Peter Sanders and Sebastian Winkel. Super scalar sample sort. In *ESA*, volume 3221 of *LNCS*, pages 784–796. Springer, 2004. doi:10.1007/978-3-540-30140-0_69.

38 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Nat. Comput.*, 7(4):615–633, 2008. doi:10.1007/s11047-008-9067-y.

39 Ernst Stadlober. Ratio of uniforms as a convenient method for sampling from classical discrete distributions. In *Winter Simulation Conference*, pages 484–489. ACM Press, 1989. doi:10.1145/76738.76801.

40 Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. Software Eng.*, 17(9):972–975, 1991. doi:10.1109/32.92917.

41 Alastair J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Math. Soft.*, 3(3), 1977. doi:10.1145/355744.355749.

An Optimal Decentralized $(\Delta + 1)$-Coloring Algorithm

Daniel Bertschinger
Department of Computer Science,
ETH Zürich, Switzerland
daniel.bertschinger@inf.ethz.ch

Johannes Lengler
Department of Computer Science,
ETH Zürich, Switzerland
johannes.lengler@inf.ethz.ch

Anders Martinsson
Department of Computer Science,
ETH Zürich, Switzerland
anders.martinsson@inf.ethz.ch

Robert Meier
Department of Computer Science,
ETH Zürich, Switzerland
robert.meier@inf.ethz.ch

Angelika Steger
Department of Computer Science,
ETH Zürich, Switzerland
steger@inf.ethz.ch

Miloš Trujić
Department of Computer Science,
ETH Zürich, Switzerland
mtrujic@inf.ethz.ch

Emo Welzl
Department of Computer Science,
ETH Zürich, Switzerland
emo@inf.ethz.ch

Abstract

Consider the following simple coloring algorithm for a graph on n vertices. Each vertex chooses a color from $\{1, \ldots, \Delta(G) + 1\}$ uniformly at random. While there exists a conflicted vertex choose one such vertex uniformly at random and recolor it with a randomly chosen color. This algorithm was introduced by Bhartia et al. [MOBIHOC'16] for channel selection in WIFI-networks. We show that this algorithm always converges to a proper coloring in expected $O(n \log \Delta)$ steps, which is optimal and proves a conjecture of Chakrabarty and de Supinski [SOSA'20].

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Mathematics of computing → Graph algorithms; Mathematics of computing → Graph coloring; Mathematics of computing → Probabilistic algorithms

Keywords and phrases Decentralized Algorithm, Distributed Computing, Graph Coloring, Randomized Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.17

Funding *Miloš Trujić*: author was supported by grant no. 200021 169242 of the Swiss National Science Foundation.

1 Introduction

It is well known that an undirected graph $G = (V, E)$ with maximum degree $\Delta = \Delta(G)$ can be properly colored by using $\Delta + 1$ colors. In fact, a simple greedy algorithm which assigns the colors successively achieves this bound by just touching each vertex once. Note that the bound $\Delta + 1$ is tight, as cliques and odd cycles require this number of colors.

In [1] Bhartia et al. introduced the use of a simple decentralized coloring algorithm as an efficient solution to the channel selection problem in wireless networks. Their algorithm can be formulated as follows.

Decentralized Graph Coloring

For a graph $G = (V, E)$

1. choose for each vertex $v \in V$ a color from $\{1, \ldots, \Delta + 1\}$ independently and uniformly at random;
2. choose a vertex $v \in V$ uniformly at random among all vertices which have a neighbor in the same color;
3. recolor v into a color chosen from $\{1, \ldots, \Delta + 1\}$ uniformly at random;
4. repeat steps 2 and 3 until a proper coloring of G is found.

They showed that this algorithm finds a proper coloring in $O(n\Delta)$ rounds in expectation. Chakrabarty and de Supinski [2] introduced a variant of the coloring algorithm: instead of recoloring a vertex v once as above, in their "Persistent Decentralized Coloring Algorithm" such a vertex v persistently (hence the name) gets recolored until it has no neighbor in the same color. They showed that this modified algorithm only requires $O(n \log \Delta)$ recolorings and conjectured that the same bound also holds for the original algorithm. In this paper we prove their conjecture.

▶ **Theorem 1.1.** *The decentralized coloring algorithm converges in expectation to a proper* $(\Delta + 1)$*-coloring in* $O(n \log \Delta)$ *recoloring steps.*

In fact, our argument shows that the same runtime bound holds true if the initial coloring is chosen adversarially. This is in contrast with the persistent version of the algorithm mentioned above, as that one takes $\Theta(n\Delta)$ recolorings in expectation when starting with an adversarial coloring (see [1, Theorem 3]). However, the question raised in [2] of 'which algorithm is faster in the random setting' remains open.

We note that the bound in Theorem 1.1 is best possible, as for the complete graph K_n the decentralized coloring algorithm essentially performs a *Coupon Collector* process. Indeed, once a color (coupon) has been acquired it remains in the graph until the end of the process and we need to see all colors. The claim thus follows from the well known fact that in expectation the coupon collector process with n coupons requires $nH_n = \Theta(n \log n)$ rounds, where $H_k = \sum_{i=1}^{k} \frac{1}{i}$ is the k-th Harmonic number. Moreover, the result is tight for every combination of n and Δ. Namely, consider a vertex-disjoint union of n/Δ complete graphs K_Δ and by the same argument the process requires $\Theta(n/\Delta \cdot \Delta \log \Delta) = \Theta(n \log \Delta)$ rounds.

Our proof of Theorem 1.1 is short and elegant, and is based on *drift analysis* [8]. It is presented in an expository way and provides insight in why our potential function is appropriate for the analysis. We complement the analyis by tail bounds in Section 3.

Finally, we conclude the paper by a brief discussion of the parallelized version of this algorithm, where all "conflicted" vertices get recolored simultaneously (instead of Step 3), and we prove that this variant takes exponential time.

2 Proof of Theorem 1.1

We start with introducing some notation. We use c_t to denote the coloring of the graph after t recoloring steps, that is c_t is a function $c_t \colon V(G) \to \{1, \ldots, \Delta + 1\}$. With $M_t \subseteq E$ we denote the set of *monochromatic edges* in c_t. Observe that c_t is a proper coloring of G if and only if $|M_t| = 0$. Our main goal is thus to establish good bounds on (the reduction of) the size of the sets M_t. In order to do so it is helpful to view the recoloring step(s) (i.e. Step 2 and Step 3) as a (slightly different) three step process:

Recoloring Step

For every $t \geqslant 1$,

- **S1** choose a monochromatic connected component $C \subseteq M_{t-1}$ at random proportional to the number of vertices in C;
- **S2** choose a vertex $v \in V(C)$ uniformly at random;
- **S3** let $c_t(v)$ be a uniformly at random chosen color from $\{1, \dots, \Delta + 1\}$ and set $c_t(u) = c_{t-1}(u)$ for all $u \neq v$.

As a main tool in bounding the expected number of recoloring steps we use a so-called drift theorem (see [8, Theorem 2.3.1]).

▶ **Theorem 2.1** (Additive Drift Theorem [6]). *Let $(X_t)_{t \geqslant 0}$ be a sequence of non-negative random variables with a finite state space $\mathcal{S} \subset \mathbb{R}_0^+$ such that $0 \in \mathcal{S}$. Let $T := \inf\{t \geqslant 0 \mid X_t = 0\}$. If there exists $\delta > 0$ such that for all $s \in \mathcal{S} \setminus \{0\}$ and for all $t > 0$,*

$$\mathbb{E}[X_t - X_{t-1} \mid X_t = s] \leqslant -\delta, \tag{1}$$

then

$$\mathbb{E}[T] \leqslant \mathbb{E}[X_0] \,/\, \delta.$$

By *drift* we refer to the expectation $\mathbb{E}[X_t - X_{t-1} \mid X_t = s]$. (For a more extensive introduction to drift analyis, we refer the reader to [8].) Our goal is to apply Theorem 2.1 by assigning to each coloring c_t a real value $\Phi(t)$ (which we plug in for X_t) so that $\Phi(t) = 0$ if and only if c_t is a proper coloring. The *potential function* $\Phi(\cdot)$ we eventually use to prove Theorem 1.1 consists of several terms (see equation (3) below) and in order to motivate each of the terms we introduce them one by one. The simplest and most natural choice is to consider just the number of monochromatic edges, i.e. $\Phi(t) := |M_t|$. (Mind that this is only for explanatory purposes and will not be the final definition of Φ.) To apply Theorem 2.1 we need to estimate the drift from a single recoloring step. In the following claim (and in fact all similar ones in this section), the expectation is always taken with respect to a single recoloring step. That is, we (implicitly) condition on the coloring c_{t-1} without stating it every time. Note that this formulation implies what is required by equation (1).

As it turns out, in the case of $\Phi(t) := |M_t|$ we do not need to make use of the fact that the component C is chosen randomly, we may assume that the component C is given arbitrarily or even by an adversary.

▷ Claim 2.2. For all $t \geqslant 1$ and any connected component C in M_{t-1} we have

$$\mathbb{E}\left[|M_t| \,\middle|\, C\right] \leqslant |M_{t-1}| - \bar{d}(C) + 1 - \frac{1}{\Delta + 1},$$

where $\bar{d}(C)$ denotes the average degree of the graph induced by $V(C)$.

Proof. The claim follows easily from the following two observations. As v is chosen uniformly at random within C (as in Step 2), we decrease the number of monochromatic edges within C by $\bar{d}(C)$ whenever the newly chosen color is different from the current color of C, which happens with probability $\Delta/(\Delta + 1)$. All edges incident to v that do not belong to C become monochromatic with probability $1/(\Delta + 1)$. Thus we have

$$\mathbb{E}\left[|M_t| \,\middle|\, C\right] \leqslant |M_{t-1}| - \bar{d}(C) \cdot \frac{\Delta}{\Delta + 1} + \frac{\Delta - \bar{d}(C)}{\Delta + 1} = |M_{t-1}| - \bar{d}(C) + 1 - \frac{1}{\Delta + 1},$$

as claimed. ◁

As the average degree of every monochromatic component in M_t is at least one, Claim 2.2 implies $\mathbb{E}[|M_t|] \leqslant |M_{t-1}| - 1/(\Delta+1)$ whenever $|M_{t-1}| > 0$. The following proposition then easily follows from Theorem 2.1.

▶ **Proposition 2.3.** *Let $D > 0$ be any fixed constant. For every graph G and every coloring c_0 of G such that $|M_0| \leqslant Dn/\Delta$ the decentralized coloring algorithm reaches a proper $(\Delta+1)$-coloring in expectation after $O(n)$ recoloring steps.*

Unfortunately, a random coloring of a graph G with $\Delta + 1$ colors has in expectation $\Theta(n)$ monochromatic edges, so Proposition 2.3 is not immediately applicable. Instead, Claim 2.2 together with Theorem 2.1 only provide us with the bound of $O(n\Delta)$ (see Bhartia et al. [1]). In order to go beyond this, observe that Claim 2.2 actually gives a drift of $-1/3$ whenever $|V(C)| \geqslant 3$, as the average degree of a connected graph on $s \geqslant 3$ vertices is at least $4/3$. Thus, the only critical case are components C that consist of only one edge. To handle these we introduce some more notation.

We denote by $I_t \subseteq M_t$ the set of *isolated edges*, that is all edges which are monochromatic components of size two. We also let $P_t \subseteq V$ stand for the set of all properly colored vertices, i.e. the vertices that are not incident to any edge in M_t. Akin to Claim 2.2, the next claim gives a bound on the expected change in the number of isolated edges in one recoloring step.

▷ Claim 2.4. For all $t \geqslant 1$ and any connected component C in M_{t-1} we have

$$\mathbb{E}\left[|I_t| \;\middle|\; C\right] \leqslant |I_{t-1}| + \bar{d}(C) + 1.$$

For components C that form an isolated edge, we have in addition

$$\mathbb{E}\left[|I_t| \;\middle|\; C = uw\right] \leqslant |I_{t-1}| - \frac{\Delta}{\Delta+1} + \frac{|N(u) \cap P_{t-1}| + |N(w) \cap P_{t-1}|}{2(\Delta+1)}.$$

Proof. By recoloring a vertex v, the only isolated edges that can be created are edges that are incident to neighbors of v within C (at most one isolated edge per neighbor of v) and edges between v and P_{t-1} (naturally at most one edge incident to v can be isolated and monochromatic). This proves the first inequality. For the second assume that $C = uw$. Clearly, after recoloring one of u and w with a different color (which happens with probability $\Delta/(\Delta+1)$), the isolated edge $C = uw$ disappears. Observe, also that a new isolated edge can only be generated if we choose as a new color for u (or w) a color of a vertex in $N(u) \cap P_{t-1}$ (or $N(w) \cap P_{t-1}$) respectively. This, together with the fact that each of u or w is chosen in Step 2 with probability $1/2$, proves the second inequality. ◁

We pause for a moment from the proof of Theorem 1.1 to showcase the use of previous claims for proving a positive result about complete bipartite graphs.

▶ **Proposition 2.5.** *For complete bipartite graphs $G = K_{n,m}$ the decentralized coloring algorithm reaches a proper $(\Delta+1)$-coloring in expectation after $O(\min\{n,m\})$ recoloring steps.*

Proof. Observe that for complete bipartite graphs vertices of $P_{t-1} \cap A$ and $P_{t-1} \cap B$ need to be colored with *different* colors (here A and B denote the two parts of the bipartite graph). Also note that an isolated edge can only be generated if a color appears only once in $P_{t-1} \cap A$ (and $P_{t-1} \cap B$). Therefore, for a monochromatic edge uv we have $|N(u) \cap P_{t-1}| + |N(w) \cap P_{t-1}| \leqslant \Delta$. We can thus replace the bound in the second inequality of Claim 2.4 by

$$\mathbb{E}\left[|I_t| \;\middle|\; C = uw\right] \leqslant |I_{t-1}| - \frac{\Delta}{\Delta+1} + \frac{\Delta}{2(\Delta+1)} \leqslant |I_{t-1}| - \frac{\Delta}{2(\Delta+1)}. \tag{2}$$

Consider now the potential function $\Phi(t) := |M_t| + \frac{1}{10}|I_t|$. In case $|V(C)| \geqslant 3$, from Claim 2.2 and Claim 2.4, we get (with room to spare)

$$\begin{aligned}\mathbb{E}\left[\Phi(t) \mid C, |V(C)| \geqslant 3\right] &\leqslant |M_{t-1}| - \bar{d}(C) + 1 - \frac{1}{\Delta+1} + \frac{1}{10}|I_{t-1}| + \frac{1}{10}\bar{d}(C) + \frac{1}{10} \\ &\leqslant \Phi(t-1) - \frac{9}{10}\bar{d}(C) + \frac{11}{10} \leqslant \Phi(t-1) - \frac{1}{20},\end{aligned}$$

where we used the fact that $\bar{d}(C) \geqslant 4/3$. On the other hand, if $C = uw$ then by Claim 2.2 and (2) we have

$$\begin{aligned}\mathbb{E}\left[\Phi(t) \mid C = uw\right] &\leqslant |M_{t-1}| - \bar{d}(C) + 1 - \frac{1}{\Delta+1} + \frac{1}{10}|I_{t-1}| - \frac{1}{10}\frac{\Delta}{2(\Delta+1)} \\ &\leqslant \Phi(t-1) - \frac{1}{20}\Big(\frac{20}{\Delta+1} + \frac{\Delta}{\Delta+1}\Big) \leqslant \Phi(t-1) - \frac{1}{20}.\end{aligned}$$

In conclusion,

$$\mathbb{E}\left[\Phi(t) \mid C\right] \leqslant \Phi(t-1) - \frac{1}{20},$$

for every component C. The proposition now follows from Theorem 2.1 together with the fact that in a random x-coloring an edge is monochromatic with probability $1/x$ and thus

$$\mathbb{E}[\Phi(0)] \leqslant \mathbb{E}[|M_0|] + \mathbb{E}[|I_0|] \leqslant 2 \cdot \frac{n \cdot m}{\max\{n,m\}+1} \leqslant 4\min\{n,m\},$$

with room to spare. ◀

We note that this proof actually shows that the assertion of Proposition 2.5 remains true, for sufficiently small $\varepsilon > 0$, if we reduce the number of colors to be used by the algorithm to $(1-\varepsilon)\Delta$, that is $(1-\varepsilon)\max\{n,m\}$. We do not elaborate further on this.

After this short detour we come back to the proof of Theorem 1.1. What one could conclude from the two claims above is that if we were to choose a component C in Step 1 which is of size at least three throughout the process, then the drift obtained (Claim 2.2) would always be less than $-1/3$. However, this is far too optimistic to hope for.

Consider an isolated edge uv and assume we recolor v. If the new color chosen *does not* belong to its properly-colored neighborhood $N(v) \cap P_{t-1}$, then the number of monochromatic isolated edges decreases by one. This happens with constant probability unless the size of $N(v) \cap P_{t-1}$ is close to Δ.

Since in Step 1 we choose C *randomly*, we expect a strong drift "towards the target" as long as we are in one of the situations from the paragraphs above. In other words, we have a desired drift unless M_{t-1} comprises mostly of isolated edges *and* most vertices $u \in V(I_{t-1})$ have almost Δ neighbors in $N(u) \cap P_{t-1}$.

Let us hence analyze what happens if in such a case we recolor a vertex v belonging to an isolated edge uv. Suppose we set $c_t(v) := c_{t-1}(x)$ for some $x \in N(v) \cap P_{t-1}$. If the color $c_{t-1}(x)$ appears multiple times in $N(v) \cap P_{t-1}$, we do not create a new isolated edge. Otherwise, the edge xv becomes isolated and $P_t := (P_{t-1} \setminus \{x\}) \cup \{u\}$. However, crucially, as we assumed that every vertex $u \in V(I_{t-1})$ had roughly Δ neighbors in P_{t-1}, we conversely have that an average vertex in P_{t-1} has roughly $\Delta|V(I_{t-1})|/|P_{t-1}|$ neighbors in $V(I_{t-1})$. Thus, we may expect that $N(x) \cap P_t$ is *smaller* than Δ. In other words, we expect that $e(V(I_t), P_t)$ is smaller than $e(V(I_{t-1}), P_{t-1})$. Here and throughout we use $e(X,Y)$ to denote the number of edges between two disjoint vertex sets X and Y.

Previous considerations motivate keeping track of $e(V(I_t), P_t)$ as well and lead us to formulate the following potential function:

$$\Phi(t) := |M_t| + \frac{|I_t|}{10} + \frac{e(V(I_t), P_t)}{100\Delta}. \tag{3}$$

Note that the value of $\Phi(t)$ is always proportional to the number of monochromatic edges.

▷ Claim 2.6. For all $t \geqslant 1$ we have

$$|M_t| \leqslant \Phi(t) \leqslant 2|M_t|.$$

Proof. The first inequality is trivial. The second follows, with room to spare, as $I_t \subseteq M_t$ and $e(V(I_t), P_t) \leqslant 2|I_t| \cdot \Delta$. ◁

With Claim 2.6 at hand we deduce from Proposition 2.3 that in order to complete the proof of Theorem 1.1 it suffices to show that the algorithm reduces the potential Φ to a value of Dn/Δ in $O(n \log \Delta)$ steps, for some arbitrarily large but fixed constant $D > 0$. This is what we do in the remainder of this section.

Note also that there is no hope to always get a *constant* drift, as by Theorem 2.1 this would then lead to a bound of $O(n)$ recoloring steps, which would contradict the bound of $\Omega(n \log n)$ for K_n. Instead we show a *multiplicative* drift.

▷ Claim 2.7. For any $t \geqslant 1$ with $\Phi(t-1) > 0$, we have

$$\mathbb{E}[\Phi(t)] \leqslant \Phi(t-1)\Big(1 - \frac{1}{1000n}\Big).$$

Proof. By linearity of expectation we can consider each term of $\Phi(\cdot)$ in (3) independently. The first two terms are handled by Claim 2.2 and Claim 2.4, so we first establish some bounds on the third. Observe that in order for an edge to be counted in $e(V(I_t), P_t)$ but not in $e(V(I_{t-1}), P_{t-1})$ it must be incident to a vertex in either $V(I_t) \setminus V(I_{t-1})$ or $P_t \setminus P_{t-1}$. Let C be a component chosen in Step 1 and v a vertex chosen in Step 2. For any vertex in $\{v\} \cup (N(v) \cap V(C))$, we either get one new isolated edge or one new properly colored vertex (or neither). In the former, the other endpoint of that edge potentially contributes by Δ to $e(V(I_t), P_t)$, and in the latter each monochromatic edge with one endpoint in $N(v) \cap V(C)$ potentially contribute by Δ to $e(V(I_t), P_t)$ for each of its endpoints. Thus we have

$$\mathbb{E}[e(V(I_t), P_t) \mid C] \leqslant e(V(I_{t-1}), P_{t-1}) + (\bar{d}(C) + 1) \cdot 2\Delta,$$

where as before $\bar{d}(C)$ denotes the average degree of the component C. Together with Claim 2.2 and Claim 2.4, for all components C on at least three vertices we get

$$\begin{aligned} \mathbb{E}\left[\Phi(t) \mid C, |V(C)| \geqslant 3\right] &\leqslant \Phi(t-1) - \Big(1 - \frac{1}{10} - \frac{2}{100}\Big)\bar{d}(C) + 1 + \frac{1}{10} + \frac{2}{100} \\ &\leqslant \Phi(t-1) - \frac{1}{25}\bar{d}(C), \end{aligned} \tag{4}$$

where the last inequality follows from $\bar{d}(C) \geqslant 4/3$.

Next we consider the third term of $\Phi(\cdot)$ conditioned on choosing a component $C \subseteq I_{t-1}$, i.e. C is an isolated edge. We first let $d(v, X) := |N(v) \cap X|$ for all $v \in V$ and sets $X \subseteq V$ and denote by

$$\bar{d}_{IP} := \frac{1}{|V(I_{t-1})|} \sum_{u \in V(I_{t-1})} d(u, P_{t-1}) \qquad \text{and} \qquad \bar{d}_{PI} := \frac{1}{|P_{t-1}|} \sum_{u \in P_{t-1}} d(u, V(I_{t-1}))$$

the average degree of vertices in $V(I_{t-1})$ into P_{t-1}, and the average degree of vertices in P_{t-1} into $V(I_{t-1})$, respectively. Note that, of course, we have $\sum_{u\in V(I_{t-1})} d(u,P_{t-1}) = \sum_{u\in P_{t-1}} d(u,V(I_{t-1}))$, and hence $\bar{d}_{IP}|V(I_{t-1})| = \bar{d}_{PI}|P_{t-1}|$.

Consider an isolated edge wv and assume v gets recolored with a new color. Then, since w is now properly colored, all $d(w,P_{t-1})$ edges incident to w which contributed to $e(V(I_{t-1}),P_{t-1})$ are not counted in $e(V(I_t),P_t)$, except possibly one in case v forms a new isolated edge with a neighbor of w. Moreover, any new edge counted in $e(V(I_t),P_t)$ must be incident to either v, w, or a vertex $x \in P_{t-1}$ for which $vx \in I_t$. There are at most $\Delta - d(v,P_{t-1})$, $\Delta - d(w,P_{t-1})$, and $\Delta - d(x,V(I_{t-1}))$ such edges respectively not already counted in $e(V(I_{t-1}),P_{t-1})$. Combining all this we get

$$\begin{aligned} e(V(I_t),P_t) \leqslant e(V(I_{t-1}),P_{t-1}) - d(w,P_{t-1}) + 1 + \Delta - d(v,P_{t-1}) \\ + \Delta - d(w,P_{t-1}) + \sum_{x\in P_{t-1}} \mathbb{1}_{vx\in I_t}\big(\Delta - d(x,V(I_{t-1}))\big), \end{aligned}$$

if $c_t(v) \neq c_{t-1}(v)$, and of course $e(V(I_t),P_t) = e(V(I_{t-1}),P_{t-1})$ if $c_t(v) = c_{t-1}(v)$.

We conclude that

$$\begin{aligned} \mathbb{E}\left[e(V(I_t),P_t) \mid C \subseteq I_{t-1}\right] \leqslant e(V(I_{t-1}),P_{t-1}) + \frac{\Delta}{\Delta+1}(2\Delta + 1 - 3\bar{d}_{IP}) \\ + \frac{1}{|V(I_{t-1})|}\sum_{x\in P_{t-1}}\sum_{v\in V(I_{t-1})\cap N(x)} \frac{1}{\Delta+1}\big(\Delta - d(x,V(I_{t-1}))\big), \end{aligned}$$

where the last term can be rewritten as

$$\frac{1}{|V(I_{t-1})|}\sum_{x\in P_{t-1}} \frac{d(x,V(I_{t-1}))\big(\Delta - d(x,V(I_{t-1}))\big)}{\Delta+1}.$$

We note that the summand above can be written as $f(d(x,V(I_{t-1}))$ where $f(y) := y(\Delta - y)/(\Delta+1)$ is a concave function. Hence, by Jensen's inequality, we can upper bound the expression by $|P_{t-1}|f(\bar{d}_{PI})/|V(I_{t-1})| = \bar{d}_{IP}(\Delta - \bar{d}_{PI})/(\Delta+1)$. Altogether we get

$$\begin{aligned} \mathbb{E}\left[e(V(I_t),P_t) \mid C \subseteq I_{t-1}\right] &\leqslant e(V(I_{t-1}),P_{t-1}) + \frac{\Delta(2\Delta+1-3\bar{d}_{IP})}{\Delta+1} + \frac{\bar{d}_{IP}(\Delta-\bar{d}_{PI})}{\Delta+1} \\ &\leqslant e(V(I_{t-1}),P_{t-1}) + \frac{\Delta}{\Delta+1}(2\Delta+1-2\bar{d}_{IP} - \bar{d}_{IP}\bar{d}_{PI}/\Delta). \end{aligned}$$

Finally, by combining this with Claim 2.2 and Claim 2.4 (and some tedious calculation) we deduce

$$\begin{aligned} \mathbb{E}\left[\Phi(t) \mid C \subseteq I_{t-1}\right] &\leqslant \Phi(t-1) - \frac{1}{\Delta+1} - \frac{1}{10}\frac{\Delta-\bar{d}_{IP}}{\Delta+1} + \frac{1}{100}\frac{2\Delta+1-2\bar{d}_{IP}-\bar{d}_{IP}\bar{d}_{PI}/\Delta}{\Delta+1} \\ &\leqslant \Phi(t-1) - \frac{2}{25}\frac{\Delta-\bar{d}_{IP}}{\Delta+1} - \frac{1}{100}\frac{\bar{d}_{IP}\bar{d}_{PI}}{\Delta(\Delta+1)}. \end{aligned} \tag{5}$$

With all these preparations we are now in a position to bound $\mathbb{E}[\Phi(t)]$. As seen in (4) and (5), both conditioning on components of size at least three or on vertices in isolated edges lead to a non-positive contribution to the drift. In order to derive an upper bound on $\mathbb{E}[\Phi(t)]$ we may thus ignore one of the terms for convenience. If we assume $|I_{t-1}| \leqslant |M_{t-1}|/2$ one would expect that the larger contribution to the change of $\Phi(t-1)$ comes from components which are not isolated edges. Indeed, in that case we may ignore the term from (5) and use (4) only to get

$$\mathbb{E}[\Phi(t)] \overset{(4)}{\leqslant} \Phi(t-1) - \sum_{C,\,|V(C)|\geqslant 3} \frac{|V(C)|}{|V(M_{t-1})|}\frac{\bar{d}(C)}{25} = \Phi(t-1) - \frac{2|M_{t-1}\setminus I_{t-1}|}{25|V(M_{t-1})|}$$
$$\leqslant \Phi(t-1) - \frac{|M_{t-1}|}{25n} \leqslant \Phi(t-1)\Big(1-\frac{1}{50n}\Big),$$

where the last inequality follows from Claim 2.6.

On the other hand, suppose $|I_{t-1}| \geqslant |M_{t-1}|/2$ and observe that this implies $|V(I_{t-1})| \geqslant |V(M_{t-1})|/2$. This means that the probability of picking a vertex in $V(I_{t-1})$ to recolor is at least $1/2$ and one may hope that the larger contribution to the change of $\Phi(t-1)$ comes from the isolated edges. Indeed, similarly as above, we now ignore the contribution from components of size at least three to get:

$$\mathbb{E}[\Phi(t)] \overset{(5)}{\leqslant} \Phi(t-1) - \frac{1}{25}\frac{\Delta-\bar{d}_{IP}}{\Delta+1} - \frac{1}{200}\frac{\bar{d}_{IP}\bar{d}_{PI}}{\Delta(\Delta+1)} \leqslant \Phi(t-1) - \frac{1}{50}\frac{\Delta-\bar{d}_{IP}}{\Delta} - \frac{1}{400}\frac{\bar{d}_{IP}\bar{d}_{PI}}{\Delta^2}. \quad (6)$$

If $\bar{d}_{IP} \leqslant \Delta - \Delta\Phi(t-1)/(30n)$, then the claim follows just from the first term. Otherwise, by Claim 2.6

$$\Phi(t-1) \leqslant 2|M_{t-1}| \leqslant 4|I_{t-1}| \leqslant 2n,$$

which in turn implies $\bar{d}_{IP} \geqslant \Delta(1-\Phi(t-1)/(30n)) \geqslant 14\Delta/15$. Recall, $\bar{d}_{PI}|P_{t-1}| = \bar{d}_{IP}|V(I_{t-1})|$, and note that $|V(I_{t-1})|/|P_{t-1}| \geqslant 2|I_{t-1}|/n \geqslant \Phi(t-1)/(2n)$. Therefore,

$$\frac{1}{400}\frac{\bar{d}_{IP}\bar{d}_{PI}}{\Delta^2} = \frac{1}{400}\frac{|V(I_{t-1})|\cdot\bar{d}_{IP}\bar{d}_{IP}}{|P_{t-1}|\Delta^2} \geqslant \Phi(t-1)\frac{1}{800n}\cdot\Big(\frac{14}{15}\Big)^2$$

and the second term in (6) is enough to conclude the proof of Claim 2.7. ◁

As mentioned in the paragraph before Claim 2.7, in order to make use of the assertion of Claim 2.7, we need a slightly different drift theorem, one for multiplicative drift.

▶ **Theorem 2.8** (Multiplicative Drift Theorem [4]). *Let $(X_t)_{t\geqslant 0}$ be a sequence of non-negative random variables with a finite state space $\mathcal{S} \subset \mathbb{R}_0^+$ such that $0 \in \mathcal{S}$. Let $s_{\min} := \min\{\mathcal{S}\setminus\{0\}\}$, let $s_0 \in \mathcal{S}\setminus\{0\}$, and let $T := \inf\{t \geqslant 0 \mid X_t = 0\}$. If there exists $\delta > 0$ such that for all $s \in \mathcal{S}\setminus\{0\}$ and for all $t > 0$,the case of*

$$\mathbb{E}[X_t - X_{t-1} \mid X_{t-1} = s] \leqslant -\delta s,$$

then

$$\mathbb{E}[T \mid X_0 = s_0] \leqslant \frac{1+\ln(s_0/s_{\min})}{\delta}.$$

Now we are ready to put things together to prove Theorem 1.1.

Proof of Theorem 1.1. For every $t \geqslant 0$, we define

$$\Phi'(t) = \begin{cases} \Phi(t), & \text{if } \Phi(t) \geqslant n/\Delta, \\ 0, & \text{otherwise.} \end{cases}$$

Note that, as long as $\Phi(t-1) \geqslant n/\Delta$, we have $\Phi'(t-1) = \Phi(t-1)$ and $\Phi'(t) \leqslant \Phi(t)$, so the deduced bound on $\Phi(t)$ in Claim 2.7 is also a bound for $\Phi'(t)$. Using Theorem 2.8 with Claim 2.7 for $T' := \inf\{t \geqslant 0 \mid \Phi'(t) = 0\} = \inf\{t \geqslant 0 \mid \Phi(t) < n/\Delta\}$, we get for all $s_0 > 0$

$$\mathbb{E}[T' \mid \Phi'(0) = s_0] \leqslant \frac{1+\ln\left(\frac{s_0}{n/\Delta}\right)}{(1000n)^{-1}}.$$

By Claim 2.6 we have $\Phi'(0) \leqslant 2|M_0| \leqslant n\Delta$, and therefore

$$\mathbb{E}[T'] \leqslant 1000n\big(1 + 2\ln\Delta\big) = O(n \log \Delta).$$

Finally, as by Claim 2.6 we then have $|M_{T'}| = O(n/\Delta)$, we conclude from Proposition 2.3 that the expected number of steps after T' to reach a legal coloring is $O(n)$. Therefore, the total number of required steps to reach a legal coloring is $O(n \log \Delta)$, which finishes the proof of Theorem 1.1. ◀

3 Tail Bounds

In this section, we prove that the runtime of the decentralized coloring algorithm is of order $O(n \log \Delta)$ not only in expectation, but also with high probability. It turns out that this does not require much additional work, as the drift theorems are accompanied with suitable tail bounds. In many situations, concentration bounds require conditions beyond the drift, for example bounds on the step size. Notably, for multiplicative drift such additional conditions are not necessary, as the following theorem holds.

▶ **Theorem 3.1** (Multiplicative Drift Tail Bound [3]). *Let $(X_t)_{t\geqslant 0}$ be a sequence of non-negative random variables with a finite state space $\mathcal{S} \subset \mathbb{R}_0^+$ such that $0 \in \mathcal{S}$. Let $s_{\min} := \min\{\mathcal{S} \setminus \{0\}\}$, let $s_0 \in \mathcal{S} \setminus \{0\}$, and let $T := \inf\{t \geqslant 0 \mid X_t = 0\}$. Suppose that $X_0 = s_0$, and that there exists $\delta > 0$ such that for all $s \in \mathcal{S} \setminus \{0\}$ and for all $t > 0$,*

$$\mathbb{E}[X_t - X_{t-1} \mid X_{t-1} = s] \leqslant -\delta s.$$

Then, for all $r \geqslant 0$

$$\Pr\left[T > \left\lceil \tfrac{r+\ln(s_0/s_{\min})}{\delta} \right\rceil\right] < e^{-r}.$$

The following proposition is a straightforward application of this theorem.

▶ **Proposition 3.2.** *Suppose $\Delta = \Omega(n^c)$ for some constant $c > 0$. Then, the decentralized coloring algorithm terminates after $O(n \log \Delta)$ steps with high probability.*

Proof. By Claim 2.6 we know that $s_0 \leqslant n\Delta$. In the proof of Theorem 1.1 we analyzed the process with multiplicative drift until the potential Φ hits n/Δ. Here, we track this potential until the end of the algorithm. Note that the smallest nonzero value Φ can attain is at least 1. Thus, under the assumption $\Delta = \Omega(n^c)$, we also have $\ln(s_0/s_{\min}) \leqslant C \log \Delta$ for large enough n, even if we do not truncate Φ. Here, $C > 0$ is a suitable constant, for example $C = 2(1 + 1/c)$. Setting $r = \log \Delta$, we can apply Theorem 3.1 to get

$$\Pr\left[T > \lceil (1 + C)1000n \log \Delta \rceil\right] \leqslant e^{-\log \Delta} = O(n^{-c}) = o(1),$$

where we used $\delta = (1000n)^{-1}$ as before, due to Claim 2.7. ◀

The proof of Proposition 3.2 fails when $\log n = \omega(\log \Delta)$. In the proof of Theorem 1.1 we switched to additive drift to analyze the second phase of the process. We use this approach again. The following tail bound for additive drift will be useful. It is a rather straightforward consequence of Azuma's inequality. Note that there is an additional assumption, namely that we have bounded step size.

▶ **Theorem 3.3** (Additive Drift Tail Bound [7]). *Let $(X_t)_{t\geqslant 0}$ be a sequence of non-negative random variables with a finite state space $\mathcal{S}\subset\mathbb{R}_0^+$ such that $0\in\mathcal{S}$. Let $T:=\inf\{t\geqslant 0\mid X_t=0\}$. Suppose there are $c,\delta>0$ such that for all $s\in\mathcal{S}\setminus\{0\}$ and for all $t>0$, we have both $\mathbb{E}[X_t-X_{t-1}\mid X_t=s]\leqslant-\delta$ and $|X_{t+1}-X_t|<c$. Then, for all $r\geqslant 2X_0/\delta$,*

$$\Pr[T\geqslant r]\leqslant\exp\Big(-\frac{r\delta^2}{8c^2}\Big).$$

The smaller Δ is, the less the potential changes at each step. Using this fact, the theorem above allows us to prove the next proposition. It gives us that the runtime of the algorithm is $O(n\log\Delta)$ for smaller Δ.

▶ **Proposition 3.4.** *If $\Delta=O(n^{1/4})$, the decentralized coloring algorithm terminates after $O(n\log\Delta)$ steps with high probability*

Proof. We go back to splitting the process in two phases as in the proof of Theorem 1.1. Let T_1 and T_2 be the duration of Phase 1 and 2 respectively. We consider Phase 1 first. To be able to apply Theorem 3.3, we use the potential function $\Psi(t):=\max\{\log(\Delta\Phi(t)/n),0\}$. As we will see, the logarithm converts multiplicative drift into additive drift. Note that $T_1=\inf\{t\geqslant 0\mid\Psi(t)=0\}$. Using Jensen's inequality and Claim 2.7, we get for all $s\in S\setminus\{0\}$ and $t\geqslant 0$

$$\begin{aligned}
\mathbb{E}[\Psi(t+1)-\Psi(t)\mid\Psi(t)=s]&=\mathbb{E}\left[\log\left(\frac{\Delta\Phi(t+1)}{n}\right)-\log\left(\frac{\Delta\Phi(t)}{n}\right)\middle|\Psi(t)=s\right]\\
&=\mathbb{E}\left[\log\left(\frac{\Phi(t+1)}{\Phi(t)}\right)\middle|\Psi(t)=s\right]\\
&\leqslant\log\mathbb{E}\left[\frac{\Phi(t+1)}{\Phi(t)}\middle|\Psi(t)=s\right]\\
&\leqslant\log\left(1-\frac{1}{1000n}\right)\\
&\leqslant-\frac{1}{1000n}.
\end{aligned}$$

The last inequality follows as $\log x\leqslant x-1$ for all $x>0$. Now that we have determined the drift, we need to bound the step size. $|M_t|$ and $|I_t|$ can change by at most Δ at each step and $e(V(I_t,P_t)$ by at most Δ^2. Thus, the step size of Φ is bounded by 2Δ. As the logarithm is concave, the largest effect of such a change in Φ on the value of Ψ is when Φ is as small as possible. In particular, this is the case if Φ goes from $2\Delta+n/\Delta$ to n/Δ. Thus we have

$$|\Psi(t+1)-\Psi(t)|\leqslant\left|\log\left(\frac{2\Delta+\frac{n}{\Delta}}{\frac{n}{\Delta}}\right)\right|=\log\left(1+\frac{2\Delta^2}{n}\right)<\frac{2\Delta^2}{n}.$$

Therefore, $2\Delta^2/n$ is a bound on the step size of Ψ. As $\Phi(0)\leqslant n\Delta$ we have $\Psi(0)\leqslant 2\log\Delta$. Hence we can use Theorem 3.3 with $r=4000n(1+\log\Delta)$. We get

$$\Pr[T_1\geqslant r]\leqslant\exp\left(-\frac{4000n(1+\log\Delta)\cdot\left(\frac{1}{1000n}\right)^2}{8\left(\frac{2\Delta^2}{n}\right)^2}\right)=\exp\left(-\frac{(1+\log\Delta)n}{8000\Delta^4}\right)=o(1).$$

We turn our attention to Phase 2. Here, we already have additive drift for $|M_t|$, so we can use Theorem 3.3 immediately. Claim 2.2 gives us

$$\mathbb{E}\left[|M_t|-|M_{t-1}|\;\middle|\;|M_{t-1}|=s\right]\leqslant-\frac{1}{\Delta+1}\leqslant-\frac{1}{2\Delta},$$

for all $s, t > 0$. We also have $\big||M_t| - |M_{t-1}|\big| \leqslant \Delta$ for all $t > 0$. By Claim 2.6 we get $|M_t| \leqslant 2n/\Delta$ at the start of Phase 2. Hence we can apply Theorem 3.3 with $r = (4 + \log \Delta)n$. We get

$$\Pr[T_2 \geqslant r] \leqslant \exp\left(-\frac{(4+\log\Delta)n \cdot \left(\frac{1}{2\Delta}\right)^2}{8\Delta^2}\right) = \exp\left(-\frac{(4+\log\Delta)n}{32\Delta^4}\right) = o(1).$$

Using a union bound gives us that the algorithm terminates in $O(n \log \Delta)$ round with high probability. ◀

Proposition 3.2 and 3.4 cover all possible values of Δ. Therefore, the algorithm has runtime $O(n \log \Delta)$ with high probability for any Δ.

4 A simultaneous-recoloring variant of the algorithm

A natural question is whether the original algorithm can be parallelized. So what if instead of choosing one conflicted vertex at a time in Step 2 *all* conflicted vertices would simultaneously want to change their color? It turns out that this process does not even have polynomial runtime on the complete graph K_n.

▶ **Proposition 4.1.** *The Decentralized Graph Colouring algorithm in which all conflicted vertices choose a new color uniformly at random needs $e^{\Omega(n)}$ rounds in expectation to terminate on a complete graph on n vertices K_n.*

Proof. Fix a sufficiently small constant $\varepsilon > 0$, e.g. $\varepsilon = 0.1$. For a round t, let X_t be the number of conflicted vertices, i.e., the number of vertices whose color is not unique. Due to symmetry, X_t is a Markov process. Let T_ε be the first round in which $X_t \leqslant \varepsilon n$. We show that T_ε has exponentially large expectation. Consider any round t with $X_t = x > \varepsilon n$. Then we show that

$$\Pr[X_{t+1} \leqslant \varepsilon n \mid X_t = x] = e^{-\Omega(n)}, \tag{7}$$

where the hidden constant is independent of x. We remark that the same argument also shows that with high probability $X_1 > \varepsilon n$, since the initial round is formally equivalent to the hypothetical case $X_0 = n$. So the proposition follows if we can show (7).

To show (7), we uncover the new colors in two batches. In the first batch, we uncover the colors of all but εn vertices. If there are more than εn vertices in conflicts from the first batch, then there is nothing to show. So in the following we may assume (and implicitly condition) on the opposite event that uncovering the first batch creates at most εn conflicted vertices. This implies that the set C_1 of colors appearing among the $(1-\varepsilon)n$ uncovered vertices, has size at least $|C_1| \geqslant (1-2\varepsilon)n$. Let $C_2 \subseteq C_1$ be the set of colors in C_1 that also appear in the second batch, i.e., for which a conflict is created by the second batch. The probability that a fixed color in C_1 does *not* occur in C_2 is $(1-1/n)^{\varepsilon n} = e^{-\varepsilon + O(1/n)} \leqslant 1 - 7\varepsilon/8$ for sufficiently large n, where we use that $e^{-\varepsilon} < 1 - 7\varepsilon/8$ for $\varepsilon < 0.2$. Hence, $\mathbb{E}[|C_2|] \geqslant 7\varepsilon/8 \cdot (1-2\varepsilon)n \geqslant 5\varepsilon/8 \cdot n$ for $\varepsilon \leqslant 1/7$.

The size of C_2 is given by the number of non-empty bins in a *Balls-and-Bins* problem, and this number is known to be concentrated around its expectation since the number of empty bins is negatively associated, and thus the Chernoff bounds are applicable. Since this is a well-known argument, we refrain from spelling out the details and refer the reader to the standard exposition [5, Proposition 29 and Section 3.3]. The result is that $\Pr[|C_2| \leqslant \varepsilon/2 \cdot n] = e^{-\Omega(n)}$.

It remains to observe that $X_{t+1} > 2|C_2|$, since every color in C_2 causes at least two conflicted vertices (one from the second batch and one from the rest). Hence, $\Pr[X_{t+1} \leqslant \varepsilon n] \leqslant \Pr[|C_2| \leqslant \varepsilon/2 \cdot n] = e^{-\Omega(n)}$, as required. ◀

References

1 Apurv Bhartia, Deeparnab Chakrabarty, Krishna Chintalapudi, Lili Qiu, Bozidar Radunovic, and Ramachandran Ramjee. IQ-Hopping: distributed oblivious channel selection for wireless networks. In *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 81–90. ACM, 2016. doi:10.1145/2942358.2942376.

2 Deeparnab Chakrabarty and Paul de Supinski. On a Decentralized $(\Delta + 1)$-Graph Coloring Algorithm. In *Symposium on Simplicity in Algorithms*, pages 91–98. SIAM, 2020. doi:10.1137/1.9781611976014.13.

3 Benjamin Doerr and Leslie Ann Goldberg. Adaptive drift analysis. *Algorithmica*, 65(1):224–250, 2013. doi:10.1007/s00453-011-9585-3.

4 Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64(4):673–697, 2012. doi:10.1007/s00453-012-9622-x.

5 Devdatt P Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996. doi:10.7146/brics.v3i25.20006.

6 Jun He and Xin Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004. doi:10.1023/b:naco.0000023417.31393.c7.

7 Timo Kötzing. Concentration of first hitting times under additive drift. *Algorithmica*, 75(3):490–506, 2016. doi:10.1007/s00453-015-0048-0.

8 Johannes Lengler. Drift analysis. In Benjamin Doerr and Frank Neumann, editors, *In: Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 89–131. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-29414-4_2.

Noisy, Greedy and Not so Greedy k-Means++

Anup Bhattacharya
Indian Statistical Institute, Kolkata, India

Jan Eube
University of Bonn, Germany

Heiko Röglin
University of Bonn, Germany

Melanie Schmidt
University of Cologne, Germany

Abstract

The k-means++ algorithm due to Arthur and Vassilvitskii [4] has become the most popular seeding method for Lloyd's algorithm. It samples the first center uniformly at random from the data set and the other $k-1$ centers iteratively according to D^2-sampling, i.e., the probability that a data point becomes the next center is proportional to its squared distance to the closest center chosen so far. k-means++ is known to achieve an approximation factor of $\mathcal{O}(\log k)$ in expectation.

Already in the original paper on k-means++, Arthur and Vassilvitskii suggested a variation called *greedy k-means++ algorithm* in which in each iteration multiple possible centers are sampled according to D^2-sampling and only the one that decreases the objective the most is chosen as a center for that iteration. It is stated as an open question whether this also leads to an $\mathcal{O}(\log k)$-approximation (or even better). We show that this is not the case by presenting a family of instances on which greedy k-means++ yields only an $\Omega(\ell \cdot \log k)$-approximation in expectation where ℓ is the number of possible centers that are sampled in each iteration.

Inspired by the negative results, we study a variation of greedy k-means++ which we call *noisy k-means++ algorithm*. In this variation only one center is sampled in every iteration but not exactly by D^2-sampling. Instead in each iteration an adversary is allowed to change the probabilities arising from D^2-sampling individually for each point by a factor between $1-\epsilon_1$ and $1+\epsilon_2$ for parameters $\epsilon_1 \in [0,1)$ and $\epsilon_2 \geq 0$. We prove that noisy k-means++ computes an $\mathcal{O}(\log^2 k)$-approximation in expectation. We use the analysis of noisy k-means++ to design a moderately greedy k-means++ algorithm.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Facility location and clustering; Theory of computation → Design and analysis of algorithms

Keywords and phrases k-means++, greedy, adaptive sampling

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.18

Funding *Anup Bhattacharya*: supported by an NPDF Fellowship, sponsored by the Government of India.
Heiko Röglin: supported by DFG grant RO 5439/1–1.
Melanie Schmidt: supported by DFG grant SCHM 2765/1–1.

Acknowledgements We thank the reviewers for their detailed comments.

1 Introduction

Clustering is a very important tool in many machine learning applications. The task is to find structure that is hidden in input data in the form of clusters, and to do this in an unsupervised way. Since clusters come with very different properties depending on the application, a variety of clustering algorithms and measures to judge clusterings have arisen in the last decades. Among those, a hugely popular method is Lloyd's algorithm [19] (also called the k-means algorithm), which for example was voted to be one of the ten most influential data mining algorithms in machine learning at the IEEE International Conference on Data Mining (ICDM) in 2006 [24].

Lloyd's algorithm is an iterative local search heuristic operating on points from Euclidean space $\mathbb{R}^d$. The measure that it implicitly strives to optimize is the k-means cost function: For a point set $X \subset \mathbb{R}^d$ and a center set $C \subset \mathbb{R}^d$, the k-means cost function is defined as

$$\Phi(X, C) = \sum_{x \in X} \min_{c \in C} ||x - c||^2,$$

the sum of the squared distances of all points to their respective center. The k-means problem asks to minimize the k-means cost over all choices of C with $|C| = k$. In an optimal solution of the k-means problem, the centers are means of their clusters, and Lloyd's algorithm iterates between computing the means of all clusters as the new center set and reassigning all points to their closest centers to form new clusters. The k-means cost function is also called *sum of squared errors* because when the means are viewed as representatives of the clusters, then the k-means cost is the squared error of this representation.

The k-means problem is NP-hard [3, 20], and it is also hard to approximate to arbitrary precision [5, 18]. On the positive side, constant-factor approximations are possible, and the best known factor is 6.357 due to a break-through result by Ahmadian et al. [2, 18]. However, the constant-factor approximation algorithms for k-means are not very practical. On the other hand, Lloyd's method is hugely popular in practice, but can produce solutions that are arbitrarily bad in the worst case.

A major result in clustering thus was the k-means++ algorithm due to Arthur and Vassilvitskii [4] in 2007, which enhances Lloyd's method with a fast and elegant initialization method that provides an $\mathcal{O}(\log k)$-approximation in expectation. The k-means++ algorithm samples k initial centers by adaptive sampling, where in each step, a point's probability of being sampled is proportional to its cost in the current solution (we will refer to this kind of sampling as D^2-sampling in the following). After sampling k centers, the solution is refined by using Lloyd's algorithm. Algorithm 1 contains pseudo code for the k-means++ algorithm.

The beauty of the algorithm is that it has a bounded approximation ratio of $\mathcal{O}(\log k)$ in expectation, and at the same time computes solutions that are good (much better than $\Theta(\log k)$) on practical tests. By feeding the computed centers into Lloyd's method, the solutions are refined to even better quality. Nevertheless, Arthur and Vassilvitskii show that the approximation ratio of k-means++ is tight in the worst case: They give an (albeit artificial) example where the expected approximation ratio is $\Omega(\log k)$, and this has been extended by now to examples where k-means++ outputs a $\Omega(\log k)$-approximate solution with high probability [8], and even in the plane [7].

Due to its beneficial theoretical and practical properties, k-means++ has by now become the de-facto standard for solving the k-means problem in practice. What is less known is that the original paper [4] and the associated PhD thesis [22] actually propose a possible

Algorithm 1 The k-means++ algorithm [4].

1: Sample a point c_1 independently and uniformly at random from X.
2: Let $C = \{c_1\}$.
3: **for** $i = 2$ to k **do**
4: **for** $x \in X$ **do**
5: $p(x) := \frac{\min_{c \in C} ||x-c||^2}{\sum_{y \in X} \min_{c \in C} ||y-c||^2}$
6: **end for**
7: Sample a point c_i from X, where every $x \in X$ has probability $p(x)$.
8: Update $C = C \cup \{c_i\}$.
9: **end for**
10: Run Lloyd's algorithm initialized with center set C and output the result.

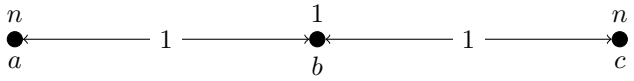


Figure 1 A bad example for the deterministic heuristic that always chooses the current cost minimizer as the next center. An optimal 2-clustering costs less than 1, while a clustering where b is a center costs $\Omega(n)$.

improvement to the k-means++ algorithm: the *greedy k-means++ algorithm*. Here in each of the adaptive sampling steps, not only one center but ℓ possible centers are chosen (independently according to the same probability distribution), and then among these l centers, the one that decreases the k-means cost the most is chosen. This is greedy because a center that reduces the cost in the current step might be a bad center later on (for example if we choose a center that lies between two optimum clusters, thus preventing us from choosing good centers for both on the long run). The original paper [4] says:

> *Also, experiments showed that k-means++ generally performed better if it selected several new centers during each iteration, and then greedily chose the one that decreased Φ (the cost function) as much as possible. Unfortunately, our proofs do not carry over to this scenario. It would be interesting to see a comparable (or better) asymptotic result proven here.*

The intuition is that k-means++ tries to find clusters in the dataset, and with each sample, it tries to find a new cluster that has not been hit by a previously sampled center. This has a failure probability, and the super-constant approximation ratio stems from the probability that some clusters are missed. In this failure event, the algorithm chooses two centers that are close to each other compared to the optimum cost. Greedy k-means++ tries to make this failure event less likely by boosting the probability to find a center from a new cluster that has not been hit previously and greedily choosing the center.

For $\ell = 1$, the greedy k-means++ becomes the k-means++ algorithm, and for very large ℓ it becomes nearly deterministic, a heuristic that always chooses the current minimizer among the whole dataset. It is easy to observe that the latter is not a good algorithm: Consider Figure 1. In the first step, the center that minimizes the overall k-means cost in the next step is b. But if we choose b, then the second greedy center is either a or c, and we end up with a clustering of cost $\Omega(n)$, while the solution $\{a, c\}$ has a cost of 1 (and the optimum solution is even slightly better).

So the crucial question is how to set ℓ, and whether there is an ℓ for which greedy k-means++ outperforms k-means++. It has been shown in [1] that for any optimal clustering of an input data set, k-means++ has in each iteration a constant probability to sample a point from a 'new' optimal cluster, where new means that no point from that cluster has previously been chosen as a center. This leads to a bicriteria approximation, since after $\mathcal{O}(k)$ centers, the algorithm has discovered all optimal clusters in expectation. Following the intuition that stems from this analysis, a natural idea would be to set $\ell = \mathcal{O}(\log k)$: This reduces the probability to pick no point from a new cluster to $\Omega(1/k)$, and by union bound, the failure probability that this event happens in one of the k samples decreases to a constant. This choice is also advertized by Celebi et al. [9], who feature greedy k-means++ in a study of initialization strategies for Lloyd's method. They report that it performs better than k-means++, for a suggested value of $\ell = \log k$. The PhD thesis [22] reports experiments with $\ell = 2$ that outperformed k-means++. It also states that the approximation guarantee of greedy k-means++ is unknown (pp. 62+63).

We initiate the analysis of the greedy k-means++ algorithm. Firstly, we prove that greedy k-means++ is *not* asymptotically better than k-means++. More precisely, we show the following statement.

▶ **Theorem 1.** *For any $k \geq 4$ and any ℓ, there exists a point set $X_{k,\ell}$ such that the expected approximation guarantee of greedy k-means++ is $\Omega(\min\{\ell, k/\log k\} \cdot \log k)$.*

Theorem 1 implies that the worst-case approximation guarantee of greedy k-means++ cannot get better by choosing $\ell > 1$. In particular for $\ell = \log k$, the approximation guarantee worsens to $\Omega(\log^2 k)$.

As indicated in the quote from [4] above, the original proof of k-means++ does not carry over to greedy k-means++, not even if we aim for a higher approximation guarantee like $\mathcal{O}(\ell \log k)$. Roughly speaking, the main problem in the analysis is that while the probability to choose a point as a center can only be increased by a factor of ℓ by the greedy procedure, there is no multiplicative lower bound on how much individual probabilities can be *de*creased. Indeed, if a point $x \in P$ is the worst greedy choice, then its probability to be chosen decreases from some $p(x)$ in the original k-means++ algorithm to $(p(x))^\ell$, which is much smaller than $p(x)$. If this happened to good centers, it could hurt the approximation factor badly.

We proceed to study a different variation of k-means++ which we call the *noisy k-means++* algorithm. This algorithm performs k-means++, but does not sample with exact probabilities. Instead of sampling a point x with probability $p(x)$ as suggested by D^2-sampling, it uses an arbitrary probability $p'(x)$ with $(1-\epsilon_1)p(x) \leq p'(x) \leq (1+\epsilon_2)p(x)$, where $\epsilon_1 \in [0, 1)$ and $\epsilon_2 \geq 0$. If we cast greedy k-means++ as a noisy k-means++ algorithm, we observe that we get a trivial upper bound of $\epsilon_2 = \ell - 1$, however, no trivial *lower* bound on how much the probabilities are skewed.

Noisy k-means++ is also interesting in its own right, since in practice, the probabilities actually computed are prone to rounding errors. Due to the iterative nature of k-means++, it is not at all clear how large the effect of a small rounding can be. We show that the following theorem holds.

▶ **Theorem 2.** *Let T_k denote the set of centers sampled by noisy k-means++ on dataset X and assume that $\frac{k}{\ln k} \geq \max\{18, \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}\}$. Then,*

$$\mathbb{E}[\Phi(X, T_k)] \leq \mathcal{O}\left(\left(\frac{1+\epsilon_2}{1-\epsilon_1}\right)^3 \cdot \log^2(k) \cdot \mathrm{OPT}_k(X)\right),$$

where $\mathrm{OPT}_k(X)$ denotes the k-means costs of an optimal k-clustering of X. If $\frac{k}{\ln k} \leq \max\{18, \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}\}$, then $\mathbb{E}[\Phi(X, T_k)] \leq \mathcal{O}\left(\left(\frac{1+\epsilon_2}{1-\epsilon_1}\right)^4 \cdot \log^2\left(\frac{1+\epsilon_2}{1-\epsilon_1}\right) \cdot \mathrm{OPT}_k(X)\right)$.

We use Theorem 2 to analyze a *moderately greedy* variant of k-means++, where the simple idea is that with probability p, we do a normal k-means++ step, and with probability $1-p$, we do a greedy k-means++ step. The idea is that in this variant, a point is never completely disregarded, so we do get a lower bound on the probabilities, yet in many steps, we do still profit from the additional power of greedy k-means++ seen in experiments. For constant p and ℓ, this variant gives an $\mathcal{O}(\log^2 k)$-approximation by Theorem 2.

Techniques

Our lower bound example for greedy k-means++ is close to the original $\Omega(\log k)$ lower bound example in [4] (we contract each cluster to a single location except for one cluster where one point is moved away from the location into the center of the instance, see Section 2).

However, the proof of the lower bound proceeds very differently. Morally, instead of missing clusters (which becomes less likely due to the multiple samples), the failure event is to choose a bad point as a center. This alone is responsible for the $\Omega(\ell \log k)$ lower bound, while the original $\Omega(\log k)$ bound stems from missing clusters.

To analyze noisy k-means++, we build upon an analysis of k-means++ by Dasgupta [12]. Analyzing k-means++ is about 'hitting' clusters. For some fixed optimal solution we call a cluster *covered* if a point is sampled that provides a good enough center for it. An iteration of k-means++ is *wasted* if a point is sampled from an already covered cluster. Dasgupta uses a potential function which accumulates costs over the wasted iterations. To make the connection between k-means++ and this potential function, it is crucial that the expected average cost of the uncovered clusters does not increase over time (in k-means++). For noisy k-means++, this is not true: The cost can increase. We show that the increase can be bounded, roughly by a factor of $\log k$. Then the key difficulty is to analyze the resulting random process which is highly dependent. We analyze an abstract version first and then show how to apply it to the setting of noisy k-means++.

Additional related work

In his master's thesis, Pago [21] shows that for $\ell = \log k$, the example in Figure 1 can be extended such that greedy k-means++ gives an $\Omega(\log k)$-approximation in expectation.

Bachem et al. [6] suggest to speed up k-means++ by replacing the exact sampling according to the probabilities $p(x)$ by a fast approximation based on Markov Chain Monte Carlo sampling. They prove that under certain assumptions on the dataset their algorithm yields the same approximation guarantee in expectation as k-means++, namely $\mathcal{O}(\log k)$. Their algorithm can be viewed as a special case of noisy k-means++. However, their analysis of the approximation factor is based on making the total variation distance between the probability distributions p and p' (in every step) so small that with high probability their algorithm behaves identically to k-means++. In contrast to this, Theorem 2 also applies to choices of ϵ_1 and ϵ_2 for which noisy k-means++ behaves differently from k-means++ with high probability.

Lattanzi and Sohler [17] propose an intermediate improvement step to be executed between the D^2-sampling and Lloyd's algorithm in order to improve the solution quality to a constant factor approximation in expectation. Their algorithm starts with a k-means++ solution and then performs $\mathcal{O}(k \log \log k)$ improvement steps: In each such step, a new center is sampled with D^2-sampling, and if swapping it with an existing center improves the solution, then this swap is performed. While this is a greedy improvement step and thus a bit related to greedy k-means++, their algorithm is closer in spirit to a known local search algorithm by Kanungo et al. [16] which uses center swaps (starting on an arbitrary solution) to obtain a constant-factor approximation, but needs a lot more rounds and is impractical. Very recently, Choo et. al. [10] improved the result by Lattanzi and Sohler and showed that $\mathcal{O}(k)$ swaps are sufficient to achieve a constant factor approximation.

The bicriteria analysis by Aggarwal et al. [1] mentioned above was improved by Wei [23] who showed that for any $\beta > 1$, sampling βk centers with D^2-sampling yields an $\mathcal{O}(1)$-approximation in expectation (with βk centers). Hsu and Telgarsky [15] show that greedy k-means++ for $\ell = \Theta(k)$ leads to a bicriteria O(1)-approximation if $\Theta(k)$ centers are chosen. All above cited works assume that k and d are input parameters; if one of them is a constant, then there exists a PTAS for the problem [11, 13, 14].

In bicriteria results (which, in a sense, also applies to [17] and [10]), the key is to show that a cluster that has not been covered by a good center is found with high probability. For the analysis of greedy k-means++ and noisy k-means++, the main challenge is to bound the expected cost after only k steps.

2 Lower Bound for Greedy k-means++

In this section we construct an instance on which greedy k-means++ yields only an $\Omega(\ell \log k)$-approximation in expectation. More precisely, we analyze Algorithm 2.

Algorithm 2 Greedy k-means++ algorithm [4].

Sample[1] a point c_1 independently and uniformly at random from X.
Let $C = \{c_1\}$.
for $i = 2$ to k **do**
  for $x \in X$ **do**
    $p(x) := \frac{\min_{c \in C} ||x-c||^2}{\sum_{y \in X} \min_{c \in C} ||y-c||^2}$.
  end for
  Sample[1] a set S of ℓ points independently according to this probability distribution.
  Let $c_i = \arg\min_{u \in S} \Phi(X, C \cup \{u\})$.
  Update $C = C \cup \{c_i\}$.
end for
Run Lloyd's algorithm initialized with center set C and output the result.

Note that we only draw one sample in the first step. This is due to the fact that in the first step, k-means++ is guaranteed to discover a new cluster, so there is no reason to draw multiple samples.

The instance is based on a regular $(k-1)$-simplex with side length $\sqrt{2}$. Let the vertices of this simplex be denoted by $v_1, \ldots, v_k$. There are k points each at vertices $v_1, \ldots, v_{k-1}$, $(k-1)$ points at vertex v_k, and there is one point at the center o of the simplex. Let X denote the set of all these points. The simplex can be constructed explicitly in $\mathbb{R}^k$ by letting v_i be the ith canonical unit vector for each i and $o = (1/k, \ldots, 1/k)$. Then it follows that the distance between the center o and any vertex v_i is $\sqrt{(k-1)/k}$.

An optimal clustering $(C_1^\star, \ldots, C_k^\star)$ of this instance is obtained as follows: The clusters $C_1^\star, \ldots, C_{k-1}^\star$ consist of the k points at vertices $v_1, \ldots, v_{k-1}$, respectively, and the cluster $C_k^\star$ consists of the $(k-1)$ points at vertex v_k and the point at the center o. The cost of this clustering is bounded from above by $||o - v_k||^2 = \frac{k-1}{k} = O(1)$.

Consider a k-clustering C obtained by greedy k-means++ that contains the point at o as one of the k centers. The cost of this clustering is at least $(k-1)^2/k = \Omega(k)$ because there exists at least one i such that C has no center at v_i. In the best case this is v_k, which generates the aforementioned cost because the $(k-1)$ points at v_k will be assigned to the center at o. The approximation guarantee of this clustering is $\Omega(k)$. We prove that with sufficiently large probability, greedy k-means++ places one of the centers at o.

Morally, we proceed as follows. We define a failure event F which captures the case that one of the points at v_k is chosen as a center during the execution of greedy k-means++. If this event happens, we cannot show a high lower bound on the approximation guarantee. So we show that F happens at most with constant probability, so with sufficient probability, F does not occur. Then we analyze the probablity that under the condition that F does not occur, o is chosen as a center during the execution of k-means++. This probability increases with every iteration (when the kth center is chosen, there are only o, the points at v_k and the points at one other location v_i left as possible choices). We analyze a simplified random experiment to lower bound the probablity that o is chosen as a center during the iterations $i = 2, \ldots, k$.

[1] In all our algorithms we do sampling with replacement.

▶ **Theorem 1.** *For any $k \geq 4$ and any ℓ, there exists a point set $X_{k,\ell}$ such that the expected approximation guarantee of greedy k-means++ is $\Omega(\min\{\ell, k/\log k\} \cdot \log k)$.*

Proof. Notice that for $\ell = 1$ there is nothing to show since a lower bound of $\Omega(\log k)$ is known for this case. So in the following, we assume that $\ell \geq 2$. Furthermore we consider first only the case that $\ell \leq \frac{k}{20\ln(k-1)}$ and defer the discussion of larger ℓ to the end of the proof.

We consider the point set X constructed above. Consider a k-clustering C obtained by greedy k-means++ that contains the point at o as one of the k centers. The cost of this clustering is at least $(k-1)^2/k = \Omega(k)$ because there exists at least one i such that C has no center at v_i. In the best case this is v_k, which generates the aforementioned cost because the $(k-1)$ points at v_k will be assigned to the center at o. The approximation guarantee of this clustering is $\Omega(k)$. We will prove that with sufficiently large probability, greedy k-means++ places one of the centers at o.

We start the analysis by defining the following events for all $i \in [k]$:

F_i = the center chosen in the ith iteration lies at v_k,

G_i = the center chosen in the ith iteration lies at o,

$H_i = F_i \cup G_i$.

We denote by Φ_i the potential after $i-1$ iterations if in these iterations no point from $C_k^\star$ has been chosen as a center. Since the probability to choose the same v_i more than once is zero, this means that $i-1$ centers from different clusters from $C_1^\star, \ldots, C_{k-1}^\star$ have been chosen. In the remaining $k-i+1$ clusters, k points pay a cost of 2, except for the one point at o which pays $1-1/k$. Thus,

$$\Phi_i = 2((k-i+1)k-1) + 1 - \frac{1}{k}$$

and

$$2((k-i+1)k-1) \leq \Phi_i \leq 2k(k-i+1).$$

We define

$$F = F_1 \cup (F_2 \cap \overline{H_1}) \cup \ldots \cup (F_{k-1} \cap \overline{H_1} \cap \ldots \cap \overline{H_{k-2}})$$

as the event that in one of the first $k-1$ iterations a point at v_k is chosen as a center and that this is the first center chosen from $C_k^\star$. We exclude the last iteration because $\mathbf{Pr}(F_k)$ is significantly higher than $\mathbf{Pr}(F_i)$ for $i \leq k-1$.

We will prove a lower bound for the probability of the event $\overline{F} \cap (G_2 \cup \ldots \cup G_{k-1})$ because if this event happens then the point at o is one of the centers computed by greedy k-means++, i.e., the approximation factor is at least $\Omega(k)$.

If the event F occurs then we cannot prove a lower bound on the approximation guarantee of greedy k-means++. Hence, we will prove an upper bound for the probability of F. Observe that

$$\mathbf{Pr}[F] \leq \sum_{i=1}^{k-1} \mathbf{Pr}[F_i \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}] \cdot \mathbf{Pr}[\overline{H_1} \cap \ldots \cap \overline{H_{i-1}}] \leq \sum_{i=1}^{k-1} \mathbf{Pr}[F_i \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}]$$

and

$$\mathbf{Pr}[F_1] = \frac{k-1}{k^2} \leq \frac{1}{k}.$$

Consider the situation that $1 \leq i-1 \leq k-2$ iterations have already been performed and that in these iterations cluster $C_k^\star$ has not been covered. Then each point from an uncovered cluster $C_j^\star$ with $j < k$ reduces the potential by $2k$. Each point at v_k reduces the potential by $2(k-1)$ and the point at o reduces the potential by

$$(\underbrace{(k-i+1)}_{\geq 2}k-1)(1+1/k)+1-1/k \geq (2k-1)(1+1/k)+1-1/k = 2(k+1-1/k) > 2k.$$

Hence, the points at v_k have the least potential reduction and thus a point at v_k is only selected as new center in iteration i if all ℓ sampled candidates are at v_k. Hence, we obtain

$$\mathbf{Pr}[F_i \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}] = \left(\frac{2(k-1)}{\Phi_i}\right)^\ell.$$

Altogether this implies

$$\mathbf{Pr}[F] \leq \mathbf{Pr}[F_1] + \sum_{i=2}^{k-1} \mathbf{Pr}[F_i \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}] \leq \frac{1}{k} + \sum_{i=2}^{k-1} \left(\frac{2(k-1)}{\Phi_i}\right)^\ell.$$

Together with $\Phi_i \geq 2((k-i+1)k-1)$ this implies

$$\begin{aligned}
\mathbf{Pr}[F] &\leq \frac{1}{k} + \sum_{i=2}^{k-1} \left(\frac{2(k-1)}{2((k-i+1)k-1)}\right)^\ell \\
&= \frac{1}{k} + \sum_{i=2}^{k-1} \left(\frac{k-1}{(k-i+1)k-1}\right)^\ell \\
&\leq \frac{1}{k} + \sum_{i=2}^{k-1} \left(\frac{k}{(k-i+1)k}\right)^\ell \\
&= \frac{1}{k} + \sum_{i=2}^{k-1} \left(\frac{1}{i}\right)^\ell,
\end{aligned}$$

where the inequality in the penultimate line of the calculation follows from $\frac{a}{b} < \frac{a+1}{b+1}$ for $0 < a < b$. Using $\ell \geq 2$ and $k \geq 4$, it follows

$$\mathbf{Pr}[F] \leq \frac{1}{k} + \sum_{i=2}^{k-1} \left(\frac{1}{i}\right)^2 \leq \frac{1}{k} + \sum_{i=2}^{\infty} \left(\frac{1}{i}\right)^2 = \frac{1}{k} + \left(\frac{\pi^2}{6} - 1\right) \leq 0.9.$$

This shows that with constant probability, the failure event F does not occur, i.e., with constant probability none of the points from v_k is chosen as a center in the first $k-1$ iterations.

Now let us consider the probability that the point at o is selected as a center. We have argued above that the potential reduction of the point at o in iteration $2 \leq i \leq k-1$ is larger than $2k$ if cluster $C_k^\star$ has not been covered in the first $i-1$ iterations. We have also seen that any other point reduces the potential by at most $2k$. Hence, in order to select the point at o as center it suffices already if it belongs to the ℓ candidates chosen in iteration i. Denote the event that the jth sample in iteration i is o by G_{ij}. Then for $i \in \{2, \ldots, k-1\}$,

$$\begin{aligned}\mathbf{Pr}[G_i \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}] &= \mathbf{Pr}[\cup_{j=1}^{\ell} G_{ij} \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}] \\ &\geq \sum_{j=1}^{\ell} \mathbf{Pr}[G_{ij} \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}] - \sum_{1 \leq j_1 < j_2 \leq j} \mathbf{Pr}[G_{ij_1} \cap G_{ij_2} \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}] \\ &= \frac{\ell(1-1/k)}{\Phi_i} - \frac{\binom{\ell}{2}(1-1/k)^2}{\Phi_i^2} \\ &\geq \frac{\ell(1-1/k)}{\Phi_i} - \frac{\ell^2(1-1/k)^2}{\Phi_i^2},\end{aligned}$$

where the first inequality follows from Bonferroni inequalities.

Since $\ell \leq k/(20\ln(k-1)) \leq k/2$, we obtain

$$\frac{\ell(1-1/k)}{\Phi_i} \leq \frac{\ell}{\Phi_i} \leq \frac{\ell}{2((k-i+1)k-1)} \leq \frac{\ell}{k} \leq \frac{1}{2}.$$

This is helpful, because for any $a \in \mathbb{R}$ with $0 \leq a \leq 1/2$, it holds that $a - a^2 \geq a/2$. Thus, the previous two inequalities imply

$$\mathbf{Pr}[G_i \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}] \geq \frac{\ell(1-1/k)}{\Phi_i} - \left(\frac{\ell(1-1/k)}{\Phi_i}\right)^2 \geq \frac{\ell(1-1/k)}{2\Phi_i}. \tag{1}$$

Let us now condition on the event $\overline{F}$, which happens with constant probability. Then we can write the probability of the event we care about as

$$\begin{aligned}\mathbf{Pr}[\overline{F} \cap (G_2 \cup \ldots \cup G_{k-1})] &= \mathbf{Pr}[\overline{F}] \cdot \mathbf{Pr}[G_2 \cup \ldots \cup G_{k-1} \mid \overline{F}] = \mathbf{Pr}[\overline{F}] \cdot \sum_{i=2}^{k-1} \mathbf{Pr}[G_i \mid \overline{F}] \\ &\geq \mathbf{Pr}[\overline{F}] \cdot \sum_{i=2}^{k-1} \mathbf{Pr}[G_i \mid \overline{F_1} \cap \ldots \cap \overline{F_{i-1}}],\end{aligned}$$

where we used in the penultimate step that the events G_i are mutually exclusive and in the last step that $\overline{F} \subseteq \overline{F_1} \cap \ldots \cap \overline{F_{i-1}}$. We cannot use (1) directly to bound $\mathbf{Pr}[G_i \mid \overline{F_1} \cap \ldots \cap \overline{F_{i-1}}]$ because the condition is different (in (1) we condition on the event that no point from $C_k^\star$ has been chosen as center in the first $i-1$ iterations while conditioning on $\overline{F_1} \cap \ldots \cap \overline{F_{i-1}}$ only implies that no point at v_k has been chosen as a center).

To prove a lower bound on $\mathbf{Pr}[G_2 \cup \ldots \cup G_{k-1} \mid \overline{F}]$, we consider a different random experiment E. This random experiment consists of $k-2$ iterations numbered from 2 to $k-1$ and each iteration i is successful with probability $\mathbf{Pr}[G_i \mid \overline{H_1} \cap \ldots \cap \overline{H_{i-1}}]$ independent of the other iterations. Then $\mathbf{Pr}[G_2 \cup \ldots \cup G_{k-1} \mid \overline{F}]$ equals the probability that at least one of the iterations of E is successful. Let E' denote the same random experiment as E only with modified success probabilities. In E' iteration i is successful with probability $\frac{\ell(1-1/k)}{2\Phi_i}$. Due to (1) and Bonferroni inequalities and using $k \geq 4$, we obtain

$$\begin{aligned}\mathbf{Pr}[G_2 \cup \ldots \cup G_{k-1} \mid \overline{F}] &= \mathbf{Pr}[\text{at least one success in } E] \\ &\geq \mathbf{Pr}[\text{at least one success in } E'] \\ &\geq \sum_{i=2}^{k-1} \frac{\ell(1-1/k)}{2\Phi_i} - \sum_{2 \leq i < j \leq k-1} \frac{\ell(1-1/k)}{2\Phi_i} \cdot \frac{\ell(1-1/k)}{2\Phi_j} \\ &\geq \sum_{i=2}^{k-1} \frac{\ell(1-1/k)}{4k(k-i+1)} - \sum_{2 \leq i < j \leq k-1} \frac{\ell}{4((k-i+1)k-1)} \cdot \frac{\ell}{4((k-j+1)k-1)}\end{aligned}$$

$$\geq \sum_{i=2}^{k-1} \frac{\ell(1-1/k)}{4k(k-i+1)} - \sum_{2\leq i<j\leq k-1} \frac{\ell}{3k(k-i+1)} \cdot \frac{\ell}{3k(k-j+1)}$$
$$= \frac{\ell(1-1/k)}{4k} \sum_{i=2}^{k-1} \frac{1}{i} - \frac{\ell^2}{9k^2} \sum_{2\leq i<j\leq k-1} \frac{1}{(k-i+1)(k-j+1)}$$
$$\geq \frac{3\ell}{16k} \sum_{i=2}^{k-1} \frac{1}{i} - \frac{\ell^2}{9k^2} \left(\sum_{i=2}^{k-1} \frac{1}{i} \right)^2$$
$$\geq \frac{3\ell}{16k} (\ln(k-1) - 1) - \frac{\ell^2}{9k^2} \ln^2(k-1).$$

For $k \geq 4$, we have $\ln(k-1) - 1 \geq 0.089 \ln(k-1)$. Together with the previous calculation we get

$$\mathbf{Pr}[G_2 \cup \ldots \cup G_{k-1} \mid \overline{F}] \geq 0.0166 \cdot \frac{\ell \cdot \ln(k-1)}{k} - \left(\frac{\ell \cdot \ln(k-1)}{3k} \right)^2$$
$$= \frac{\ell \cdot \ln(k-1)}{k} \cdot \left(0.0166 - \frac{\ell \cdot \ln(k-1)}{9k} \right)$$
$$\geq 0.01 \cdot \frac{\ell \cdot \ln(k-1)}{k},$$

where we used $\ell \leq \frac{0.05 \cdot k}{\ln(k-1)}$ for the last inequality.

Overall we obtain

$$\mathbf{Pr}[\overline{F} \cap (G_2 \cup \ldots \cup G_{k-1})] = \mathbf{Pr}[\overline{F}] \cdot \mathbf{Pr}[G_2 \cup \ldots \cup G_{k-1} \mid \overline{F}]$$
$$\geq 0.1 \cdot 0.01 \cdot \frac{\ell \cdot \ln(k-1)}{k} = \Omega\left(\frac{\ell \cdot \log(k)}{k} \right).$$

If this event happens, then the costs of the clustering are $\Omega(k)$. Hence the expected costs of the clustering computed by greedy k-means++ are $\Omega(\ell \cdot \log(k))$.

Finally let us consider the case $\ell > \frac{k}{20 \ln(k-1)}$. We argue that in this case the approximation guarantee cannot be better than for $\ell = \frac{k}{20 \ln(k-1)}$. To see that this is true, one has to have a closer look at where the upper bound on ℓ has been used in the argument above. It is used twice: once for proving an upper bound on the conditional probability of G_i and once for proving an upper bound on the conditional probability of $G_2 \cup \ldots \cup G_{k-1}$. Both these probabilities increase with ℓ so if ℓ is larger one could simply replace it by $\frac{k}{20 \ln(k-1)}$, leading to a lower bound of $\Omega(k/\log(k) \cdot k) = \Omega(k)$ for the approximation guarantee. ◀

3 Analysis of Noisy k-means++ Seeding

In this section we analyze a noisy seeding procedure, which we call *noisy k-means++* in the following. This procedure iteratively selects k centers from the data set in a similar fashion as k-means++. The only difference is that the probability of sampling a point as the next center is no longer exactly proportional to its squared distance to the closest center chosen so far. The probabilities are only approximately correct. To be more precise, consider an iteration of noisy k-means++. For any point $x \in X$, we denote by p_x the probability that x is chosen by k-means++ as the next center (i.e., p is the uniform distribution in the first iteration and the distribution that results from D^2-sampling in the following iterations). In noisy k-means++ an adversary can choose an arbitrary probability distribution q on X

with $q_x \in [(1-\epsilon_1)p_x, (1+\epsilon_2)p_x]$ for all $x \in X$ where $\epsilon_1 \in [0,1)$ and $\epsilon_2 \geq 0$ are parameters[2]. Then the next center is sampled according to q. This is repeated in every iteration of noisy k-means++ and in every iteration the adversary can decide arbitrarily how to choose q based on the current distribution p that results from D^2-sampling. We analyze the worst-case approximation guarantee provided by noisy k-means++.

The difficulty with noisy k-means++ is that a) it has a high probability to differ from k-means++, and b) the steps are highly dependent on each other, so once the algorithm has deviated, this propagates in the subsequent steps. It may be surprising that such a little change to the algorithm has such a huge effect. After some considerations it is even unclear if noisy k-means++ has any approximation guarantee at all. While we achieve worse guarantees compared to k-means++, we do at least answer this question affirmatively, showing that noisy k-means++ achieves an expected approximation guarantee of $\mathcal{O}(\log^2 k)$. Achieving this requires an intricate analysis of the highly dependent algorithm. We could not make it work with the original proof, so we use an alternative proof by Dasgupta [12] as a starting point. Also in this proof, a crucial step breaks down (the expected average cost of uncovered clusters can now increase, which is not the case for k-means++). This makes the process difficult to analyze and solving this challenge is the main technical contribution of this paper.

Let us first introduce some notation. We denote by $\Phi(X, C)$ the k-means costs of data set X with respect to center set C, i.e.,

$$\Phi(X, C) = \sum_{x \in X} \min_{c \in C} ||x - c||^2.$$

For $c \in \mathbb{R}^d$ we also write $\Phi(X, c)$ instead of $\Phi(X, \{c\})$ and similarly for $x \in \mathbb{R}^d$ we write $\Phi(x, C)$ instead of $\Phi(\{x\}, C)$. Let $\mathrm{OPT}_k(X)$ denote the optimal k-means costs of dataset X. In the following we assume that a data set X is given and we denote by $(C_1^\star, \dots, C_k^\star)$ an optimal k-clustering of X. For a finite set $X \subset \mathbb{R}^d$, we denote by $\mu(X) = \frac{1}{|X|} \sum_{x \in X} x$ its mean. The following lemma is well-known.

▶ **Lemma 3.** *For any finite $X \subset \mathbb{R}^d$ and any $z \in \mathbb{R}^d$,*

$$\Phi(C, z) = \Phi(C, \mu(C)) + |C| \cdot ||z - \mu(C)||^2 = \mathrm{OPT}_1(C) + |C| \cdot ||z - \mu(C)||^2.$$

We call an optimal cluster $C_i^\star$ *covered* by (noisy) k-means++ if at least one point from $C_i^\star$ is selected as a center. Arthur and Vassilvitskii [4] observe that covered clusters are well approximated by k-means++ in expectation. In particular, they show that the expected costs of an optimal cluster $C_i^\star$ with respect to the center set computed by k-means++ are at most $2 \cdot \mathrm{OPT}_1(C_i^\star)$ and $8 \cdot \mathrm{OPT}_1(C_i^\star)$ if the cluster is covered in the first or any of the following iterations, respectively. First of all, we carry these observations over to noisy k-means++. The following two lemmata are straightforward adaptations of Lemma 3.2 and Lemma 3.3 in [4].

▶ **Lemma 4.** *Let c_1 denote the first center chosen by noisy k-means++. For each optimal cluster $C_i^\star$,*

$$\mathbb{E}[\Phi(C_i^\star, c_1) \mid c_1 \in C_i^\star] \leq \frac{2(1+\epsilon_2)}{1-\epsilon_1} \cdot \mathrm{OPT}_1(C_i^\star).$$

[2] For better readability, whenever we write $q(x) \leq (1+\epsilon_2)p_x$, we implicitly require $q(x) \leq \min\{1, (1+\epsilon_2)p_x\}$

Proof. In k-means++ the first center is chosen uniformly at random, i.e., each point from X has a probability of $1/|X|$ of being chosen. In noisy k-means++, all points have a probability in $[(1-\epsilon_1)/|X|, (1+\epsilon_2)/|X|]$ of being chosen. Hence, the probability of choosing a point $x \in C_i^\star$ as the first center conditioned on the first center being chosen from $C_i^\star$ is at most $\frac{1+\epsilon_2}{(1-\epsilon_1)|C_i^\star|}$. This implies

$$\begin{aligned}
\mathbb{E}[\Phi(C_i^\star, \{c_1\})] &\leq \sum_{c \in C_i^\star} \frac{1+\epsilon_2}{(1-\epsilon_1)|C_i^\star|}\, \Phi(C_i^\star, c) \\
&= \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \frac{1}{|C_i^\star|} \sum_{c \in C_i^\star} \Phi(C_i^\star, c) \\
&= \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \frac{1}{|C_i^\star|} \sum_{c \in C_i^\star} (\mathrm{OPT}_1(C_i^\star) + |C_i^\star| \cdot ||c - \mu(C_1^\star)||^2) \qquad \text{(Lemma 3)} \\
&= \frac{2(1+\epsilon_2)}{1-\epsilon_1} \cdot \mathrm{OPT}_1(C_i^\star)
\end{aligned}$$

◀

▶ **Lemma 5.** *Consider an iteration of noisy k-means++ after the first one and let $C \neq \emptyset$ denote the current set of centers. We denote by z the center sampled in the considered iteration. Then for any $C \neq \emptyset$ and any optimal cluster $C_i^\star$,*

$$\mathbb{E}[\Phi(C_i^\star, C \cup \{z\}) \mid C, z \in C_i^\star] \leq \frac{8(1+\epsilon_2)}{1-\epsilon_1} \cdot \mathrm{OPT}_1(C_i^\star).$$

Proof. Conditioned on sampling a point from $C_i^\star$, the probability of choosing point $x \in C_i^\star$ as the next center is at most $\frac{1+\epsilon_2}{1-\epsilon_1} \cdot \frac{\Phi(x,C)}{\Phi(C_i^\star, C)}$. If x is chosen as the next center, the costs of any point $p \in C_i^\star$ become $\min\{\Phi(p,C), ||p-x||^2\}$. This implies

$$\begin{aligned}
\mathbb{E}[\Phi(C_i^\star, C \cup \{z\}) \mid C, z \in C_i^\star] &= \sum_{x \in C_i^\star} \mathbf{Pr}[z = x \mid C] \cdot \Phi(C_i^\star, C \cup \{x\}) \\
&\leq \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \sum_{x \in C_i^\star} \frac{\Phi(x,C)}{\Phi(C_i^\star, C)} \sum_{p \in C_i^\star} \min\{\Phi(p,C), ||p-x||^2\}.
\end{aligned} \tag{2}$$

For any two points $x, p \in C_i^\star$, we can write

$$\Phi(x,C) = \Big(\min_{c \in C} ||x-c||\Big)^2 \leq \Big(\min_{c \in C}(||x-p|| + ||p-c||)\Big)^2 \leq 2\Phi(p,C) + 2||x-p||^2.$$

By summing over all p in $C_i^\star$, we get

$$\Phi(x,C) \leq \frac{2}{|C_i^\star|} \sum_{p \in C_i^\star} \Phi(p,C) + \frac{2}{|C_i^\star|} \sum_{p \in C_i^\star} ||x-p||^2.$$

With (2), this implies that $\mathbb{E}[\Phi(C_i^\star, C \cup \{z\}) \mid C, z \in C_i^\star]$ is bounded from above by

$$\begin{aligned}
&\frac{1+\epsilon_2}{1-\epsilon_1} \cdot \sum_{x \in C_i^\star} \frac{\frac{2}{|C_i^\star|}\sum_{p \in C_i^\star} \Phi(p,C) + \frac{2}{|C_i^\star|}\sum_{p \in C_i^\star} ||x-p||^2}{\Phi(C_i^\star, C)} \sum_{p \in C_i^\star} \min\{\Phi(p,C), ||p-x||^2\} \\
&= \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \sum_{z \in C_i^\star} \frac{\frac{2}{|C_i^\star|}\sum_{p \in C_i^\star} \Phi(p,C)}{\sum_{p \in C_i^\star} \Phi(p,C)} \sum_{p \in C_i^\star} \min\{\Phi(p,C), ||p-z||^2\}
\end{aligned}$$

$$
\begin{aligned}
&+ \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \sum_{z \in C_i^\star} \frac{\frac{2}{|C_i^\star|} \sum_{p \in C_i^\star} ||p-z||^2}{\sum_{p \in C_i^\star} \Phi(p,C)} \sum_{p \in C_i^\star} \min\{\Phi(p,C), ||p-z||^2\} \\
&\leq \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \sum_{z \in C_i^\star} \frac{2}{|C_i^\star|} \sum_{p \in C_i^\star} ||p-z||^2 + \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \sum_{z \in C_i^\star} \frac{2}{|C_i^\star|} \sum_{p \in C_i^\star} ||p-z||^2 \\
&= \frac{4(1+\epsilon_2)}{1-\epsilon_1} \cdot \sum_{z \in C_i^\star} \frac{1}{|C_i^\star|} \sum_{p \in C_i^\star} ||p-z||^2 \\
&= \frac{4(1+\epsilon_2)}{1-\epsilon_1} \cdot \sum_{z \in C_i^\star} \frac{1}{|C_i^\star|} (\mathrm{OPT}_1(C_i^\star) + |C_i^\star| \cdot ||z - \mu(C_i^\star)||^2) \qquad \text{(Lemma 3)} \\
&= \frac{8(1+\epsilon_2)}{1-\epsilon_1} \cdot \mathrm{OPT}_1(C_i^\star)
\end{aligned}
$$

◀

Consider a run of noisy k-means++. For $t \in [k]$, let H_t and U_t denote the set of all points from X that belong after iteration i to covered and uncovered optimal clusters, respectively. Let u_t denote the number of uncovered clusters after iteration t. Furthermore let T_t denote the set of centers chosen by noisy k-means++ in the first t iterations. We say that iteration t is *wasted* if the center chosen in iteration t comes from H_{t-1}, i.e., if in iteration t no uncovered cluster becomes covered.

▶ **Corollary 6.** *For any $t \in [k]$,*

$$\mathbf{E}\left[\Phi(H_t, T_t)\right] \leq \frac{8(1+\epsilon_2)}{1-\epsilon_1} \cdot \mathrm{OPT}_k(X)$$

Proof. Using Lemma 4 and Lemma 5 we obtain

$$
\begin{aligned}
\mathbf{E}\left[\Phi(H_t, T_t)\right] &= \sum_{i=1}^{k} \mathbf{Pr}[C_i^\star \subseteq H_t] \cdot \mathbf{E}\left[\Phi(C_i^\star, T_t) \mid C_i^\star \subseteq H_t\right] \\
&\leq \sum_{i=1}^{k} \mathbf{Pr}[C_i^\star \subseteq H_t] \cdot \frac{8(1+\epsilon_2)}{1-\epsilon_1} \cdot \mathrm{OPT}_1(C_i^\star) \\
&\leq \frac{8(1+\epsilon_2)}{1-\epsilon_1} \cdot \sum_{i=1}^{k} \mathrm{OPT}_1(C_i^\star) \\
&= \frac{8(1+\epsilon_2)}{1-\epsilon_1} \cdot \mathrm{OPT}_k(X).
\end{aligned}
$$

◀

Corollary 6 implies that the covered clusters contribute in expectation at most $O(\mathrm{OPT}_k(X))$ to the costs of the solution computed by noisy k-means++ (assuming ϵ_1 and ϵ_2 to be constants). The not straightforward part is to prove an upper bound for the costs of the clusters that are not covered by noisy k-means++. For this, we adapt the analysis of k-means++ due to Dasgupta [12]. This analysis is based on a potential function that accumulates costs in every wasted iteration. The potential function has the property that the expected value of the potential function in the end can be bounded and that the total costs accumulated are in expectation at least the costs of the uncovered clusters in the end.

Dasgupta crucially uses that the expected average costs of the uncovered clusters do not increase in k-means++. For noisy k-means++ this is not true anymore in general. Hence, we have to adapt the potential function and the analysis. We define $W_i = 1$ if iteration i is wasted and $W_i = 0$ otherwise. We define the potential function as

$$\Psi_k = \sum_{i=2}^{k} W_i \cdot \frac{\Phi(U_i, T_i)}{u_i}.$$

The easier part is to show that the potential can be bounded from above.

▶ **Lemma 7.** *It holds*

$$\mathbf{E}[\Psi_k] \leq \frac{8(1+\epsilon_2)^2}{(1-\epsilon_1)^2} \cdot (\ln(k)+1) \cdot \mathrm{OPT}_k(X).$$

Proof. Let $i \in \{2,\dots,t\}$. In the following calculation we sum over all realizations $\mathcal{F}_{i-1}$ of the first $i-1$ iterations of noisy k-means++. Any realization $\mathcal{F}_{i-1}$ determines the value of $\Phi(U_{i-1},T_{i-1})$ and u_{i-1}. We use the notation $[\dots]_{\mathcal{F}_{i-1}}$ to express that all terms inside the brackets take the values determined by $\mathcal{F}_{i-1}$. Then

$$\begin{aligned}\mathbf{E}\left[W_i \cdot \frac{\Phi(U_i,T_i)}{u_i}\right] &= \sum_{\mathcal{F}_{i-1}} \mathbf{Pr}[\mathcal{F}_{i-1}] \cdot \mathbf{E}\left[W_i \cdot \frac{\Phi(U_i,T_i)}{u_i} \;\middle|\; \mathcal{F}_{i-1}\right] \\ &= \sum_{\mathcal{F}_{i-1}} \mathbf{Pr}[\mathcal{F}_{i-1}] \cdot \mathbf{Pr}[W_i = 1 \mid \mathcal{F}_{i-1}] \cdot \mathbf{E}\left[\frac{\Phi(U_i,T_i)}{u_i} \;\middle|\; \mathcal{F}_{i-1} \cap (W_i = 1)\right]\end{aligned}$$

Since under the condition that iteration i is wasted the average costs of the uncovered clusters cannot increase, we can upper bound the term above by

$$\begin{aligned}&\sum_{\mathcal{F}_{i-1}} \mathbf{Pr}[\mathcal{F}_{i-1}] \cdot \mathbf{Pr}[W_i = 1 \mid \mathcal{F}_{i-1}] \cdot \left[\frac{\Phi(U_{i-1},T_{i-1})}{u_{i-1}}\right]_{\mathcal{F}_{i-1}} \\ &\leq \sum_{\mathcal{F}_{i-1}} \mathbf{Pr}[\mathcal{F}_{i-1}] \cdot \left[\frac{(1+\epsilon_2)\Phi(H_{i-1},T_{i-1})}{(1-\epsilon_1)\Phi(U_{i-1},T_{i-1})} \cdot \frac{\Phi(U_{i-1},T_{i-1})}{u_{i-1}}\right]_{\mathcal{F}_{i-1}} \\ &= \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \sum_{\mathcal{F}_{i-1}} \mathbf{Pr}[\mathcal{F}_{i-1}] \cdot \left[\frac{\Phi(H_{i-1},T_{i-1})}{u_{i-1}}\right]_{\mathcal{F}_{i-1}} \\ &\leq \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \sum_{\mathcal{F}_{i-1}} \mathbf{Pr}[\mathcal{F}_{i-1}] \cdot \left[\frac{\Phi(H_{i-1},T_{i-1})}{k-i+1}\right]_{\mathcal{F}_{i-1}} \\ &= \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \frac{\mathbf{E}[\Phi(H_{i-1},T_{i-1})]}{k-i+1}.\end{aligned}$$

This implies

$$\begin{aligned}\mathbf{E}[\Psi_k] &= \sum_{i=2}^{k} \mathbf{E}\left[W_i \cdot \frac{\Phi(U_i,T_i)}{u_i}\right] \\ &\leq \frac{1+\epsilon_2}{1-\epsilon_1} \cdot \sum_{i=2}^{k} \frac{\mathbf{E}[\Phi(H_{i-1},T_{i-1})]}{k-i+1}.\end{aligned}$$

With Corollary 6 this yields

$$\mathbf{E}[\Psi_k] \leq \frac{8(1+\epsilon_2)^2}{(1-\epsilon_1)^2} \cdot \mathrm{OPT}_k(X) \sum_{i=2}^{k} \frac{1}{k-i+1} \leq \frac{8(1+\epsilon_2)^2}{(1-\epsilon_1)^2} \cdot (\ln(k)+1) \cdot \mathrm{OPT}_k(X). \quad \blacktriangleleft$$

Now our goal is to prove a lower bound on the potential, namely, we want to prove the following lemma, where we use $D(\epsilon_1,\epsilon_2,\ln k) = \max\{18, \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}\} \cdot \ln k$.

▶ **Lemma 8.** *If $k \geq D(\epsilon_1,\epsilon_2,\ln k)$, then set $B(\epsilon_1,\epsilon_2,\ln k) = \frac{4(1+\epsilon_2)}{1-\epsilon_1} \cdot \ln(k) + 2$, and otherwise, set $B(\epsilon_1,\epsilon_2,\ln k) = D(\epsilon_1,\epsilon_2,\ln k)$. Then*

$$\mathbf{E}[\Psi_k] \geq \frac{\mathbf{E}[\Phi(U_k,T_k)]}{B(\epsilon_1,\epsilon_2,\ln k)}.$$

The main technical challenge is to bound the increase in the expected average cost of uncovered clusters, namely, the proof of Lemma 8 heavily depends on the following lemma. We state the lemma below and prove it later. We use the notation $[\ldots]_{\mathcal{F}_i}$ to express that all terms inside the brackets take the values determined by $\mathcal{F}_i$.

▶ **Lemma 9.** *Set* $D(\epsilon_1, \epsilon_2, \ln k) = \max\{18, \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}\} \cdot \ln k$. *If* $k \geq D(\epsilon_1, \epsilon_2, \ln k)$, *then set* $B(\epsilon_1, \epsilon_2, \ln k) = \frac{4(1+\epsilon_2)}{1-\epsilon_1} \cdot \ln(k) + 2$, *and otherwise, set* $B(\epsilon_1, \epsilon_2, \ln k) = D(\epsilon_1, \epsilon_2, \ln k)$. *Then for any* $i \in [k]$ *and any realization* $\mathcal{F}_i$ *of the first* i *iterations*

$$\mathbf{E}\left[\frac{\Phi(U_k, T_k)}{u_k} \,\middle|\, \mathcal{F}_i\right] \leq B(\epsilon_1, \epsilon_2, \ln k) \cdot \left[\frac{\Phi(U_i, T_i)}{u_i}\right]_{\mathcal{F}_i}.$$

We assume Lemma 9 to prove Lemma 8 as follows. Later, we give the proof of Lemma 9.

Proof. For any $i \in \{2, \ldots, k\}$, we obtain using Lemma 9

$$\begin{aligned}
\mathbf{E}\left[W_i \cdot \frac{\Phi(U_i, T_i)}{u_i}\right] &= \sum_{\mathcal{F}_i, [W_i]_{\mathcal{F}_i}=1} \mathbf{Pr}[\mathcal{F}_i] \cdot \left[\frac{\Phi(U_i, T_i)}{u_i}\right]_{\mathcal{F}_i} \\
&\geq \sum_{\mathcal{F}_i, [W_i]_{\mathcal{F}_i}=1} \mathbf{Pr}[\mathcal{F}_i] \cdot \frac{1}{B(\epsilon_1, \epsilon_2, \ln k)} \cdot \mathbf{E}\left[\frac{\Phi(U_k, T_k)}{u_k} \,\middle|\, \mathcal{F}_i\right] \\
&= \frac{1}{B(\epsilon_1, \epsilon_2, \ln k)} \cdot \sum_{\mathcal{F}_i} \mathbf{Pr}[\mathcal{F}_i] \cdot \mathbf{E}\left[W_i \cdot \frac{\Phi(U_k, T_k)}{u_k} \,\middle|\, \mathcal{F}_i\right] \\
&= \frac{1}{B(\epsilon_1, \epsilon_2, \ln k)} \cdot \mathbf{E}\left[W_i \cdot \frac{\Phi(U_k, T_k)}{u_k}\right].
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{E}[\Psi_k] &= \sum_{i=2}^{k} \mathbf{E}\left[W_i \cdot \frac{\Phi(U_i, T_i)}{u_i}\right] \\
&\geq \frac{1}{B(\epsilon_1, \epsilon_2, \ln k)} \cdot \sum_{i=2}^{k} \mathbf{E}\left[W_i \cdot \frac{\Phi(U_k, T_k)}{u_k}\right] \\
&= \frac{1}{B(\epsilon_1, \epsilon_2, \ln k)} \cdot \mathbf{E}\left[\left(\sum_{i=2}^{k} W_i\right) \cdot \frac{\Phi(U_k, T_k)}{u_k}\right] \\
&= \frac{1}{B(\epsilon_1, \epsilon_2, \ln k)} \cdot \mathbf{E}\left[u_k \cdot \frac{\Phi(U_k, T_k)}{u_k}\right] \\
&= \frac{\mathbf{E}[\Phi(U_k, T_k)]}{B(\epsilon_1, \epsilon_2, \ln k)}
\end{aligned}$$

◀

With Lemma 7, Lemma 8 and Corollary 6, we prove the main theorem.

▶ **Theorem 2.** *Let* T_k *denote the set of centers sampled by noisy* k*-means++ on dataset* X *and assume that* $\frac{k}{\ln k} \geq \max\{18, \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}\}$. *Then,*

$$\mathbb{E}[\Phi(X, T_k)] \leq \mathcal{O}\left(\left(\frac{1+\epsilon_2}{1-\epsilon_1}\right)^3 \cdot \log^2(k) \cdot \mathrm{OPT}_k(X)\right),$$

where $\mathrm{OPT}_k(X)$ *denotes the* k*-means costs of an optimal* k*-clustering of* X. *If* $\frac{k}{\ln k} \leq \max\{18, \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}\}$, *then* $\mathbb{E}[\Phi(X, T_k)] \leq \mathcal{O}\left(\left(\frac{1+\epsilon_2}{1-\epsilon_1}\right)^4 \cdot \log^2\left(\frac{1+\epsilon_2}{1-\epsilon_1}\right) \cdot \mathrm{OPT}_k(X)\right)$.

Proof. For $k \geq \max\{18, \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}\} \cdot \ln k$, Lemma 7 and Lemma 8 imply

$$\begin{aligned}\mathbf{E}\left[\Phi(U_k, T_k)\right] &\leq B(\epsilon_1, \epsilon_2, \ln k) \cdot \mathbf{E}\left[\Psi_k\right] \\ &\leq B(\epsilon_1, \epsilon_2, \ln k) \cdot \frac{8(1+\epsilon_2)^2}{(1-\epsilon_1)^2} \cdot (\ln(k)+1) \cdot \mathrm{OPT}_k(X) \\ &= O\left(\frac{(1+\epsilon_2)^3}{(1-\epsilon_1)^3} \cdot \log^2(k) \cdot \mathrm{OPT}_k(X)\right).\end{aligned}$$

With Corollary 6 this implies

$$\begin{aligned}\mathbf{E}\left[\Phi(H_k, T_k) + \Phi(U_k, T_k)\right] &\leq O\left(\frac{1+\epsilon_2}{1-\epsilon_1} \cdot \mathrm{OPT}_k(X)\right) + O\left(\frac{(1+\epsilon_2)^3}{(1-\epsilon_1)^3} \cdot \log^2(k) \cdot \mathrm{OPT}_k(X)\right) \\ &\leq O\left(\frac{(1+\epsilon_2)^3}{(1-\epsilon_1)^3} \cdot \log^2(k) \cdot \mathrm{OPT}_k(X)\right)\end{aligned}$$

For $k \leq \max\{18, \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}\} \cdot \ln k$, we get

$$\begin{aligned}&B(\epsilon_1, \epsilon_2, \ln k) \cdot \frac{8(1+\epsilon_2)^2}{(1-\epsilon_1)^2} \cdot (\ln(k)+1) \cdot \mathrm{OPT}_k(X) \\ &= O\left(\frac{(1+\epsilon_2)^4}{(1-\epsilon_1)^4} \cdot \log^2(k) \cdot \mathrm{OPT}_k(X)\right),\end{aligned}$$

where we use that $\epsilon_1 < 1$, so $\epsilon_1 + \epsilon_2 \leq 1 + \epsilon_2$. This implies

$$\begin{aligned}\mathbf{E}\left[\Phi(H_k, T_k) + \Phi(U_k, T_k)\right] &\leq O\left(\frac{(1+\epsilon_2)^4}{(1-\epsilon_1)^4} \cdot \log^2(k) \cdot \mathrm{OPT}_k(X)\right) \\ &\leq O\left(\frac{(1+\epsilon_2)^4}{(1-\epsilon_1)^4} \cdot \log^2\left(\frac{(1+\epsilon_2)^2}{(1-\epsilon_1)^2}\right) \cdot \mathrm{OPT}_k(X)\right) \\ &= O\left(\left(\frac{1+\epsilon_2}{1-\epsilon_1}\right)^4 \cdot \log^2\left(\frac{1+\epsilon_2}{1-\epsilon_1}\right) \cdot \mathrm{OPT}_k(X)\right),\end{aligned}$$

where we use for the second inequality that either $\sqrt{k} \leq \frac{k}{\ln k} \leq 18$ and then $\ln k \leq O(1)$, or

$$\begin{aligned}\sqrt{k} \leq \frac{k}{\ln k} \leq \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2} &\Rightarrow \ln\sqrt{k} \leq \ln\left(\frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}\right) \\ &\Rightarrow \log k \leq O\left(\log\left(\frac{(1+\epsilon_2)^2}{(1-\epsilon_1)^2}\right)\right).\end{aligned}$$

◀

We conclude this section by discussing Lemma 9. It says that the average potential of uncovered clusters increases by at most a logarithmic multiplicative factor. We first consider the following abstract random experiment whose connection to noisy k-means++ we discuss in the actual proof of Lemma 9 below. Let $a_1, \ldots, a_z \in \mathbb{R}_{\geq 0}$ denote numbers with average value 1. Since there are z numbers with the average equal to one, their sum equals z. We assume that in each step of our experiment with probability $\epsilon \in [0, 1)$ an adversary chooses one of the numbers to be removed and with probability $1 - \epsilon$ a number is removed by proportional sampling (i.e., if number a_i still exists then it is removed with probability a_i/S, where S denotes the sum of the remaining numbers). Note that in this process the number a_i is sampled with probability at least $(1-\epsilon)\frac{a_i}{S}$. Additionally after each step an adversary can arbitrarily lower the value of some numbers. This process is run for ℓ steps and we are interested in an upper bound for the expected average of the numbers remaining after these ℓ steps. We denote this average by A_ℓ.

▶ **Lemma 10.** *Let $\epsilon \in (0,1)$, assume that $\frac{z}{\ln z} \geq \max\{18, \frac{24\epsilon}{(1-\epsilon)^2}\}$, and $\ell \geq z/2$. Then $\mathbf{E}[A_\ell] \leq \frac{4}{1-\epsilon} \cdot \ln(z) + 2$. For $z \leq \max\{18, \frac{24\epsilon}{(1-\epsilon)^2}\} \cdot \ln z$, we observe that $\mathbf{E}[A_\ell] \leq z \leq \max\{18, \frac{24\epsilon}{(1-\epsilon)^2}\} \cdot \ln z$.*

Proof. Let Z denote the number of adversarial steps among the first ℓ steps. Then $\mathbf{E}[Z] = \epsilon\ell$. We denote by $\mathcal{F}_1$ the event that $Z \geq \frac{1+\epsilon}{2} \cdot \ell$. Note that $(1+\epsilon)/2 = \epsilon + (1-\epsilon)/2$ always lies between ϵ and 1.

By Chernoff bound we get

$$\begin{aligned}\mathbf{Pr}[\mathcal{F}_1] = \mathbf{Pr}\left[Z \geq \frac{1+\epsilon}{2}\ell\right] &= \mathbf{Pr}\left[Z \geq \frac{1+\epsilon}{2\epsilon}\epsilon\ell\right] \\ &= \mathbf{Pr}\left[Z \geq \left(1 + \frac{1-\epsilon}{2\epsilon}\right) \cdot \mathbf{E}[Z]\right] \leq \exp\left(-\frac{\min\{\delta, \delta^2\} \cdot \mathbf{E}[Z]}{3}\right)\end{aligned}$$

for $\delta = \frac{1-\epsilon}{2\epsilon}$. We make a case analysis. For $\frac{1-\epsilon}{2\epsilon} \geq 1 \Leftrightarrow \epsilon \leq \frac{1}{3}$, $\min\{\delta, \delta^2\} = \delta$, so we have

$$\mathbf{Pr}[\mathcal{F}_1] \leq \exp\left(-\frac{\delta \cdot \mathbf{E}[Z]}{3}\right) = \exp\left(-\frac{\frac{1-\epsilon}{2\epsilon} \cdot \epsilon\ell}{3}\right) = \exp\left(-\frac{(1-\epsilon)}{6}\ell\right) \leq \exp\left(-\frac{(1-\epsilon)}{12}z\right),$$

where the last inequality follows from $\ell \geq z/2$. We observe that

$$\exp\left(-\frac{(1-\epsilon)}{12}z\right) \leq \frac{1}{z} \Leftrightarrow \frac{z}{\ln z} \geq \frac{12}{1-\epsilon},$$

and by $\epsilon \leq 1/3$ and by our lower bound on $z/\ln z$, we have $\frac{z}{\ln z} \geq 18 \geq \frac{12}{1-\varepsilon}$.

If $\epsilon > 1/3$, we compute similarly that

$$\mathbf{Pr}[\mathcal{F}_1] \leq \exp\left(-\frac{\frac{(1-\epsilon)^2}{(2\epsilon)^2} \cdot \epsilon\ell}{3}\right) \leq \exp\left(-\frac{(1-\epsilon)^2}{12\epsilon}\ell\right) \leq \exp\left(-\frac{(1-\epsilon)^2}{24\epsilon}z\right) \leq \frac{1}{z},$$

where the last inequality follows from $z/(\ln z) \geq \frac{24\epsilon}{(1-\epsilon)^2}$.

If the event $\mathcal{F}_1$ does not happen then in at least $(1 - \frac{1+\epsilon}{2})\ell = \frac{1-\epsilon}{2}\ell \geq \frac{1-\epsilon}{4}z =: z/c'$ steps proportional sampling is used to remove one of the numbers (we set $c' = 4/(1-\epsilon)$). We will show that with high probability after these steps all remaining numbers are at most $2c' \ln z$. Let $\mathcal{F}_2$ denote the event that after z/c' steps of proportional sampling at least one number with final value at least $2c' \ln z$ is remaining. Furthermore, let $\mathcal{E}_i$ denote the event that the ith number a_i remains after z/c' steps of proportional sampling and its final value $\tilde{a}_i$ is at least $2c' \ln z$ (remember that the adversary can decrease numbers during the process but not increase and hence $\tilde{a}_i \leq a_i$). Then $\mathcal{F}_2 = \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_z$. If $\mathcal{E}_i$ occurs then the ith number is in every step at least $\tilde{a}_i \geq 2c' \ln z$. Since the numbers $a_1, \ldots a_z$ have average 1, their sum is z. The sum of the remaining numbers cannot increase during the process. Hence, in every step the probability of taking the ith number is at least $(2c' \ln z)/z$. This implies

$$\mathbf{Pr}[\mathcal{E}_i] \leq \left(1 - \frac{2c' \ln z}{z}\right)^{z/c'} \leq \exp(-2 \ln z) = \frac{1}{z^2}.$$

We use a union bound to obtain

$$\mathbf{Pr}[\mathcal{F}_2] = \mathbf{Pr}[\exists i \in [z] : \mathcal{E}_i] \leq \frac{1}{z}.$$

If neither $\mathcal{F}_1$ nor $\mathcal{F}_2$ occurs then the final value of each remaining number is at most $2c' \ln z$. Hence, in this case, also the average is bounded from above by $2c' \ln z$. Otherwise we only use the trivial upper bound of z for the average of the remaining numbers (observe that initially each a_i is at most z because the average is 1). Altogether we obtain

$$\begin{aligned}\mathbf{E}[A_\ell] &\leq \mathbf{Pr}[\neg\mathcal{F}_1 \wedge \neg\mathcal{F}_2] \cdot 2c' \ln z + \mathbf{Pr}[\mathcal{F}_1 \vee \mathcal{F}_2] \cdot z \\ &\leq 2c' \ln z + (\mathbf{Pr}[\mathcal{F}_1] + \mathbf{Pr}[\mathcal{F}_2]) \cdot z \\ &\leq 2c' \ln z + \left(\frac{1}{z} + \frac{1}{z}\right) \cdot z \\ &= 2c' \ln(z) + 2 = 4/(1-\epsilon) \ln z + 2.\end{aligned}$$

For the second inequality stated in the lemma, we only observe that even if we draw all but one number, the average cannot increase beyond z since the sum of the numbers is z. Thus $\mathbf{E}[A_\ell] \leq z$ is true for any $1 \leq \ell \leq z$. ◀

We prove below that if $\ell < z/2$ then $\mathbf{E}[A_\ell] \leq 2$.

▶ **Lemma 11.** *Let $\ell < z/2$. Then $\mathbf{E}[A_\ell] \leq 2$.*

Proof. In the worst case all steps are adversarial and the ℓ smallest numbers are removed. Then the average of the remaining numbers is at most

$$\frac{z}{z-\ell} < \frac{z}{z-z/2} = 2.$$ ◀

Using Lemma 11, we obtain the following corollary.

▶ **Corollary 12.** *Let $\epsilon \in (0,1)$ and $1 \leq \ell \leq z-1$. Then for $z \geq \max\{18, \frac{24\epsilon}{(1-\epsilon)^2}\} \cdot \ln z$, we get*

$$\mathbf{E}[A_\ell] \leq \frac{4}{1-\epsilon} \cdot \ln z + 2,$$

and for $z \leq \max\{18, \frac{24\epsilon}{(1-\epsilon)^2}\} \cdot \ln z$, we have $\mathbf{E}[A_\ell] \leq \max\{18, \frac{24\epsilon}{(1-\epsilon)^2}\} \cdot \ln z$.

Proof. Follows from Lemma 10 and Lemma 11. ◀

Now we are ready to prove Lemma 9.

Proof of Lemma 9. Given realization F_i, after the first i iterations there are $z = u_i \leq k$ uncovered clusters. Each of them has certain costs with respect to the center set after the first i iterations. The costs of each cluster do not increase in the following iterations anymore because only new centers are added. In any iteration the costs of these clusters may decrease and one uncovered cluster may become covered. If the latter happens, the average costs of the uncovered clusters can increase (if the costs of the uncovered cluster that becomes covered are less than the average costs of the uncovered clusters). Hence, only the non-wasted iterations are of interest.

The costs of the uncovered clusters after the first i iterations correspond to the numbers $a_1, \ldots, a_z$ in the random experiment above. We scaled the instance such that the sum of the a_i is equal to z. This is without loss of generality. In each iteration of noisy k-means++ either a covered cluster is hit again, which can only reduce the numbers a_i, or an uncovered cluster becomes covered, in which case the corresponding number is removed. Conditioned on covering an uncovered cluster, the probability p_i that a_i is removed is at least $\frac{1-\epsilon_1}{1+\epsilon_2} \cdot \frac{a_i}{S}$, where S denotes the sum of the costs of the uncovered clusters (i.e., the sum of the remaining a_i). We can simulate the probability distribution induced by the probabilities p_i

by mixing two distributions: with probability $\frac{1-\epsilon_1}{1+\epsilon_2}$ we do proportional sampling, i.e., we choose a_i with probability $\frac{a_i}{S}$, and with probability $1-\frac{1-\epsilon_1}{1+\epsilon_2}$ we sample according to some other distribution to obtain the right probabilities p_i. In the abstract random experiment analyzed above this second distribution is selected by an adversary. For $\epsilon = \frac{\epsilon_1+\epsilon_2}{1+\epsilon_2} \in (0,1)$ we have

$$1-\epsilon = 1-\frac{\epsilon_1+\epsilon_2}{1+\epsilon_2} = \frac{1+\epsilon_2-(\epsilon_1+\epsilon_2)}{1+\epsilon_2} = \frac{1-\epsilon_1}{1+\epsilon_2}.$$

Hence, Corollary 12 applies to noisy k-means++ with $\epsilon = \frac{\epsilon_1+\epsilon_2}{1+\epsilon_2}$. Observe that then

$$\frac{24\epsilon}{(1-\epsilon)^2} = \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)^2}{(1+\epsilon_2)(1-\epsilon_1)^2} = \frac{24(\epsilon_1+\epsilon_2)(1+\epsilon_2)}{(1-\epsilon_1)^2}.$$ ◀

Bicriteria Approximation

We remark that noisy k-means++ still gives a bicriteria approximation because the probability that an uncovered cluster is hit can only be decreased by a constant factor, and the probability to pick a good center is also still comparably high. The theorem mentioned below follows from [1]. We omit the proof of this theorem in this paper as it easily follows from [1].

▶ **Theorem 13.** *Let S denote a set of $16(\frac{1+\epsilon_2}{1-\epsilon_1})^2(k+\sqrt{k})$ centers sampled using noisy k-means++, then $\Phi(X,S) \leq 20\ \mathrm{OPT}_k(X)$ with probability at least $1-\exp\left(-0.0157\cdot\frac{1-\epsilon_1}{1+\epsilon_2}\right)$.*

Not so greedy k-means++

Consider the following variant of the greedy k-means++ algorithm (Algorithm 3).

Algorithm 3 Moderately greedy k-means++.

Input: Set $X \subseteq \mathbb{R}^d$, integers k, l
Output: $C \subseteq X, |C| = k$
$C = \emptyset$
Sample a point c_1 independently and uniformly at random from X.
Let $C = \{c_1\}$.
for $i = 2$ to k **do**
    With probability p, sample one point c_i with D^2-sampling and set $C = C \cup \{c_i\}$.
    With the remaining probability:
        Sample a set S of ℓ points independently with D^2-sampling from X wrt C.
        Let $c_i = \arg\min_{u\in S} \Phi(X, C\cup\{u\})$.
        Update $C = C \cup \{c_i\}$.
end for
Return C

Let $x \in P$ be any point. Say that $p_i(x)$ is the probability to draw x with one D^2-sample from X based on the center set $c_1,\ldots,c_{i-1}$. Then the probability $q_i(x)$ to sample x in iteration i of the above algorithm satisfies

$$p\cdot p_i(x) \leq q_i(x) \leq [(1-p)\cdot\ell + p]\cdot p_i(x),$$

since with probability p, we do exactly the same as k-means++, and with probability $(1-p)$, we sample ℓ times, which can at most boost the probability by a factor of $(1-p)\cdot\ell$. Assume that p is a constant. Then by Theorem 2, moderately greedy k-means++ has an expected approximation guarantee of $\mathcal{O}(\ell^3\cdot\log^2 k)$ (for large k).

References

1 Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *Proceedings of the 12th and 13th APPROX-RANDOM*, pages 15–28, 2009.

2 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, 2017.

3 Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.

4 David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1027–1035, 2007.

5 Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of euclidean k-means. In *Proceedings of the 31st International Symposium on Computational Geometry (SoCG)*, pages 754–767, 2015.

6 Olivier Bachem, Mario Lucic, S. Hamed Hassani, and Andreas Krause. Approximate k-means++ in sublinear time. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 1459–1467, 2016. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12147`.

7 Anup Bhattacharya, Ragesh Jaiswal, and Nir Ailon. Tight lower bound instances for k-means++ in two dimensions. *Theoretical Computer Science*, 634:55–66, 2016.

8 Tobias Brunsch and Heiko Röglin. A bad instance for k-means++. *Theoretical Computer Science*, 505:19–26, 2013.

9 M. Emre Celebi, Hassan A. Kingravi, and Patricio A. Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, 40(1):200–210, 2013.

10 Davin Choo, Christoph Grunau, Julian Portmann, and Václav Rozhon. k-means++: few more steps yield constant approximation. *CoRR*, abs/2002.07784, 2020. `arXiv:2002.07784`.

11 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 353–364, 2016.

12 Sanjoy Dasgupta. Lecture 3 – Algorithms for k-means clustering, 2013. accessed May 8th, 2019. URL: `http://cseweb.ucsd.edu/~dasgupta/291-geom/kmeans.pdf`.

13 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 569–578, 2011.

14 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. *SIAM Journal on Computing*, 48(2):452–480, 2019.

15 Daniel J. Hsu and Matus Telgarsky. Greedy bi-criteria approximations for k-medians and k-means. *CoRR*, abs/1607.06203, 2016. `arXiv:1607.06203`.

16 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2-3):89–112, 2004.

17 Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3662–3671, 2019.

18 Euiwoong Lee, Melanie Schmidt, and John Wright. Improved and simplified inapproximability for k-means. *Information Processing Letters*, 120:40–43, 2017.

19 Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. originally published as Bell Laboratories Technical Memorandum in 1957.

20 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi R. Varadarajan. The Planar k-means Problem is NP-Hard. In *Proceedings of the 3rd Workshop on Algorithms and Computation (WALCOM)*, pages 274–285, 2009.

21 Benedikt Pago. Upper and lower bounds for the approximation ratios of incremental and hierarchical clustering algorithms. Master's thesis, University of Bonn, 2018.

22 Sergei Vassilvitskii. *k-means: Algorithms, Analyses, Experiments*. PhD thesis, Stanford University, 2007.

23 Dennis Wei. A constant-factor bi-criteria approximation guarantee for k-means++. In *Proceedings of the Annual Conference on Neural Information Processing Systems 2016 (NIPS)*, pages 604–612, 2016.

24 Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

An Algorithmic Study of Fully Dynamic Independent Sets for Map Labeling

Sujoy Bhore
Algorithms and Complexity Group, TU Wien, Austria
sujoy@ac.tuwien.ac.at

Guangping Li
Algorithms and Complexity Group, TU Wien, Austria
guangping@ac.tuwien.ac.at

Martin Nöllenburg
Algorithms and Complexity Group, TU Wien, Austria
noellenburg@ac.tuwien.ac.at

Abstract

Map labeling is a classical problem in cartography and geographic information systems (GIS) that asks to place labels for area, line, and point features, with the goal to select and place the maximum number of independent, i.e., overlap-free, labels. A practically interesting case is point labeling with axis-parallel rectangular labels of common size. In a fully dynamic setting, at each time step, either a new label appears or an existing label disappears. Then, the challenge is to maintain a maximum cardinality subset of pairwise independent labels with sub-linear update time. Motivated by this, we study the maximal independent set (MIS) and maximum independent set (Max-IS) problems on fully dynamic (insertion/deletion model) sets of axis-parallel rectangles of two types – (i) uniform height and width and (ii) uniform height and arbitrary width; both settings can be modeled as rectangle intersection graphs.

We present the first deterministic algorithm for maintaining a MIS (and thus a 4-approximate Max-IS) of a dynamic set of uniform rectangles with amortized sub-logarithmic update time. This breaks the natural barrier of $\Omega(\Delta)$ update time (where Δ is the maximum degree in the graph) for *vertex updates* presented by Assadi et al. (STOC 2018). We continue by investigating Max-IS and provide a series of deterministic dynamic approximation schemes. For uniform rectangles, we first give an algorithm that maintains a 4-approximate Max-IS with $O(1)$ update time. In a subsequent algorithm, we establish the trade-off between approximation quality $2(1 + \frac{1}{k})$ and update time $O(k^2 \log n)$, for $k \in \mathbb{N}$. We conclude with an algorithm that maintains a 2-approximate Max-IS for dynamic sets of unit-height and arbitrary-width rectangles with $O(\omega \log n)$ update time, where ω is the maximum size of an independent set of rectangles stabbed by any horizontal line. We have implemented our algorithms and report the results of an experimental comparison exploring the trade-off between solution quality and update time for synthetic and real-world map labeling instances.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Dynamic graph algorithms

Keywords and phrases Independent Sets, Dynamic Algorithms, Rectangle Intersection Graphs, Approximation Algorithms, Experimental Evaluation

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.19

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.07611`.

Supplementary Material Source code and benchmark data at `https://dyna-mis.github.io/dynaMIS/`.

Funding Research supported by the Austrian Science Fund (FWF), grant P 31119.

1 Introduction

Map Labeling is a classical problem in cartography and geographic information systems (GIS), that has received significant attention in the past few decades and is concerned with selecting and positioning labels on a map for area, line, and point features. The focus in the computational geometry community has been on labeling point features [3, 20, 39, 40]. The labels are typically modeled as the bounding boxes of short names, which correspond precisely to unit height, but arbitrary width rectangles; alternatively, labels can be standardized icons or symbols, which correspond to rectangles of uniform size. In map labeling, a key task is in fact to select an independent (i.e., overlap-free) set of labels from a given set of candidate labels. Commonly the optimization goal is related to maximizing the number of labels. Given a set $\mathcal{R}$ of rectangular labels, Map Labeling is essentially equivalent to the problem of finding a maximum independent set in the intersection graph induced by $\mathcal{R}$.

The independent set problem is a fundamental graph problem with a wide range of applications. Given a graph $G = (V, E)$, a set of vertices $M \subset V$ is *independent* if no two vertices in M are adjacent in G. A *maximal independent set* (MIS) is an independent set that is not a proper subset of any other independent set. A *maximum independent set* (Max-IS) is a maximum cardinality independent set. While Max-IS is one of Karp's 21 classic NP-complete problems [31], computing a MIS can easily be done by a simple greedy algorithm in $O(|E|)$ time. The MIS problem has been studied in the context of several other prominent problems, e.g., graph coloring [33], maximum matching [30], and vertex cover [36]. On the other hand, Max-IS serves as a natural model for many real-life optimization problems, including map labeling [3], computer vision [6], information retrieval [37], and scheduling [38].

Stronger results for independent set problems in geometric intersection graphs are known in comparison to general graphs. For instance, it is known that Max-IS on general graphs cannot be approximated better than $|V|^{1-\epsilon}$ in polynomial time for any $\epsilon > 0$ unless NP=ZPP [27]. In contrast, a randomized polynomial-time algorithm exists that computes for rectangle intersection graphs an $O(\log \log n)$-approximate solution to Max-IS with high probability [12], as well as QPTASs [2,16]. The Max-IS problem is already NP-Hard on unit square intersection graphs [21], however, it admits a polynomial-time approximation scheme (PTAS) for unit square intersection graphs [19] and more generally for pseudo disks [13]. Moreover, for rectangles with either uniform size or at least uniform height and bounded aspect ratio, the size of an MIS is not arbitrarily worse than the size of a Max-IS. For instance, any MIS of a set of uniform rectangles is a 4-approximate solution to the Max-IS problem, since each rectangle can have at most four independent neighbors.

Past research has mostly considered static label sets in static maps [3, 20, 39, 40] and in dynamic maps allowing zooming [7] or rotations [25], but not fully dynamic label sets with insertions and deletions of labels. Recently, Klute et al. [32] proposed a framework for semi-automatic label placement, where domain experts can interactively insert and delete labels. In their setting an initially computed large independent set of labels can be interactively modified by a cartographer, who can easily take context information and soft criteria such as interactions with the background map or surrounding labels into account. Standard map labeling algorithms typically do not handle such aspects well. Based on these modifications (such as deletion, forced selection, translation, or resizing), the solution is updated by a dynamic algorithm while adhering to the new constraints. Another scenario for dynamic labels are maps, in which features and labels (dis-)appear over time, e.g., based on a stream of geotagged, uniform-size photos posted on social media or, more generally, maps with labels of dynamic spatio-temporal point sets [22]. For instance, a geo-located event

that happens at time t triggers the availability of a new label for a certain period of time, after which it vanishes again. Examples beyond social media are reports of earth quakes, forest fires, or disease incidences. Motivated by this, we study the independent set problem for dynamic rectangles of two types – (i) uniform height and width and (ii) uniform height and arbitrary width. We consider fully dynamic algorithms for maintaining independent sets under insertions and deletions of rectangles, i.e., vertex insertions and deletions in the corresponding dynamic rectangle intersection graph.

Dynamic graphs are subject to discrete changes over time, i.e., insertions or deletions of vertices or edges [18]. A dynamic graph algorithm solves a computational problem, such as the independent set problem, on a dynamic graph by updating efficiently the previous solution as the graph changes over time, rather than recomputing it from scratch. A dynamic graph algorithm is called *fully dynamic* if it allows both insertions and deletions, and *partially dynamic* if only insertions or only deletions are allowed. While general dynamic independent set algorithms can obviously also be applied to rectangle intersection graphs, our goal is to exploit their geometric properties to obtain more efficient algorithms.

Related Work. There has been a lot of work on dynamic graph algorithms in the last decade and dynamic algorithms still receive considerable attention in theoretical computer science. We point out some of these works, e.g., on spanners [9], vertex cover [10], set cover [1], graph coloring [11], and maximal matching [23]. In particular, the maximal independent set problem on dynamic graphs with edge updates has attracted significant attention in the last two years [4, 5, 8, 15, 17]. Recently, Henzinger et al. [28] studied the Max-IS problem for intervals, hypercubes and hyperrectangles in d dimensions, with special assumptions. They assumed that the objects are axis-parallel and contained in the space $[0, N]^d$; the value of N is given in advance, and each edge of an input object has length at least 1 and at most N. Moreover, they have designed dynamic approximation algorithms and lower bounds, where the update time depends on N and is of high complexity. Gavruskin et al. [24] studied the Max-IS problem for dynamic proper intervals (intervals cannot contain one another), and showed how to maintain a Max-IS with polylogarithmic update time.

Results and Organization. We study MIS and Max-IS problems for dynamic sets of $O(n)$ axis-parallel rectangles of two types: (i) congruent rectangles of uniform height and width and (ii) rectangles of uniform height and arbitrary width. For both classes of rectangles a MIS can be maintained in $\Omega(\Delta)$ update time by using the recent algorithm of Assadi et al. [4], where Δ is the maximum degree of the intersection graph. A $(1+\epsilon)$-approximate Max-IS can be maintained for unit squares in $O(n^{1/\epsilon^2})$ time [19], and a $(1+\frac{1}{k})$-approximate Max-IS can be maintained for unit height and arbitrary width rectangles in $O(n^{2k-1})$ update time [3] for any integer $k \geq 1$. In this paper we design and implement algorithms for dynamic MIS and Max-IS that demonstrate the trade-off between update time and approximation factor, both from a theoretical perspective and in an experimental evaluation. In contrast to the recent dynamic MIS algorithms, which are randomized [4, 5, 8, 15], our algorithms are deterministic.

In Section 3 we present an algorithm that maintains a MIS of a dynamic set of unit squares in amortized $O(\log^{2/3+o(1)} n)$ update time, improving the best known update time $\Omega(\Delta)$ [4]. A major, but generally unavoidable bottleneck of that algorithm is that the entire graph is stored explicitly, and thus insertions/deletions of vertices take $\Omega(\Delta)$ time. We use structural geometric properties of the unit squares along with a dynamic orthogonal range searching data structure to bypass the explicit intersection graph and break this bottleneck.

In Section 4, we study the Max-IS problem. For dynamic unit squares, we give an algorithm that maintains a 4-approximate Max-IS with $O(1)$ update time. We generalize this algorithm and improve the approximation factor to $2(1+\frac{1}{k})$, which increases the update time to $O(k^2 \log n)$. We conclude with an algorithm that maintains a 2-approximate Max-IS for a dynamic set of unit-height and arbitrary-width rectangles (in fact, for a dynamic interval graph, which is of independent interest) with $O(\omega \log n)$ output-sensitive update time, where ω is the maximum size of an independent set of rectangles stabbed by any horizontal line.

Finally, Section 5 provides an experimental evaluation of the proposed Max-IS approximation algorithms on synthetic and real-world map labeling data sets using unit squares. The experiments explore the trade-off between solution size and update time, as well as the speed-up of the dynamic algorithms over their static counterparts. See the supplemental material for source code and benchmark data.

Proofs marked ($\star$) *are missing due to space constraints; refer to the full version for all details.*

2 Model and Notation

Let $\mathcal{R} = \{r_1, \ldots, r_\nu\}$ be a set of ν axis-parallel, unit-height rectangles in the plane. If the rectangles are of uniform height and width, we can use an affine transformation to map $\mathcal{R}$ to a set of unit squares $\mathcal{S} = \{s_1, \ldots, s_\nu\}$ instead. We use the shorthand notation $[n] = \{1, 2, \ldots, n\}$. In our setting we assume that $\mathcal{R}$ is dynamically updated by a sequence of $N \in \mathbb{N}$ insertions and deletions. We denote the set of rectangles at step $i \in [N]$ as $\mathcal{R}_i$. For a set of unit squares $\mathcal{S}_i = \{s_1, \ldots, s_\nu\}$ at step $i \in [N]$ we further define the set $\mathcal{C}_i = \{c_1, \ldots, c_\nu\}$ of the corresponding square centers. Let $n = \max\{|\mathcal{R}_i| \mid i \in [N]\}$ be the maximum number of rectangles over all steps. The rectangle intersection graph defined by $\mathcal{R}_i$ at time step i is denoted as $G_i = (\mathcal{R}_i, E_i)$, where two rectangles $r, r' \in \mathcal{R}_i$ are connected by an edge $\{r, r'\} \in E_i$ if and only if $r \cap r' \neq \emptyset$. We use M_i to denote a maximal independent set in G_i, and OPT_i to denote a maximum independent set in G_i. For a graph $G = (V, E)$ and a vertex $v \in V$, let $N(v)$ denote the set of neighbors of v in G. This notation also extends to any subset $U \subseteq V$ by defining $N(U) = \bigcup_{v \in U} N(v)$. We use $\deg(v)$ to denote the degree of a vertex $v \in V$. For any vertex $v \in V$, let $N^r(v)$ be the r-neighborhood of v, i.e., the set of vertices that are within distance at most r from v (excluding v).

3 Dynamic MIS with Sub-Logarithmic Update Time

In this section, we study the MIS problem for dynamic uniform rectangles. As stated before we can assume w.l.o.g. that the rectangles are unit squares. We design an algorithm that maintains a MIS for a dynamic set of $O(n)$ unit squares in sub-logarithmic update time. Assadi et al. [4] presented an algorithm for maintaining a MIS on general dynamic graphs with $\Omega(\Delta)$ update time, where Δ is the maximum degree in the graph. In the worst case, however, that algorithm takes $O(n)$ update time. In fact, it seems unavoidable for an algorithm that explicitly maintains the (intersection) graph to perform a MIS update in less than $\Omega(\deg(v))$ time for an insertion/deletion of a vertex v. In contrast, our proposed algorithm in this section does not explicitly maintain the intersection graph $G_i = (\mathcal{S}_i, E_i)$ (for any $i \in [N]$), but rather only the set of squares $\mathcal{S}_i$ in a suitable dynamic geometric data structure. For the ease of explanation, however, we do use graph terms at times.

Let $i \in [N]$ be any time point in the sequence of updates. For each square $s_v \in \mathcal{S}_i$, let s_v^a be a square of side length a concentric with s_v. Further, let M_i denote the MIS that we compute for $G_i = (\mathcal{S}_i, E_i)$, and let $\mathcal{C}(M_i) \subseteq \mathcal{C}_i$ be their corresponding square centers. We

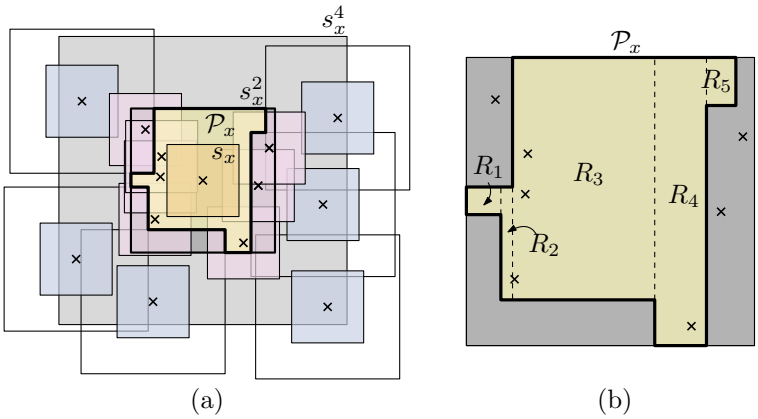


Figure 1 Example for the deletion of a square s_x. (a) Square s_x, its neighborhood with centers in s_x^2, its 2-neighborhood with centers in s_x^4, and the polygon $\mathcal{P}_x$. (b) Vertical slab partition of $\mathcal{P}_x$.

maintain two fully dynamic orthogonal range searching data structures throughout: (i) a dynamic range tree $T(\mathcal{C}_i)$ for the entire point set $\mathcal{C}_i$ and (ii) a dynamic range tree $T(\mathcal{C}(M_i))$ for the point set $\mathcal{C}(M_i)$ corresponding to the centers of M_i. They can be implemented with dynamic fractional cascading [34], which yields $O(\log n \log\log n)$ update time and $O(k + \log n \log\log n)$ query time for reporting k points. The currently best fully dynamic data structure for orthogonal range reporting requires $O(\log^{2/3+o(1)} n)$ amortized update time and $O(k + \frac{\log n}{\log\log n})$ amortized query time [14].

We compute the initial MIS M_1 for $G_1 = (\mathcal{S}_1, E_1)$ by using a simple linear-time greedy algorithm. First we initialize the range tree $T(\mathcal{C}_i)$. Then we iterate through the set $\mathcal{S}_1$ as long as it is not empty, select a square s_v for M_1 and insert its center into $T(\mathcal{C}(M_i))$, find its neighbors $N(s_v)$ by a range query in $T(\mathcal{C}_i)$ with the concentric square s_v^2, and delete $N(s_v)$ from $\mathcal{S}_1$. It is clear that once this process terminates, M_1 is a MIS.

When we move in the next step from $G_i = (\mathcal{S}_i, E_i)$ to $G_{i+1} = (\mathcal{S}_{i+1}, E_{i+1})$, either a square is inserted into $\mathcal{S}_i$ or deleted from $\mathcal{S}_i$. Let s_x be the square that is inserted or deleted.
Insertion: When we insert a square s_x into $\mathcal{S}_i$ to obtain $\mathcal{S}_{i+1}$, we do the following operations. First, we obtain $T(\mathcal{C}_{i+1})$ by inserting the center of s_x into $T(\mathcal{C}_i)$. Next, we have to detect whether s_x can be included in M_{i+1}. If there exists a square s_u from M_i intersecting s_x, we should not include s_x; otherwise we will add it to the MIS. To check this, we search with the range s_x^2 in $T(\mathcal{C}(M_i))$. By a simple packing argument, we know that no more than four points (the centers of four independent squares) of $\mathcal{C}(M_i)$ can be in the range s_x^2. If the query returns such a point, then s_x would intersect with another square in M_i and we set $M_{i+1} = M_i$. Otherwise, we insert s_x into $T(\mathcal{C}(M_i))$ to get $T(\mathcal{C}(M_{i+1}))$.
Deletion: When we delete a square s_x from $\mathcal{S}_i$, it is possible that $s_x \in M_i$. In this case we may have to add squares from $N(s_x)$ into M_{i+1} to keep it maximal. Since any square can have at most four independent neighbors, we can add in this step up to four squares to M_{i+1}.

First, we check if $s_x \in M_i$. If not, then we simply delete s_x from $T(\mathcal{C}_i)$ to get $T(\mathcal{C}_{i+1})$ and set $M_{i+1} = M_i$. Otherwise, we delete again s_x from $T(\mathcal{C}_i)$ and also from $T(\mathcal{C}(M_i))$. In order to detect which neighbors of s_x can be added to M_i, we use suitable queries in the data structures $T(\mathcal{C}(M_i))$ and $T(\mathcal{C}_i)$. Figure 1a illustrates the next observations. The centers of all neighbors in $N(s_x)$ must be contained in the square s_x^2. But some of these neighbors may intersect other squares in M_i. In fact, these squares would by definition belong to the 2-neighborhood, i.e., be in the set $Q_x = N^2(s_x) \cap M_i$. We can obtain Q_x by querying

$T(\mathcal{C}(M_i))$ with the range s_x^4. Since $s_x \in M_i$, we know that $Q_x \cap s_x^2 = \emptyset$ and hence the center points of the squares in Q_x lie in the annulus $s_x^4 - s_x^2$. A simple packing argument implies that $|Q_x| \leq 12$ and therefore querying $T(\mathcal{C}(M_i))$ will return at most 12 points.

Next we define the rectilinear polygon $\mathcal{P}_x = s_x^2 - \bigcup_{s_y \in Q_x} s_y^2$, which contains all possible center points of squares that are neighbors of s_x but do not intersect any square $s_y \in M_i \setminus \{s_x\}$.

▶ **Observation 1** (⋆). *The polygon $\mathcal{P}_x$ has at most 28 corners.*

Next we want to query $T(\mathcal{C}_i)$ with the range $\mathcal{P}_x$, which we do by vertically partitioning $\mathcal{P}_x$ into rectangular slabs $R_1, \ldots, R_c$ for some $c \leq 28$ (see Figure 1b). For each slab R_j, where $1 \leq j \leq c$, we perform a range query in $T(\mathcal{C}_i)$. If a center p is returned, we can add the corresponding square s_p into M_{i+1}, and p into $T(\mathcal{C}(M_i))$ to obtain $T(\mathcal{C}(M_{i+1}))$. Moreover, we have to update $\mathcal{P}_x \leftarrow \mathcal{P}_x - s_p$, refine the slab partition and continue querying $T(\mathcal{C}_i)$ with the slabs of $\mathcal{P}_x$. We know that the deleted square s_x can have at most four independent neighbors. So after adding at most four new squares to M_{i+1} we know that $\mathcal{P}_x = \emptyset$ and we can stop searching.

▶ **Lemma 2** (⋆). *The set M_i is a maximal independent set of $G_i = (\mathcal{S}_i, E_i)$ for each step $i \in [N]$.*

Both the Insertion and the Deletion operations consist of (i) a constant number of insertions or deletions in the two fully dynamic orthogonal range searching data structures and (ii) of a constant number of orthogonal range reporting queries. These queries return at most 12 points in $T(\mathcal{C}(M_{i-1}))$. While the query range $\mathcal{P}_x$ for $T(\mathcal{C}_{i-1})$ in the Deletion operation may contain many points, an arbitrary point in $\mathcal{P}_x$ is sufficient for adding a new square to the independent set. Thus we do not need to enumerate all contained points, but just a single witness. In the orthogonal range searching data structure of Chan and Tsakalidis [14], the amortized update time for an insertion/deletion is $O(\log^{2/3+o(1)} n)$; this dominates the query time and together with Lemma 2 yields:

▶ **Theorem 3.** *We can maintain a maximal independent set of a dynamic set of unit squares, deterministically, in amortized $O(\log^{2/3+o(1)} n)$ update time.*

Proof. The correctness follows from Lemma 2. It remains to show the running time for the fully dynamic updates. At each step i we perform either an Insertion or a Deletion operation. Let us first discuss the update time for the insertion of a square. As described above, an insertion performs one or two insertions of the center of the square into the range trees and one range query in $T(\mathcal{C}(M_{i-1}))$, which will return at most four points. Using dynamic fractional cascading [34], this requires $O(\log n \log\log n)$ time; with the data structure of Chan and Tsakalidis [14], the amortized update time for inserting a square is $O(\log^{2/3+o(1)} n)$, the time for inserting a new point into their range searching data structure; this dominates the query time. The deletion of a square triggers either just a single deletion from the range tree $T(\mathcal{C}_{i-1})$ or, if it was contained in the MIS M_{i-1}, two deletions, up to four insertions, and a sequence of range queries: one query in $T(\mathcal{C}(M_{i-1}))$, which can return at most 12 points and a constant number of queries in $T(\mathcal{C}_{i-1})$ with the constant-complexity slab partition of $\mathcal{P}_x$. Note that while the number of points in $\mathcal{P}_x$ can be large, for our purpose it is sufficient to return a single point in each query range if it is not empty. Therefore, the update time for a deletion is again $O(\log n \log\log n)$ with dynamic fractional cascading [34] or amortized $O(\log^{2/3+o(1)} n)$ [14], depending on the selected data structure. ◀

For unit square intersection graphs, recall that any square in a MIS can have at most four mutually independent neighbors. Therefore, maintaining a dynamic MIS immediately implies maintaining a dynamic 4-approximate Max-IS.

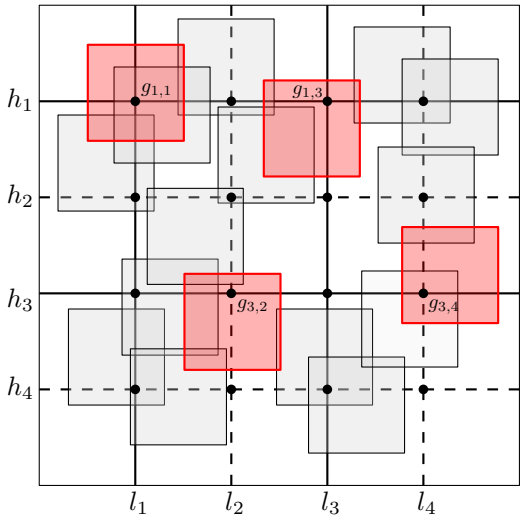


Figure 2 Example instance with bounding square $\mathcal{B}$ partitioned into a 5×5 grid. Red squares represent the computed 4-approximate solution, which here is $M(O(H))$.

▶ **Corollary 4.** *We can maintain a 4-approximate maximum independent set of a dynamic set of unit squares, deterministically, in amortized* $O(\log^{2/3+o(1)} n)$ *update time.*

4 Approximation Algorithms for Dynamic Maximum Independent Set

In this section, we study the Max-IS problem for dynamic unit squares as well as for unit-height and arbitrary-width rectangles. In a series of dynamic schemes proposed in this section, we establish the trade-off between the update time and the solution size, i.e., the approximation factors. First, we design a 4-approximation algorithm with $O(1)$ update time for Max-IS on dynamic unit squares (Section 4.1). We improve this to an algorithm that maintains a $2(1+\frac{1}{k})$-approximate Max-IS with $O(k^2 \log n)$ update time (Section 4.2). Finally, we conclude with an algorithm that deterministically maintains a 2-approximate Max-IS with output-sensitive $O(\omega \log n)$ update time, where ω is the maximum size of an independent set of the unit-height rectangles stabbed by any horizontal line (Section 4.3).

Let $\mathcal{B}$ be a bounding square of the dynamic set of 1×1-unit squares $\bigcup_{i\in[N]} \mathcal{S}_i$ of side length $\kappa \times \kappa$; we can assume that $\kappa = O(n)$; otherwise we could contract empty horizontal/vertical strips of $\mathcal{B}$. Let $H = \{h_1, \ldots, h_\kappa\}$ and $L = \{l_1, \ldots, l_\kappa\}$ be a set of top-to-bottom and left-to-right ordered equidistant horizontal and vertical lines partitioning $\mathcal{B}$ into a square grid of side-length-1 cells, see Figure 2. Let $E_H = \{h_i \in H \mid i = 0 \pmod 2\}$ and $O_H = \{h_i \in H \mid i = 1 \pmod 2\}$ be the set of even and odd horizontal lines, respectively.

4.1 4-Approximation Algorithm with Constant Update Time

We design a 4-approximation algorithm for the Max-IS problem on dynamic unit square intersection graphs with constant update time. Our algorithm is based on a grid partitioning approach. Consider the square grid on $\mathcal{B}$ induced by the sets H and L of horizontal and vertical lines. We denote the grid points as $g_{p,q}$ for $p, q \in [\kappa]$, where $g_{p,q}$ is the intersection point of lines h_p and l_q. Under a general position assumption (otherwise we slightly perturb the grid position to handle degenerate cases), each unit square in any set $\mathcal{S}_i$, for $i \in [N]$ contains exactly one grid point. For each $g_{p,q}$, we store a Boolean *activity value* 1 or 0 based

on its intersection with $\mathcal{S}_i$ (for any step $i \in [N]$). If $g_{p,q}$ intersects at least one square of S_i, we say that it is *active* and set the value to 1; otherwise, we set the value to 0. Observe that for each grid point $g_{p,q}$ and each time step i at most one square of $\mathcal{S}_i$ intersecting $g_{p,q}$ can be chosen in any MAX-IS. This holds because all squares that intersect the same grid point form a clique in G_i, and at most one square from a clique can be chosen in any independent set.

We first initialize an independent set M_1 for $G_1 = (\mathcal{S}_1, E_1)$ with $|M_1| \geq |OPT_1|/4$. For each horizontal line $h_j \in H$, we compute two independent sets $M^1_{h_j}$ and $M^2_{h_j}$, where $M^1_{h_j}$ (resp. $M^2_{h_j}$) contains an arbitrary square intersecting each odd (resp. even) grid point on h_j. Since every other grid point is omitted in these sets, any two selected squares are independent. Let $M(h_j) = \arg\max\{|M^1_{h_j}|, |M^2_{h_j}|\}$ be the larger of the two independent sets. We define $p(h_j) = |M^1_{h_j}|$ and $q(h_j) = |M^2_{h_j}|$, as well as $c(h_j) = |M(h_j)| = \max\{p(h_j), q(h_j)\}$.

We construct the independent sets $M(E_H) = \bigcup_{j=1}^{\lfloor \kappa/2 \rfloor}(M(h_{2j}))$ for E_H and $M(O_H) = \bigcup_{j=1}^{\lfloor \kappa/2 \rfloor}(M(h_{2j-1})$ for O_H. We return $M_1 = \arg\max\{|M(E_H)|, |M(O_H)|\}$ as the independent set for G_1. See Figure 2 for an illustration. The initialization of all $O(\kappa^2)$ variables and the computation of the first set M_1 take $O(\kappa^2)$ time. (Alternatively, a hash table would be more space efficient, but could not provide the $O(1)$-update time guarantee.)

▶ **Lemma 5 (⋆).** *The set M_1 is an independent set of $G_1 = (\mathcal{S}_1, E_1)$ with $|M_1| \geq |OPT_1|/4$ and can be computed in $O(\kappa^2)$ time.*

In the following step, when we move from G_i to G_{i+1}, for any $i \in [N]$, a square s_x is inserted into $\mathcal{S}_i$ or deleted from $\mathcal{S}_i$. Intuitively, we check the activity value of the grid point that s_x intersects. If the update has no effect on its activity value, we keep $M_{i+1} = M_i$. Otherwise, we update the activity value, the corresponding cardinality counters, and report the solution accordingly. All of these operations can be performed in $O(1)$-time. For a more detailed description see the full version of this paper.

▶ **Lemma 6 (⋆).** *The set M_i is an independent set of $G_i = (\mathcal{S}_i, E_i)$ for each $i \in [N]$ and $|M_i| \geq |OPT_i|/4$.*

Lemmas 5 and 6 together with the $O(1)$ update time yield:

▶ **Theorem 7 (⋆).** *We can maintain a 4-approximate maximum independent set in a dynamic unit square intersection graph, deterministically, in $O(1)$ update time.*

4.2 $2(1 + \frac{1}{k})$-Approximation Algorithm with $O(k)$ Update Time

Next, we improve the approximation factor from 4 to $2(1 + \frac{1}{k})$, for any $k \geq 1$, by combining the shifting technique [29] with the insights gained from Section 4.1. This comes at the cost of an increase of the update time to $O(k^2 \log n)$, which illustrates the trade-off between solution quality and update time. We reuse the grid partition and some notations from Section 4.1. We first describe how to obtain a solution M_1 for the initial graph G_1 that is of size at least $|OPT_1|/2(1 + \frac{1}{k})$ and then discuss how to maintain this under dynamic updates.

Let $h_j \in H$ be a horizontal stabbing line and let $\mathcal{S}(h_j) \subseteq \mathcal{S}$ be the set of squares stabbed by h_j. Since they are all stabbed by h_j, the intersection graph of $\mathcal{S}(h_j)$ is equivalent to the unit interval intersection graph obtained by projecting each unit square $s_x \in \mathcal{S}(h_j)$ to a unit interval i_x on the line h_j; we denote this set of unit intervals as $I(h_j)$. First, we sort the intervals in $I(h_j)$ from left to right. Next we define $k+1$ *groups* with respect to h_j that are formed by deleting those squares and their corresponding intervals from $S(h_j)$ and $I(h_j)$, respectively, that intersect every $k+1$-th grid point on h_j, starting from some $g_{j,\alpha}$ with $\alpha \in [k+1]$. Now consider the k consecutive grid points on h_j between two deleted grid points

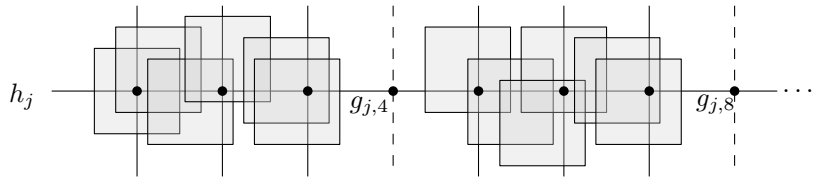


Figure 3 Illustration of a group on line h_j for $k = 3$ with the two subgroups $I_1^3(h_j)$ and $I_5^3(h_j)$.

in one such group, say, $\{g_{j,\ell}, \ldots, g_{j,\ell+k-1}\}$ for some $\ell \in [\kappa]$. Let $I_\ell^k(h_j) \subseteq I(h_j)$ be the set of unit intervals intersecting the k grid points $g_{j,\ell}$ to $g_{j,\ell+k-1}$. We refer to them as *subgroups*. See Figure 3 for an illustration. Observe that the maximum size of an independent set of each subgroup is at most k, since the width of each subgroup is strictly less than $k+1$ and each interval has unit length.

We compute M_1 for G_1 as follows. For each stabbing line $h_j \in H$, we form the $k+1$ different groups of $I(h_j)$. For each group, a MAX-IS is computed optimally and separately inside each subgroup. Since any two subgroups are horizontally separated and thus independent, we can then take the union of the independent sets of the subgroups to get an independent set for the entire group. This is done with the linear-time greedy algorithm to compute maximum independent sets for interval graphs [26]. Let $\{M_{h_j}^1, \ldots, M_{h_j}^{k+1}\}$ be $k+1$ maximum independent sets for the $k+1$ different groups and let $M(h_j) = \arg\max\{|M_{h_j}^1|, |M_{h_j}^2|, \ldots, |M_{h_j}^{k+1}|\}$ be one with maximum size. We store its cardinality as $c(h_j) = \max\{|M_{h_j}^i| \mid i \in [k+1]\}$. Next, we compute an independent set for E_H, denoted by $M(E_H)$, by composing it from the best solutions $M(h_j)$ from the even stabbing lines, i.e., $M(E_H) = \bigcup_{j=1}^{\lfloor \kappa/2 \rfloor} M(h_{2j})$ and its cardinality $|M(E_H)| = \sum_{j=1}^{\lfloor \kappa/2 \rfloor} c(h_{2j})$. Similarly, we compute an independent set for O_H as $M(O_H) = \bigcup_{j=1}^{\lfloor \kappa/2 \rfloor} M(h_{2j-1})$ and its cardinality $|M(O_H)| = \sum_{j=1}^{\lfloor \kappa/2 \rfloor} c(h_{2j-1})$. Finally, we return $M_1 = \arg\max\{|M(E_H)|, |M(O_H)|\}$ as the solution for G_1.

▶ **Lemma 8** (⋆). *The independent set M_1 of $G_1 = (\mathcal{S}_1, E_1)$ can be computed in $O(n \log n + kn)$ time and $|M_1| \geq |OPT_1|/2(1 + \frac{1}{k})$.*

Next, we describe a pre-processing step, which is required for the dynamic updates.
PRE-PROCESSING: For each horizontal line $h_j \in H$, consider a group. For each subgroup $I_\ell^k(h_j)$ (for some $\ell \in [k+1]$), we construct a balanced binary tree $T(I_\ell^k(h_j))$ storing the intervals of $I_\ell^k(h_j)$ in left-to-right order (indexed by their left endpoints) in the leaves. This process is done for each group of every horizontal line $h_j \in H$. We mark those leaves in $T(I_\ell^k(h_j))$ as *selected* that correspond to an independent interval in the solution and maintain a list of pointers to those independent intervals. This tree also lets us quickly identify the location of an interval that is inserted or deleted. In fact, while we run the greedy algorithm on $I_\ell^k(h_j)$, we can already mark precisely the selected intervals for the independent set.

When we perform the update step from $G_i = (\mathcal{S}_i, E_i)$ to $G_{i+1} = (\mathcal{S}_{i+1}, E_{i+1})$, either a square is inserted into $\mathcal{S}_i$ or deleted from $\mathcal{S}_i$. Let s_x and i_x be this square and its corresponding interval. Let $g_{u,v}$ (for some $u, v \in [\kappa]$) be the grid point that intersects s_x. We describe here the INSERTION and refer to the full version of this paper for the DELETION.
INSERTION: The insertion of i_x affects all but one of the groups on line h_u. We describe the procedure for one such group on h_u; it is then repeated for the other groups. In each group, i_x appears in exactly one subgroup and the other subgroups remain unaffected. This subgroup, say $I_\ell^k(h_u)$, is determined by the index v of the grid point $g_{u,v}$ intersecting i_x. First, we locate i_x in the sorted list of intervals of $I_\ell^k(h_u)$, which can be done in $O(\log n)$ time by searching in the associated tree $T(I_\ell^k(h_u))$. If i_x is immediately left of a selected interval

i_y, but does not intersect the previous selected interval, then i_x becomes a new selected interval that replaces i_y and triggers a sequence of updates of the later selected intervals. Let us first consider the case that i_x is not selected as a new independent interval. Then we simply insert i_x into $T(I_\ell^k(h_u))$ in $O(\log n)$ time. Otherwise, we mark i_x as selected, remove the selection mark from its successor interval i_y, and replace i_y by i_x in the maintained subsolution. Since the right endpoint of i_x is left of the right endpoint of i_y, this change possibly triggers a sequence of updates to the subsequent selected intervals.

We thus identify in $T(I_\ell^k(h_u))$ the leftmost interval i_z that starts to the right of the right endpoint of i_x. This takes $O(\log n)$ time. If this interval i_z is not yet marked as selected, we replace the previous successor of i_x in the current list of selected intervals by i_z and repeat the update process for i_z. Otherwise, if i_z is already selected, we can stop the update of the subsolution as there would be no further changes.

Since a maximum independent set in each subgroup contains at most k intervals, the update time is $O(k \log n)$ per group and $O(k^2 \log n)$ for all k affected groups.

While doing the updates, we collect the new selected intervals as the MAX-IS for the subgroup $I_\ell^k(h_u)$. For all groups affected by the insertion of i_x we update the corresponding independent sets $M_{h_u}^p$ for $p \in [k+1]$, whenever some updates of selected intervals were necessary. Then we select the largest independent set of all $k+1$ groups as $M(h_j)$ and update its new cardinality in $c(h_j)$. Finally, we update the independent sets $M(E_H)$ and $M(O_H)$ and their cardinalities and return $M_{i+1} = \arg\max\{|M(E_H)|, |M(O_H)|\}$ as the solution for G_{i+1}.

▶ **Lemma 9** (⋆). *The set M_i is an independent set of $G_i = (\mathcal{S}_i, E_i)$ for each $i \in [N]$ and $|M_i| \geq |OPT_i|/2(1 + \frac{1}{k})$.*

With Lemma 9 and the update time discussion in the full version of the paper we obtain:

▶ **Theorem 10** (⋆). *We can maintain a $2(1 + \frac{1}{k})$-approximate maximum independent set in a dynamic unit square intersection graph, deterministically, in $O(k^2 \log n)$ update time.*

4.3 2-Approximation Algorithm with $O(\omega \log n)$ Update Time

We finally design a 2-approximation algorithm for the MAX-IS problem on dynamic axis-aligned unit height, but arbitrary width rectangles. Let $\mathcal{B}$ be the bounding box of the dynamic set of rectangles $\widetilde{\mathcal{R}} = \bigcup_{i \in [N]} \mathcal{R}_i$. We begin by dividing $\mathcal{B}$ into horizontal strips of height 1 defined by the set $H = \{h_1, \ldots, h_\kappa\}$ of $\kappa = O(n)$ horizontal lines. We assume, w.l.o.g., that every rectangle in $\widetilde{\mathcal{R}}$ is stabbed by exactly one line in H. For a set of rectangles $\mathcal{R}$, we denote the subset stabbed by a line h_j as $\mathcal{R}(h_j) \subseteq \mathcal{R}$.

We first describe how to obtain an independent set M_1 for the initial graph $G_1 = (\mathcal{R}_1, E_1)$ such that $|M_1| \geq |OPT_1|/2$ by using the following algorithm of Agarwal et al. [3]. For each horizontal line $h_j \in H$, we compute a maximum independent set for $\mathcal{R}_1(h_j)$. The set $\mathcal{R}_i(h_j)$ (for any $i \in [N]$ and $j \in [\kappa]$) can again be seen as an interval graph. For a set of n intervals, a MAX-IS can be computed by a left-to-right greedy algorithm visiting the intervals in the order of their right endpoints in $O(n \log n)$ time. So for each horizontal line $h_j \in H$, let $M(h_j)$ be a MAX-IS of $\mathcal{R}_1(h_j)$, and let $c(h_j) = |M(h_j)|$. Then we construct the independent set $M(E_H) = \bigcup_{j=1}^{\lfloor \kappa/2 \rfloor}(M(h_{2j}))$ for E_H. Similarly, we construct the independent set $M(O_H) = \bigcup_{j=1}^{\lfloor \kappa/2 \rfloor}(M(h_{2j-1})$ for O_H. We return $M_1 = \arg\max\{|M(E_H)|, |M(O_H)|\}$ as the independent set for $G_1 = (\mathcal{R}_1, E_1)$. See Figure 4 for an illustration.

▶ **Lemma 11** (Theorem 2, [3]). *The set M_1 is an independent set of $G_1 = (\mathcal{R}_1, E_1)$ with $|M_1| \geq |OPT_1|/2$ and can be computed in $O(n \log n)$ time.*

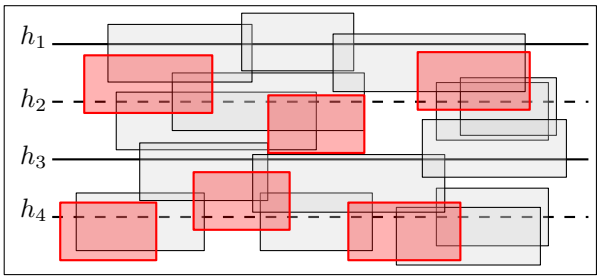


Figure 4 Example instance with four horizontal lines. Red rectangles represent the computed 2-approximate solution, which here is $M(E(H))$.

We describe the following pre-processing step to initialize in $O(n \log n)$ time the data structures that are required for the subsequent dynamic updates.
Pre-Processing: Consider a stabbing line h_j and the stabbed set of rectangles $\mathcal{R}_i(h_j)$ for some $i \in [N]$. We denote the corresponding set of intervals as $I(h_j)$. We build a balanced binary search tree $T_l(I(h_j))$, storing the intervals in $I(h_j)$ in left-to-right order based on their left endpoints. This is called the *left tree* of $I(h_j)$. We augment the left tree such that each tree node additionally stores a pointer to the interval with leftmost right endpoint in its subtree. This pointer structure can easily be computed by a bottom-up pass through $T_l(I(h_j))$. Note that a leaf update in $T_l(I(h_j))$ takes $O(\log n)$ time as for standard binary search trees, but we can in the same $O(\log n)$ time propagate the change that potentially affects the leftmost right endpoints of the tree nodes along the path to the root. Additionally, we store the set of selected independent intervals for the Max-IS of $I(h_j)$ in left-to-right order in another balanced binary search tree $T_s(I(h_j))$ (the so-called *solution tree*). Let ω_j be the cardinality of the maximum independent of $I(h_j)$ for $j \in [\kappa]$, and let $\omega = \max_j \omega_j$ be the maximum of these cardinalities over all stabbing lines h_j.

When we move from G_i to G_{i+1} (for some $1 \leq i < N$), either we insert a new rectangle into $\mathcal{R}_i$ or delete one rectangle from $\mathcal{R}_i$. Let r_x be the rectangle that is inserted or deleted, let i_x be its corresponding interval, and let h_j (for some $j \in [\kappa]$) be the horizontal line that intersects r_x. In what follows, we describe how to maintain a 2-approximate Max-IS with $O(\omega_j \log n) = O(\omega \log n)$ update time. We distinguish Insertion and Deletion.
Insertion: We first determine whether i_x should be a new selected interval or not. Because the greedy algorithm for constructing the Max-IS visits the intervals in left-to-right order based on the right endpoints, we need to reconstruct the state of the algorithm when it would visit i_x. We query the solution tree $T_s(I(h_j))$ with both endpoints of i_x in $O(\log \omega_j)$ time. If and only if both search paths end up between the same two leaves belonging to two consecutive selected intervals i_y and i_z (considering their right endpoints), then i_x would have been chosen as the next selected interval after i_y and before i_z in the greedy algorithm. This implies that i_y and i_x are independent, but i_x and i_z may or may not intersect.

If the two search paths in the solution tree are different, then i_x does not become a new selected interval, and we simply insert it into $T_l(I(h_j))$ in $O(\log n)$ time. Else we also insert i_x into $T_l(I(h_j))$, but we also have to perform a sequence of selection update operations, which are more involved for intervals of arbitrary length compared to the updates in Section 4.2. Figure 5 shows an example. First we mark i_x as selected and insert it into $T_s(I(h_j))$. Now we need to identify the next selected interval right of i_x that would have been found by the greedy algorithm. We use the left tree $T_l(I(h_j))$ to search in $O(\log n)$ time for the interval i'_z with leftmost right endpoint, whose left endpoint is right of the right endpoint p of i_x.

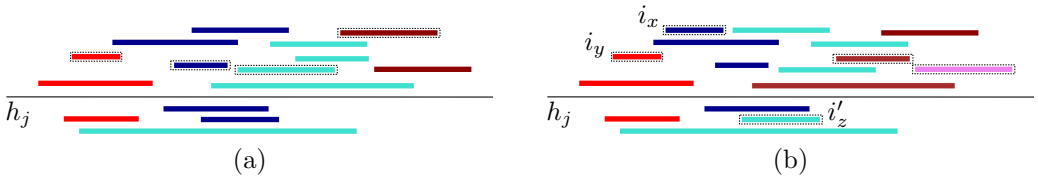


Figure 5 Illustration of the updates triggered by the insertion of an interval; selected independent intervals are marked by a dotted bounding box and intervals intersected by a selected interval have the same color as the rightmost such interval. (a) Before insertion of i_x, (b) after insertion of i_x.

More precisely, we search for p in $T_l(I(h_j))$ and whenever the search path branches into the left subtree, we compare whether the leftmost right endpoint stored in the root of the right subtree is left of the right endpoint of the current candidate interval. Once a leaf is reached, the leftmost found candidate interval is the desired interval i'_z. This interval i'_z is precisely the first interval after i_x in the order considered by the greedy algorithm that is independent of i_x and thus must be the next selected interval. If $i_z \neq i'_z$, we repeat the update process for i'_z as if it would have been the newly inserted interval until we reach the end of $I(h_j)$; otherwise we keep i_z as the successor of i_x and stop the update process.

For each update of a selected interval, we perform one search in $T_l(I(h_j))$ in $O(\log n)$ time. There are at most ω_j updates, so the update time is $O(\omega_j \log n)$. Finally, we need to delete $O(\omega_j)$ old selected intervals from and insert $O(\omega_j)$ new selected intervals into the solution tree $T_s(I(h_j))$, which takes $O(\log \omega_j)$ time for each insertion and deletion. We now re-evaluate the new Max-IS $M(h_j)$ and its cardinality $c(h_j)$, which possibly affects $M(E_H)$ or $M(O_H)$. We obtain the new independent set $M_{i+1} = \arg\max\{|M(E_H)|, |M(O_H)|\}$ for $G_{i+1} = (\mathcal{R}_{i+1}, E_{i+1})$.

Deletion: If the interval i_x to be deleted is not a selected interval, it is sufficient to delete it from the left tree $T_l(I(h_j))$ in $O(\log n)$ time. Otherwise, if i_x is a selected interval, let i_y be the selected interval preceding i_x in the solution tree $T_s(I(h_j))$. We first delete i_x from $T_l(I(h_j))$ and $T_s(I(h_j))$. Then we need to select a new interval to replace i_x according to the greedy Max-IS algorithm, which is the interval i_z whose right endpoint is leftmost among all intervals that are completely to the right of i_y. We find this interval i_z again by a search in the left tree $T_l(I(h_j))$ with the right endpoint of i_y as the query point. We make i_z a new selected interval and use the right endpoint of i_z as the query point for finding the next selected interval in $T_l(I(h_j))$. We repeat this process until we have reached the last interval of $I(h_j)$.

As for the Insertion step, each update of a selected interval requires $O(\log n)$ time due to the query for the next selected interval in $T_l(I(h_j))$. There are $O(\omega_j)$ such updates. Further, we need to update the solution tree $T_s(I(h_j))$ by performing $O(\omega_j)$ insertions and deletions of seeds, each in $O(\log \omega_j)$ time. Once all updates to the selected intervals and the data structures for $I(h_j)$ are done, we re-evaluate the new Max-IS $M(h_j)$ and its cardinality $c(h_j)$, which possibly affects $M(E_H)$ or $M(O_H)$. This yields the new independent set $M_{i+1} = \arg\max\{|M(E_H)|, |M(O_H)|\}$ for $G_{i+1} = (\mathcal{R}_{i+1}, E_{i+1})$.

▶ **Lemma 12.** *The set M_i is an independent set of $G_i = (\mathcal{R}_i, E_i)$ for each $i \in [N]$ and $|M_i| \geq |OPT_i|/2$.*

Proof. We prove the lemma by induction. From Lemma 11 we know that M_1 satisfies the claim, and in particular each set $M(h)$ for $h \in H$ is a Max-IS of the interval set $I(h)$. So let us consider the set M_i for $i \geq 2$ and assume that M_{i-1} satisfies the claim by the induction hypothesis. Let r_x and i_x be the updated rectangle and its interval, and assume that it

belongs to the stabbing line h_j. Then we know that for each $h_k \in H$ with $k \neq j$ the set $M(h_k)$ is not affected by the update to r_x and thus is a MAX-IS by the induction hypothesis. It remains to show that the update operations described above restore a MAX-IS $M(h_j)$ for the set $I(h_j)$. But in fact the updates are designed in such a way that the resulting set of selected intervals is identical to the set of intervals that would be found by the greedy MAX-IS algorithm for $I(h_j)$. Therefore $M(h_j)$ is a MAX-IS for $I(h_j)$ and by the pigeonhole principle $|M_i| \geq |OPT_i|/2$. ◀

Each update of a rectangle r_x (and its interval i_x) triggers either an INSERTION or a DELETION operation on the unique stabbing line of r_x. As we have argued in the description of these two update operations, the insertion or deletion of i_x requires one $O(\log n)$-time update in the left tree data structure. If i_x is a selected independent interval, the update further triggers a sequence of at most ω_j selection updates, each of which requires $O(\log n)$ time. Hence the update time is bounded by $O(\omega_j \log n) = O(\omega \log n)$. Recall that ω_j and ω are output-sensitive parameters describing the maximum size of an independent set of $I(h)$ for a specific stabbing line $h = h_j$ or any stabbing line h.

▶ **Theorem 13.** *We can maintain a 2-approximate maximum independent set in a dynamic unit-height arbitrary-width rectangle intersection graph, deterministically, in $O(\omega \log n)$ time, where ω is the cardinality of a maximum independent set of the rectangles stabbed by the horizontal stabbing line affected by the dynamic update.*

▶ Remark 14. We note that Gavruskin et al. [24] gave a dynamic algorithm for maintaining a MAX-IS on *proper* interval graphs. Their algorithm runs in amortized time $O(\log^2 n)$ for insertion and deletion, and $O(\log n)$ for element-wise decision queries. The complexity to report a MAX-IS J is $\Theta(|J|)$. Whether the same result holds for general interval graphs was posed as an open problem [24]. Our algorithm in fact solves the MAX-IS problem on arbitrary dynamic interval graphs, which is of independent interest. Moreover, it explicitly maintains a MAX-IS at every step.

5 Experiments

We implemented all our MAX-IS approximation algorithms presented in Sections 3 and 4 in order to empirically evaluate their trade-offs in terms of *solution quality*, i.e., the cardinality of the computed independent sets, and *update time* measured on a set of suitable synthetic and real-world map-labeling benchmark instances with unit squares. The goal is to identify those algorithms that best balance the two performance criteria. Moreover, for smaller benchmark instances with up to 2 000 squares, we compute exact MAX-IS solutions using a MaxSAT model by Klute et al. [32] that we solve with MaxHS 3.0 (see `www.maxhs.org`). These exact solutions allow us to evaluate the optimality gaps of the different algorithms in light of their worst-case approximation guarantees. Finally, we investigate the speed-ups gained by using our dynamic update algorithms compared to the baseline of recomputing new solutions from scratch with their respective static algorithm after each update.

5.1 Experimental Setup

We have implemented the five algorithms (and their greedy augmentation variants) listed below in C++. The experiments were run on a server equipped with two Intel Xeon E5-2640 v4 processors (2.4 GHz 10-core) and 160GB RAM. The machine ran the 64-bit version of Ubuntu Bionic (18.04.2 LTS). The code was compiled with g++ 7.4.0.

MIS-graph A naive graph-based dynamic MIS algorithm, explicitly maintaining the square intersection graph and a MIS [4, Sec. 3]. In order to evaluate and compare the performance of our algorithm *MIS-ORS* (Section 3) for the MIS problem, we have implemented this alternative dynamic algorithm. This algorithm, instead of maintaining the current instance in a dynamic geometric data structure, maintains the rectangle intersection graph explicitly as a baseline approach. We use standard adjacency lists to represent the intersection graph, implemented as unordered sets in C++. Now, to obtain a MIS at the first step, we add an arbitrary (unmarked) vertex v to the solution and mark $N(v)$ in the corresponding intersection graph. This process is repeated iteratively until there is no unmarked vertex left in the intersection graph. Clearly, by following this greedy method, we obtain a MIS. Moreover, for each vertex v, we maintain an augmenting *counter* that stores the number of vertices from its neighborhood $N(v)$ that are contained in the current MIS.

This approach handles the updates in a straightforward manner. When a new vertex is inserted, its corresponding rectangle introduces new intersections in the current intersection graph. Therefore, when adding this vertex, we also determine the edges that are required to be added to the intersection graph. Notice that, unlike the canonical vertex update operation defined in the literature, where the adjacencies of the new vertex are part of the dynamic update, here, we actually need to figure out the neighborhood of a vertex. This is done by iterating over each vertex and checking whether its corresponding rectangle is overlapping with the newly inserted rectangle. Thus, it takes $O(n)$ time to obtain the neighborhood of this vertex. If the newly inserted rectangle has no intersection with any rectangle from the current solution, then we simply add its vertex to the solution; otherwise, we ignore it. Finally, we update the counters. If a vertex is deleted, we update the intersection graph by deleting its corresponding rectangle. If the deleted vertex was in the solution, then we decrease the counters of its neighbors by 1. Once the counter of a vertex is updated to 0, we add this vertex into the solution. Both the insertion (after computing $N(v)$) and deletion operation for a vertex v take $O(\deg(v))$ time each to update the intersection graph and the MIS solution.

MIS-ORS The dynamic MIS algorithm based on orthogonal range searching (Section 3); this algorithm provides a 4-approximation. In the implementation we used the dynamic orthogonal range searching data structure implemented in CGAL (version 4.11.2), which is based on a dynamic Delaunay triangulation [35, Chapter 10.6]. Hence, this implementation does not provide the sub-logarithmic worst-case update time of Theorem 3.

grid The grid-based 4-approximation algorithm (Section 4.1).

grid-k The shifting-based $2(1+\frac{1}{k})$-approximation algorithm (Section 4.2). In the experiments we use $k = 2$ (i.e., a 3-approximation) and $k = 4$ (i.e., a 2.5-approximation).

line The stabbing-line based 2-approximation algorithm (Section 4.3).

Since the algorithms *grid*, *grid-k*, and *line* are based on partitioning the set of squares and considering only sufficiently segregated subsets, they produce a lot of white space in practice. For instance, they ignore the squares stabbed by either all the even or all the odd stabbing lines completely in order to create isolated subinstances. In practice, it is therefore interesting to augment the computed approximate MAX-IS by greedily adding independent, but initially discarded squares. We have also implemented the greedy variants of these algorithms, which are denoted as *g-grid*, *g-grid-k*, and *g-line*.

We created three types of benchmark instances. The synthetic data sets consist of n 30×30-pixel squares placed inside a bounding rectangle $\mathcal{B}$ of size $1\,080 \times 720$ pixels, which also creates different densities. The real-world instances use the same square size, but geographic feature distributions. For the updates we consider three models: *insertion-only*, *deletion-only*, and *mixed*, where the latter selects insertion or deletion uniformly at random.

Table 1 Specification of the six OSM instances.

	post-CH	peaks-AT	hotels-CH	hotels-AT	peaks-CH	hamlets-CH
features (n)	646	652	1 788	2 209	4 320	4 326
overlaps (m)	5 376	5 418	28 124	68 985	107 372	159 270
density (m/n)	8.32	8.31	15.73	31.23	24.85	36.92

Gaussian In the Gaussian model, we generate n squares randomly in $\mathcal{B}$ according to an overlay of three Gaussian distributions, where 70% of the squares are from the first distribution, 20% from the second one, and 10% from the third one. The means are sampled uniformly at random in $\mathcal{B}$ and the standard deviation is 100 in both dimensions.

Uniform In the uniform model, we generate n squares in $\mathcal{B}$ uniformly at random.

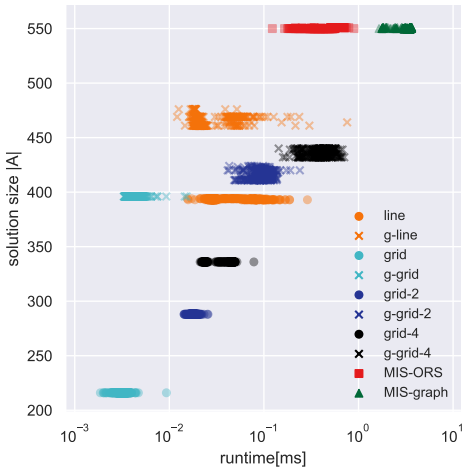
Real-world We created six real-world data sets by extracting point features from OpenStreetMap (OSM), see Table 1 for their detailed properties.

5.2 Experimental Results

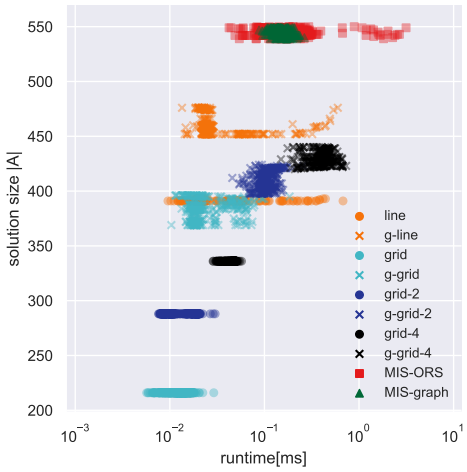
Time-quality trade-offs. For our first set of experiments we compare the five implemented algorithms, including their greedy variants, in terms of update time and size of the computed independent sets. Figure 6 shows scatter plots of runtime vs. solution size on uniform and Gaussian benchmarks, where algorithms with dots in the top-left corner perform well in both measures.

We first consider the results for the uniform instances with $n = 10\,000$ squares in the top row of Figure 6. Each algorithm performed $N = 400$ updates, either insertions (Figure 6a) or deletions (Figure 6b) and each update is shown as one point in the respective color. Both plots show that the two MIS algorithms compute the best solutions with almost the same size and well ahead of the rest. While *MIS-ORS* is clearly faster than *MIS-graph* on insertions, they are comparably fast for deletions, with some slower outliers of *MIS-ORS*. The approximation algorithms *grid*, *grid*-2, *grid*-4, and *line* (without the greedy optimizations) show their predicted relative behavior: The better the solution quality, the worse the update times. Algorithms *line* and *g-line* show a wide range of update times, spanning almost two orders of magnitude. Adding the greedy optimization drastically improves the solution quality in all cases, but typically at the cost of higher runtimes. For *g-grid-k* the algorithms get slower by an order of magnitude and increase the solution size by 30–50%. For *g-grid*, the additional runtime is not as significant (but deletions are slower than insertions), and the solution size almost doubles. Finally, *g-line* is nearly as fast as *line*, and reaches the best quality among the approximation algorithms with about 80% of the MIS solutions, but faster by one or two orders of magnitude.

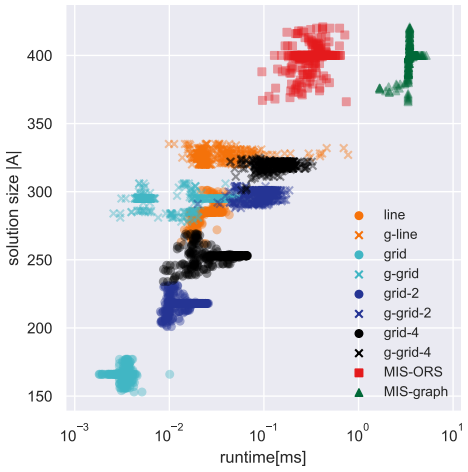
For the results of the Gaussian instances with $n = 10\,000$ squares and $N = 400$ updates plotted in Figures 6c (insertions) and 6d (deletions) we observe the same ranking between the different algorithms. However, due to the non-uniform distribution of squares, the solution sizes are more varying, especially for the insertions. For the deletions it is interesting to see that *grid* and *MIS-graph* have more strongly varying runtimes, which is in contrast to the deletions in the uniform instance, possibly due to the dependence on the vertex degree. The best solutions are computed by *MIS-ORS* and *MIS-graph*, which show similar deletion times, but the insertion times of *MIS-ORS* are one order of magnitude faster than *MIS-graph*. Algorithm *g-line* again reaches more than 80% of the quality of the MIS algorithms, with a speed-up between one and two orders of magnitude.



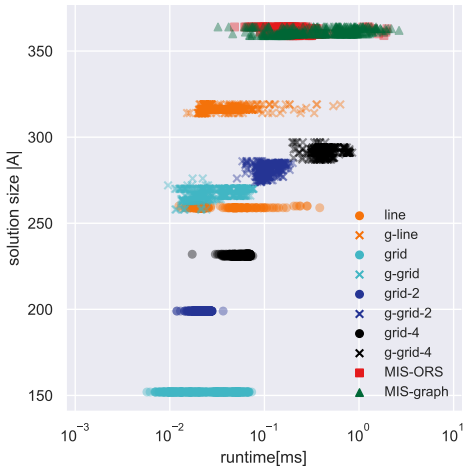
(a) Uniform, $n = 10\,000$, 400 insertions.



(b) Uniform, $n = 10\,000$, 400 deletions.



(c) Gaussian, $n = 10\,000$, 400 insertions.



(d) Gaussian, $n = 10\,000$, 400 deletions.

Figure 6 Time-quality scatter plots for synthetic benchmark instances. The x-axis (log-scale) shows runtime, the y-axis shows the solution size.

Optimality gaps. Next, let us look at the results of the real-world instances in Figure 7. The first four instances in Figure 7a–d, were small enough so that we could compute each Max-IS exactly with MaxHS and compare the solutions of the approximation algorithms with the optimum on the y-axis. The largest two instances in Figure 7e and 7f plot the solution size on the y-axis. First, let us consider Figure 7c as a representative, which is based on a data set of 1 788 hotels and hostels in Switzerland with mixed updates of 10% of the squares ($N = 179$). Generally speaking, the results of the different algorithms are much more overlapping in terms of quality than for the synthetic instances. The plot shows that the MIS algorithms reach consistently between 80% and 85% of the optimum, but are sometimes outperformed by *g-grid*-4 and *g-line*. Regarding the runtime, *MIS-ORS* has more homogeneous update times ranging between the extrema of *MIS-graph*, which suffers from the rather slow insertions. The original approximations are well above their respective worst-case ratios, but stay between 45% and 65% of the optimum. The greedy extensions

push this towards larger solutions, at the cost of higher runtimes. However, *g-line* seems to provide a very good balance between quality and speed. We point out that because the updates comprise insertions and deletions, the marks for algorithms that are sensitive to the update type, such as *g-grid* and *MIS-graph* form two separate runtime clusters. The same relative observations of the algorithms' performance can be made in Figures 7a–d, yet they show different absolute quality offsets and variance.

Let us next consider the largest OSM instance in Figure 7f. It again reflects the same findings as obtained from the smaller instances. The instance consists of $n = 4\,326$ hamlets in Switzerland with 10% mixed updates ($N = 433$) and is denser by a factor of about 2.3 than hotels-CH (see Table 1). There is quite some overlap of the different algorithms in terms of the solution size, yet the algorithms form the same general ranking pattern as observed before. Interestingly, while the MIS algorithms contribute some of the best solutions, they also show a variance of ± 50 squares. In contrast, *g-line*, the best of the approximation algorithms, is competing well and is more stable in terms of solution size and again about an order of magnitude faster than the MIS algorithms. The update-type dependent behavior of *MIS-graph* with its significantly slower insertions is observed once more, making *MIS-ORS* the better choice for a mixed update model.

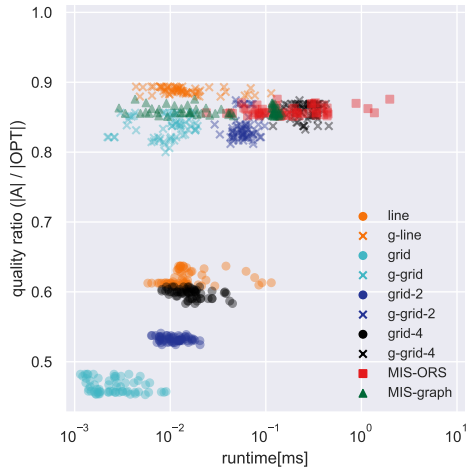
Finally, Figure 8 shows the optimality ratios of the algorithms for small uniform and Gaussian instances with $n = 1\,000$ squares. They confirm our earlier observations, but also show that for these small instances, *MIS-graph* is about as fast as *MIS-ORS* for insertions and faster than *MIS-ORS* for deletions. This is because the graph size and vertex degrees do not yet influence the running time of *MIS-graph* strongly. Yet, as the next experiment shows, this changes drastically, as the instance size grows.

Runtimes. In our last experiment, we explore in more detail the scalability of the algorithms for larger instances, both relative to each other and in comparison to the re-computation times of their corresponding static algorithms. We generated one random instance with $n = 1\,000k$ squares for each $k \in \{1, 2, 4, 8, 16, 32\}$ and measured the average update times over $n/10$ insertions or deletions. The results for the Gaussian and uniform model are plotted in Figure 9. Considering the update times for insertions, we confirm the observations from the scatter plots in terms of the performance ranking. Most algorithms grow only very slowly in terms of their running time, with the notable exception of *MIS-graph*, but that was to be expected. For deletions, *MIS-graph* is initially faster than *MIS-ORS*, but again shows the steepest increase in runtime. Deletions in the Gaussian model also affect the runtime of *grid* and *g-grid* quite noticeably, yet one order of magnitude below *MIS-graph*.

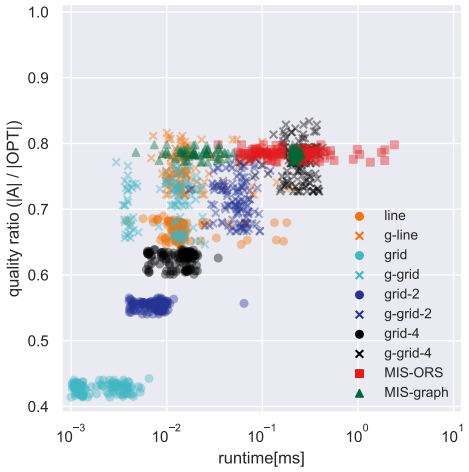
In the comparison with their non-dynamic versions, i.e., re-computing solutions after each update, the dynamic algorithms indeed show a significant speed-up in practice, already for small instance sizes of $n = 1\,000$, and even more so as n grows (notice the different y-offsets). For some algorithms, including *MIS-ORS* and *g-line*, this can be as high as 3–4 orders of magnitude for $n = 32\,000$. It clearly confirms that the investigation of algorithms for dynamic MIS and Max-IS problems for rectangles is well justified also from a practical point of view.

5.3 Discussion

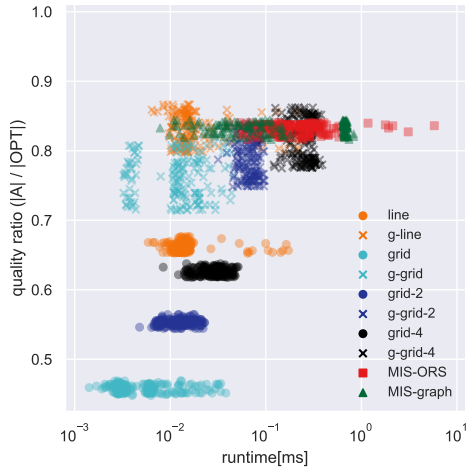
Our experimental evaluation provides several interesting insights into the practical performance of the different algorithms. First of all, both MIS-based algorithms generally showed the best solution quality in the field, reaching 85% of the exact Max-IS size, where we could compare against optimal solutions. This is in strong contrast to their factor-4 worst-case approximation guarantee of only 25%.



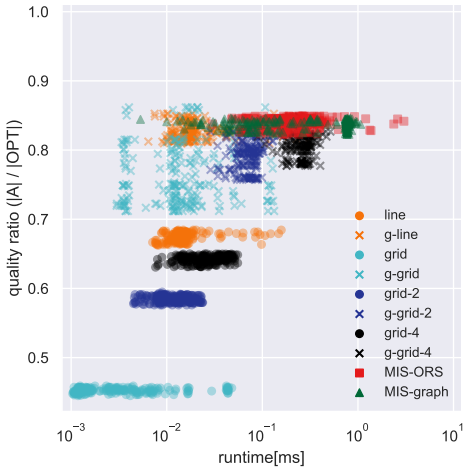
(a) post-CH, 10% mixed updates.



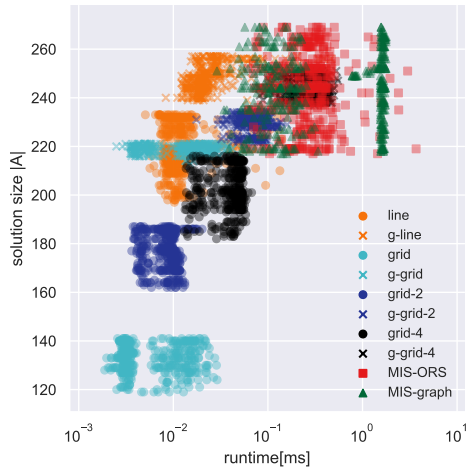
(b) peaks-AT, 10% mixed updates.



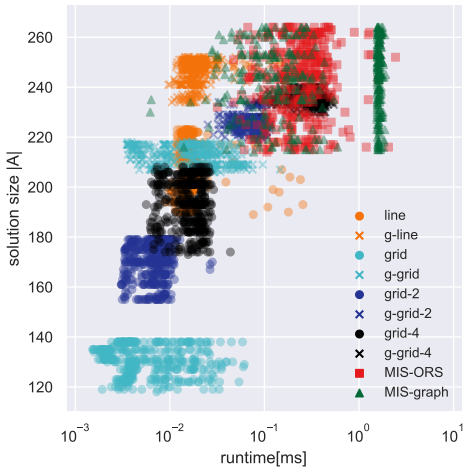
(c) hotels-CH, 10% mixed updates.



(d) hotels-AT, 10% mixed updates.

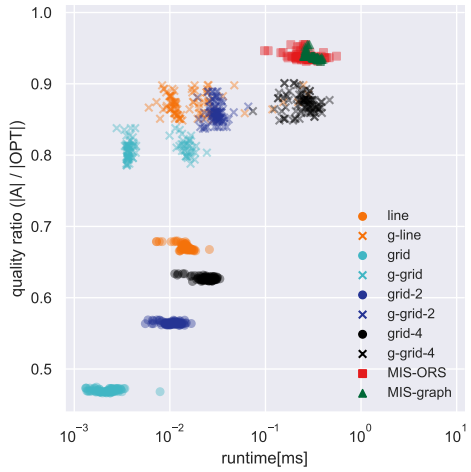


(e) peaks-CH, 10% mixed updates.

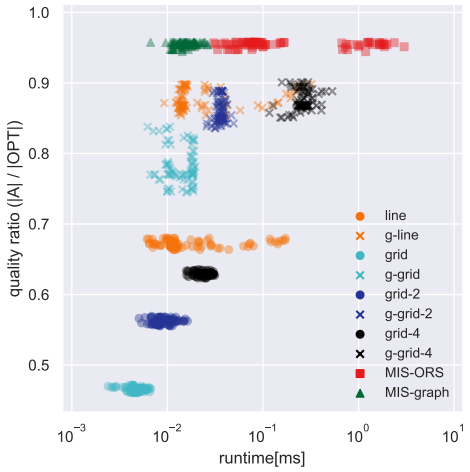


(f) hamlets-CH, 10% mixed updates.

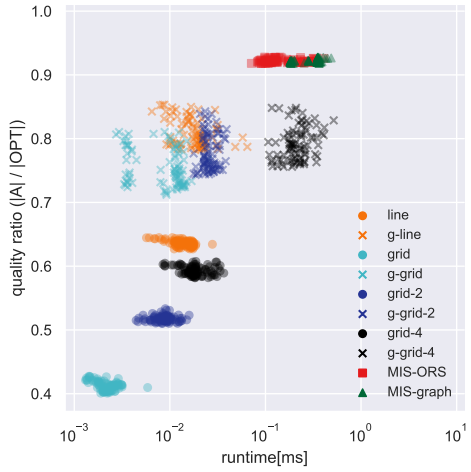
Figure 7 Time-quality scatter plots for the OSM instances. The x-axis (log-scale) shows runtime. The y-axis shows the quality ratio compared to an optimal MAX-IS solution for the smaller instances (a)–(d), and the solution size for the larger instances (e)–(f).



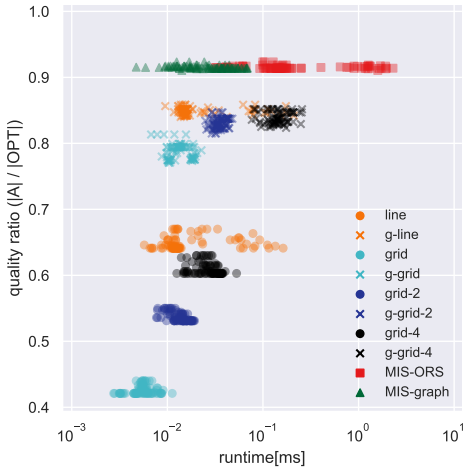
(a) Uniform, 100 insertions.



(b) Uniform, 100 deletions.



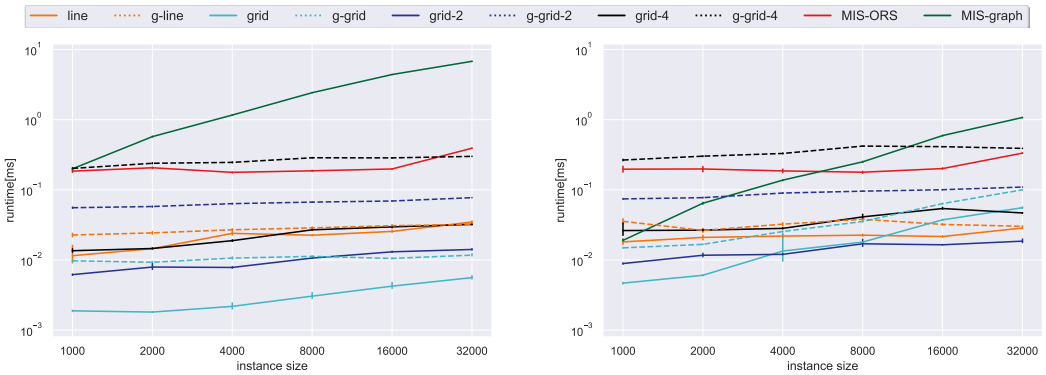
(c) Gaussian, 100 insertions.



(d) Gaussian, 100 deletions.

Figure 8 Time-quality scatter plots for uniform and Gaussian instances with $n = 1\,000$ squares. The x-axis (log-scale) shows runtime. The y-axis shows the quality ratio compared to an optimal Max-IS solution.

Our algorithm *MIS-ORS* avoids storing the intersection graph explicitly. Instead, we only store the relevant geometric information in a dynamic data structure and derive edges on demand. Therefore it breaks the natural barrier of $\Omega(\Delta)$ (amortized) vertex update in a dynamic graph, where Δ is the maximum degree in the graph. However, it has to find the intersections using the complex range query, which takes $O(\log n)$ time. We did not involve any geometric data structure in the baseline MIS approach *MIS-graph*. Recall that, the update of the intersection graph when adding a new rectangle includes the time to figure out the neighborhood of the newly added vertex. Therefore, the graph-based algorithm showed a slow insertion update and was quite sensitive to the size of instances in insertion updates. However, the deletion update only depends on the degree of the involved vertex, not the size of instances directly. And as expected, the graph-based algorithm was indeed much faster for small instances, but *MIS-ORS* was more scalable in our experiment. However, the intersection graph update for *MIS-graph* can be improved by using additionally a geometric



(a) Update times for insertions (Gaussian).

(b) Update times for deletions (Gaussian).

(c) Re-computation times for insertions (Gaussian).

(d) Re-computation times for deletions (Gaussian).

(e) Update times for insertions (uniform).

(f) Update times for deletions (uniform).

(g) Re-computation times for insertions (uniform).

(h) Re-computation times for deletions (uniform).

Figure 9 Log-log runtime plots (notice the different y-offsets) for dynamic updates and re-computation on Gaussian instances (a)–(d) and uniform instances (e)–(h) of size $n = 1\,000$ to $32\,000$, averaged over $n/10$ updates. Error bars indicate the standard deviation.

data structure to store the rectangle set and detect intersections. We expect that it would show improvements for insertion updates, but may slow down deletions, since the state-of-the art data structure provides only an amortized update time guarantee. Therefore, it is an open question whether the performance of *MIS-graph* can indeed be improved by using a suitable dynamic geometric data structure. Note, *MIS-ORS* too, can sometimes show slower deletions, due to the necessary complex orthogonal range search in some cases. Recall that, in our implementation, we used a dynamic range searching data structure from CGAL, which does not provide the theoretical sub-logarithmic worst-case update time of Chan et al. [14] used in Theorem 3. Exploring how *MIS-ORS* can benefit from such a state-of-the-art dynamic data structure in practice remains to be investigated in future work. Notwithstanding, it remains to state that even with the suboptimal data structure, *MIS-ORS* was able to compute its solutions for up to 32 000 squares in less than 1ms. So if solution quality is the priority, then the *MIS-ORS* algorithm is the method of choice. It provides the best solutions (together with *MIS-graph*), but is significantly more scalable.

An expected observation is that while consistently exceeding their theoretical guarantees, the approximation algorithms do not perform too well in practice due to their pigeonhole choice of too strictly separated subinstances. However, a simple greedy augmentation of the approximate solutions can boost the solution size significantly, and for some algorithms even to almost that of the MIS algorithms. Of course, at the same time this increases the runtime of the algorithms. We want to point out *g-line*, the greedy-augmented version of the 2-approximation algorithm *line*, as it computes very good solutions, even comparable or better than *MIS-ORS* and *MIS-graph* for the real-world instances, and at 80% of the MIS solutions for the synthetic instances. At the same time, *g-line* is still significantly faster than *MIS-ORS* and *MIS-graph* and thus turns out to be a well-balanced compromise between time and quality. It would be our recommended method if *MIS-ORS* or *MIS-graph* are too slow for an application.

6 Conclusions

We investigated the MIS and Max-IS problems on dynamic sets of uniform rectangles and uniform-height rectangles from an algorithm engineering perspective, providing both theoretical results for maintaining a MIS or an approximate Max-IS and reporting insights from an experimental study. Open problems for future work include (i) finding Max-IS sublinear-update-time approximation algorithms for dynamic unit squares with approximation ratio better than 2, (ii) studying similar questions for dynamic disk graphs, and (iii) implementing improvements such as a sub-logarithmic dynamic range searching data structure to speed-up our algorithm *MIS-ORS*. Moreover, it would be interesting to design dynamic approximation schemes for Max-IS that maintain stability in a solution.

References

1 Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *Symposium on Theory of Computing (STOC'19)*, pages 114–125. ACM, 2019. doi:10.1145/3313276.3316376.

2 Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Foundations of Computer Science (FOCS'13)*, pages 400–409. IEEE, 2013. doi:10.1109/FOCS.2013.50.

3 Pankaj K Agarwal, Marc Van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Computational Geometry*, 11(3-4):209–218, 1998. doi:10.1016/S0925-7721(98)00028-5.

4 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Symposium on Theory of Computing (STOC'18)*, pages 815–826, 2018. doi:10.1145/3188745.3188922.

5 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *Symposium on Discrete Algorithms (SODA'19)*, pages 1919–1936. SIAM, 2019. doi:10.1137/1.9781611975482.116.

6 Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4):1054–1068, 1986. doi:10.1137/0215075.

7 Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. Optimizing active ranges for consistent dynamic map labeling. *Comput. Geom. Theory Appl.*, 43(3):312–328, 2010. doi:10.1016/j.comgeo.2009.03.006.

8 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *Foundations of Computer Science (FOCS'19)*, pages 382–405. IEEE, 2019. doi:10.1109/FOCS.2019.00032.

9 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *Symposium on Discrete Algorithms (SODA'19)*, pages 1899–1918. SIAM, 2019. doi:10.1137/1.9781611975482.115.

10 Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in $o(1)$ amortized update time. In *Integer Programming and Combinatorial Optimization (IPCO'17)*, volume 10328 of *LNCS*, pages 86–98. Springer, 2017. doi:10.1007/978-3-319-59250-3_8.

11 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Symposium on Discrete Algorithms (SODA'18)*, pages 1–20. SIAM, 2018. doi:10.1137/1.9781611975031.1.

12 Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *Symposium on Discrete Algorithms (SODA'09)*, pages 892–901. SIAM, 2009. doi:10.1137/1.9781611973068.97.

13 Timothy M Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.

14 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the RAM, revisited. In Boris Aronov and Matthew J. Katz, editors, *Computational Geometry (SoCG'17)*, volume 77 of *LIPIcs*, pages 28:1–28:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.SoCG.2017.28.

15 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *Foundations of Computer Science (FOCS'19)*, pages 370–381. IEEE, 2019. doi:10.1109/FOCS.2019.00031.

16 Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *Foundations of Computer Science (FOCS'16)*, pages 820–829. IEEE, 2016. doi:10.1109/FOCS.2016.92.

17 Graham Cormode, Jacques Dark, and Christian Konrad. Independent sets in vertex-arrival streams. In *International Colloquium on Automata, Languages, and Programming (ICALP'19)*, volume 132 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.45.

18 David Eppstein, Zvi Galil, and Giuseppe F. Italiano. Dynamic graph algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 8. CRC Press, 1999. doi:10.1.1.43.8372.

19 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005. doi:10.1137/s0097539702402676.

20 Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Symposium on Computational Geometry (SoCG'91)*, pages 281–288. ACM, 1991. doi:10.1145/109648.109680.

21 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.

22 Edith Gabriel. Spatio-temporal point pattern analysis and modeling. In Shashi Shekhar, Hui Xiong, and Xun Zhou, editors, *Encyclopedia of GIS*, pages 1–8. Springer, 2015. doi:10.1007/978-3-319-23519-6_1646-1.

23 Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *Foundations of Computer Science (FOCS'19)*, pages 26–37, 2019. doi:10.1109/FOCS.2019.00011.

24 Alexander Gavruskin, Bakhadyr Khoussainov, Mikhail Kokho, and Jiamou Liu. Dynamic algorithms for monotonic interval scheduling problem. *Theoretical Computer Science*, 562:227–242, 2015. doi:10.1016/j.tcs.2014.09.046.

25 Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Consistent labeling of rotating maps. *J. Computational Geometry*, 7(1):308–331, 2016. doi:10.20382/jocg.v7i1a15.

26 U. I. Gupta, D. T. Lee, and Joseph Y.-T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12(4):459–467, 1982. doi:10.1002/net.3230120410.

27 Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999. doi:10.1007/BF02392825.

28 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In Sergio Cabello and Danny Z. Chen, editors, *Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *LIPIcs*, pages 51:1–51:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.51.

29 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, 1985. doi:10.1145/2455.214106.

30 John E Hopcroft and Richard M Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

31 Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

32 Fabian Klute, Guangping Li, Raphael Löffler, Martin Nöllenburg, and Manuela Schmidt. Exploring semi-automatic map labeling. In *Advances in Geographic Information Systems (SIGSPATIAL'19)*, pages 13–22. ACM, 2019. doi:10.1145/3347146.3359359.

33 Nathan Linial. Distributive graph algorithms global solutions from local data. In *Foundations of Computer Science (SFCS'87)*, pages 331–335. IEEE, 1987. doi:10.1109/SFCS.1987.20.

34 Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(1–4):215–241, 1990. doi:10.1007/BF01840386.

35 Kurt Mehlhorn and Stefan Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999. doi:10.1145/204865.204889.

36 Huy N Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *Foundations of Computer Science (FOCS'08)*, pages 327–336. IEEE, 2008. doi:10.1109/FOCS.2008.81.

37 Panos M Pardalos and Jue Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, 1994. doi:10.1007/BF01098364.

38 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, 2015. doi:10.1007/s10951-014-0398-5.

39 Marc J. van Kreveld, Tycho Strijk, and Alexander Wolff. Point set labeling with sliding labels. In Ravi Janardan, editor, *Proceedings of the Fourteenth Annual Symposium on Computational Geometry, Minneapolis, Minnesota, USA, June 7-10, 1998*, pages 337–346. ACM, 1998. doi:10.1145/276884.276922.

40 Frank Wagner and Alexander Wolff. A practical map labeling algorithm. *Comput. Geom. Theory Appl.*, 7:387–404, 1997. doi:10.1016/S0925-7721(96)00007-7.

Lower Bounds and Approximation Algorithms for Search Space Sizes in Contraction Hierarchies

Johannes Blum
University of Konstanz, Germany
blum@inf.uni-konstanz.de

Sabine Storandt
University of Konstanz, Germany
storandt@inf.uni-konstanz.de

Abstract

Contraction hierarchies (CH) is a prominent preprocessing-based technique that accelerates the computation of shortest paths in road networks by reducing the search space size of a bidirectional Dijkstra run. To explain the practical success of CH, several theoretical upper bounds for the maximum search space size were derived in previous work. For example, it was shown that in minor-closed graph families search space sizes in $\mathcal{O}(\sqrt{n})$ can be achieved (with n denoting the number of nodes in the graph), and search space sizes in $\mathcal{O}(h \log D)$ in graphs of highway dimension h and diameter D. In this paper, we primarily focus on lower bounds. We prove that the average search space size in a so called weak CH is in $\Omega(b_\alpha)$ for $\alpha \geq 2/3$ where b_α is the size of a smallest α-balanced node separator. This discovery allows us to describe the first approximation algorithm for the average search space size. Our new lower bound also shows that the $\mathcal{O}(\sqrt{n})$ bound for minor-closed graph families is tight. Furthermore, we deeper investigate the relationship of CH and the highway dimension and skeleton dimension of the graph, and prove new lower bound and incomparability results. Finally, we discuss how lower bounds for strong CH can be obtained from solving a HittingSet problem defined on a set of carefully chosen subgraphs of the input network.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases contraction hierarchies, search space size, balanced separator, tree decomposition

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.20

1 Introduction

The concept of contraction hierarchies (CH) was introduced by Geisberger et al. [15] to accelerate shortest path planning in road networks. The basic idea is to precompute an overlay graph in which the search space size of a bidirectional Dijkstra run can be drastically reduced. For example, on the road network of Western Europe with 18 million nodes, a bidirectional Dijkstra run in the original graph scans almost 5 million nodes on average, while in the CH overlay graph only 280 nodes are scanned. This decreases the query time from over two seconds to about one millisecond [5].

There exist two CH variants, referred to as strong and weak contraction hierarchies. In a strong CH, the overlay graph construction takes the edge weights directly into account. In a weak CH, the preprocessing phase is split into a metric-independent overlay graph construction phase, and a subsequent customization phase in which the edges are augmented with weights. Weak CH is used in practice to deal with dynamically changing edge weights, as changes in the metric only require to repeat the customization phase but not the overlay graph construction [12]. In [7], the first theoretical upper bounds for search space sizes in weak CH were described, which are valid for strong CH as well. Other lines of research focus on strong CH directly. Graph parameters such as the highway dimension [1] or the skeleton dimension [17] were explictly introduced with the purpose of analyzing search space sizes

of preprocessing-based route planning techniques. However, most existing work focuses on upper bounding the maximum search space. In this paper, we are particularily interested in constructing tight bounds for the average search space size, as those are more expressive for judging whether the CH technique is useful for a given graph (family). We provide a multitude of novel results, including several lower bounding techniques as well as the first approximation algorithm for the average search space size in a weak CH.

1.1 Related Work

In [2, 1], the *highway dimension* h was introduced as a novel graph parameter to capture the shortest path structure of road networks. It was proven that in a strong CH maximum search space sizes in $\mathcal{O}(h \log D)$ can be achieved where D denotes the diameter of the graph. At the same time, the size of the overlay graph is bounded by $\mathcal{O}(n \cdot h \log D)$ where n is the number of nodes in the graph. In [7], the first upper bounds for weak CH were proven. There, *nested dissection* was used to guide the overlay graph construction; a technique that relies on recursive decomposition of the graph into smaller subgraphs by removing balanced node separators. It was shown that in graphs of *treewidth* tw an upper bound of $\mathcal{O}(tw \log n)$ for the maximum search space size S_{max} can be achieved with an overlay graph size of $\mathcal{O}(n \cdot tw \log n)$. For minor-closed graph families that exhibit $\mathcal{O}(\sqrt{n})$ balanced separators, $S_{max} \in \mathcal{O}(\sqrt{n})$ and an overlay graph size in $\mathcal{O}(n \log n)$ was proven. In [14, 8], the *bounded growth model* was proposed as a theoretical framework for studying road networks. There, CH search space sizes in $\mathcal{O}(\sqrt{n} \log n)$ and overlay graph sizes in $\mathcal{O}(n \log D)$ were shown for graphs with unit edge costs that adhere to the model.

Concerning lower bounds, White [22] proved that for any h, D and n, there exists a graph with at least n nodes that has a highway dimension h, a diameter of $\Theta(D)$, and in which the average search space size in a strong CH is in $\Omega(h \log D)$. Therefore, the upper bound by Abraham et al. [1] for strong CH search space sizes parametrized by h is tight. An analogue result was proven for the related route planning technique *hub labeling*. In [20], it was shown on the example of a carefully weighted grid graph that there exists graphs for which the average search space is in $\Omega(\sqrt{n})$ for any strong CH (and for hub labeling as well). In addition, a greedy algorithm was proposed to compute lower bounds for the average search space size of any strong CH in a given weighted graph. In [7], it was observed that the maximum search space size in a weak CH coincides with the so called elimination tree height. As proven by Bodlaender et al. [9], the minimum elimination tree height of a graph equals the *treedepth* td, and the following relations to other graph parameters hold

$$b - 1 \leq tw \leq pw \leq td$$

where b is the *balanced separator number* of the graph, tw is the *treewidth*, and pw the *pathwidth*. Accordingly, the balanced separator number, the treewidth, the pathwidth, and the treedepth are all valid lower bounds for S_{max}. In [6], it was proven by reduction from VertexCover that it is NP-hard to find a contraction order for a strong CH that minimizes the average search space size while adding at most K shortcuts to the graph. Furthermore it was observed that in a complete graph $S_{avg} \in \Omega(n)$ holds, and in path graphs $S_{avg} \in \Omega(\log n)$. For the special case of trees, a linear-time optimal preprocessing algorithm is known that minimizes S_{max} [21]. Complementarily, it was shown in [7] that a CH-graph constructed with a nested dissection contraction order provides a 2-approximation for S_{avg} in trees.

1.2 Contribution

We significantly extend the set of known results on (average) search space sizes in weak and strong contraction hierarchies. Our main findings are listed in the following.

- Although the balanced separator number, the treewidth, the pathwidth and the treedepth are all lower bounds for the maximum search space size S_{max} in a weak CH, we prove that in general none of them is a valid lower bound for S_{avg}. However, we prove that the average search space size S_{avg} in a weak CH is in $\Omega(b_\alpha)$ for $\alpha \geq 2/3$ where b_α is the size of a smallest α-balanced node separator in G.
- We establish the first approximation result for S_{avg} in weak CHs which applies to general graphs. In particular, we prove that a nested dissection CH construction scheme leads to an average search space size within a factor of $1 + 1.5 \log_{1.5} n$ of the optimum. This answers an open question from [7]. We also discuss how to turn the nested dissection approach into an efficient CH construction algorithm on general graphs.
- We furthermore show that for every unweighted graph, one can choose metric edge weights such that the highway dimension h of the graph equals 1, and hence constitutes a trivial lower bound for the average search space size in a strong CH. On such graphs, the algorithms described in [2, 1] with an upper bound on the search space size of $\mathcal{O}(h \log D)$ are close-to-optimal. However, we also show that in case the metric is given, there can be a gap of size $\Omega(n)$ between the highway dimension and the maximum search space size.
- For a given weighted graph, we prove that lower bounds on the average search space size in the respective strong CH can be obtained by solving a (hierarchical) HittingSet problem defined on a set of specific subgraphs of G.

2 Preliminaries

In this section, we describe the concepts of strong and weak CH more formally and provide the definitions of some separator-related graph parameters that will be relevant in our analysis.

2.1 Strong and Weak Contraction Hierarchies

Given an undirected[1], connected graph $G(V, E)$ with edge costs $c : E \to \mathbb{R}^+$, the preprocessing phase for *strong CH* works as follows: First a node permutation is fixed which constitutes the so called contraction order of the nodes. Note that any contraction order is feasible but that the overlay graph size as well as the search space sizes crucially depend on that order. Let $r : V \to \{1, \dots, n\}$ with $n = |V|$ be the respective ranks of the nodes in the contraction order. Then an overlay graph is constructed with the following property: A shortcut edge $\{v, w\}$ is introduced between v and w if and only if there exists a shortest path π between them that except for v and w only contains nodes u with $r(u) < \min\{r(v), r(w)\}$. The cost of that shortcut edge is set to $c(\pi)$. When constructing a *weak CH* instead, shortcuts are introduced between nodes v and w iff there exists any path between them that except for v and w only contains nodes u with $r(u) < \min\{r(v), r(w)\}$. In either case, the set of shortcuts E^+ is then added to the original graph G to obtain the CH-graph $G^+(V, E \cup E^+)$. For strong CH, this completes the preprocessing phase. For weak CH, a so called customization phase follows in which edge weights are first assigned to the original edges, and are then propagated to the shortcuts. For further details, we refer to [12].

[1] With slight modifications, CH also works on directed graphs. To simplify the presentation, we only consider undirected graphs throughout the paper.

In practical implementations, the node contraction order is not always fixed a priori but might be determined in a process which interleaves the overlay graph construction and the node rank assignments. The insertion of shortcut edges is then based on the so called node contraction operation: Here, a node v and its incident edges are removed from the graph and new edges are inserted between certain pairs of its former neighbors. In a strong CH, a shortcut between the neighbors u, w of v is only inserted if the path u, v, w is a shortest path. In a weak CH, shortcuts are inserted between all pairs of neighbors (as without knowledge about the metric, any of those paths could become a shortest path later). The preprocessing then consists of contracting all nodes in the graph one after the other, using e.g. the current degree of the nodes as guidance which of them to contract next (or more complicated criteria, see [15]). But independently of the chosen implementation, the formal characterization of the resulting overlay CH-graph given above always applies.

To compute a shortest path between nodes $s, t \in V$ in the CH-graph G^+, a bidirectional Dijkstra run is started from s and t in G^+ with the restriction that from any node v incident edges $\{v, w\}$ are only relaxed if $r(w) > r(v)$. The set of nodes that can be reached from a node $v \in V$ via paths which go strictly upwards with respect to the contraction order is called the search space $SS(v)$. By construction of G^+ one can show that there always exists a node $p \in SS(s) \cap SS(t)$ which lies on a shortest path between s and t in G, and the shortest path distance from s to p as well as the shortest path distance from p to t are correctly computed in G^+. Accordingly, the computed shortest path distance in G^+ equals the shortest path distance in G.

Note that in general there is a difference between the search space size and the query time. The search space size $|SS(v)|$ of a node v is the number of nodes settled in the Dijkstra run from v in the CH-graph while the query time also accounts for edge relaxation and priority queue operations. However, an upper bound U on the search space size also yields an upper bound of U^2 on the query time as there can be at most a quadratic number of edges between U nodes. The other way around, a lower bound on the search space size is automatically also a lower bound for the query time. Therefore, we stick to the notion of search space sizes throughout the paper. We distinguish between the maximum CH search space size $S_{max} = \max_{v \in V} |SS(v)|$ and the average CH search space size $S_{avg} = \frac{1}{n} \sum_{v \in V} |SS(v)| \leq S_{max}$.

2.2 Balanced Separators

Existing upper bounds for search space sizes in weak CH were obtained by using nested dissection based overlay graph construction, which involves the recursive computation of balanced node separators. We will investigate which separator-related graph parameters can be used to lower bound the average search space size. The *balanced separator number* is defined as the smallest integer b such that for every $V' \subseteq V$, the induced subgraph $G[V']$ admits a balanced separator of size at most b, i.e. after the removal of at most b nodes every remaining connected component of $G[V']$ contains at most $\lceil (|V'| - b)/2 \rceil$ nodes. While $b - 1$ is known to lower bound the maximum CH search space size in any given graph, we will show that the same is not true for S_{avg}. However, we will prove a lower bound of $\alpha \cdot b_\alpha$ for $\alpha \geq 2/3$ which later will be used to get an approximation guarantee for S_{avg} when constructing the CH with a variant of nested dissection. The parameter b_α, for an $\alpha \in [0, 1]$, is the size of a minimum α-balanced node separator of $G(V, E)$, i.e., the smallest number of nodes that have to be removed from the input graph $G(V, E)$ such that all remaining connected components have size at most $\alpha \cdot |V|$. Note that the value of b_α is solely determined by considering the input graph as a whole, while for b all induced subgraphs are relevant as well. By definition,

the following hierarchy holds: $b_{2/3} \leq b_{1/2} \leq b$.

3 Bounds and Algorithms for Search Space Sizes in Weak CH

In this section, we discuss at the beginning how S_{avg}, S_{max}, and the balanced separator number relate to each other in a weak CH. Subsequently, we describe new lower bounding techniques and present the first approximation algorithm for S_{avg}.

3.1 Maximum versus Average Search Space Size

As discussed above, the smallest possible maximum search space size of a weak CH is equal to the treedepth of the input graph. However, there can be a large gap between treedepth and average search space size. In fact, not even the balanced separator number b yields a lower bound for S_{avg} as the following lemma shows.

▶ **Lemma 1.** *There exist graphs $G(V, E)$ with $S_{avg} \in o(b)$.*

Proof. Take a star graph with k leaves and replace one leaf with a clique C of size $\sqrt{k}$. Every balanced separator of the subgraph induced by C contains $\Omega(\sqrt{k})$ vertices, which implies that $b \in \Omega(\sqrt{k})$. Consider a contraction order where the central node obtains rank $n = |V|$. Then the search space size of the central node is 1, the search space size of every leaf is 2 and the search space size of a node in the clique is at most $\sqrt{k} + 1$. It follows that the average search space size is

$$S_{avg} \leq \frac{1 + (k-1) \cdot 2 + \sqrt{k} \cdot (\sqrt{k} + 1)}{n} = \frac{3k + \sqrt{k} - 1}{n} = \frac{3k + \sqrt{k} - 1}{k + \sqrt{k}} < 3 \in o(\sqrt{k}). \quad \blacktriangleleft$$

With $b - 1 \leq tw \leq pw \leq td$, the lemma implies that none of those parameters is a valid lower bound for S_{avg}; and that there can be an exponential gap between the average and the maximum search space size.

3.2 Balanced Separators and Average Search Space Size

In a weak CH, a node $w \in V$ of rank R is in the search space $SS(v)$ of another node $v \in V$ with rank $r < R$ if there exists a path from v to w which – except for w – exclusively contains nodes of rank $< R$. This fact leads to the following central observation.

▶ **Observation 2.** *Let $G(V, E)$ be a connected graph. In a weak CH on G, for a node $w \in V$ of rank R to* not *be in the search space $SS(v)$ of a node $v \in V$ of rank $r < R$, there needs to be a set of nodes of rank higher than R that separates w from v.*

We will now present our first main theorem, which exploits this observation to show a general relationship between the average serach space size S_{avg} in a weak CH and balanced node separators in G.

▶ **Theorem 3.** *For any weak CH and any $\alpha \geq \frac{2}{3}$, at least αn nodes in G have a search space of size at least b_α.*

Proof. Consider some contraction order of the nodes. We identify the smallest integer k such that the k nodes with highest rank in the given order form an α-balanced node separator S in G. Hence, if we remove the $k - 1$ nodes of highest rank, there is still a connected component C of size $> \alpha n$. If we now remove the node v_k with kth highest rank, we can

split C into two subgraphs of size at most αn each, which are not connected to each other. We observe that the larger of the two subgraphs C^* (which is not necessarily connected) has to contain at least $\frac{\alpha}{2}n$ nodes. Therefore, there are at most $n - \frac{\alpha}{2}n$ nodes that are not contained in C^*, which for any $\alpha \geq \frac{2}{3}$ is at most αn. It follows that the separator nodes $S^* \subseteq S$ that are adjacent to C^* form an α-balanced separator already. By definition of b_α, we have $|S^*| \geq b_\alpha$. It remains to be shown that between any node v in $C \setminus v_k$ (which has rank $r(v) < n-k$) and any separator node $s \in S^*$ (which has rank $r(s) \geq n-k$) there exists a path in G on which all nodes have rank at most $r(s)$. For $s = v_k$ this is true, because C is connected and v_k has maximum rank among all nodes of C. If we have $s \neq v_k$, the node s has a neighbor w contained in C. As C is connected, there is a path from v to w in C and hence, the maximum rank on this path is at most $n - k < r(s)$. It follows that G contains a path from v to s of maximum rank $r(s)$. This means that there are αn nodes (recall that C has size $> \alpha n$), which have a search space size of at least $|S^*| \geq b_\alpha$. ◀

It follows that for any $\alpha \geq \frac{2}{3}$ the sum of the search space sizes of all nodes in G is at least $\alpha n \cdot b_\alpha$ and hence the average search space size is lower bounded by $\alpha \cdot b_\alpha$.

▶ **Corollary 4.** *In any weak CH graph, $S_{avg} \geq \alpha \cdot b_\alpha$ for $\alpha \in \left[\frac{2}{3}, 1\right)$.*

The theorem implies that weak CH performs poorly on graph families with no small balanced separators. For the class of planar graphs, and $\alpha = \frac{2}{3}$, there exist graphs with a smallest α-balanced separator of size at least $1.55\sqrt{n}$ [13]. Hence, according to Corollary 4, their induced average CH search spaces are in $\frac{2}{3}1.55\sqrt{n} = 1.0\overline{3}\sqrt{n} \in \Omega(\sqrt{n})$. This matches the known upper bounds for S_{max} in minor-closed graph families with $\mathcal{O}(\sqrt{n})$ sized balanced separators, proving them to be tight.

We prove next that $S_{avg} \in \Omega(b_{1/2})$ holds as well which will later be important for establishing our approximation guarantee. Note that if we just consider balanced separators in G directly, there could be an arbitrary large gap between $b_{1/2}$ and $b_{2/3}$. Hence, we will consider separators in G and in a selected subgraph of G simultaneously to get a meaningful bound. We first show a helping lemma which might also be of independent interest.

▶ **Lemma 5.** *Let G' be a connected subgraph of G with n' nodes and let b'_α be the size of a smallest α-balanced separator in G' for an $\alpha \geq \frac{2}{3}$, then it follows that $S_{avg} \geq \frac{n'}{n}\alpha b'_\alpha$.*

Proof. The proof is the same as the proof of Theorem 3 with the only modification that we consider the k nodes with highest rank in G' that form an α-balanced separator in G' instead of considering G as a whole. ◀

The lemma improves the lower bound on S_{avg} shown in Theorem 6 in case a subgraph of G has a larger balanced separator than G as a whole and this subgraph G' is not too small compared to the total size of G. And it also allows us to prove a general relationship between S_{avg} and $b_{1/2}$ as manifested in the following theorem.

▶ **Theorem 6.** *The average search space size in a weak CH is lower bounded by $\frac{2}{9}b_{1/2}$.*

Proof. Let $b_{1/2}$ and $b_{2/3}$ be minimum balanced separator sizes in G for the respective α-values. Obviously, $b_{2/3} \leq b_{1/2}$ holds. Let G^* be the larger of the two parts that results from removing the $b_{2/3}$ nodes in the respective separator from G. The smaller part has to contain less then $\frac{1}{2}n$ nodes; but G^* may contain between $\frac{1}{3}n$ and $\frac{2}{3}n$ nodes. Now let $b'_{2/3}$ be the minimum size of a $2/3$-balanced separator in G^*. The largest part of G^* after the removal of such a separator has size at most $\frac{2}{3} \cdot \frac{2}{3}n = \frac{4}{9}n < \frac{1}{2}n$. Accordingly, the union of the nodes in the $2/3$-balanced separator in G and the $2/3$-balanced separator in G^* form a $1/2$-balanced separator in G. It

follows that $b_{1/2} \leq b_{2/3} + b'_{2/3} \leq 2\max(b_{2/3}, b'_{2/3})$. Using Lemma 5 together with the relation between the separators in G and G^* leads to the following set of inequalities that lower bound the average search space size: $S_{avg} \geq \max(\frac{2}{3}b_{2/3}, \frac{2}{3} \cdot \frac{2}{3}b'_{2/3}) \geq \frac{4}{9}\max(b_{2/3}, b'_{2/3}) \geq \frac{2}{9}b_{1/2}$ ◀

3.3 An Approximation Algorithm for the Average Search Space Size

The crucial part of CH construction is fixing the contraction order of the nodes. When using nested dissection, as proposed in [7], the input graph G is partitioned recursively into smaller subgraphs via a balanced separator decomposition, and for each obtained subgraph the nodes in the separator are contracted before the other nodes. It was proven that this contraction order allows to upper bound the resulting maximum search space size S^{ND}_{max}. More precisely, an (α, f_α)-balanced separator decomposition of $G(V, E)$ for $\alpha \in (0, 1)$ is a rooted tree T whose nodes are disjoint subsets of V and that is recursively defined as follows. If $n = 1$, then T consists of a single node $X = V$. If $n > 1$, then the root of T is a set $X \subseteq V$ of size at most $f_\alpha(n)$ whose removal separates G into $d \geq 2$ subgraphs $G_1, \ldots, G_d$ with at most αn vertices, each. Moreover, the children of X are the roots of (α, f_α)-balanced separator decompositions of $G_1, \ldots, G_d$. A nested dissection order can then be obtained via a post-order traversal of such a (α, f_α)-balanced decomposition (nodes within one separator X can be contracted in an arbitrary order). It follows that $S^{ND}_{max} \leq (1 + \log_{1/\alpha} n) \cdot B$ where B is an upper bound on the separator sizes f_α in all subgraphs. As every graph has a $(1/2, b)$-balanced separator decomposition where b is the balanced separator number of G, it follows that $S^{ND}_{max} \in \mathcal{O}(b \log n)$. Combined with $S^{ND}_{max} \geq S^*_{max} = td > b$, where S^*_{max} is the smallest possible value of S_{max}, this proves an $\mathcal{O}(\log n)$ approximation guarantee for the maximum search space size[2].

For trees, a $(1/2, 1)$-balanced separator decomposition exists. Accordingly, $S^{ND}_{max} \in \mathcal{O}(\log n)$ holds for trees. Furthermore, it was proven in [7] that the nested dissection contraction order for trees also is an approximation algorithm for S_{avg}. In particular, $S^{ND}_{avg} \leq 2 \cdot S^*_{avg}$ was shown where S^*_{avg} is the optimal average search space size. It was posed as one of the main open questions whether there are ways to approximate the average search space size in general graphs. Note that the approximation guarantee proof given for trees explicitly leverages the fact that trees are cycle free, and that balanced separators in trees always consist of a single node. Hence new proof concepts are required for generalizing the result.

Exploiting our novel lower bound shown in Corollary 4, we will now prove that for general graphs, the contraction order induced by a (α, b_α)-balanced decomposition indeed comes with an approximation guarantee for S_{avg}. Our proof consists of the following three steps:

1. *Upper bounding* $S^{ND}_{avg}(G)$. We show that $S^{ND}_{avg}(G) \leq \frac{1}{n}\sum_{i=1}^{d} n_i \cdot S^{ND}_{avg}(G_i) + b_\alpha$ where b_α is the size of an α-balanced separator B in G, whose removal splits G into the connected subgraphs $G_1, \ldots, G_d$ of sizes $n_1, \ldots, n_d$ (cf. Lemma 7).
2. *Lower bounding* $S^*_{avg}(G)$. We prove that for the optimal average search space size S^*_{avg} it holds that $S^*_{avg}(G) \geq \frac{1}{n}\sum_{i=1}^{d} n_i \cdot S^*_{avg}(G_i)$ (cf. Lemma 8).
3. *Combining upper and lower bounds.* Combining the outcomes of steps 1 and 2 as well as Corollary 4, we show that with every level of the separator decomposition, the ratio between S^{ND}_{avg} and S^*_{avg} increases at most by $\frac{1}{\alpha}$. This results in an overall approximation factor of $1 + \frac{1}{\alpha}\log_{1/\alpha}$ for $\alpha \in [\frac{2}{3}, 1)$ (cf. Theorem 9).

[2] This result does not hold in directed graphs as shown in [11].

▶ **Lemma 7.** *Given a graph G, it yields $S^{ND}_{avg}(G) \leq \frac{1}{n}\sum_{i=1}^{d} n_i \cdot S^{ND}_{avg}(G_i) + b_\alpha$ where b_α is the size of a given α-balanced separator B in G, and $G_1, \ldots, G_d$ are the connected subgraphs of G that remain after removing B, with $n_1, \ldots, n_d$ being their respective node set sizes.*

Proof. The average search space size S_{avg} is defined as the sum of the individual search space sizes divided by n. For each node in the separator B, the search space size can be at most $|B| = b_\alpha$ as the nodes in B all have higher rank in the contraction order than the nodes in $V \setminus B$. For $i = 1, \ldots, d$, the n_i nodes of G_i have a total search space sizes of $n_i \cdot S^{ND}_{avg}(G_i)$ if we do not count the nodes from B. The search space size of each individual node increases by at most b_α when adding B to the graph. Hence in total we get $n \cdot S^{ND}_{avg} \leq \sum_{i=1}^{d} n_i \cdot S^{ND}_{avg}(G_i) + n \cdot b_\alpha$. Dividing both sides by n concludes the proof. ◀

▶ **Lemma 8.** *Given a graph G, the optimal average search space size in a weak CH is lower bounded by $S^*_{avg}(G) \geq \frac{1}{n}\sum_{i=1}^{d} n_i \cdot S^*_{avg}(G_i)$ where G_i are disjoint subgraphs of G.*

Proof. Let r^* be the optimal contraction order for G, leading to an average search space size of $S^*_{avg}(G) = \frac{1}{n}\sum_{v \in V} |SS^*(v)|$. By r^*_i we denote the restriction of r^* to subgraph G_i, that is, the nodes in G_i are sorted by their r^* values and $r^*_i : V(G_i) \to \{1, \ldots, n_i\}$ then assigns each node $v \in V(G_i)$ the rank of v in the obtained order. Let $v \in V(G_i)$ and denote by $SS'(v)$ the search space of v in G_i when using contraction order r^*_i. We show that $|SS^*(v) \cap V(G_i)| \geq |SS'(v)|$, implying that the part of a search space of a node v that intersects a certain subgraph is as least as large as the search space of v in that subgraph when using the globally optimal contraction order restricted to that subgraph. For proving this property, consider a node $w \in SS'(v)$. As w is contained in this search space, there exists a path from v to w in G_i with all nodes on the path having a rank of at most $r^*_i(w)$. This path then also exists in G and as for nodes x, y with $r^*(x) > r^*(y)$, we have $r^*_i(x) > r^*_i(y)$ by construction, it follows that $w \in SS^*(v) \cap V(G_i)$ holds as well. Therefore, $SS^*(v) \cap V(G_i) \supseteq SS'(v)$ applies. Plugging the resulting size inequality into the definition of the average search space results in the following lower bound:

$$\begin{aligned} n \cdot S^*_{avg}(G) &= \sum_{v \in V} |SS^*(v)| \geq \sum_{i=1}^{d} \sum_{v \in V(G_i)} |SS^*(v)| \geq \sum_{i=1}^{d} \sum_{v \in V(G_i)} |SS^*(v) \cap V(G_i)| \\ &\geq \sum_{i=1}^{d} \sum_{v \in V(G_i)} |SS'(v)| \end{aligned}$$

As the application of any contraction order to G_i results in summed search space sizes that are at least as large as $n_i \cdot S^*_{avg}(G_i)$ where $S^*_{avg}(G_i)$ is the optimal average search space in G_i, we get $S^*_{avg}(G) \geq \frac{1}{n}\sum_{i=1}^{d}\sum_{v \in V(G_i)} |SS'(v)| \geq \frac{1}{n}\sum_{i=1}^{d} n_i \cdot S^*_{avg}(G_i)$. ◀

▶ **Theorem 9.** *Given a graph G, nested dissection contraction results in search space sizes $S^{ND}_{avg} \leq (1 + \frac{1}{\alpha}\log_{1/\alpha} n) \cdot S^*_{avg}$ for any $\alpha \geq \frac{2}{3}$, if optimal separators are used.*

Proof. Consider the (α, b_α)-balanced separator decomposition which induces the nested dissection order. For every leaf X in the decomposition we have $S^{ND}_{avg}(X) = S^*_{avg}(X)$ as X contains only one vertex. Consider now some non-leaf node X from the decomposition. Let H be the subgraph of G induced by X and its descendants in the separator decomposition and denote the connected components of $H \setminus X$ by $H_1, \ldots, H_d$. Denote the size of H_i by n_i and assume that for the average search spaces of $H_1, \ldots, H_d$ we have an approximation

factor of γ, i.e. $S^{ND}_{avg}(H_i) \leq \gamma \cdot S^*_{avg}(H_i)$. Lemma 7 implies

$$S^{ND}_{avg}(H) \leq \frac{1}{n}\sum_{i=1}^{d} n_i \cdot S^{ND}_{avg}(H_i) + |X| \leq \gamma \cdot \frac{1}{n}\sum_{i=1}^{d} n_i \cdot S^*_{avg}(H_i) + |X|$$

Moreover, X is an optimal α-balanced separator, so Corollary 4 implies that $S^*_{avg}(H) \geq \alpha \cdot |X|$, which can be rearranged to $|X| \leq \frac{1}{\alpha} S^*_{avg}(H)$. In combination with Lemma 8 we obtain

$$\gamma \cdot \frac{1}{n}\sum_{i=1}^{d} n_i \cdot S^*_{avg}(H_i) + |X| \leq \gamma \cdot S^*_{avg}(H) + \frac{1}{\alpha} \cdot S^*_{avg}(H) \leq (\gamma + \frac{1}{\alpha}) \cdot S^*_{avg}(H)$$

As for every leaf X we have $S^{ND}_{avg}(X) = S^*_{avg}(X)$ and the height of the separator decomposition is $\log_{1/\alpha} n$, it follows by induction that $S^{ND}_{avg}(G) \leq (1 + \frac{1}{\alpha}\log_{1/\alpha} n) \cdot S^*_{avg}(G)$. ◀

▶ **Corollary 10.** *The average search space size in a weak CH using nested dissection based on recursive decomposition with $b_{2/3}$ is at most $1 + 1.5\log_{1.5} n$ times the optimal size.*

However, as computing optimal balanced separators is NP-hard in general, nested dissection does not directly yield a polynomial time approximation algorithm. But we can exploit the existence of a pseudo-approximation for balanced separators. In [18] it was proven that one can find a 3/4-balanced separator that has size at most $\mathcal{O}(\log n) \cdot b_{2/3}$ in polynomial time[3]. In general, if we have a γ-approximation for $b_{2/3}$, Lemma 7 can be modified to show that $S^{ND}_{avg} \leq \sum_{i=1}^{d} n_i \cdot S^{ND}_{avg}(G_i) + \gamma b_{2/3}$ which plugged into Theorem 9 results in an average search space size of $S^{ND}_{avg} \leq (1 + 1.5\gamma\log_{1.5} n) S^*_{avg}$. Using the pseudo-approximation result, we have $\gamma = \mathcal{O}(\log n)$, and additionally the depth of the recursion increases from $\log_{3/2} n$ to $\log_{4/3} n$. We call the nested dissection based contraction algorithm which leverages the pseudo-approximation algorithm to compute the node separators the pND-algorithm. Combining the aforementioned observations, we get the following theorem.

▶ **Theorem 11.** *Given a graph $G(V, E)$, then a weak CH obtained from the pND-algorithm in polynomial time leads to an average search space size of $S^{pND}_{avg}(G)$ with $S^{pND}_{avg}(G) \leq \mathcal{O}(\log^2 n) S^*_{avg}(G)$, where $S^*_{avg}(G)$ denotes the minimum average search space size.*

Note that for some graph classes better approximation algorithms for balanced node separators exist, which then can be plugged into our analysis to achieve tighter overall results. For example, for any graph class for which optimal balanced separators can be approximated by some constant factor $c > 0$ in polynomial time (as it is the case e.g., for planar graphs [4]), we achieve an overall approximation ratio of $\mathcal{O}(\log n)$ for S_{avg}. For the special case of $\sqrt{n} \times \sqrt{n}$ rectangular grids, the results from [7] imply that nested dissection leads to search space sizes of at most $3\sqrt{n}$. This matches our lower bound of $\alpha \cdot b_\alpha$ (translating for grids to $\frac{2}{3}\sqrt{n}$) up to a factor of 4.5. As the respective separators can be found efficiently in grids, this implies that nested dissection yields a constant factor approximation algorithm for this graph class.

Regarding space consumption, it was argued in [7] that a weak CH-graph constructed by nested dissection contains at most $n \cdot S_{avg}$ shortcut edges. However, this bound is rather loose on some graphs. For example, it is known for planar graphs that contraction orders exist such that the CH-graph size is in $\mathcal{O}(n \log n)$ while e.g., for grids we know by

[3] This is called a pseudo-approximation or bicriteria approximation as the approximation factor compares the output for a relaxed problem version to the optimum of the unrelaxed version.

Corollary 4 that $n \cdot S_{avg}$ amounts to $\Omega(n\sqrt{n})$. In [3], it was proven that nested dissection indeed approximates the number of shortcut edges within a factor of $\mathcal{O}(\sqrt{d}\log^4 n)$ where d is the maximum degree of the input graph. To achieve polynomial running time, the nested dissection variant described there relies on a pseudo-approximation for $1/2$-balanced separators. As proven in Theorem 6, the average search space size is also lower bounded by $2/9 \cdot b_{1/2}$. This can be plugged in our approximation algorithm analysis to also get an $\mathcal{O}(\log^2 n)$ overall approximation factor for S_{avg}. For road networks, where the maximum node degree is a small constant, it therefore yields that a polynomial time variant of nested dissection approximates the three important aspects of a CH-graph – space consumption (number of inserted shortcuts), maximum search space size, and average search space size – all by polylogarithmic factors.

4 Relation to Road Network Dimensions

So far, we considered search space sizes in weak CHs. Now we shift our focus to strong CH. In particular, we now study the relationship between the average search space size in a strong CH and the highway dimension h as well as the skeleton dimension k of the graph. Those two parameters were both previously used to show upper bounds on the search space size of preprocessing-based route planning techniques.

4.1 Highway Dimension Lower Bound

The highway dimension of a weighted graph $G(V, E)$ is defined as follows [1]: For $u \in V$ and $r > 0$, the ball $B_r(u)$ consists of all nodes at distance at most r from u. Consider now the set $\mathcal{P}$ of all shortest paths longer than r that are contained within the ball $B_{4r}(u)$ and let $H_r^u \subseteq V$ be a minimum hitting set for $\mathcal{P}$. The highway dimension of G is defined as $h = \max_{u,r} |H_r^u|$. For a graph with highway dimension h, $S_{max} \in \mathcal{O}(h \log D)$ was proven for strong CH [1]. Furthermore, it was shown in [7] that there exist graphs and edge weights such that $h \in \Omega(pw/\log n)$ holds. We now establish a complementary result where the aim is to find edge weights such that the value of h is as small as possible.

▶ **Lemma 12.** *For any unweighted graph G there are metric edge weights such that the highway dimension of G is 1.*

Proof. Let $V = \{1, \dots, n\}$ and choose the weight of an edge $\{u, v\}$ as $9^{\max\{u,v\}}$. To bound the highway dimension consider some ball $B_{4r}(v)$ and choose the largest j such that $9^j \leq 4r$. If $v > j$, any edge incident to v has length at least $9^v \geq 9^{j+1} > 4r$. This means that $B_{4r}(v) = \{v\}$ and the ball contains no shortest path that needs to be hit. Let now $v \leq j$. Consider some shortest path π that passes only through vertices $i < j$. The path π is simple and hence by the choice of the edge weights it contains no three edges of same length. This means that we can bound the length of π by $\sum_{i=j/2}^{j-1} 2 \cdot 9^i < 2 \cdot \sum_{i=0}^{j-1} 9^i = (9^j - 1)/4 \leq r$. Moreover, any $u > j$ has distance $9^u \geq 9^{j+1} > 4r$ from v and hence no shortest path passing through u is contained in $B_{4r}(v)$. This means that every relevant shortest path contains j and can be hit by the set $\{j\}$. ◀

The lemma shows that the known upper bound on the maximum search space size of $\mathcal{O}(h \log D)$ is almost tight when the metric is chosen appropriately. White [22] showed for a special family of graphs called $G_{t,k,q}$ graphs (introduced in [19]) that the average search space size is in $\Omega(h \log D)$, which matches the upper bound asymptotically. His proof strongly relies on the characteristics of that graph family, though, and especially on the graph topology.

Our lower bound, however, is independent of the graph structure but just considers the metric. Our result also generalizes to the hub labeling technique where a matching upper bound on the maximum number of hubs per node was proven in [1].

4.2 General Incomparability

While we showed above that we can always choose edge weights such that the highway dimension lower bounds the average search space size, this is not necessarily true if the edge weights are given. In fact, we will prove that in general, it is not possible to lower bound the maximum (or average) CH search space size in terms of the highway dimension or the skeleton dimension. For the skeleton dimension k, no upper bounds for CH are known so far, but for the related technique of hub labeling [17], upper bounds of $\mathcal{O}(k \log n)$ were shown.

Intuitively, the skeleton dimension measures how many "important" branches every shortest path tree contains. For a concise definition we refer to [17]. Our incomparability result just exploits the fact that the maximum degree is a lower bound for the skeleton dimension.

▶ **Lemma 13.** *There are graphs with skeleton dimension k and highway dimension h such that $S_{\max} \in o(k)$ and $S_{\max} \in o(h)$.*

Proof. Take a star graph, subdivide every edge by inserting one vertex and assume unit edge weights. The maximum degree Δ of the graph is linear in the number of nodes n and as Δ is a lower bound for the skeleton dimension k, it follows that $k \in \Omega(n)$. Moreover half of the edges are incident to a leaf node. All these edges are pairwise disjoint and every such edge forms a shortest path of length $1 =: 2r > r$, which intersects the ball of radius $4r = 2$ around the central vertex of the graph. This means that the highway dimension is in $\Omega(n)$.

Consider a contraction order where the highest rank is assigned to the central vertex of the graph and the lowest ranks are assigned to the leaves. Then the maximum search space size is $3 \in o(n)$, which is assumed in the leaves. ◀

Accordingly, search space sizes might be significantly smaller than the discussed road network dimensions. Therefore, the upper bounds derived in dependency of those parameters might be very loose on some graphs. This motivates further research into finding other graph parameters where the possible gap between the average/maximum search space size and the parameter value is in $o(n)$; and to investigate these gaps on real-world networks.

5 Lower Bounds for Strong CH

In this section, we present algorithms to obtain lower bounds for the average search space size of a strong CH in a given weighted graph, e.g., to judge how large the gaps to the road network dimensions are, or to investigate whether a contraction order used in practice produces search space sizes close to the optimum.

5.1 HittingSet Lower Bound

In a strong CH, a node w of rank R is in the search space of a node v with rank $r < R$, if on the shortest path between v and w there is no node with a rank higher than R. (Note that this is a sufficient but not necessary condition.) Accordingly, if we consider the node with maximum rank in the contraction order, we know that it has to be contained in the search space of each node in G. This leads us to the definition of the inverse search space of a node v as $ISS(v) := \{w \in V | v \in SS(w)\}$ which can be used as an alternative mean to determine the average search space $S_{avg} = \frac{1}{n} \sum_{v \in V} |SS(v)| = \frac{1}{n} \sum_{v \in V} |ISS(v)|$. For the node v_{max}

with the highest rank, we know that $|ISS(v_{max})| = n$. Now the goal is to show large inverse search spaces also for other nodes of sufficiently high rank. Note however, that the inverse search space sizes are not necessarily proportional to the rank, as nodes of even higher rank might block many shortest paths. Therefore, we have to take the topology of the graph as well as the shortest path structure into account.

▶ **Observation 14.** *Let G' be the subgraph of G induced by a node set $V' \subseteq V$ and let $v'_{max} \in V'$ be the node with the highest rank in the contraction order among all nodes in V'. Then the number of nodes in V' with their shortest path towards v'_{max} being completely contained in G' is a lower bound for $|ISS(v'_{max})|$.*

To get a lower bound which adheres to all possible contraction orders we need to deal with the fact that we do not know which node in V' will become v'_{max}. To make the bound more general, we can iterate through all nodes $v \in V'$ and compute the number of nodes in V' with the shortest path towards v being completely contained in G', keeping track of the minimum. We call this value the minimum shortest path tree size in G' or $misp(G')$.

▶ **Theorem 15.** *For any $\beta \in (0, 1]$, let $\mathcal{G}$ be a collection of subgraphs of G with $misp(G') \geq \beta n$ for all $G' \in \mathcal{G}$. Furthermore, let H be a minimum HittingSet for the set system $(V, \mathcal{G})$. Then it yields $S_{avg} \geq \beta |H|$.*

Proof. We want to count the inverse search space sizes induced by the nodes of highest rank within each subgraph. To that end, we interpret H as the set of nodes contracted last. We know by definition of $\mathcal{G}$ and minimality of H that for every node $h \in H$, we have $|ISS(h)| \geq \beta n$. Furthermore, by H being a minimum HittingSet, we conclude that there can't be fewer than $|H|$ nodes with that property. Accordingly, we get $S_{avg} = \frac{1}{n} \sum_{v \in V} |ISS(v)| \geq \frac{1}{n} \sum_{h \in H} |ISS(h)| \geq \frac{1}{n} \beta n |H| = \beta |H|$ which completes the proof. ◀

▶ **Example 16.** To illustrate the usefulness of the HittingSet lower bound, we consider a complete graph G with n nodes and metric edge weights, and we choose $\beta = \frac{1}{2}$. We observe that any induced subgraph G' of G of size $\geq \frac{n}{2}$ has $misp(G') \geq \frac{n}{2}$ as well. It follows that we need a HittingSet H of size $\frac{n}{2} + 1$ to hit all these subgraphs. According to Theorem 15, we hence get $S_{avg} \geq \frac{n}{4}$ which matches the upper bound on S_{max} of $n - 1$ asymptotically.

For practical exploitation in real networks, we remark that for every shortest path π of length βn in G, we automatically have $misp(\pi) = \beta n$ and that for any selected subgraph the misp-value can be computed in polytime.

5.2 Hierarchical Lower Bound

In general graphs, the question arises how to choose β in practice to get the best bound. For large β the size of H is expected to be small, while gains in the HittingSet size for small values β might be diminished by the multiplication with β itself. To get a useful lower bound nevertheless, we propose a hierarchical scheme based on the following observation.

▶ **Observation 17.** *Let $\beta_1, \beta_2 \in (0, 1]$ be two parameters with $\beta_1 > \beta_2$ and $\mathcal{G}_1$, H_1 as well as $\mathcal{G}_2, H_2$ the respective subgraph collections and minimum HittingSets. Then we have $S_{avg} \geq \beta_1 |H_1| + \beta_2 (|H_2| - |H_1|)$.*

This observation can be generalized to an arbitrarily fine-grained succession of β-values.

▶ **Corollary 18.** *Given a weighted graph $G(V, E)$, as well as $\beta_1 > \beta_2 > \cdots > \beta_k$ with $\beta_i \in (0, 1]$, let H_i be the respective HittingSet sizes for all subgraphs G' of G with $misp(G') \geq \beta_i n$. Then the average search space size S_{avg} is lower bounded by $\beta_1 |H_1| + \sum_{i=2}^{k} \beta_i (|H_i| - |H_{i-1}|)$.*

▶ **Example 19.** To illustrate that the hierarchical scheme can be beneficial, we consider a path graph with $n = 2^q$ nodes. For any choice of β, we know that a trivial HittingSet of size $\frac{1}{\beta}$ exists. Therefore the bound we get from Theorem 15 only amounts to $S_{avg} = \beta \cdot \frac{1}{\beta} = 1$. If we now use Corollary 18 with $\beta_i = 2^{-i}$ for $i = 0, \ldots, \log_2 n$, we get

$$S_{avg} \geq 1 + \sum_{i=1}^{q} 2^{-i}(2^i - 2^{i-1}) = 1 + \sum_{i=1}^{q} \tfrac{1}{2} = \tfrac{1}{2} \log_2 n$$

which matches the known upper bound up to constant factors.

6 Conclusions and Future Work

We described several novel lower bounding techniques for average and maximum search space sizes in contraction hierarchies. Lower bounds are an important mean to judge the quality of existing construction schemes and theoretical upper bounds. While for hub labeling (a preprocessing-based route planning technique closely related to contraction hierarchies), approximation algorithms for the average number of nodes that have to be scanned in a query were known for some time [10], we proved the first general approximation result for the average search space size in weak contraction hierarchies. As the construction of weak contraction hierarchies is closely related to graph triangulation and to solving systems of linear equations [3], there might be cross-implications to explore.

Future work could also consider weak contraction hierarchies on directed graphs. As the graph parameters used in our analysis are classically defined on undirected graphs only, the established relationships to CH search space sizes do not automatically transfer to directed graphs. One possibility would be to consider parameter variants, as the directed treewidth [16], which are explicitly defined on directed graphs and to check whether approximation results can be obtained in dependency of those parameters.

Another interesting open question is whether there are efficient approximation algorithms for the average or maximum search space size in strong CHs. While upper bounds for search space sizes in weak CHs are valid for strong CHs as well, it is the other way around for lower bounds. Therefore, strong CHs are not covered by our approximation results for weak CHs. Our lower bounds for average search space sizes in strong CHs rely on HittingSet computation, though, as do some strong CH construction schemes which come with provable upper bounds [1]. It hence might be possible to link those results.

Finally, it might be interesting to experimentally investigate the strength of our lower bounds and the efficiency of the proposed algorithms on real-world (road) networks.

References

1 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway dimension and provably efficient shortest path algorithms. Technical Report MSR-TR-2013-91, Microsoft Research, September 2013.

2 Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 782–793, 2010.

3 Ajit Agrawal, Philip Klein, and R Ravi. Cutting down on fill using nested dissection: provably good elimination orderings. In *Graph Theory and Sparse Matrix Computation*, pages 31–55. Springer, 1993.

4 Eyal Amir, Robert Krauthgamer, and Satish Rao. Constant factor approximation of vertex-cuts in planar graphs. In *ACM Symposium on Theory of Computing (STOC)*, pages 90–99, 2003.

5 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

6 Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug, and Dorothea Wagner. Preprocessing speed-up techniques is hard. In *International Conference on Algorithms and Complexity (CIAC)*, pages 359–370. Springer, 2010.

7 Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. *Theoretical Computer Science*, 645:112–127, 2016.

8 Johannes Blum, Stefan Funke, and Sabine Storandt. Sublinear search spaces for shortest path planning in grid and road networks. In *AAAI Conference on Artificial Intelligence*, 2018.

9 Hans L. Bodlaender, John R. Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995. doi:10.1006/jagm.1995.1009.

10 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

11 Tobias Columbus. Search space size in contraction hierarchies. Diploma thesis, Karlsruhe Institute of Technology, 2012.

12 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. In *International Symposium on Experimental Algorithms (SEA)*, pages 271–282, 2014.

13 Hristo Nicolov Djidjev. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):229–240, 1982.

14 Stefan Funke and Sabine Storandt. Provable efficiency of contraction hierarchies with randomized preprocessing. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 479–490, 2015.

15 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012. doi:10.1287/trsc.1110.0401.

16 Thor Johnson, Neil Robertson, Paul D Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

17 Adrian Kosowski and Laurent Viennot. Beyond highway dimension: Small distance labels using tree skeletons. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1462–1478, 2017.

18 Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE MICROSYSTEMS RESEARCH CENTER, 1989.

19 Nikola Milosavljević. On optimal preprocessing for contraction hierarchies. In *ACM SIGSPATIAL International Workshop on Computational Transportation Science (IWCTS)*, pages 33–38. ACM, 2012.

20 Tobias Rupp and Stefan Funke. A lower bound for the query phase of contraction hierarchies and hub labels. In *Computer Science in Russia (CSR)*, 2020.

21 Alejandro A Schäffer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33(2):91–96, 1989.

22 Colin White. Lower bounds in the preprocessing and query phases of routing algorithms. In *Annual European Symposium on Algorithms (ESA)*, pages 1013–1024, 2015.

The Minimization of Random Hypergraphs

Thomas Bläsius
Hasso Plattner Institute, University of Potsdam, Germany

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Germany

Martin Schirneck[1]
Hasso Plattner Institute, University of Potsdam, Germany

Abstract

We investigate the maximum-entropy model $\mathcal{B}_{n,m,p}$ for random n-vertex, m-edge multi-hypergraphs with expected edge size pn. We show that the expected size of the minimization $\min(\mathcal{B}_{n,m,p})$, i.e., the number of inclusion-wise minimal edges of $\mathcal{B}_{n,m,p}$, undergoes a phase transition with respect to m. If m is at most $1/(1-p)^{(1-p)n}$, then $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|]$ is of order $\Theta(m)$, while for $m \geq 1/(1-p)^{(1-p+\varepsilon)n}$ for any $\varepsilon > 0$, it is $\Theta(2^{(\mathrm{H}(\alpha)+(1-\alpha)\log_2 p)\,n}/\sqrt{n})$. Here, H denotes the binary entropy function and $\alpha = -(\log_{1-p} m)/n$. The result implies that the maximum expected number of minimal edges over all m is $\Theta((1+p)^n/\sqrt{n})$. Our structural findings have algorithmic implications for minimizing an input hypergraph. This has applications in the profiling of relational databases as well as for the Orthogonal Vectors problem studied in fine-grained complexity. We make several technical contributions that are of independent interest in probability. First, we improve the Chernoff–Hoeffding theorem on the tail of the binomial distribution. In detail, we show that for a binomial variable $Y \sim \mathrm{Bin}(n,p)$ and any $0 < x < p$, it holds that $\mathrm{P}[Y \leq xn] = \Theta(2^{-\mathrm{D}(x\,\|\,p)\,n}/\sqrt{n})$, where D is the binary Kullback–Leibler divergence between Bernoulli distributions. We give explicit upper and lower bounds on the constants hidden in the big-O notation that hold for all n. Secondly, we establish the fact that the probability of a set of cardinality i being minimal after m i.i.d. maximum-entropy trials exhibits a sharp threshold behavior at $i^* = n + \log_{1-p} m$.

2012 ACM Subject Classification Mathematics of computing → Information theory; Mathematics of computing → Hypergraphs; Theory of computation → Random network models; Mathematics of computing → Random graphs

Keywords and phrases Chernoff–Hoeffding theorem, maximum entropy, maximization, minimization, phase transition, random hypergraphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.21

Related Version A full version of the paper is available at `https://arxiv.org/abs/1910.00308`.

Acknowledgements The authors thank Benjamin Doerr, Timo Kötzing, and Martin Krejca for the fruitful discussions on the Chernoff–Hoeffding theorem, including valuable pointers to the literature.

1 Introduction

A plethora of work has been dedicated to the analysis of random graphs. Random hypergraphs, however, received much less attention. For many types of data, hypergraphs provide a much more natural model. This is especially true if the data has a hierarchical structure or reflects interactions between groups of entities. In non-uniform hypergraphs, where edges can have different numbers of vertices, a phenomenon occurs that is unknown to graphs: an edge may be contained in another, with multiple edges even forming chains of inclusion. We are often only interested in the endpoints of those chains, namely, the collections of inclusion-wise minimal or maximal edges. This is the *minimization* or *maximization* of the hypergraph.

[1] Corresponding author: `martin.schirneck@hpi.de`.

We investigate the maximum-entropy model $\mathcal{B}_{n,m,p}$ for random multi-hypergraphs with n vertices and m edges and expected edge size pn for some constant sampling probability p. In other words, out of all probability distributions on hypergraphs with expected edge size pn, $\mathcal{B}_{n,m,p}$ is the one of maximum entropy.[2] This is equivalent to sampling m independent edges by adding any vertex $v \in [n]$ independently with probability p (see Section 2 for details). We are interested in the expected size of the minimization/maximization of $\mathcal{B}_{n,m,p}$, that is, the expected number of minimal/maximal edges. Most of our results are phrased in terms of the minimization, but replacing the probability p with $1-p$ immediately transfers them to the maximization. We show that the size of the minimization undergoes a phase transition with respect to m with the point of transition at $m = 1/(1-p)^{(1-p)n}$. While the number of edges is still small, a constant fraction of them is minimal and the minimization grows linearly in the total sample sizes. For m beyond the transition, we can instead characterize the size of the minimization in terms of the entropy function of $\log_{1-p} m$, see Theorem 1.2 for a precise statement. This characterization shows that the minimality ratio goes down dramatically when m increases. It also allows us to prove that the maximum expected number of minimal edges over all m is of order $\Theta((1+p)^n/\sqrt{n})$. These results draw from another, more hidden, threshold behavior. The probability of a set to be minimal in the hypergraph $\mathcal{B}_{n,m,p}$ depends only on its cardinality i and we show that this probability falls sharply from almost 1 to almost 0 at $i^* = n + \log_{1-p} m$.

The main tool in our analysis is the Chernoff–Hoeffding theorem bounding the tail of the binomial distribution via the Kullback–Leibler divergence from information theory. However, the existing inequalities are not sharp enough to derive tight statements on the expected size of the minimization. So far, there is a gap of order $\sqrt{n}$ between the best-known upper and lower estimates. In this work, we improve these bounds such that they match up to constant factors. We give an explicit interval for the constants involved that holds for all positive integers n making the result useful also in a non-asymptotic setting.

Our structural findings have algorithmic implications for the computation of the minimization $\min(\mathcal{H})$ from an input hypergraph $\mathcal{H}$. We discuss two examples in the context of fine-grained complexity as well as data profiling. There are reasons to believe that there exists no minimization algorithm running in time $m^{2-\varepsilon} \cdot \mathrm{poly}(n)$ for any $\varepsilon > 0$ on m-edge, n-vertex hypergraphs. The reason is as follows: The Sperner Family problem is to decide whether $\mathcal{H}$ comprises two edges such that one is contained in the other, i.e., whether $|\min(\mathcal{H})| < |\mathcal{H}|$. It is equivalent under subquadratic reductions to the more prominent Orthogonal Vectors problem [14, 25]. Hence, a truly subquadratic algorithm would falsify the Orthogonal Vectors Conjecture[3] and in turn the Strong Exponential Time Hypothesis [52]. Partitioning the edges by the number of vertices and processing them in order of increasing cardinality gives an algorithm running in $\mathrm{O}(mn\,|\min(\mathcal{H})| + mn)$, which is $\mathrm{O}(m^2 n)$ in the worst case. However, when looking at the average-case complexity for $\mathcal{B}_{n,m,p}$, we get a run time of $\mathrm{O}(mn\,\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] + mn)$. Our main result therefore shows that the algorithm is subquadratic for all m beyond the phase transition, and even linear for $m \geq 1/(1-p)^n$.

There is also a connection to the profiling of relational databases. Data scientists regularly need to compile and output a comprehensive list of metadata, like unique column combinations, functional dependencies, or, more general, denial constraints, cp. [1]. These multi-column dependencies can all be described as the minimal hitting sets of certain hypergraphs created from comparing pairs of rows in the database and recording the sets

[2] The notation $\mathcal{B}_{n,m,p}$ is mnemonic of the binomial distribution emerging in the sampling process.
[3] Precisely, we mean the Orthogonal Vectors Conjecture for moderate dimensions, see [25].

of attributes in which they differ [24, 11, 10, 40]. Computing these difference sets one by one generates an incoming stream of seemingly random subsets. Filtering the inclusion-wise minimal ones from this stream does not affect the solution, but reduces the number of sets to store and the complexity of the resulting hitting set instance. Minimizing the input is therefore a standard preprocessing technique in data profiling. It has been observed in practice that the minimal sets make up only a small fraction of the whole input [45]. Usually there are fewer minimal difference sets than rows in the database, let alone pairs thereof [11]. The upper bounds given in the Theorems 1 and 2 provide a way to explain this phenomenon. We show that only a few edges can be expected to be minimal, their number may even shrink as the database grows, provided that the number of rows is large enough compared to the number of columns. The respective lower bounds can further be seen as the smallest amount of data any dependency enumeration algorithm needs to hold in memory.

Related Work. Erdős–Rényi graphs $\mathcal{G}_{n,m}$ [23] and Gilbert graphs $\mathcal{G}_{n,p}$ [27] are arguably the most-discussed random graph models in the literature. We refer the reader to the monograph by Bollobás [12] for an overview. A majority of the work on these models concentrates on various phase transitions with respect to the number of edges m or the sample probability p, respectively. This intensive treatment is fueled by the appealing property that Erdős–Rényi graphs are "maximally random" in that they do not assume anything but the number of vertices and edges. More formally, among all probability distributions on graphs with n vertices and m edges, $\mathcal{G}_{n,m}$ is the unique distribution of maximum entropy. The same holds for $\mathcal{G}_{n,p}$ under the constraint that the expected number of edges is $p\binom{n}{2}$, see [2].

The intuition of being maximally random is captured by the Shannon entropy, which is the central concept in information theory [18, 50]. A discrete stochastic system that can be described by the probability distribution $(p_i)_i$ has a (binary) *entropy* of $H((p_i)_i) = -\sum_i p_i \log_2 p_i$. The *self-information* of a single state with probability p is $-\log_2 p$, the entropy is thus the expected information of the whole system. It is a measure of surprisal or how "spread out" the distribution is. Originally stemming from thermodynamics [39], the versatility of this definition is key to the successful application of information theory to fields as diverse as cryptography [15], machine learning [28], quantum computing [44], and of course network analysis [43], to name only a few topics close to computer science. The *principle of maximum entropy* states that out of an ensemble of probability distributions that all describe the observed phenomena equally well, the one of maximum entropy is to be preferred in order to minimize any outside bias. The principle is usually attributed to Jaynes [37, 32, 33]. In the context of random graphs, it is mainly used to define so-called *null models* [53]. One fixes certain graph statistics to mimic those of an observed network and then chooses the maximum-entropy distribution that meets these constraints. By comparing the original network with a "typical graph" drawn from the null model, one can infer whether other observed properties are correlated with the constraints. This method was made rigorous by Park and Newman [46] building on earlier work in general statistics. Prescribing the exact or expected number of edges leads to the $\mathcal{G}_{n,m}$ or $\mathcal{G}_{n,p}$ distributions, respectively. The configuration model fixes the whole degree sequence [13], and in the soft configuration model the degrees hold at least in expectation [8, 26].

Many early attempts to transfer the concept of null models to hypergraphs were only indirect in that they studied hypergraphs via their clique-expansion [42] or as bipartite graphs [48]. This is unsatisfactory since these projections alter relevant observables, like node degrees or the number of triangles. Only recently, Chodrow generalized the configuration model directly to multi-hypergraphs [16]. There also seems to be not much literature on

hypergraph models that can be cast into the maximum-entropy framework without being intentionally designed as such. A notable early exception is the work by Schmidt-Pruzan and Shamir [49]. They fixed the exact/expected edge sequence such that the largest edge has cardinality $\mathrm{O}(\log n)$ and showed a "double jump" phase transition in the size of the largest connected component. Most of the recent literature on random hypergraphs concentrates on the k-uniform model where every edge has exactly k vertices [34, 5, 6] or, equivalently, on random binary matrices with k 1s per column [17]. In our model, we do not prescribe the exact cardinalities of the edges and neither do we bound their maximum size, instead we only require that the *expected* edge size is pn.

Probably closest to our work is a string of articles by Demetrovics et al. [20] as well as Katona [35, 36]. They investigated random databases and connected the Rényi entropy of *order 2* of the logarithmic number of rows with the probability that certain unique column combinations or functional dependencies hold. In contrast, we connect the Shannon entropy of the logarithmic number of *pairs* of rows with the expected number of minimal difference sets. Unique column combinations and functional dependencies are dual to the difference sets of record pairs, one are the minimal hitting sets of the other [1, 9]. Also, the Shannon entropy is the same as the Rényi entropy of *order 1* [18]. In this sense, we complement the result by Demetrovics et al. by showing that the duality also pertains to the order of entropy.

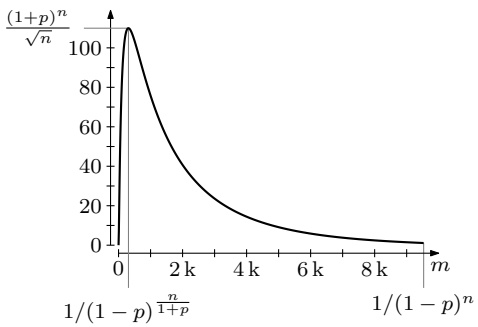
The analysis of random (hyper-)graphs naturally uses tools from combinatorics and probability theory. Conversely, it has always helped to advance the fields by sharpening those tools [7, 12, 31]. In this work, we improve the bounds of the Chernoff–Hoeffding theorem [30] on the tail of the binomial distribution. We use an observation by Klar [38] on the relation between the distribution function and the probability mass function. There were some refined inequalities known before. By Cramér's theorem [19], Chernoff–Hoeffding is asymptotically tight up to subexponential factors. The gap was subsequently reduced to order $\mathrm{O}(\sqrt{n})$, cp. [3], we close it down to a constant. There also exist some comparatively tight bounds based on the normal limit of the binomial distribution, contributions by Prokhorov [47] and Slud [51] founded major lines of research. However, we avoid this approach since the normal approximation cannot be expressed in terms of elementary functions. Also, it tends to place unnecessary restrictions on the success probability p when deriving non-asymptotic results.

Outline. Next, we introduce the hypergraph model and state our results in full detail. We review some notation in Section 3. Section 4 is dedicated to the Chernoff–Hoeffding theorem. Section 5 adds further technical contributions, including the sharp threshold of minimal sets at a certain cardinality. The main theorem is proven in Section 6. Section 7 discusses the phase transition and concludes the work.

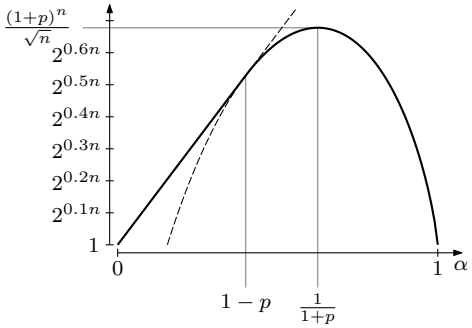
2 Model and Main Theorem

Fix a probability p and positive integers n and m. The random multi-hypergraph $\mathcal{B}_{n,m,p}$ is defined by independently sampling m (not necessarily distinct) subsets of $[n]$. Each set is generated by including a vertex $v \in [n]$ with probability p independently of all other choices.

We quickly argue that this is indeed the maximum-entropy model. Besides the size of the universe n and the number of edges m, the only other constraint is the expected edge size pn. The independence bound on the entropy reads as follows: Let X_1 to X_m be random variables with joint distribution $\mathrm{P}_{X_1,\dots,X_m}$ and marginal distributions P_{X_j}. Then, their entropies observe the inequality $H(\mathrm{P}_{X_1,\dots,X_m}) \leq \sum_{j=1}^{m} H(\mathrm{P}_{X_j})$, equality holds if and only if the X_j are independent, see [18]. This suggests that we should choose the edges independently if we



(a) Our bound as a function of m for $n = 10$ and $p = 0.6$ in the information-theoretic regime. The vertical line at $m = 1/(1-p)^{\frac{n}{1+p}}$ indicates the maximum (Theorem 2). For $m > 1/(1-p)^n$, the size goes to 1. The linear bound for $m \leq 1/(1-p)^{(1-p)n}$ is not shown as it is too close to 0.



(b) Our bound as a function of α for $p = 0.6$ (the plot is independent of n). The vertical line at $\alpha = 1/(1+p)$ indicates the maximum (Theorem 2). For $\alpha \leq 1-p$, the linear bound holds, for larger α, we get the information-theoretic bound. They are continued as dashed lines into the other regime.

Figure 1 Illustration of Theorem 1 showing the expected size of the minimization of a random hypergraph depending on the number of edges m (a) and on α (b). As α grows logarithmically in m, (b) shows the same plot as (a) but with both axes being logarithmic.

want to maximize the entropy and the same is true for the vertices inside an edge. Finally, the fact that setting the sampling probability to be equal for all vertices indeed maximizes the entropy under a given mean set size was proven by Harremoës [29].

We are interested in the expected number of inclusion-wise minimal sets in $\mathcal{B}_{n,m,p}$, denoted by $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|]$. We describe the asymptotic behavior of this expectation with respect to n. In more detail, we view $m = m(n)$ as a function of n assuming integer values and bound the univariate asymptotics of $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|]$ in n for different choices of m. The probability p is considered to be a constant throughout. We show that the size of the minimization can be described precisely in terms of p and the Shannon entropy of the logarithm of m.

We let $\mathrm{H}(x) = H((x, 1-x))$ denote the binary entropy function and define the quantity $\alpha = \log_{\frac{1}{(1-p)^n}} m = -(\log_{1-p} m)/n$. The quantity α is well-defined for all $0 < p < 1$ and $n, m \geq 1$. It is always non-negative and asymptotically of order $\Theta((\log m)/n)$. If p and n are fixed, choosing a value for α determines m since we can rewrite m as $1/(1-p)^{\alpha n}$.

▶ **Theorem 1.** *Let p be a probability, and n, m be two positive integers. If $p = 0$ or $p = 1$, then $|\min(\mathcal{B}_{n,m,p})| = 1$ holds deterministically. For $0 < p < 1$, the following statements hold.*

1. *If $m \leq 1/(1-p)^{(1-p)n}$, then $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] = \Theta(m)$.*
2. *For any two $\varepsilon, \varepsilon' > 0$ and all m such that $1/(1-p)^{(1-p+\varepsilon)n} \leq m \leq 1/(1-p)^{(1-\varepsilon')n}$, i.e., all α such that $1 - p + \varepsilon \leq \alpha \leq 1 - \varepsilon'$, we have*

$$\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] = \Theta\Big(2^{(\mathrm{H}(\alpha)+(1-\alpha)\log_2 p)\,n}\Big/\sqrt{n}\Big) = \Theta\bigg(\bigg(\frac{p^{1-\alpha}}{(1-\alpha)^{1-\alpha}\ \alpha^{\alpha}}\bigg)^n\Big/\sqrt{n}\bigg);$$

3. *If $m = 1/(1-p)^{n+\omega(\log n)}$, then $1 \leq \mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] = 1 + \mathrm{o}(1)$.*

The bounds in the distinct cases are very different in nature. They are visualized in Figure 1 showing the expectation both as a function of the number of trials m and of α. To distinguish the behavior also in writing, we use the term *linear regime* if m is between 1 and $1/(1-p)^{(1-p)n}$, corresponding to $0 \leq \alpha \leq 1-p$, likewise, we refer to m being between $1/(1-p)^{(1-p)n}$ and $1/(1-p)^n$, i.e., $1-p \leq \alpha \leq 1$, as the *information-theoretic regime*.

All asymptotic estimates in Theorem 1 are at least tight up to constants, the third statement is even tight up to lower-order terms. The constants hidden in the big-O-notation are universal in the sense that they do not depend on m or n, and also not on α describing the relation between the former two. However, they may depend on the probability p and, in case of Statement 2, on the particular choices for ε and ε'. We note that the bounds for the information-theoretic regime have two gaps at $m = 1/(1-p)^{(1-p)n}$ and $m = 1/(1-p)^n$. These gaps can be made arbitrarily small: Let $c = 1/(1-p)$, then Statement 2 holds if $m \leq (c-\gamma)^n$ for any constant $\gamma > 0$ and Statement 3 takes over at $m \geq (c+\delta(n))^n$, where $\delta(n)$ is a function converging to 0 as n increases.

From the main theorem, we derive bounds on the maximum expectation over all m.

▶ **Theorem 2.** *If $p = 0$ or $p = 1$, then $\max_{m\geq 1} |\min(\mathcal{B}_{n,m,p})| = 1$. For $0 < p < 1$, we have $\max_{m\geq 1} \mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] = \Theta((1+p)^n/\sqrt{n})$, attained at $m = 1/(1-p)^{\frac{n}{1+p}}$.*

3 Preliminaries and Notation

Multi-Hypergraphs. A *hypergraph* on $[n] = \{1, \dots, n\}$ is a set of subsets $\mathcal{H} \subseteq \mathcal{P}([n])$, called the *(hyper-)edges*. If $\mathcal{H}$ is a multiset instead, we have a *multi-hypergraph*. We do not allow multiple copies of the same vertex in one edge. The *minimization* of a hypergraph $\mathcal{H}$ is the collection of its inclusion-wise minimal edges, $\min(\mathcal{H}) = \{E \in \mathcal{H} \mid \forall E' \in \mathcal{H} \colon E' \subseteq E \Rightarrow E' = E\}$. We extend this notion to multi-hypergraphs by requiring that whenever a minimal edge has multiple copies, only one of them is included in the minimization. This way $\min(\mathcal{H})$ is always a mere hypergraph (a set). For a multi-hypergraph $\mathcal{H}$, we use $|\mathcal{H}|$ to denote the total number of edges counting multiplicities, and $\|\mathcal{H}\|$ for the number of *distinct* edges, i.e., the cardinality of the support of $\mathcal{H}$. Evidently, we have $|\min(\mathcal{H})| \leq \|\mathcal{H}\| \leq |\mathcal{H}|$.

Information Theory. We intend the expressions $0 \cdot \log_a 0$ and $0 \cdot \log_a(\frac{0}{0})$ to all mean 0 for any positive real base $a > 0$. Note that this convention also implies $0^0 = a^{0 \log_a 0} = 1$ and $(\frac{0}{0})^0 = 1$. We use $\operatorname{ld} x$ for the binary (base-2) logarithm of x. The (binary) *entropy function* H is defined for all probabilities x as $\mathrm{H}(x) = -x \operatorname{ld} x - (1-x)\operatorname{ld}(1-x)$. It describes the Shannon entropy or, equivalently, the Rényi entropy of order 1, of the Bernoulli distribution with parameter x. In the notation of the previous sections, $\mathrm{H}(x) = H((x, 1-x))$. Evidently, the entropy function is symmetric around $1/2$ with $\mathrm{H}(x) = \mathrm{H}(1-x)$. On the open unit interval, H is positive and differentiable with derivative $\frac{\mathrm{d}}{\mathrm{d}x}\mathrm{H}(x) = \operatorname{ld}\left(\frac{1-x}{x}\right)$. This is the negative (binary) *logit function*, also dubbed *log-odds* in statistics. H is strictly concave and has its maximum at $1/2$ with value $\mathrm{H}(1/2) = 1$. The *perplexity* of x is $2^{\mathrm{H}(x)} = 1/(x^x(1-x)^{1-x})$. We utilize it to estimate binomial coefficients. The bounds are well-known in the literature [18].

▶ **Lemma 3.** *Let n be a positive integer and $0 < x < 1$ such that xn is an integer, then $2^{\mathrm{H}(x)n}/\sqrt{8nx(1-x)} \leq \binom{n}{xn} \leq 2^{\mathrm{H}(x)n}/\sqrt{\pi nx(1-x)}$.*

Let $(p_i)_i$ and $(q_i)_i$ be two distributions on the same state space such that $(p_i)_i$ is absolutely continuous with respect to $(q_i)_i$, i.e., $q_i = 0$ implies $p_i = 0$ for all i. The (binary) *Kullback–Leibler divergence*[4] from $(q_i)_i$ to $(p_i)_i$ is given by $D((p_i)_i \,\|\, (q_i)_i) = -\sum_i p_i \operatorname{ld}(\frac{q_i}{p_i})$. It is the expected information loss when assuming that the distribution is $(q_i)_i$ while the system actually follows $(p_i)_i$. The divergence is a premetric in that it is non-negative and 0 iff the distributions are the same. However, it is neither symmetric nor does it observe the triangle

[4] The divergence is sometimes also called *relative entropy*, we avoid this term due to ambiguities, cf. [18].

inequality. In this work, we only need the divergence between Bernoulli distributions. For any two probabilities x, y, the divergence between two Bernoulli distributions with respective parameters x and y is $\mathrm{D}(x \,\|\, y) = D((x,1-x) \,\|\, (y,1-y)) = -x \operatorname{ld}\left(\frac{y}{x}\right) - (1-x) \operatorname{ld}\left(\frac{1-y}{1-x}\right)$. The function $\mathrm{D}(x \,\|\, y)$ is convex in both x and y, attains its minimum 0 for $x = y$, and observes $\mathrm{D}(x \,\|\, y) = \mathrm{D}(1-x \,\|\, 1-y)$. We often use the derived quantity $2^{-\mathrm{D}(x \,\|\, y)} = \left(\frac{y}{x}\right)^x \left(\frac{1-y}{1-x}\right)^{1-x}$.

Polynomials of Probabilities.

▶ **Lemma 4.** *Let n be a non-negative integer and x a probability, then it holds that $e^{-nx}\left(1-nx^2\right) \leq (1-x)^n \leq e^{-nx}$.*

▶ **Lemma 5** (Lemma 10 in [4])**.** *Let n be a non-negative integer and x a probability, then $nx/(1+nx) \leq 1-(1-x)^n \leq nx$.*

▶ **Lemma 6.** *Consider a random experiment with outcomes A, B, and C, where $\mathrm{P}[B] > 0$. In a series of m i.i.d. trials, let A_j denote the event that the outcome of the j-th trial is A, same with B. Then, we have $\mathrm{P}[\forall j \leq m\colon \neg A_j \mid \exists k \leq m\colon B_k] \leq \mathrm{P}[\forall j \leq m\colon \neg A_j \mid B_m]$.*

4 The Chernoff–Hoeffding Theorem

In this section, we tighten the Chernoff–Hoeffding theorem bounding the tail of the binomial distribution. The result will later help us with the random hypergraphs, but more importantly it provides a powerful tool of general interest in probability theory. Fix a positive integer n and probabilities x and p. Recall that the Kullback–Leibler divergence between the respective Bernoulli distributions is $\mathrm{D}(x \,\|\, p) = -x \operatorname{ld}\left(\frac{p}{x}\right) - (1-x) \operatorname{ld}\left(\frac{1-p}{1-x}\right)$. The Chernoff–Hoeffding theorem [30, 22] employs the divergence to bound the probability that a binomially distributed random variable $Y \sim \mathrm{Bin}(n,p)$ deviates from its expected value $\mathrm{E}[Y] = pn$. If $x \leq p$, then $\mathrm{P}[Y \leq xn] \leq 2^{-\mathrm{D}(x \,\|\, p)\, n} = \left(\frac{p}{x}\right)^{xn} \left(\frac{1-p}{1-x}\right)^{(1-x)n}$. Similarly, if $p \leq x$, we have $\mathrm{P}[Y \geq xn] \leq 2^{-\mathrm{D}(x \,\|\, p)\, n}$. Several weaker but more practical inequalities have been inferred from this, summarized as *Chernoff bounds* [41, 21]. We sharpen these inequalities by a $\sqrt{n}$-factor for all but the extreme values of x. While the upper bound of Chernoff–Hoeffding holds for all probabilities x, there are some lower bounds known for $\mathrm{P}[Y \leq xn]$ if the product xn is an integer, c.f. the textbook by Ash [3, Lemma 4.7.2]. We use a proposition by Klar [38] to improve the upper bound such that it matches the lower one up to constants. We then extend both bounds to the general case of arbitrary products xn.

▶ **Theorem 7.** *Let n be a positive integer, x and p two probabilities with $0 < p < 1$, and $Y \sim \mathrm{Bin}(n,p)$ a binomial random variable.*

1. *If $1/n \leq x < p$, then $\frac{(1-p)\sqrt{x}}{2e\sqrt{2\,(1-x)}} \cdot \frac{2^{-\mathrm{D}(x \,\|\, p)\, n}}{\sqrt{n}} \leq \mathrm{P}[Y \leq xn] \leq \frac{\sqrt{1-x}}{(p-x)\sqrt{\pi\, x}} \cdot \frac{2^{-\mathrm{D}(x \,\|\, p)\, n}}{\sqrt{n}}$.*
2. *If $p < x \leq 1 - 1/n$, then $\frac{p\sqrt{1-x}}{2e\sqrt{2\,x}} \cdot \frac{2^{-\mathrm{D}(x \,\|\, p)\, n}}{\sqrt{n}} \leq \mathrm{P}[Y \geq xn] \leq \frac{\sqrt{x}}{(x-p)\sqrt{\pi\,(1-x)}} \cdot \frac{2^{-\mathrm{D}(x \,\|\, p)\, n}}{\sqrt{n}}$.*

Proof sketch. The second statement of the theorem is implied by the first one by applying it to the complementary variable $\overline{Y} \sim \mathrm{Bin}(n, 1-p)$. Let $x \leq p$ be a probability. We mainly confine ourselves here to the case that the product xn is integral and show that then we get $\frac{1}{\sqrt{8\,x(1-x)}} \cdot \frac{2^{-\mathrm{D}(x \,\|\, p)\, n}}{\sqrt{n}} \leq \mathrm{P}[Y \leq xn] \leq \frac{p\sqrt{1-x}}{(p-x)\sqrt{\pi\, x}} \cdot \frac{2^{-\mathrm{D}(x \,\|\, p)\, n}}{\sqrt{n}}$.

Lemma 3 provides the following error bounds for the probability mass function of Y: $1/\sqrt{8nx(1-x)} \leq \mathrm{P}[Y = xn]/(2^{\mathrm{H}(x)n} \cdot p^{xn}(1-p)^{(1-x)n}) \leq 1/\sqrt{\pi nx(1-x)}$. We further have $2^{\mathrm{H}(x)n} \cdot p^{xn}(1-p)^{(1-x)n} = 2^{-\mathrm{D}(x\,\|\,p)\,n}$. This proves the first part that $\mathrm{P}[Y \leq xn] \geq \mathrm{P}[Y = xn] \geq 2^{-\mathrm{D}(x\,\|\,p)\,n}/\sqrt{8\,n\,x(1-x)}$ holds.

A result by Klar [38, Proposition 1(c)] states that the ratio $\mathrm{P}[Y \leq xn]/\mathrm{P}[Y = xn]$ is at most $f_{n,xn}(p) = p\,(1 - \frac{xn}{n+1})/(p - \frac{xn}{n+1})$. The partial discrete derivative of $f_{n,xn}$ with respect to n, that is, $\Delta_n(f_{n,xn})(p) = f_{n+1,x(n+1)}(p) - f_{n,xn}(p)$ can be shown to be positive whenever $x < p$. Thus $f_{n,xn}(p)$ converges from below to $p(1-x)/(p-x)$ as n increases. Combined with the error bounds this is $\mathrm{P}[Y \leq xn] \leq \frac{p\,(1-x)}{p-x} \cdot \frac{1}{\sqrt{\pi\,n\,x(1-x)}} \cdot 2^{-\mathrm{D}(x\,\|\,p)\,n} = \frac{p\sqrt{1-x}}{(p-x)\sqrt{\pi\,x}} \cdot \frac{2^{-\mathrm{D}(x\,\|\,p)\,n}}{\sqrt{n}}$.

Transferring the improvements also to non-integral products xn is not straightforward. A careful analysis of the monotonicity of the entropy function H as well as that of the divergence D reveals that this transition weakens the upper bound only by a additional factor of $1/p$ and the lower bound by $x\,(1-p)/e$, independently of n. ◀

We showed that for all values x strictly between 0 and p, the Chernoff–Hoeffding theorem can be asymptotically improved to $\mathrm{P}[Y \leq xn] = \Theta(2^{-\mathrm{D}(x\,\|\,p)\,n}/\sqrt{n})$. However, the constants hidden in the big-O notation diverge at the boundaries. This caveat cannot be healed, there is no way to extend the improvement also to $x = 0$ or p. Simply put, the original formulation of the theorem is tight. First, pn is not only the mean but also the median of the binomial distribution, whence $\mathrm{P}[Y \leq pn] \geq 1/2$ is constant and *not* of order $\mathrm{O}(2^{-\mathrm{D}(p\,\|\,p)\,n}/\sqrt{n}) = \mathrm{O}(1/\sqrt{n})$. Secondly, the initial bound $\mathrm{P}[Y \leq 0] = (1-p)^n = 2^{-\mathrm{D}(0\,\|\,p)\,n}$ even is exact.

5 Distinct Sets and Minimality

We now return to the main topic of this work, which is determining the expected size of the minimization $\min(\mathcal{B}_{n,m,p})$ of the maximum-entropy multi-hypergraph $\mathcal{B}_{n,m,p}$. The sampling probabilities $p = 0$ or $p = 1$ are trivial, we thus assume $0 < p < 1$ in this work unless explicitly stated otherwise. Every subset of $[n]$ then has a non-vanishing chance to be sampled. Such a set is minimal for $\mathcal{B}_{n,m,p}$ iff it is generated in one of the trials and no proper subset ever occurs. Both of these aspects influence the chance of minimality, but their impact varies depending on the cardinality of the set in question. The number of vertices per edge is heavily concentrated around pn and the more vertices there are in an edge, the less likely it is minimal. Intuitively, almost no sets with very low cardinalities are sampled, but if so, they are often included in $\min(\mathcal{B}_{n,m,p})$. There are plenty of edges with a medium number of vertices and there is a good chance they are minimal. Finally, sets of very high cardinality rarely occur and usually they are then dominated by smaller ones. This disparity is exacerbated by a large number of trials. Boosting m increases the probability that also sets of cardinality a bit further away from pn are sampled, at the same time the process generates more duplicates of sets that occurred before. More importantly though, the likelihood of a larger set being minimal is even smaller with many trials. Eventually, the last effect outweighs all others, creating a situation in which the only minimal edge is empty.

We start making this intuition rigorous by giving preliminary bounds on the number of minimal edges as a first step towards the proof of Theorem 1. The results are binomial sums of polynomials of probabilities, depending on which factors we choose, we get an upper or a lower bound. The estimates are already tight up to constants but are rather unwieldy. They will serve as the basis for our further analysis. Let $\mathcal{D}_{n,p}$ denote the maximum-entropy distribution on the power set $\mathcal{P}([n])$ provided that $\mathrm{E}_{X\sim\mathcal{D}_{n,p}}[|X|] = pn$, meaning each vertex is included independently with probability p.

▶ **Lemma 8.** *Let $0 < p < 1$ be a probability, n, m positive integers, and let $X_j \sim \mathcal{D}_{n,p}$ denote the outcome of the j-th independent trial. For any integer i with $0 \leq i \leq n$, define $s_{n,p}(i,m) = \mathrm{P}[\exists j \leq m \colon X_j = [i]]$ and $w_{n,p}(i,m) = \mathrm{P}[\forall j \leq m \colon \neg(X_j \subsetneq [i])]$ to be the respective probabilities[5] that some trial produces the set $[i]$ and no trial produces a proper subset of $[i]$. Then, we have $s_{n,p}(i,m) = 1 - (1 - p^i(1-p)^{n-i})^m$ and $w_{n,p}(i,m) = \left(1 - (1-p)^{n-i}(1-p^i)\right)^m$. Furthermore, the following statements hold.*

1. $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] \geq \sum_{i=0}^{n} \binom{n}{i} s_{n,p}(i,m) \cdot w_{n,p}(i,m)$.
2. $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] \leq \sum_{i=0}^{n} \binom{n}{i} s_{n,p}(i,m) \cdot w_{n,p}(i,m-1)$.
3. $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] \leq 1 + \frac{1}{p} \sum_{i=0}^{n} \binom{n}{i} s_{n,p}(i,m) \cdot w_{n,p}(i,m)$.

Proof sketch. The formula for $w_{n,p}(i,m) = \mathrm{P}[\forall j \leq m \colon \neg(X_j \subsetneq [i])]$ can be seen as follows. The random set $X_j \sim \mathcal{D}_{n,p}$ is a subset of $[i]$ iff it does not contain an element of $[n] \setminus [i]$, which happens with probability $(1-p)^{n-i}$. Conditioned on being any subset, X_j is a *proper* subset if it is missing at least one element of $[i]$, having conditional probability $1 - p^i$.

Regarding the main statements, a set $S \subseteq [n]$ is in $\min(\mathcal{B}_{n,m,p})$ iff it is sampled in one of the m trials and no proper subset is sampled. The probability for both events depends only on the cardinality $|S|$: $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] = \sum_{S \subseteq [n]} \mathrm{P}[\exists k \leq m \colon X_k = S \wedge \forall j \leq m \colon \neg(X_j \subsetneq S)] = \sum_{i=0}^{n} \binom{n}{i} \cdot \mathrm{P}[\exists k \leq m \colon X_k = [i]] \cdot \mathrm{P}[\forall j \leq m \colon \neg(X_j \subsetneq [i]) \mid \exists k \leq m \colon X_k = [i]]$. Generating any *other* set than $[i]$ in a single trial has probability $1 - p^i(1-p)^{n-i}$, over the independent trials we thus get $s_{n,p}(i,m) = \mathrm{P}[\exists j \leq m \colon X_j = [i]] = 1 - (1 - p^i(1-p)^{n-i})^m$

The last factor $\mathrm{P}[\forall j \leq m \colon \neg(X_j \subsetneq [i]) \mid \exists k \leq m \colon X_k = [i]]$ in each term describes the likelihood that the set $[i]$ is minimal, conditioned on it being sampled at all. Conditioning on at least one trial producing $[i]$ itself only increases the chances of never sampling a proper subset, which gives Statement 1. To prove Statement 2, we apply Lemma 6. Statement 3 follows from the ratio between $w_{n,p}(i,m)$ and $w_{n,p}(i,m-1)$ being the probability that a non-subset of $[i]$ is sampled in a single trial. ◀

The part that all three bounds of Lemma 8 have in common describes the expected number of *distinct* sets in $\mathcal{B}_{n,m,p}$. Recall that we use $\|\mathcal{H}\|$ to denote the number of distinct sets of some multi-hypergraph $\mathcal{H}$. That means, we have $\mathrm{E}[\|\mathcal{B}_{n,m,p}\|] = \sum_{i=0}^{n} \binom{n}{i} s_{n,p}(i,m)$. We weighted the terms of this sum by $w_{n,p}(i,m)$ or $w_{n,p}(i,m-1)$, respectively. In the following, we analyze the two parts separately, starting with the *weighting factors* $w_{n,p}$. They are of interest beyond their application to random multi-hypergraphs. Consider m trials according to the maximum-entropy distribution $\mathcal{D}_{n,p}$ on subsets of $[n]$ with expected set size pn. The quantity $w_{n,p}(i,m)$ is, by definition, the probability that any fixed subset of cardinality i survives as minimal after all trials. Equivalently, $1 - w_{n,p}(i,m)$ is the probability of any proper subset being sampled. We prove next that the weighting factors are in fact threshold functions falling abruptly from almost 1 to almost 0 as i increases from 0 to n, the position of the transition depends on n, m, and p. Recall that α abbreviates $-(\log_{1-p} m)/n$. Lemma 9 below establishes a sharp threshold at $i^* = n + \log_{1-p} m = (1-\alpha)n$. Note that i^* is always at most n since $\log_{1-p} m$ is non-positive. The definition ensures the equality $m = 1/(1-p)^{n-i^*} = 1/(1-p)^{\alpha n}$. For increasing m, the threshold gets smaller relative to n. Once m grows beyond $1/(1-p)^n$, i.e., $\alpha > 1$, the quantity i^* can no longer be interpreted as a cardinality as it becomes negative. Later, in Lemma 12, we will see that m being this large is in fact irrelevant for the minimization.

[5] The notation $s_{n,p}$ refers the set being *sampled*; these probabilities are then *weighted* by the factors $w_{n,p}$.

▶ **Lemma 9.** *Let $0 < p < 1$ be a probability, and n, m positive integers, then $w_{n,p}(0,m) = 1$, and $w_{n,p}(n,m) = p^{nm}$. Now let $i = i(n)$ with $0 < i < n$ be a function taking integer values.*

1. *We have $\exp(-m\,(1-p)^{n-i}) \cdot (1 - m\,(1-p)^{2(n-i)}) \leq w_{n,p}(i,m) \leq \exp(-m\,(1-p)^{n-i+1})$.*

In particular, the following statements hold.[6]

2. *If $i = n + \log_{1-p} m + \omega(1)$, then $\lim_{n\to\infty} w_{n,p}(i,m) = 0$.*
3. *If $i = n + \log_{1-p} m - \omega(1)$, then $\lim_{n\to\infty} w_{n,p}(i,m) = 1$.*
4. *If $i = n + \log_{1-p} m \pm \Theta(1)$, then $w_{n,p}(i,m) = \Theta(1)$.*

Proof. Suppose $0 < i < n$, we estimate $w_{n,p}(i,m)$ using mainly Lemma 4. This yields $w_{n,p}(i,m) = (1-(1-p)^{n-i}(1-p^i))^m \leq (1-(1-p)^{n-i}(1-p))^m \leq \exp\Big(-m\,(1-p)^{n-i}\cdot(1-p)\Big)$. Since $1-p$ is constant, the limit behavior is entirely determined by the product $m\,(1-p)^{n-i}$. If $i = n + \log_{1-p} m + \omega(1)$, then $m\,(1-p)^{n-i} = m\,(1-p)^{n-n-(\log_{1-p} m)-\omega(1)} = (1-p)^{-\omega(1)}$ diverges and thus the weighting factor $w_{n,p}(i,m)$ converges to 0. Conversely, from $1-p^i \leq 1$ we get that $w_{n,p}(i,m) \geq (1-(1-p)^{n-i})^m \geq \exp\Big(-m\,(1-p)^{n-i}\Big)\cdot(1-m\,(1-p)^{2(n-i)})$. If $i = n + \log_{1-p} m - \omega(1)$, both $m\,(1-p)^{n-i} = (1-p)^{\omega(1)}$ and $m\,(1-p)^{2(n-i)} = (1-p)^{\omega(1)}/m$ tend to 0, implying $\lim_{n\to\infty} w_{n,p}(i,m) = 1$.

Finally, if the cardinality i is around the threshold $i^* = n + \log_{1-p} m$, the limit may not exist. We show that $w_{n,p}(i,m)$ is still bounded away from 0. Suppose $i = n + \log_{1-p} m \pm \Theta(1)$; in particular, the difference $i^* - i$ is bounded for all n. If m is constant w.r.t. n, so is $w_{n,p}(i,m) \geq (1-(1-p)^{n-i})^m \geq p^m$. Here, we used the assumption $i < n$. Finally, if m diverges, then $n - i = \log_{1-p} m \mp \Theta(1) = \omega(1)$ diverges with it. Together with the fact that $m\,(1-p)^{n-i} = (1-p)^{i^*-i}$ holds by the definition of i^*, we get that $w_{n,p}(i,m)$ is bounded since $w_{n,p}(i,m) \geq \exp\Big(-(1-p)^{i^*-i}\Big)\cdot(1-(1-p)^{(i^*-i)+(n-i)}) = \Omega(1)$. ◀

After we have shown the existence of a sharp threshold for the weighting factors, we now treat the number of distinct sets $\|\mathcal{B}_{n,m,p}\|$ in the multi-hypergraph. This is a natural upper bound for the size of the minimization. In turn, a trivial cap for the number of distinct sets is the total number of sets $|\mathcal{B}_{n,m,p}| = m$. When starting the sampling, many different sets are generated and $\|\mathcal{B}_{n,m,p}\|$ is indeed close to m. As the number of trials increases though, duplicates occur in the sample and the two quantities grow apart.

To discuss this in more detail, we introduce some notation. For a pair of integers ℓ, u with $0 \leq \ell \leq u \leq n$, let $\|\mathcal{B}_{n,m,p}(\ell,u)\|$ denote the number of distinct sampled sets whose cardinality is between ℓ and u, including. This is also at most as large as the total number of samples in that range. It thus makes sense to expect an upper bound in terms of the binomial distribution. We confirm this below and further prove that there is also a lower bound of the same flavor.

▶ **Lemma 10.** *Let $0 < p < 1$ be a probability, n, m positive integers, and $Y \sim \mathrm{Bin}(n,p)$ a binomially distributed random variable with parameters n and p. Let ℓ, u be integers such that $0 \leq \ell \leq u \leq n$ and define $\mathbf{p} = \max_{\ell\leq i\leq u}\{p^i(1-p)^{n-i}\}$. Then, $\mathbf{p}$ is equal to $p^\ell(1-p)^{n-\ell}$ if $p \leq 1/2$; otherwise, we have $\mathbf{p} = p^u(1-p)^{n-u}$. Further, the expected number of distinct sets in $\mathcal{B}_{n,m,p}$ with cardinality between ℓ and u observes $\frac{m}{1+m\mathbf{p}} \cdot \mathrm{P}[\ell \leq Y \leq u] \leq \mathrm{E}[\,\|\mathcal{B}_{n,m,p}(\ell,u)\|\,] \leq m \cdot \mathrm{P}[\ell \leq Y \leq u]$.*

[6] We understand $\omega(1)$ as the class of all non-negative unbounded functions of n. In particular, the classes $n + \log_{1-p} m + \omega(1)$ and $n + \log_{1-p} m - \omega(1)$ are disjoint.

6 Proof of the Main Theorem

We prove the main results on the expected size of the minimization of the hypergraph $\mathcal{B}_{n,m,p}$ in this section with the help of the tools above. The key observation is that the minimization is dominated by the sets with cardinalities around the threshold $i^* = n + \log_{1-p} m$.

6.1 The Lower Bound

We prove the main results, Theorem 1, with the help of the tools above. The key observation is that the minimization is dominated by the sets with cardinalities around the threshold $i^* = n + \log_{1-p} m$ of the weighting factors. We will see that the distinct edges make up a constant fraction of $\mathcal{B}_{n,m,p}$ as long as m is at most $1/(1-p)^{(1-p)n}$. In turn, a constant fraction of those distinct edges are indeed minimal. However, the linear growth of $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|]$ cannot be maintained for a larger sample size. We prove that once m is so large that the threshold i^* is below pn, the ratio of minimal edges decreases significantly. The minimization then enters a regime governed by the entropy of $\alpha = -(\log_{1-p} m)/n$.

The next lemma shows both lower bounds of Theorem 1 together. The information-theoretic one is slightly more general than what was stated in Theorem 1.2 in that it pertains to all m between $1/(1-p)^{(1-p)n}$ and $1/(1-p)^{(1-\varepsilon')n}$. Let H denote the entropy function.

▶ **Lemma 11** (Theorem 1.1 and the lower bound of Theorem 1.2)**.** *Let $0 < p < 1$. If $m \leq 1/(1-p)^{(1-p)n}$, then $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] = \Theta(m)$. For any $\varepsilon' > 0$ and m such that $1/(1-p)^{(1-p)n} \leq m \leq 1/(1-p)^{(1-\varepsilon')n}$, corresponding to $1-p \leq \alpha \leq 1-\varepsilon'$, we have $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] = \Omega\big(2^{(\mathrm{H}(\alpha)+(1-\alpha)\operatorname{ld} p)\,n}/\sqrt{n}\big)$.*

Proof sketch. The sought expectation is at least as large as the number of distinct sets up to some cardinality i that are minimal after m trials for arbitrary values of i. As an ansatz, we choose this to be the threshold $i^* = n + \log_{1-p} m$. Lemmas 8 and 10 together then imply that $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] \geq (m/(1+m\mathbf{p})) \cdot \mathrm{P}[Y \leq i^*] \cdot w_{n,p}(i^*, m)$. It can be shown via Lemma 10 that the denominator $1 + m\mathbf{p}$ is at most 2 as long as $m \leq 1/(1-p)^n$. Lemma 9.1 shows that there exists a universal constant $\delta > 0$ (again for all $m \leq 1/(1-p)^n$) such that $w(i^*, m) \geq \delta$.

The bounds in the two regimes differ in the way the product $m \cdot \mathrm{P}[Y \leq i^*]$ is estimated. If $m \leq 1/(1-p)^{(1-p)n}$, then $i^* \geq pn$ is at least as large as the median of Y, whence $m \cdot \mathrm{P}[Y \leq i^*] \geq m/2$. This gives the lower bound in the linear regime. In the information-theoretic regime, we use the rewrite $m = 1/(1-p)^{\alpha n}$. Suppose first that there are constants $\varepsilon, \varepsilon' > 0$ such that $1 - p + \varepsilon \leq \alpha \leq 1 - \varepsilon'$ holds. These are exactly the prerequisites of Theorem 1.2. We apply the improved lower bound of the Chernoff–Hoeffding theorem, Theorem 7.1. Let D denote the Kullback–Leibler divergence. There exists a positive constant $C > 0$–independent of n and m but possibly dependent on p, ε, and ε'–such that

$$\begin{aligned} m \cdot \mathrm{P}[Y \leq i^*] &= m \cdot \mathrm{P}[Y \leq (1-\alpha)n] \geq m \cdot C\,\frac{2^{-\mathrm{D}(1-\alpha\,\|\,p)\,n}}{\sqrt{n}} \\ &= \frac{1}{(1-p)^{\alpha n}} \cdot \frac{C}{\sqrt{n}} \left(\frac{p}{1-\alpha}\right)^{(1-\alpha)n} \left(\frac{1-p}{\alpha}\right)^{\alpha n} = \frac{C}{\sqrt{n}} \cdot \left(\frac{p^{1-\alpha}}{(1-\alpha)^{1-\alpha}\,\alpha^{\alpha}}\right)^n. \end{aligned}$$

The latter expression equals $C \cdot 2^{(\mathrm{H}(\alpha)+(1-\alpha)\operatorname{ld} p)\,n}/\sqrt{n}$. Finally, if $m(1-p)^{(1-p)n}$ converges to 1 from above, i.e., $\alpha \searrow 1-p$, then the result follows from a direct application of Lemma 3. ◀

6.2 The Upper Bound

The upper bound draws from the same core observations as the lower one: the threshold position of the weighting factors $w_{n,p}$ and the proportion of distinct sets in the sample. First, we show that once m is more than a polynomial factor larger than $1/(1-p)^n$, the minimization essentially consists of a single edge, the empty set. Lemma 13 then proves our claim that the information-theoretic lower bound is tight beyond the phase transition.

▶ **Lemma 12** (Theorem 1.3). *If* $m = 1/(1-p)^{n+\omega(\log n)}$, *then* $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] = 1 + \mathrm{o}(1)$.

▶ **Lemma 13** (Upper bound of Theorem 1.2). *Let* $0 < p < 1$ *be a probability,* $\varepsilon, \varepsilon' > 0$ *positive reals, and* n, m *positive integers such that* m *is between* $1/(1-p)^{(1-p+\varepsilon)n}$ *and* $1/(1-p)^{(1-\varepsilon')n}$, *i.e.,* $1 - p + \varepsilon \le \alpha \le 1 - \varepsilon'$. *Then, we have* $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] = \mathrm{O}\big(2^{(\mathrm{H}(\alpha)+(1-\alpha)\,\mathrm{ld}\,p)\,n}/\sqrt{n}\big)$.

Proof sketch. We get $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|] \le \sum_{i=0}^{n} \binom{n}{i}(1-(1-p^i(1-p)^{n-i})^m) \cdot w_{n,p}(i, m-1)$ from Lemma 8.2. The idea of this proof is to split the sum at the threshold $i^* = (1-\alpha)n$ and handle the two parts separately. Let $Y \sim \mathrm{Bin}(n,p)$ be a binomial variable. Lemma 10 shows for the first part that $\sum_{i=0}^{i^*} \binom{n}{i}(1-(1-p^i(1-p)^{n-i})^m) \cdot w_{n,p}(i, m-1) \le m \cdot \mathrm{P}[Y \le i^*]$. The new Chernoff–Hoeffding theorem (Theorem 7.1) gives a constant $C' = C'(p, \varepsilon, \varepsilon')$ with

$$m \cdot \mathrm{P}[Y \le i^*] = m \cdot \mathrm{P}[Y \le (1-\alpha)n] \le m \cdot C' \frac{2^{-\mathrm{D}(1-\alpha\,\|\,p)\,n}}{\sqrt{n}} = \mathrm{O}\left(\frac{2^{(\mathrm{H}(\alpha)+(1-\alpha)\,\mathrm{ld}\,p)\,n}}{\sqrt{n}}\right).$$

The lemma follows from the second part of the sum being at most a constant factor larger than the first one. This is shown using the assumption $\alpha \le 1 - \varepsilon'$ and the weighting factors $w_{n,p}(i,m)$ going doubly exponentially to 0 if i crosses the threshold i^*, see Lemma 9.1. ◀

7 Conclusion

We examined the expected number of minimal edges of the maximum-entropy multi-hypergraph model with expected edge size pn. We discovered a phase transition with respect to the total number of edges at $m = 1/(1-p)^{(1-p)n}$. Now that we have tight upper and lower bounds in place, we can discuss the transition in full detail. For small m, $\mathrm{E}[|\min(\mathcal{B}_{n,m,p})|]$ is linear in m. Beyond that point, the minimization instead follows $2^{(\mathrm{H}(\alpha)+(1-\alpha)\,\mathrm{ld}\,p)\,n}/\sqrt{n}$ with $\alpha = -(\log_{1-p} m)/n$. In the information-theoretic regime the size of the minimization is decoupled from the number of edges. It continues to grow initially, but now sublinearly in m and only until $m = 1/(1-p)^{\frac{n}{1+p}}$. From there on, the size of the minimization decays rapidly although the total number of trials increases. Once m exceeds $1/(1-p)^n$, the minimization collapses under the sheer likelihood of sampling the empty set.

We gain additional insights by contrasting the results in the two regimes (ignoring constant factors here). The ratio between the two bounds at $m = 1/(1-p)^{\alpha n}$ for any $0 \le \alpha \le 1$ is $((2^{(\mathrm{H}(\alpha)\,+\,(1-\alpha)\,\mathrm{ld}\,p)\,n})/\sqrt{n})/m = (2^{-\mathrm{D}(1-\alpha\,\|\,p)\,n})/\sqrt{n}$. This is exponentially small in n when α lies strictly between 0 and $1-p$. Therefore, the information-theoretic *lower* bound of Theorem 1.2 also pertains to the linear regime, but is unnecessarily loose there. If the number of trials m is close to $1/(1-p)^{(1-p)n}$, the two bounds coincide, up to a factor of $\sqrt{n}$, since the divergence vanishes at $1-\alpha = p$. This overlap is indicated in Figure 1b by dashed lines. Finally, for α beyond $1-p$ the relative share of minimal edges becomes exponentially small.

The Chernoff–Hoeffding theorem played an integral role in verifying these results. We tightened the tail bounds on the binomial distribution and provided explicit upper and lower bounds on the constants involved. We are convinced that this sharpened tool can help researchers in all of probability beyond the scope of this paper.

There is more work needed for the upper and lower bounds in the information-theoretic regime. Currently, α has to be bounded away from from $1-p$ and 1 for the bounds to be tight. For $\alpha \searrow 1-p$ the lower bound goes to $\Omega(2^{(\mathrm{H}(1-p)+p\operatorname{ld} p)\,n}/\sqrt{n}) = \Omega(m/\sqrt{n})$. Here, we profit from the hidden constant not depending on α. In actuality though, the minimization at $m = 1/(1-p)^{(1-p)n}$ has size $\Theta(m)$, so the share of minimal edges in the sample moves from order $1/\sqrt{n}$ to a constant. The speed of this shift depends on how fast $\alpha = 1-p+\mathrm{o}(1)$ converges. The situation for $\alpha \nearrow 1$ is different as in this parameter range there is a huge disparity between the number of minimal edges $|\min(\mathcal{B}_{n,m,p})|$ and the number of distinct edges $||\mathcal{B}_{n,m,p}||$. Thus, the expected size of the minimization is not completely captured by the binomial distribution and additional tools are needed for tight estimates. An immediate extension of our work would therefore be to pinpoint the exact behavior of the minimization at the two transitions points. Another interesting question in light of the original motivation of random databases is to allow different sample probabilities per vertex as well as dependencies between the elements. To fit the maximum-entropy setting, this would require the model to incorporate additional constraints.

References

1 Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. *Data Profiling*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael, CA, USA, 2018. doi:10.2200/S00878ED1V01Y201810DTM052.

2 Kartik Anand and Ginestra Bianconi. Entropy Measures for Networks: Toward an Information Theory of Complex Topologies. *Physical Review E*, 80:045102, 2009. doi:10.1103/PhysRevE.80.045102.

3 Robert B. Ash. *Information Theory*. Dover Books on Mathematics. Dover Publications, Mineola, NY, USA, 1990. Reprint of the Interscience Publishers 1965 edition.

4 Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Black-box Complexity of Parallel Search with Distributed Populations. In *Proceedings of the 2015 Conference on Foundations of Genetic Algorithms (FOGA)*, pages 3–15, 2015. doi:10.1145/2725494.2725504.

5 Michael Behrisch, Amin Coja-Oghlan, and Mihyun Kang. The Order of the Giant Component of Random Hypergraphs. *Random Structures and Algorithms*, 36:149–184, 2010. doi:10.1002/rsa.v36:2.

6 Michael Behrisch, Amin Coja-Oghlan, and Mihyun Kang. Local Limit Theorems for the Giant Component of Random Hypergraphs. *Combinatorics, Probability and Computing*, 23:331—366, 2014. doi:10.1017/S0963548314000017.

7 Claude Berge. *Hypergraphs - Combinatorics of Finite Sets*, volume 45 of *North-Holland Mathematical Library*. North-Holland, Amsterdam, Netherlands, 1989.

8 Ginestra Bianconi. The Entropy of Randomized Network Ensembles. *Europhysics Letters*, 81:28005, 2007. doi:10.1209/0295-5075/81/28005.

9 Thomas Bläsius, Tobias Friedrich, and Martin Schirneck. The Parameterized Complexity of Dependency Detection in Relational Databases. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 6:1–6:13, 2016. doi:10.4230/LIPIcs.IPEC.2016.6.

10 Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. Efficient Denial Constraint Discovery with Hydra. *Proceedings of the VLDB Endowment*, 11:311–323, 2017. doi:10.14778/3157794.3157800.

11 Thomas Bläsius, Tobias Friedrich, Julius Lischeid, Kitty Meeks, and Martin Schirneck. Efficiently Enumerating Hitting Sets of Hypergraphs Arising in Data Profiling. In *Proceedings of the 21st Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 130–143, 2019. doi:10.1137/1.9781611975499.11.

12 Béla Bollobás. *Random Graphs*. Studies in Advanced Mathematics. Cambridge University Press, Cambridge, UK, 2 edition, 2001. doi:10.1017/CBO9780511814068.

13 Béla Bollobás. A Probabilistic Proof of an Asymptotic Formula for the Number of Labelled Regular Graphs. *European Journal of Combinatorics*, 1:311–316, 1980. doi:10.1016/S0195-6698(80)80030-8.

14 Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the Square: On the Complexity of Some Quadratic-time Solvable Problems. *Electronic Notes in Theoretical Computer Science*, 322:51–67, 2016. doi:10.1016/j.entcs.2016.03.005.

15 Aiden A. Bruen and Mario A. Forcinito. *Cryptography, Information Theory, and Error-Correction: A Handbook for the 21st Century*. Wiley-Interscience, New York, NY, USA, 2004. doi:10.1002/9781118033296.

16 Philip Samuel Chodrow. Configuration Models of Random Hypergraphs. *ArXiv e-prints*, 2019. arXiv:1902.09302.

17 Colin Cooper, Alan M. Frieze, and Wesley Pegden. On the Rank of a Random Binary Matrix. *Electronic Journal of Combinatorics*, 26:P4.12, 2019. doi:10.37236/8092.

18 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications and Signal Processing. Wiley-Interscience, New York, NY, USA, 2nd edition, 2006.

19 Harald Cramér. Sur un nouveau théorème-limite de la théorie des probabilités. In *Actualités scientifiques et industrielles*, volume 763, pages 5–23, 1938. Colloque consacré à la théorie des probabilités. (On a New Limit Theorem in Probability.) In French.

20 J. Demetrovics, Gyula O. H. Katona, D. Miklos, O. Seleznjev, and B. Thalheim. Asymptotic Properties of Keys and Functional Dependencies in Random Databases. *Theoretical Computer Science*, 190:151–166, 1998. doi:10.1016/S0304-3975(97)00089-3.

21 Benajmin Doerr. Probabilistic Tools for the Analysis of Randomized Optimization Heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation*, chapter 1. Springer International, Basel, Switzerland, 2020. doi:10.1007/978-3-030-29414-4.

22 Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 2009.

23 Paul Erdős and Alfréd Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

24 Vincent Froese, René van Bevern, Rolf Niedermeier, and Manuel Sorge. Exploiting Hidden Structure in Selecting Dimensions That Distinguish Vectors. *Journal of Computer and System Sciences*, 82:521–535, 2016. doi:10.1016/j.jcss.2015.11.011.

25 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for First-Order Properties on Sparse Structures With Algorithmic Applications. *Transactions on Algorithms*, 15:23:1–23:35, 2018. doi:10.1145/3196275.

26 Diego Garlaschelli and Maria I. Loffredo. Maximum Likelihood: Extracting Unbiased Information from Complex Networks. *Physical Review E*, 78:015101, 2008. doi:10.1103/PhysRevE.78.015101.

27 Edgar N. Gilbert. Random Graphs. *Annals of Mathematical Statistics*, 30:1141–1144, 1959. doi:10.1214/aoms/1177706098.

28 Yves Grandvalet and Yoshua Bengio. Entropy Regularization. In Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, editors, *Semi-Supervised Learning*, chapter 9. MIT Press, Cambridge, MA, USA, 2006. doi:10.7551/mitpress/9780262033589.001.0001.

29 Peter Harremoës. Binomial and Poisson Distributions as Maximum Entropy Distributions. *Transactions on Information Theory*, 47:2039–2041, 2001. doi:10.1109/18.930936.

30 Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

31 Remco van der Hofstad. *Random Graphs and Complex Networks*, volume 1 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, UK, 2016. doi:10.1017/9781316779422.

32 Edwin Thompson Jaynes. Information Theory and Statistical Mechanics. *Phyical Review Series II*, 106:620–630, 1957. doi:10.1103/PhysRev.106.620.

33 Edwin Thompson Jaynes. Information Theory and Statistical Mechanics II. *Phyical Review Series II*, 108:171–190, 1957. doi:10.1103/PhysRev.108.171.

34 Michał Karoński and Tomasz Łuczak. The Phase Transition in a Random Hypergraph. *Journal of Computational and Applied Mathematics*, 142:125–135, 2002. doi:10.1016/S0377-0427(01)00464-2.

35 Gyula O. H. Katona. Random Databases with Correlated Data. In Antje Düsterhöft, Meike Klettke, and Klaus-Dieter Schewe, editors, *Conceptual Modelling and Its Theoretical Foundations: Essays Dedicated to Bernhard Thalheim on the Occasion of His 60th Birthday*, pages 29–35. Springer, Berlin and Heidelberg, Germany, 2012. Festschrift. doi:10.1007/978-3-642-28279-9_4.

36 Gyula O. H. Katona. Testing Functional Connection Between Two Random Variables. In Albert N. Shiryaev, S. R. S. Varadhan, and Ernst L. Presman, editors, *Prokhorov and Contemporary Probability Theory*, pages 335–348. Springer, Berlin and Heidelberg, Germany, 2013. Festschrift. doi:10.1007/978-3-642-33549-5_20.

37 Hiremagalur Krishnaswamy Kesavan. Jaynes' Maximum Entropy Principle. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1779–1782. Springer, Boston, MA, USA, 2009. doi:10.1007/978-0-387-74759-0_312.

38 Bernhard Klar. Bounds on Tail Probabilities of Discrete Distributions. *Probability in the Engineering and Informational Sciences*, 14:161–171, 2000.

39 Elliott H. Lieb and Jakob Yngvason. The Physics and Mathematics of the Second Law of Thermodynamics. *Physics Reports*, 310:1–96, 1999. doi:10.1016/S0370-1573(98)00082-9.

40 Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. Approximate Denial Constraints. *CoRR*, abs/2005.08540, 2020. Arxiv preprint. To apprear in PVLDB 13. arXiv:2005.08540.

41 Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, New York, NY, USA, 2nd edition, 2017.

42 Mark E. J. Newman. Scientific Collaboration Networks. I. Network Construction and Fundamental Results. *Physical Review E*, 64:016131, 2001. doi:10.1103/PhysRevE.64.016131.

43 Mark E. J. Newman. *Networks: An Introduction*. Oxford University Press, New York, NY, USA, 2010. doi:10.1093/acprof:oso/9780199206650.001.0001.

44 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, Cambridge, UK, 2010. doi:10.1017/CBO9780511976667.

45 Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proceedings of the VLDB Endowment*, 8:1082–1093, 2015. doi:10.14778/2794367.2794377.

46 Juyong Park and Mark E. J. Newman. The Statistical Mechanics of Networks. *Physical Review E*, 70:066117, 2004. doi:10.1103/PhysRevE.70.066117.

47 Yuri Vasilyevich Prokhorov. Asymptotic Behavior of the Binomial Distribution. *Uspekhi Matematicheskikh Nauk*, 8:135–142, 1953. In Russian.

48 Fabio Saracco, Riccardo Di Clemente, Andrea Gabrielli, and Tiziano Squartini. Randomizing Bipartite Networks: The Case of the World Trade Web. *Scientific Reports*, 5:10595, 2015. doi:10.1038/srep10595.

49 Jeanette Schmidt-Pruzan and Eli Shamir. Component Structure in the Evolution of Random Hypergraphs. *Combinatorica*, 5:81–94, 1985. doi:10.1007/BF02579445.

50 Claude Elwood Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423, 1948. doi:10.1002/j.1538-7305.1948.tb01338.x.

51 Eric V. Slud. Distribution Inequalities for the Binomial Law. *The Annals of Probability*, 5:404–412, 1977. URL: https://projecteuclid.org/euclid.aop/1176995801.

52 Ryan Williams. A New Algorithm for Optimal 2-Constraint Satisfaction and Its Implications. *Theoretical Computer Science*, 348:357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

53 Katharina Anna Zweig. *Network Analysis Literacy: A Practical Approach to the Analysis of Networks*. Lecture Notes in Social Networks. Springer, Vienna, Austria, 2014. doi:10.1007/978-3-7091-0741-6.

Acyclic, Star and Injective Colouring: A Complexity Picture for H-Free Graphs

Jan Bok
Computer Science Institute, Charles University, Prague, Czech Republic
bok@iuuk.mff.cuni.cz

Nikola Jedličková
Department of Applied Mathematics, Charles University, Prague, Czech Republic
jedlickova@kam.mff.cuni.cz

Barnaby Martin
Department of Computer Science, Durham University, UK
barnaby.d.martin@durham.ac.uk

Daniël Paulusma
Department of Computer Science, Durham University, UK
daniel.paulusma@durham.ac.uk

Siani Smith
Department of Computer Science, Durham University, UK
siani.smith@durham.ac.uk

Abstract

A k-colouring c of a graph G is a mapping $V(G) \to \{1, 2, \ldots\ k\}$ such that $c(u) \neq c(v)$ whenever u and v are adjacent. The corresponding decision problem is Colouring. A colouring is acyclic, star, or injective if any two colour classes induce a forest, star forest or disjoint union of vertices and edges, respectively. Hence, every injective colouring is a star colouring and every star colouring is an acyclic colouring. The corresponding decision problems are Acyclic Colouring, Star Colouring and Injective Colouring (the last problem is also known as $L(1,1)$-Labelling).

A classical complexity result on Colouring is a well-known dichotomy for H-free graphs, which was established twenty years ago (in this context, a graph is H-free if and only if it does not contain H as an *induced* subgraph). Moreover, this result has led to a large collection of results, which helped us to better understand the complexity of Colouring. In contrast, there is no systematic study into the computational complexity of Acyclic Colouring, Star Colouring and Injective Colouring despite numerous algorithmic and structural results that have appeared over the years.

We initiate such a systematic complexity study, and similar to the study of Colouring we use the class of H-free graphs as a testbed. We prove the following results:

1. We give almost complete classifications for the computational complexity of Acyclic Colouring, Star Colouring and Injective Colouring for H-free graphs.
2. If the number of colours k is fixed, that is, not part of the input, we give full complexity classifications for each of the three problems for H-free graphs.

From our study we conclude that for fixed k the three problems behave in the same way, but this is no longer true if k is part of the input. To obtain several of our results we prove stronger complexity results that in particular involve the girth of a graph and the class of line graphs.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases acyclic colouring, star colouring, injective colouring, H-free, dichotomy

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.22

Funding *Jan Bok*: Supported by GAUK 1580119 and SVV–2020–260578.
Nikola Jedličková: Supported by GAUK 1198419 and SVV–2020–260578.
Daniël Paulusma: Supported by the Leverhulme Trust (RPG-2016-258).

1 Introduction

We study the complexity of three classical colouring problems. We do this by focusing on *hereditary* graph classes, i.e., classes closed under vertex deletion, or equivalently, classes characterized by a (possibly infinite) set $\mathcal{F}$ of forbidden induced subgraphs. As evidenced by numerous complexity studies in the literature, even the case where $|\mathcal{F}| = 1$ captures a rich family of graph classes suitably interesting to develop general methodology. Hence, we usually first set $\mathcal{F} = \{H\}$ and consider the class of *H-free* graphs, i.e., graphs that do not contain H as an induced subgraph. We then investigate how the complexity of a problem restricted to H-free graphs depends on the choice of H and try to obtain a *complexity dichotomy*.

To give a well-known and relevant example, the Colouring problem is to decide, given a graph G and integer $k \geq 1$, if G has a *k-colouring*, i.e., a mapping $c : V(G) \to \{1, \ldots, k\}$ such that $c(u) \neq c(v)$ for every two adjacent vertices u and v. Král' et al. [37] proved that Colouring on H-free graphs is polynomial-time solvable if H is an induced subgraph of P_4 or $P_1 + P_3$ and NP-complete otherwise. Here, P_n denotes the n-vertex path and $G_1 + G_2 = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$ the disjoint union of two vertex-disjoint graphs G_1 and G_2. If k is fixed (not part of the input), then we obtain the k-Colouring problem. No complexity dichotomy is known for k-Colouring if $k \geq 3$. In particular, the complexities of 3-Colouring for P_t-free graphs for $t \geq 8$ and k-Colouring for sP_4-free graphs for $s \geq 2$ and $k \geq 3$ are still open. Here, we write sG for the disjoint union of s copies of G. We refer to the survey of Golovach et al. [27] for further details and to [13, 36] for updated summaries.

For a colouring c of a graph G, a *colour class* consists of all vertices of G that are mapped by c to a specific colour i. We consider the following special graph colourings. A colouring of a graph G is *acyclic* if the union of any two colour classes induces a forest. The $(r+1)$-vertex *star* $K_{1,r}$ is the graph with vertices $u, v_1, \ldots, v_r$ and edges uv_i for every $i \in \{1, \ldots, r\}$. An acyclic colouring is a *star colouring* if the union of any two colour classes induces a *star forest*, that is, a forest in which each connected component is a star. A star colouring is *injective* (or an *$L(1,1)$-labelling*) if the union of any two colour classes induces an $sP_1 + tP_2$ for some integers $s \geq 0$ and $t \geq 0$. An alternative definition is to say that all the neighbours of every vertex of G are uniquely coloured. These definitions lead to the following three decision problems:

Acyclic Colouring	
Instance:	A graph G and an integer $k \geq 1$
Question:	Does G have an acyclic k-colouring?

Star Colouring	
Instance:	A graph G and an integer $k \geq 1$
Question:	Does G have a star k-colouring?

Injective Colouring	
Instance:	A graph G and an integer $k \geq 1$
Question:	Does G have an injective k-colouring?

If k is fixed, we write Acyclic k-Colouring, Star k-Colouring and Injective k-Colouring, respectively.

All three problems have been extensively studied. We note that in the literature on the Injective Colouring problem it is often assumed that two adjacent vertices may be coloured alike by an injective colouring (see, for example, [29, 30, 33]). However, in our

paper, we do **not** allow this; as reflected in their definitions we only consider colourings that are proper. This enables us to compare the results for the three different kinds of colourings with each other.

So far, systematic studies mainly focused on structural characterizations, exact values, lower and upper bounds on the minimum number of colours in an acyclic colouring or star colouring (i.e., the *acyclic* and *star chromatic number*); see, e.g., [2, 9, 19, 20, 21, 34, 35, 50, 51, 53], to name just a few papers, whereas injective colourings (and the *injective chromatic number*) were mainly considered in the context of the distance constrained labelling framework (see the survey [11] and Section 6 therein). The problems have also been studied from a complexity perspective, but apart from a study on Acyclic Colouring for graphs of bounded maximum degree [45], known results are scattered and relatively sparse. We perform a *systematic* and *comparative* complexity study by focusing on the following research question both for k part of the input and for fixed k:
What are the computational complexities of Acyclic Colouring, Star Colouring and Injective Colouring for H-free graphs?
Before discussing our new results and techniques, we first briefly discuss some known results.

Coleman and Cai [14] proved that for every $k \geq 3$, Acyclic k-Colouring is NP-complete for bipartite graphs. Afterwards, a number of hardness results appeared for other hereditary graph classes. Alon and Zaks [3] showed that Acyclic 3-Colouring is NP-complete for line graphs of maximum degree 4. Angelini and Frati [4] showed that Acyclic 3-Colouring is NP-complete for planar graphs of maximum degree 4. Mondal et al. [45] proved that Acyclic 4-Colouring is NP-complete for graphs of maximum degree 5 and for planar graphs of maximum degree 7. Albertson et al. [1] and recently, Lei et al. [38] proved that Star 3-Colouring is NP-complete for planar bipartite graphs and line graphs, respectively. Bodlaender et al. [7], Sen and Huson [48] and Lloyd and Ramanathan [41] proved that Injective Colouring is NP-complete for split graphs, unit disk graphs and planar graphs, respectively. Mahdian [44] proved that for every $k \geq 4$, Injective k-Colouring is NP-complete for line graphs, whereas Injective 4-Colouring is known to be NP-complete for cubic graphs (see [11]); observe that Injective 3-Colouring is trivial for general graphs.

On the positive side, Lyons [43] showed that every acyclic colouring of a P_4-free graph is, in fact, a star colouring. Lyons [43] also proved that Acyclic Colouring and Star Colouring are polynomial-time solvable for P_4-free graphs; we note that Injective Colouring is trivial for P_4-free graphs, as every injective colouring must assign each vertex of a connected P_4-free graph a unique colour. The results of Lyons have been extended to P_4-tidy graphs and $(q, q-4)$-graphs [40]. Cheng et al. [12] complemented the aforementioned result of Alon and Zaks [3] by proving that Acyclic Colouring is polynomial-time solvable for claw-free graphs of maximum degree at most 3. Calamoneri [11] observed that Injective Colouring is polynomial-time solvable for comparability and co-comparability graphs. Zhou et al. [52] proved that Injective Colouring is polynomial-time solvable for graphs of bounded treewidth (which is best possible due to the W[1]-hardness result of Fiala et al. [22]).

Our Complexity Results and Methodology

The *girth* of a graph G is the length of a shortest cycle of G (if G is a forest, then its girth is ∞). To answer our research question we focus on two important graph classes, namely the classes of graphs of high girth and line graphs, which are interesting classes on their own. If a problem is NP-complete for both classes, then it is NP-complete for H-free graphs whenever H has a cycle or a claw. It then remains to analyze the case when H is a *linear forest*, i.e., a disjoint union of paths; see [8, 10, 25, 37] for examples of this approach, which we discuss in detail below.

The construction of graph families of high girth and large chromatic number is well studied in graph theory (see, e.g. [18]). To prove their complexity dichotomy for Colouring on H-free graphs, Král' et al. [37] first showed that for every integer $g \geq 3$, 3-Colouring is NP-complete for the class of graphs of girth at least g. This approach can be readily extended to any integer $k \geq 3$ [17, 42]. The basic idea is to replace edges in a graph by graphs of high girth and large chromatic number, such that the resulting graph has sufficiently high girth and is k-colourable if and only if the original graph is so (see also [28, 32]).

By a more intricate use of the above technique we are able to prove that for every $g \geq 3$, Acyclic 3-Colouring is NP-complete for the class of graphs of girth at least g. This implies that Acyclic 3-Colouring is NP-complete for H-free graphs whenever H has a cycle. We prove the same result for every $k \geq 4$ by combining known results, just as we use known results to prove that Star k-Colouring ($k \geq 3$) and Injective k-Colouring ($k \geq 4$) are NP-complete for H-free graphs if H has a cycle.

A classical result of Holyer [31] is that 3-Colouring is NP-complete for line graphs (and Leven and Galil [39] proved the same for $k \geq 4$). As line graphs are claw-free, Král' et al. [37] used Holyer's result to show that 3-Colouring is NP-complete for H-free graphs whenever H has an induced claw. For Acyclic 3-Colouring, this follows from Alon and Zaks' result [3], which we extend to work for $k \geq 4$. For Injective k-Colouring ($k \geq 4$) we can use the aforementioned result on line graphs of Mahdian [44].

The above hardness results leave us to consider the case where H is a linear forest. In Section 2 we will use a result of Atminas et al. [5] to prove a general result from which it follows that for fixed k, all three problems are polynomial-time solvable for H-free graphs if H is a linear forest. Hence, we have full complexity dichotomies for the three problems when k is fixed. However, these positive results do not extend to the case where k is part of the input: we prove NP-completeness for graphs that are P_r-free for some small value of r or have a small independence number, i.e., that are sP_1-free for some small integer s.

Our complexity results for H-free graphs are summarized in the following three theorems, proven in Sections 3–5, respectively; see Table 1 for a comparison. For two graphs F and G, we write $F \subseteq_i G$ or $G \supseteq_i F$ to denote that F is an *induced* subgraph of G.

▶ **Theorem 1.** *Let H be a graph. For the class of H-free graphs it holds that:*

- **(i)** Acyclic Colouring *is polynomial-time solvable if $H \subseteq_i P_4$ and NP-complete if H is not a forest or $H \supseteq_i 19P_1, 3P_3$ or $2P_5$;*
- **(ii)** *For every $k \geq 3$,* Acyclic k-Colouring *is polynomial-time solvable if H is a linear forest and NP-complete otherwise.*

▶ **Theorem 2.** *Let H be a graph. For the class of H-free graphs it holds that:*

- **(i)** Star Colouring *is polynomial-time solvable if $H \subseteq_i P_4$ and NP-complete for any $H \neq 2P_2$.*
- **(ii)** *For every $k \geq 3$,* Star k-Colouring *is polynomial-time solvable if H is a linear forest and NP-complete otherwise.*

▶ **Theorem 3.** *Let H be a graph. For the class of H-free graphs it holds that:*

- **(i)** Injective Colouring *is polynomial-time solvable if $H \subseteq_i P_4$ or $H \subseteq_i P_1 + P_3$ and NP-complete if H is not a forest or $2P_2 \subseteq_i H$ or $6P_1 \subseteq_i H$.*
- **(ii)** *For every $k \geq 4$,* Injective k-Colouring *is polynomial-time solvable if H is a linear forest and NP-complete otherwise.*

In Section 6 we give a number of open problems that resulted from our systematic study; in particular we will discuss the distance constrained labelling framework in more detail.

Table 1 The state-of-the-art for the three problems in this paper and the original Colouring problem; both when k is fixed and when k is part of the input.

	polynomial time	NP-complete
Colouring [37]	$H \subseteq_i P_4$ or $P_1 + P_3$	else
Acyclic Colouring	$H \subseteq_i P_4$	else except for at most 1991 open cases
Star Colouring	$H \subseteq_i P_4$	else except for 1 open case
Injective Colouring	$H \subseteq_i P_4$ or $P_1 + P_3$	else except for 10 open cases
k-Colouring (see [13, 27, 36])	depends on k	infinitely many open cases for all $k \geq 3$
Acyclic k-Colouring ($k \geq 3$)	H is a linear forest	else
Star k-Colouring ($k \geq 3$)	H is a linear forest	else
Injective k-Colouring ($k \geq 4$)	H is a linear forest	else

2 A General Polynomial Result

A *biclique* or *complete bipartite graph* is a bipartite graph on vertex set $S \cup T$, such that S and T are independent sets and there is an edge between every vertex of S and every vertex of T; if $|S| = s$ and $|T| = t$, we denote this graph by $K_{s,t}$, and if $s = t$, the biclique is *balanced* and of *order* s. We say that a colouring c of a graph G satisfies the *balance biclique condition* (BB-condition) if c uses at least $k + 1$ colours to colour G, where k is the order of a largest biclique that is contained in G as a (not necessarily induced) subgraph.

Let π be some colouring property; e.g., π could mean being acyclic, star or injective. Then π *can be expressed in MSO*$_2$ (monadic second-order logic with edge sets) if for every $k \geq 1$, the graph property of having a k-colouring with property π can be expressed in MSO$_2$. The general problem Colouring(π) is to decide, on a graph G and integer $k \geq 1$, if G has a k-colouring with property π. If k is fixed, we write k-Colouring(π). We now prove the following result.

▶ **Theorem 4.** *Let H be a linear forest, and let π be a colouring property that can be expressed in MSO$_2$, such that every colouring with property π satisfies the BB-condition. Then, for every integer $k \geq 1$, k-Colouring(π) is linear-time solvable for H-free graphs.*

Proof. Atminas, Lozin and Razgon [5] proved that that for every pair of integers ℓ and k, there exists a constant $b(\ell, k)$ such that every graph of treewidth at least $b(\ell, k)$ contains an induced P_ℓ or a (not necessarily induced) biclique $K_{k,k}$. Let G be an H-free graph, and let ℓ be the smallest integer such that $H \subseteq_i P_\ell$; observe that ℓ is a constant. Hence, we can use Bodlaender's algorithm [6] to test in linear time if G has treewidth at most $b(\ell, k) - 1$.

First suppose that the treewidth of G is at most $b(\ell, k) - 1$. As π can be expressed in MSO$_2$, the result of Courcelle [15] allows us to test in linear time whether G has a k-colouring with property π. Now suppose that the treewidth of G is at least $b(\ell, k)$. As G is H-free, G is P_ℓ-free. Then, by the result of Atminas, Lozin and Razgon [5], we find that G contains $K_{k,k}$ as a subgraph. As π satisfies the BB-condition, G has no k-colouring with property π. ◀

We now apply Theorem 4 to obtain the polynomial cases for fixed k in Theorem 1–3.

▶ **Corollary 5.** *Let H be a linear forest. For every $k \geq 1$,* Acyclic k-Colouring, Star k-Colouring *and* Injective k-Colouring *are polynomial-time solvable for H-free graphs.*

Proof. All three kinds of colourings use at least s colours to colour $K_{s,s}$ (as the vertices from one bipartition class of $K_{s,s}$ must receive unique colours). Hence, every acyclic, star and injective colouring of every graph satisfies the BB-condition. Moreover, it is readily seen that the colouring properties of being acyclic, star or injective can all be expressed in MSO$_2$. Hence, we may apply Theorem 4. ◀

3 Acyclic Colouring

In this section, we prove Theorem 1. For a graph G and a colouring c, the pair (G, c) has a *bichromatic* cycle C if C is a cycle of G with $|c(V(C)| = 2$, i.e., the vertices of C are coloured by two alternating colours (so C is even). A path P in G is an *i-j-path* if the vertices of P have alternating colours i and j. We now prove the following result.

▶ **Lemma 6.** *For every $g \geq 3$,* Acyclic 3-Colouring *is NP-complete for graphs of girth at least g.*

Proof. We reduce from Acyclic 3-Colouring, which is known to be NP-complete [14]. We start by taking a graph F that has a 4-colouring but no 3-colouring and that is of girth at least g. By a seminal result of Erdős [18], such a graph F exists (and its size is constant, as it only depends on g which is a fixed integer). We now repeatedly remove edges from F until we obtain a graph F' that is acyclically 3-colourable. Let xy be the last edge that we removed. As F has no 3-colouring, the edge xy exists. Moreover, by our construction, the graph $F' + xy$ is not acyclically 3-colourable. As edge deletions do not decrease the girth, $F' + xy$ and F' have girth at least g.

The basic idea (Case 1) is as follows. Let G be an instance of Acyclic 3-Colouring. We pick an edge $uv \in E(G)$. In $G - uv$ we "glue" F' by identifying u with x and y with v; see also Figure 1. We then prove that G has an acyclic 3-colouring if and only if G' has an acyclic 3-colouring. Then, by performing the same operation for each other edge of G as well, we obtain a graph G'', such that G has an acyclic 3-colouring if and only if G'' has so. As the size of G'' is polynomial in the size of G and the girth of G'' is at least g, we have proven the theorem. The challenge in this technique is that we do not know how the graph F' looks. We can only prove its existence and therefore have to consider several possibilities for the properties of the acyclic 3-colourings of F'. Hence, we distinguish between Cases 1–3, 4a, and 4b.

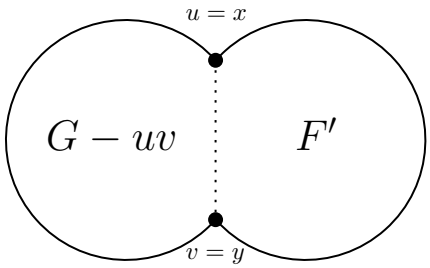


Figure 1 The graph G' from Case 1.

Case 1: *Every acyclic 3-colouring of F' assigns different colours to x and y.*
We construct the graph G' as described above and in Figure 1. We claim that G is a yes-instance of Acyclic 3-Colouring if and only if G' is a yes-instance of Acyclic 3-Colouring.

First suppose that G has an acyclic 3-colouring c. Let c^* be an acyclic 3-colouring of F'. We may assume without loss of generality that $c(u) = c^*(x)$ and $c(v) = c^*(y)$. Hence, we can define a vertex colouring c' of G' with $c'(w) = c(w)$ if $w \in V(G)$ and $c'(w) = c^*(w)$ if $w \in V(F')$. As c and c^* are 3-colourings of G and F', respectively, c' is a 3-colouring of G'. We claim that c' is acyclic. For contradiction, assume that (G', c') has a bichromatic cycle C. If all edges of C are in G or all edges of C are in F', then (G, c) or (F', c^*) has a bichromatic

cycle, which is not possible as c and c^* are acyclic. Hence, at least one edge of C belongs to G and at least one edge of C belongs to F'. This means that C contains both $u = x$ and $v = y$. Recall that G contains the edge uv. Consequently, (G, c) has a bichromatic cycle, namely the cycle induced by $V(C) \cap V(G)$, a contradiction.

Now suppose that G' has an acyclic 3-colouring c'. Let c and c^* be the restrictions of c' to $V(G)$ and $V(F')$, respectively. Then c and c^* are acyclic 3-colourings of $G - uv$ and F', respectively. By our assumption and because c^* is an acyclic 3-colouring of F', we find that $c^*(x) \neq c^*(y)$, or equivalently, $c(u) \neq c(v)$. This means that c is also a 3-colouring of G and c^* is also a 3-colouring of $F' + xy$. We claim that c is acyclic on G. For contradiction, assume that (G, c) has a bichromatic cycle C. As c is an acyclic 3-colouring of $G - uv$, we deduce that C must contain the edge $uv = xy$. As $F' + xy$ has no acyclic 3-colouring by construction and c^* is a 3-colouring of $F' + xy$, we find that $(F' + xy, c^*)$ has a bichromatic cycle D. As c^* is an acyclic 3-colouring of F', this means that D contains the edge $xy = uv$. However, then (G', c') has a bichromatic cycle, namely the cycle induced by $V(C) \cup V(D)$, a contradiction.

Let F^* be the graph obtained from F' by adding a new vertex x' and edges xx' and $x'y$. As $F' + xy$ has girth at least g, we find that F^* and $F^* - x'y$ have girth at least g. As x' has degree 1 in $F^* - x'y$ and F' has an acyclic 3-colouring, $F^* - x'y$ has an acyclic 3-colouring.

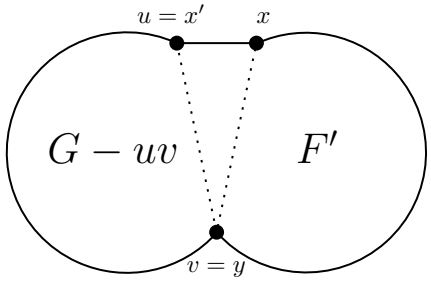


Figure 2 The graph G' from Case 2.

Case 2: *All acyclic 3-colourings of F' assign the same colour to x and y and F^* has no acyclic 3-colouring.*

In this case we let G' be the graph obtained from $G - uv$ and $F^* - x'y$ by identifying u with x' and v with y; see also Figure 2. We claim that G is a yes-instance of Acyclic 3-Colouring if and only if G' is a yes-instance of Acyclic 3-Colouring.

First suppose that G has an acyclic 3-colouring c. Let c^* be an acyclic 3-colouring of $F^* - x'y$. Then the restriction of c^* to F' is an acyclic 3-colouring of F'. By our assumption, it holds therefore that $c^*(x) = c^*(y)$ and thus $c^*(x') \neq c^*(y)$. We may assume without loss of generality that $c(u) = c^*(x')$ and $c(v) = c^*(y)$. Hence, we can define a vertex labelling c' of G' with $c'(w) = c(w)$ if $w \in V(G)$ and $c'(w) = c^*(w)$ if $w \in V(F^*)$. As c and c^* are 3-colourings of G and $F^* - x'y$, respectively, c' is a 3-colouring of G'. We claim that c' is acyclic. For contradiction, assume that (G', c') has a bichromatic cycle C. If the edges of C are all in G or all in $F^* - x'y$, then (G, c) or $(F^* - x'y, c^*)$ has a bichromatic cycle, which is not possible as c and c^* are acyclic. Hence, at least one edge of C belongs to G and at least one edge of C belongs to F'. This means that C contains both $u = x'$ and $v = y$. Recall that G contains the edge uv. Consequently, (G, c) has a bichromatic cycle, namely the cycle induced by $V(C) \cap V(G)$, a contradiction.

Now suppose that G' has an acyclic 3-colouring c'. Let c and c^* be the restrictions of c' to $V(G-uv)$ and $V(F^*-x'y)$, respectively. Then c and c^* are acyclic 3-colourings of $G-uv$ and $F^*-x'y$, respectively. Moreover, the restriction of c' to $V(F')$ is an acyclic 3-colouring of F'. By our assumption, this means that $c'(x)=c'(y)$ and thus $c^*(x')\neq c^*(y)$, or equivalently, $c(u)\neq c(v)$. Consequently, c is also a 3-colouring of G and c^* is also a 3-colouring of F^*. We claim that c is acyclic. For contradiction, assume that (G,c) has a bichromatic cycle C. As c is an acyclic 3-colouring of $G-uv$, we deduce that C must contain the edge $uv=x'y$. As F^* does not have an acyclic 3-colouring by our assumption and c^* is a 3-colouring of F^*, we find that (F^*,c^*) has a bichromatic cycle D. As c^* is an acyclic 3-colouring of $F^*-x'y$, this means that D must contain the edge $x'y=uv$. However, then (G',c') has a bichromatic cycle, namely the cycle induced by $V(C)\cup V(D)$, a contradiction.

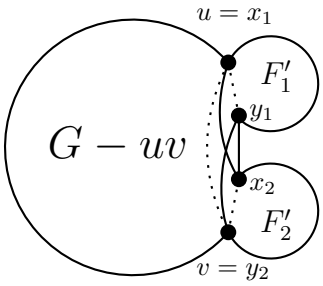


Figure 3 The graph G' with the graph F^+ from Case 3 (before we recursively repeat g times the operation of placing the graph F^+ on the y_1x_2-edge).

Case 3: *All acyclic* 3*-colourings of* F' *assign the same colour to* x *and* y *and* F^* *has an acyclic* 3*-colouring.*

We first construct a new graph F^+ as follows. We take the disjoint union of two copies F_1' and F_2' of F', where we denote the vertices x and y as x_1 and y_1 in F_1' and as x_2 and y_2 in F_2'. We add edges x_1x_2, x_2y_1, and y_1y_2 to $F_1'+F_2'$; see also Figure 3.

We claim that F^+ has an acyclic 3-colouring. First, observe that F^+ is the union of two copies of F^* sharing exactly one edge, namely y_1x_2. That is, $F_1'+x_1x_2,y_1x_2$ and $F_2'+y_1y_2,y_1x_2$ are both isomorphic to F^*. By our assumption on F^*, graphs $F_1'+x_1x_2,x_2y_1$ and $F_2'+y_1y_2,y_1x_2$ have acyclic 3-colourings c_1 and c_2, respectively. By our assumption on F', the restriction of c_1 to F_1' gives x_1,y_1 the same colour and the restriction of c_2 to F_2' gives x_2 and y_2 the same colour. We may assume without loss of generality that c_1 assigns colour 1 to x_1 and y_1 and colour 2 to x_2, and that c_2 assigns colour 2 to x_2 and y_2 and colour 1 to y_1. This yields a 3-colouring c^+ of F^+. We claim that c^+ is acyclic. For contradiction, suppose (F^+,c^+) has a bichromatic cycle C. As the restrictions of c^+ to $F_1'+x_1x_2,y_1x_2$ and $F_2'+y_1y_2,y_1x_2$ (the 3-colourings c_1 and c_2) are acyclic, C must contain the edges x_1x_2 and y_1y_2, so C has the chord y_1x_2. Hence, $(F_1'+x_1x_2,y_1x_2,c_1)$ has a bichromatic cycle on vertex set $(V(C)\setminus V(F_2))\cup\{x_2\}$, a contradiction.

We now essentially reduce to Case 1. Set $x=x_1$, $y=y_2$ and take the graph F^+. We proved above that F^+ has an acyclic 3-colouring. As every acyclic 3-colouring c of F^+ colours x_1 and y_1 alike, c colours $x=x_1$ and $y=y_2$ differently (as y_1x_2 is an edge). Finally, the graph $F^++xy=F^++x_1y_2$ has no acyclic 3-colouring, as for every 3-colouring c of $F^++x_1y_2$, the 4-vertex cycle $x_1x_2y_1y_2x_1$ is bichromatic for $(F^++x_1y_2,c)$. The only difference with Case 1 is that the graph $F^++x_1y_2$ has girth 4 due to the cycle $x_1x_2y_1y_2x_1$ whereas we need the girth to be at least g just as the graph $F'+xy$ in Case 1 has girth g. Hence, before reducing to Case 1, we first recursively repeat g times the operation of placing the graph F^+ on the y_1x_2-edge; note that the size of the resulting graph G' is still polynomial in the size of G.

Case 4: *There exist acyclic* 3*-colourings* c_1 *and* c_2 *of* F' *with* $c_1(x) = c_1(y)$ *and* $c_2(x) \neq c_2(y)$. We first construct a new graph J. We take two disjoint copies F'_1 and F'_2 of F' and identify the two x-vertices with each other and also the two y-vertices with each other. We write $x = x_1 = x_2$ and $y = y_1 = y_2$; see also Figure 4 (left).

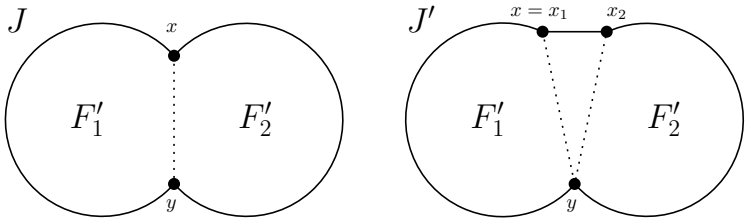


Figure 4 The graph J from Case 4 (left) and the graph J' from Case 4b (right).

We distinguish between two sub-cases.

Case 4a: *J has an acyclic* 3*-colouring.*
Our goal is to reduce either to Case 2 or 3 by using J instead of F'. We first observe that J and $J + xy$ have girth at least g. We also note that $J + xy$ has no acyclic 3-colouring, as otherwise $F' + xy$, being an induced subgraph of $J + xy$, has an acyclic 3-colouring. Hence, in order to reduce to Case 2 or 3 it remains to show that every acyclic 3-colouring of J assigns the same colour to x and y. For contradiction, suppose that J has an acyclic 3-colouring c such that $c(x) \neq c(y)$, say $c(x) = 1$ and $c(y) = 2$. Then in at least one of the two subgraphs F'_1 and F'_2 of J, say F'_1, there exists no 1-2 path from x to y; otherwise (J, c) has a bichromatic cycle formed by the union of the two 1-2-paths, which is not possible as c is acyclic. Let c' be the restriction of c to $V(F'_1)$. Then, as $c(x) = 1$ and $c(y) = 2$, we find that c' is a 3-colouring of $F'_1 + xy$. As there is no 1-2 path from x to y in F'_1, we find that c' is even an acyclic 3-colouring of $F'_1 + xy$, a contradiction (recall that $F' + xy$ has no acyclic 3-colouring by construction).

Case 4b: *J has no acyclic* 3*-colouring.*
By assumption, F' has an acyclic 3-colouring that gives x and y different colours. We first prove a claim.[1]

▷ Claim 1. For every triple (h, i, j) with $\{h, i, j\} = \{1, 2, 3\}$, every acyclic 3-colouring c of F' with $c(x) = c(y) = h$ yields an h-i path and h-j path from x to y.

We prove Claim 1 as follows. For contradiction, suppose that F' has an acyclic 3-colouring c that colours x and y alike, say $c(x) = c(y) = 1$, such that F' contains no 1-2-path or no 1-3-path, say F' contains no 1-2-path from x to y. Then by swapping colours 2 and 3, we obtain another acyclic 3-colouring c' of F' such that F' contains no 1-3-path from x to y. In J we now colour the vertices of F'_1 by c and the vertices of F'_2 by c'. As $c(x) = c(x') = 1$ and $c(y) = c(y') = 1$, this yields a 3-colouring c_J. By assumption, c_J is not acyclic. Hence, (J, c_J) contains a bichromatic cycle C with colours 1 and i for some $i \in \{2, 3\}$. As the restrictions of c_J to F'_1 and F'_2 are acyclic, C must contain at least one vertex of $V(F'_1) \setminus \{x, y\}$ and at least one vertex of $V(F'_2) \setminus \{x, y\}$. Thus C consists of 1-i-paths from x to y in both F'_1 and F'_2. As at least one of these paths is missing in F'_1 or F'_2, this yields a contradiction.

[1] Claim 1 only holds for $k = 3$ and is the reason we cannot generalize Lemma 6 to $k \geq 3$.

We now construct a new graph J' as follows. We take two disjoint copies F_1' and F_2' of F' and still identify y_1 and y_2 as y, but instead of identifying x_1 and x_2 we add an edge between x_1 and x_2; see also Figure 4 (right).
We now prove some more claims that will enable us to reduce to Case 1.

(i) The graphs J' and $J' + x_1y$ have girth at least g.
This follows directly from the fact that respectively, F' and $F' + xy$ have girth at least g.

(ii) The graph $J' + x_1y$ has no acyclic 3-colouring.
This follows directly from the fact that $F' + xy$ is an induced subgraph of $J' + x_1y$ and has no acyclic 3-colouring by construction.

(iii) The graph J' has an acyclic 3-colouring.
This can be seen as follows. By assumption, F' has an acyclic 3-colouring c that gives x and y different colours, say $c(x) = 1$ and $c(y) = 3$. By swapping colours 1 and 2 we obtain an acyclic 3-colouring c' of F' with $c'(x) = 2$ and $c'(y) = 3$. As $c(y) = c'(y) = 3$, this yields a 3-colouring $c_{J'}$ of J'. As the restrictions of $c_{J'}$ to F_1' and F_2' are acyclic, any bichromatic cycle of $(J', c_{J'})$ must pass through x_1, x_2 and y. However, x_1, x_2 and y have colours 1, 2, 3, respectively. Hence, this is not possible.

(iv) Every acyclic 3-colouring of J' gives x_1 and y different colours.
For contradiction, assume J' has an acyclic 3-colouring c that colours x_1 and y alike, say $c(x_1) = c(y) = 1$ and $c(x_2) = 2$. The restriction of c to $V(F_1')$ is an acyclic 3-colouring of F_1' that gives x_1 and y colour 1. Hence, by Claim 1, F_1' contains a 1-2 path from x_1 to y. The restriction of c' to $V(F_2')$ is an acyclic 3-colouring of F_2' that gives x_2 colour 2 and y colour 1. Then F_2' must contain a 1-2 path from x_2 to y; otherwise we found an acyclic 3-colouring of $F_2' + x_2y$, which is not possible by construction. The two 1-2 paths now form, with the edge x_1x_2, a bichromatic cycle of (J', c). As c is acyclic, this is not possible.

By (i)-(iv) we may take J' with x_1 and y instead of F' with x and y and reduce to Case 1. ◀

The *line graph* of a graph G has vertex set $E(G)$ and an edge between two vertices e and f if and only if e and f share an end-vertex of G. In Lemma 7 we modify the construction of [3] for line graphs from $k = 3$ to $k \geq 3$. In Lemma 8 we give a new construction for proving hardness when k is part of the input.

▶ **Lemma 7.** *For every $k \geq 3$,* Acyclic k-Colouring *is NP-complete for line graphs.*

Proof. For an integer $k \geq 1$, a *k-edge colouring* of a graph $G = (V, E)$ is a mapping $c : E \to \{1, \ldots, k\}$ such that $c(e) \neq c(f)$ whenever the edges e and f share an end-vertex. A *colour class* consists of all edges of G that are mapped by c to a specific colour i. The pair (G, c) has a *bichromatic* cycle C if C is a cycle of G with its edges coloured by two alternating colours. The notion of a *bichromatic* path is defined in a similar manner. We say that c is *acyclic* if (G, c) has no bichromatic cycle. For a fixed integer $k \geq 1$, the Acyclic k-Edge Colouring problem is to decide if a given graph has an acyclic k-edge colouring. Alon and Zaks proved that Acyclic 3-Edge Colouring is NP-complete for multigraphs. We note that a graph has an acyclic k-edge colouring if and only if its line graph has an acyclic k-colouring. Hence, it remains to generalize the construction of Alon and Zaks [3] from $k = 3$ to $k \geq 3$. Our main tool is the gadget graph F_k, depicted in Figure 5, about which we prove the following two claims.

- **(i)** *The edges of F_k can be coloured acyclically using k colours, with no bichromatic path between v_1 and v_{14}.*
- **(ii)** *Every acyclic k-edge colouring of F_k using k colours assigns e_1 and e_2 the same colour.*

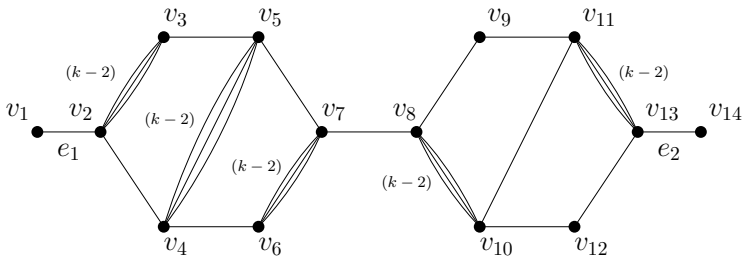


Figure 5 The gadget multigraph F_k. The labels on edges are multiplicities.

We first prove (ii). We assume, without loss of generality, that v_1v_2 is coloured by 1, v_2v_4 by 2 and the edges between v_2 and v_3 by colours $3, \ldots, k$. The edge v_3v_5 has to be coloured by 1, otherwise we have a bichromatic cycle on $v_2v_3v_5v_4$. This necessarily implies that

- the edges between v_4 and v_5 are coloured by $3, \ldots, k$,
- the edge v_5v_7 is coloured by 2,
- the edge v_4v_6 is coloured by 1,
- the edges between v_6 and v_7 are coloured by $3, \ldots, k$, and
- the edge v_7v_8 is coloured by 1.

Now assume that the edge v_8v_9 is coloured by $a \in \{2, \ldots, k\}$ and the edges between v_8 and v_{10} by colours from the set A, where $A = \{2, \ldots, k\} \setminus a$. The edge $v_{10}v_{11}$ is either coloured a or 1. However, if it is coloured 1, v_9v_{11} is assigned a colour $b \in A$ and necessarily we have either a bichromatic cycle on $v_8v_9v_{11}v_{13}v_{12}v_{10}$, coloured by b and a, or a bichromatic cycle on $v_{10}v_{11}v_{13}v_{12}$, coloured by a and 1. Thus $v_{10}v_{11}$ is coloured by a. To prevent a bichromatic cycle on $v_8v_9v_{11}v_{10}$, the edge v_9v_{11} is assigned colour 1. The rest of the colouring is now determined as $v_{10}v_{12}$ has to be coloured by 1, the edges between v_{11} and v_{13} by A, $v_{12}v_{13}$ by a, and $v_{13}v_{14}$ by 1. We then have a k-colouring with no bichromatic cycles of size at least 3 in F_k for every possible choice of a. This proves that v_1v_2 and $v_{13}v_{14}$ are coloured alike under every acyclic k-edge colouring.

We prove (i) by choosing a different from 2. Then there is no bichromatic path between v_1 and v_{14}.

We now reduce from k-Edge-Colouring to Acyclic k-Edge Colouring as follows. Given an instance G of k-edge Colouring we construct an instance G' of Acyclic k-Edge Colouring by replacing each edge uv in G by a copy of F_k where u is identified with v_1 and v is identified with v_{14}.

If G' has an acyclic k-edge colouring c' then we obtain a k-edge colouring c of G by setting $c(uv) = c'(e_1)$ where e_1 belongs to the gadget F_k corresponding to the edge uv. If G has a k-edge colouring c then we obtain an acyclic k-edge colouring c' of G' by setting $c'(e_1) = c(uv)$ where e_1 belongs to the gadget corresponding to the edge uv. The remainder of each gadget F_k can then be coloured as described above. ◀

In our next result, k is part of the input.

▶ **Lemma 8.** Acyclic Colouring *is NP-complete for* $(19P_1, 3P_3, 2P_5)$*-free graphs.*

Proof. We reduce from 3-Colouring with maximum degree 4 which is known to be NP-complete [26]. Let G be an instance of 3-Colouring with $|V(G)| = n$ vertices and maximum degree 4. We will construct an instance G' of Acyclic Colouring where $k = 4n$. Our vertex gadget is built from two k-cliques, J_0 and J_1, with a matching between them. We number the vertices of each of the cliques 0 to $k-1$. The matching we insert into the graph

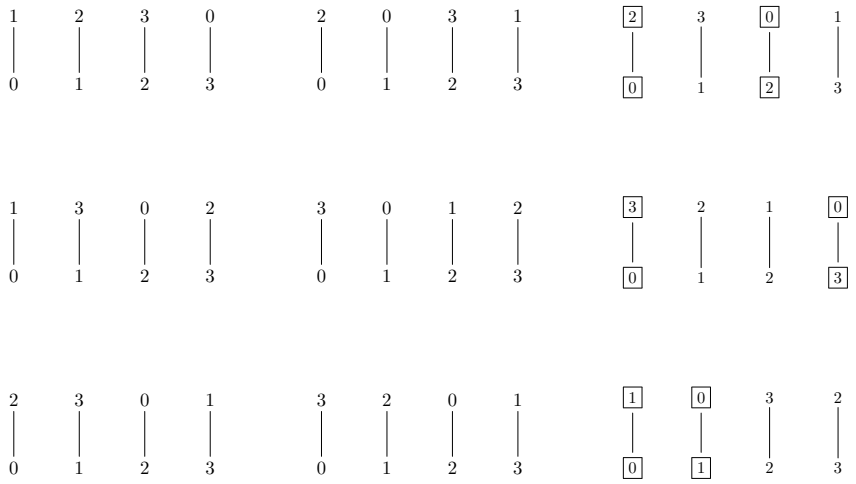


Figure 6 Acyclic colourings in the proof of Lemma 8 for a vertex representing one of the three colours (left and middle). Sample failures for an acyclic colouring from other permutations of $(0, 1, 2, 3)$ together with a failure cycle (right). Note that each row of quadruples is joined in a clique.

is $(0, 0), \ldots, (k-1, k-1)$. In addition, we place an edge from i in J_0 to j in J_1 if and only if $\lfloor i/4 \rfloor < \lfloor j/4 \rfloor$. Suppose that some assignment of colours is given to J_0. By recolouring, we assume it is the identity colouring of i to i on J_0. Then the possible acyclic k-colourings of vertices $(\lfloor i/4 \rfloor + 0, \lfloor i/4 \rfloor + 1, \lfloor i/4 \rfloor + 2, \lfloor i/4 \rfloor + 3)$ in J_1 are

$$
\begin{aligned}
&(\lfloor i/4 \rfloor + 1, \lfloor i/4 \rfloor + 2, \lfloor i/4 \rfloor + 3, \lfloor i/4 \rfloor + 0),\\
&(\lfloor i/4 \rfloor + 1, \lfloor i/4 \rfloor + 3, \lfloor i/4 \rfloor + 0, \lfloor i/4 \rfloor + 2),\\
&(\lfloor i/4 \rfloor + 2, \lfloor i/4 \rfloor + 3, \lfloor i/4 \rfloor + 1, \lfloor i/4 \rfloor + 0),\\
&(\lfloor i/4 \rfloor + 2, \lfloor i/4 \rfloor + 0, \lfloor i/4 \rfloor + 3, \lfloor i/4 \rfloor + 1),\\
&(\lfloor i/4 \rfloor + 3, \lfloor i/4 \rfloor + 0, \lfloor i/4 \rfloor + 1, \lfloor i/4 \rfloor + 2),\\
&(\lfloor i/4 \rfloor + 3, \lfloor i/4 \rfloor + 2, \lfloor i/4 \rfloor + 0, \lfloor i/4 \rfloor + 1).
\end{aligned}
$$

They are built from the permutations of $(0, 1, 2, 3)$ that do not contain a transposition. We draw all of them, to demonstrate it is not an acyclic colouring, in Figure 6 (keep in mind that vertices in a row are joined in a clique).

In our reduction, the first two acyclic k-colourings will represent colour 1, the second two colour 2 and the third two colour 3 of the sought 3-colouring of G. To force similarly coloured copies of J_0 we add a new k-clique J_2 with edges from i in J_0 to j in J_2 if and only if $i < j$. To prevent the existence of bichromatic cycles in our later construction, we add a k-clique J_3 with edges from i in J_2 to j in J_3 if and only if $i < j$. This enforces that in any acyclic k-colouring of G', the i-th vertices (where $i \in \{0, \ldots, k-1\}$) in cliques J_0, J_2, J_3 would have the same colour. Therefore, by the way we placed the edges between different cliques from $\{J_0, J_2, J_3\}$, there is no bichromatic path with vertices from more than one clique in $\{J_0, J_2, J_3\}$.

We now construct edge gadgets. We take another two k-cliques to join J_2, say J_4 and J_5. We will want them coloured exactly like J_0, so for i in J_2 and j in J_4 or J_5, where $i < j$, we will add an edge ij. Suppose we have an edge in G between p and q for some $p, q \in \{0, \ldots, n-1\}$. Then we place an edge from the vertex $4p$ in J_1 to $4q+1$ in J_3 and from $4q$ in J_1 to $4p+1$ in J_3 (recall that $p, q \in \{0, \ldots, n-1\}$ and cliques J_1 and J_3 are of size $4n$, so these edges are well defined). See Figure 7. Now we place an edge from $4p$ in J_1 to $4q+2$ in J_4 and of $4q$ in J_1 to $4p+2$ in J_4. Finally, we place an edge from $4p$ in J_1 to $4q+3$ in J_5 and from $4q$ in J_1 to $4p+3$ in J_5. This concludes the construction for the edge pq in $E(G)$.

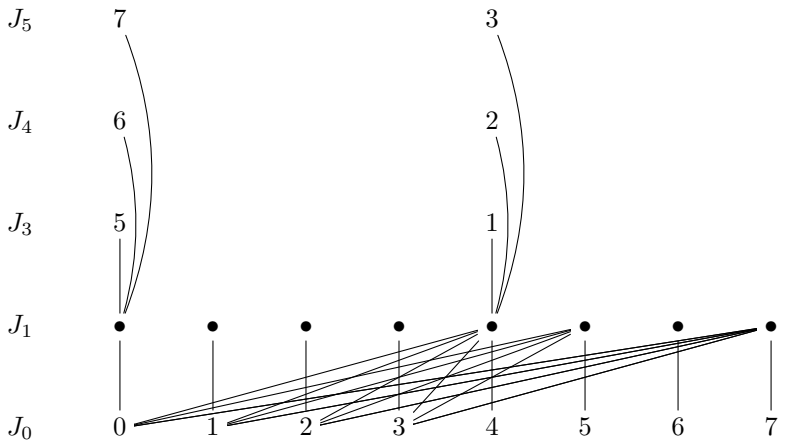


Figure 7 Edge construction in the proof of Lemma 8 between vertices 0 and 1 of G. Everything in a row is joined in a clique. Edges are omitted between J_0 and J_3, J_4, J_5, though they enforce the colouring.

Suppose we have an edge $rs \in E(G)$ so that $\{p,q\} \cap \{r,s\} = \emptyset$. Then we build a gadget for rs using the same additional three cliques that we used for the edge pq. However, if we have edges with a common endpoint, e.g. $pq, ps \in E(G)$, then by adding the edges from $4p$ in J_1 to $4q+1$ in J_3, from $4q$ in J_1 to $4p+1$ in J_3, from $4p$ in J_1 to $4s+1$ in J_3, and from $4s$ in J_1 to $4p+1$ in J_3 we introduce new 4-cycles, one of them induced by the vertices $4q$ and $4p$ in J_1 and $4p+1$ and $4s+1$ in J_3. To avoid this, we add three additional k-cliques to build the gadget for ps. By Vizing's Theorem [49], we obtain in polynomial time a 5-edge colouring of G (as G has maximum degree 4). Using this 5-edge colouring, we build gadgets for all the edges with at most $5 \times 3 = 15$ additional k-cliques (we use 3 additional cliques for each colour class).

The clique structure of G' is drawn in Figure 8. As G' consists of at most 18 cliques, G' is $19P_1$-free. Furthermore, any induced linear forest where each connected component has size at least 3 contains vertices in at most five cliques. Hence G' is $(3P_3, 2P_5)$-free. It remains to prove that G has a 3-colouring if and only if G' has an acyclic k-colouring.

First, suppose that G' has an acyclic k-colouring c'. Then each k-clique of G' has to use each colour exactly once. We can permute colours so that vertex i in J_0 (where $0 \leq i \leq 4n-1$) has colour i. It follows from the connections between cliques that the i-th vertices in cliques $J_2, \ldots, J_{17}$ also have colour i and the vertices $4j, 4j+1, 4j+2, 4j+3$, $(0 \leq j \leq n-1)$ in J_1 have colours from the set $\{4j, 4j+1, 4j+2, 4j+3\}$. For each vertex i in G, set $c(i) = 1$ if the colours of $(4i, 4i+1, 4i+2, 4i+3)$ in J_1 under c' correspond to one of the first two possible colourings (listed above); set $c(i) = 2$ if it corresponds to one of the second two possible colourings; set $c(i) = 3$ if it corresponds to one of the last two colourings. We claim that c is a 3-colouring of G. Suppose that pq is an edge in G with edge gadget using cliques J_3, J_4, J_5. Since c' is acyclic and c' is identity on J_3, we have $c'(4p) \neq 4p+1$ in J_1 or $c'(4q) \neq 4q+1$ in J_1. Both $4p$ and $4q$ are the first vertices of the respective quadruples, so p and q are not both coloured 1. Similarly, the edges going between cliques J_1 and J_4 ensure that they are not both coloured 2 and the edges going between cliques J_1 and J_5 ensure that they are not both coloured 3. Hence, $c(p) \neq c(q)$ and c is a 3-colouring of G.

Now suppose G has a 3-colouring c. We construct a labelling c' of G' where we colour each quadruple in J_1 corresponding to a vertex of G by the first of each pair of colourings listed in the table for each of the three colours, respectively. The labelling c' in other cliques of G' is the identity. By the construction of G' and particularly by the properties of edge gadgets in G', we find that c' is a k-colouring of G'.

Finally, we need to verify that c' is acyclic. We will begin with bichromatic cycles between two cliques. No bichromatic cycle can appear in J_0 and J_1 forming the vertex gadget. This is due to the edges from the former to the latter always pointing to a higher number (or the same but here we chose a 3-colouring to avoid such situation). A similar explanation works for all the clique pairs $(0,2),(2,3),\ldots,(2,17)$ in Figure 8. The last possibility is a bichromatic cycle formed through J_1 from one of the cliques J_3 to J_{17}. However, such a cycle would have to pass through an actual edge gadget (where it is forbidden by the 3-colouring) or through vertices in different edge gadgets, where it must form a cycle with four colours. Now we need to consider bichromatic cycles passing through three or more cliques, but they would have to involve a bichromatic path through J_0, J_2, J_3 which is not possible. This completes the proof. ◀

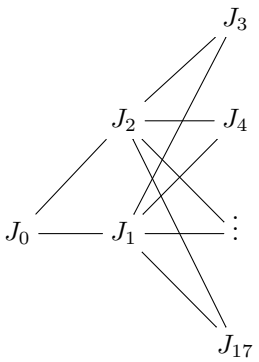


Figure 8 Connections between cliques in the construction from the proof of Lemma 8.

We combine the above results with results of Coleman and Cai [14] and Lyons [43] to prove Theorem 1.

▶ **Theorem 1 (restated).** *Let H be a graph. For the class of H-free graphs it holds that:*

(i) Acyclic Colouring *is polynomial-time solvable if $H \subseteq_i P_4$ and NP-complete if H is not a forest or $H \supseteq_i 19P_1, 3P_3, 2P_5$ or P_{11};*

(ii) *For every $k \geq 3$,* Acyclic k-Colouring *is polynomial-time solvable if H is a linear forest and NP-complete otherwise.*

Proof. We first prove (ii). First suppose that H contains an induced cycle C_p. If $p = 3$, then we use the result of Coleman and Cai [14], who proved that for every $k \geq 3$, Acyclic k-Colouring is NP-complete for bipartite graphs. Suppose that $p \geq 3$. If $k = 3$, then we let $g = p + 1$ and use Lemma 6. If $k \geq 4$, we reduce from Acyclic 3-Colouring for graphs of girth $p + 1$ by adding a dominating clique of size $k - 3$. Now assume H has no cycle so H is a forest. If H has a vertex of degree at least 3, then H has an induced $K_{1,3}$. As every line graph is $K_{1,3}$-free, we can use Lemma 7. Otherwise H is a linear forest and we use Corollary 5.

We now prove (i). Due to (ii), we may assume that H is a linear forest. If $H \subseteq_i P_4$, then we use the result of Lyons [43] that states that Acyclic Colouring is polynomial-time solvable for P_4-free graphs. If $H \supseteq_i 19P_1, 3P_3, 2P_5$ or P_{11}, then we use Lemma 8. ◀

Figure 9 The gadget replacing edges uv (on the left) and its natural star 3-colouring (on the right) in the proof of Lemma 9.

4 Star Colouring

In this section we prove Theorem 2. We first prove the following lemma.

▶ **Lemma 9.** *Let H be a graph with an* even *cycle. Then, for every $k \geq 3$,* Star k-Colouring *is NP-complete for H-free graphs.*

Proof. We reduce from 3-Colouring for graphs of girth at least $p+1$. Given an instance G of this problem, we construct an instance G' of Star 3-Colouring as follows. Take three vertex disjoint copies of P_3 and form a triangle using one endpoint of each; see Figure 9. Replace each edge uv in G by this gadget with u and v identified with the non-adjacent endpoints of two paths. Then G' is C_p-free since, aside from triangles, the construction cannot introduce any cycle shorter than those present in G.

We first show that any star 3-colouring of G' colours u and v differently. Assume not, their neighbours must be coloured differently since otherwise any 3-colouring of the remainder of the gadget will result in a bichromatic P_4. Without loss of generality, assume that u and v are coloured 1, the neighbour u' of u is coloured 2 and the neighbour v' of v is coloured 3. Let x be the neighbour of u' in the triangle and y the neighbour of v' in the triangle. Neither x or y can be coloured 1 since this will result in a bichromatic P_4. Therefore x is coloured 3, y is coloured 2 and the third vertex z of the triangle is coloured 1. This is a contradiction since we have a bichromatic P_4 on the vertices u', x, y, v'. Therefore, we obtain a 3-colouring c of G by setting $c(v) = c'(v)$ for some star 3-colouring c' of G'.

We extend a given 3-colouring of G to a star 3-colouring of G', by locally star 3-colouring as in the right hand side of Figure 9 (or automorphically). Hence, G is 3-colourable if and only if G' is star 3-colourable.

We obtain NP-completeness for $k \geq 4$ by a reduction from Star 3-Colouring for C_p-free graphs by adding a dominating clique of size $k-3$. ◀

In Lemma 10 we extend the recent result of Lei et al. [38] from $k = 3$ to $k \geq 3$. In Lemma 11 we show a result where k is part of the input. A graph is *co-bipartite* if it is the complement of a bipartite graph.

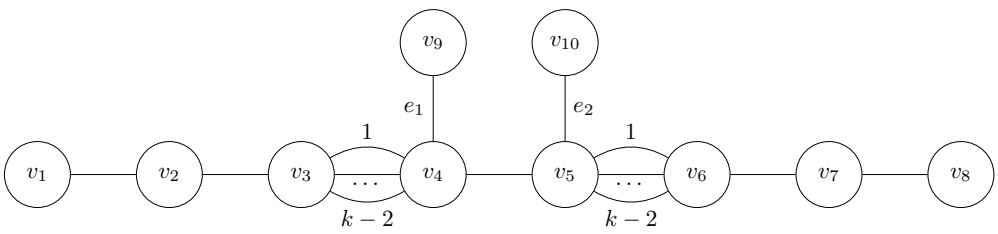


Figure 10 The gadget F_k in the proof of Lemma 10.

▶ **Lemma 10.** *For every $k \geq 3$,* Star k-Colouring *is NP-complete for line graphs.*

Proof. Recall that for an integer $k \geq 1$, a *k-edge colouring* of a graph $G = (V, E)$ is a mapping $c : E \to \{1, \ldots, k\}$ such that $c(e) \neq c(f)$ whenever the edges e and f share an end-vertex. Recall also that the notions of a colour class and bichromatic subgraph for colourings has its natural analogue for edge colourings. An edge k-colouring c is a *star* k-edge colouring if the union of any two colour classes induces a star forest. For a fixed integer $k \geq 1$, the Star k-Edge Colouring problem is to decide if a given graph has an star k-edge colouring. Lei et al. [38] proved that Star 3-Edge Colouring is NP-complete. Dvořák et al. [16] observed that a graph has a star k-edge colouring if and only if its line graph has a star k-colouring. Hence, it suffices to follow the proof of Lei et al.[38] in order to generalize the case $k = 3$ to $k \geq 3$. As such, we give a reduction from k-Edge Colouring to Star k-Edge Colouring which makes use of the gadget F_k in Figure 10. First we consider separately the case where the edges $e_1 = v_4v_9$ and $e_2 = v_5v_{10}$ are coloured alike and the case where they are coloured differently to show that in any star k-edge colouring of the gadget F_k shown in Figure 10, v_1v_2 and v_7v_8 are assigned the same colour.

Assume $c(e_1) = c(e_2) = 1$. We may then assume that the edge v_4v_5 is assigned colour 2 and the remaining $k - 2$ colours are used for the multiple edges v_3v_4 and v_5v_6. The edge v_2v_3, and similarly v_6v_7, must then be assigned colour 1 to avoid a bichromatic P_5 on the vertices $\{v_2, v_3, v_4, v_5, v_6\}$ using any two of the multiple edges in a single colour. The edge v_1v_2, and similarly v_7v_8 must then be assigned colour 2 to avoid a bichromatic P_5 on the vertices $\{v_1, v_2, v_3, v_4, v_9\}$.

Next assume e_1 and e_2 are coloured differently. Without loss of generality, let $c(e_1) = 1$, $c(e_2) = 2$ and $c(v_4v_5) = 3$. The multiple edges v_3v_4 must then be assigned colours 2 and $4 \ldots k$ and v_5v_6 colour 1 and colours $4 \ldots k$. To avoid a bichromatic P_5 on the vertices $\{v_2, v_3, v_4, v_5, v_6\}$, v_2v_3 must be coloured 1. Similarly, v_6v_7 must be assigned colour 2. Finally, to avoid a bichromatic P_5 on the vertices $\{v_1, v_2, v_3, v_4, v_9\}$, v_1v_2 must be coloured 3. By a similar argument, v_7v_8 must also be coloured 3, hence v_1v_2 and v_7v_8 must be coloured alike.

We can then replace every edge e in some instance G of k-Edge-Colouring by a copy of F_k, identifying its endpoints with v_1 and v_8, to obtain an instance G' of Star k-Edge-Colouring. If G is k-edge-colourable we can star k-edge-colour G' by setting $c'(v_1v_2) = c'(v_7v_8) = c(e)$. If G' is star k-edge-colourable, we obtain a k-edge-colouring of G by setting $c(e) = c'(v_1v_2)$. ◀

We now let k be part of the input. The *complement* of a graph G is the graph $\overline{G}$ with vertex set $V(G)$ and an edge between two vertices u and v if and only if $uv \notin E(G)$. A k-colouring of G can be seen as a partition of $V(G)$ into k independent sets. Hence, a k-colouring of G corresponds to a *clique-covering* of $\overline{G}$, which is a partition of $V(\overline{G}) = V(G)$ into k cliques. A graph is *co-bipartite* if it is the complement of a bipartite graph.

▶ **Lemma 11.** *Star Colouring is NP-complete for co-bipartite graphs.*

Proof. We show that finding an optimal star colouring of a co-bipartite graph G is equivalent to finding a maximum balanced biclique in its complement $\overline{G}$. An optimal star colouring of G corresponds to an optimal clique-covering of $\overline{G}$ such that the graph induced by the vertices of any two cliques in the covering partition is $\overline{P_4} = P_4$-free and $\overline{C_4} = 2P_2$-free. Since $\overline{G}$ is triangle-free, the clique-covering number of $\overline{G}$ is $n - M$ where n is the number of vertices of G and M is the number of edges in a maximum matching such that no two edges induce either $2P_2$ or P_4. Since $\overline{G}$ is bipartite, a maximum matching of this form is a maximum balanced biclique. It is NP-complete to find the maximum size of a balanced biclique in a bipartite graph [26]. Therefore Star Colouring is NP-complete for co-bipartite graphs. ◀

We combine the above results with results of Albertson et al. [1] and Lyons [43] to prove Theorem 2.

▶ **Theorem 2** (restated). *Let H be a graph. For the class of H-free graphs it holds that:*

(i) Star Colouring *is polynomial-time solvable if $H \subseteq_i P_4$ and NP-complete for any $H \neq 2P_2$.*

(ii) *For every $k \geq 3$,* Star k-Colouring *is polynomial-time solvable if H is a linear forest and NP-complete otherwise.*

Proof. We first prove (ii). First suppose that H contains an induced odd cycle. Then the class of bipartite graphs is contained in the class of H-free graphs. Lemma 7.1 in Albertson et al. [1] implies, together with the fact that for every $k \geq 3$, k-Colouring is NP-complete, that for every $k \geq 3$, Star k-Colouring is NP-complete for bipartite graphs. If H contains an induced even cycle, then we use Lemma 9. Now assume H has no cycle, so H is a forest. If H contains a vertex of degree at least 3, then H contains an induced $K_{1,3}$. As every line graph is $K_{1,3}$-free, we can use Lemma 10. Otherwise H is a linear forest, in which case we use Corollary 5.

We now prove (i). Due to (ii), we may assume that H is a linear forest. If $H \subseteq_i P_4$, then we use the result of Lyons [43] that states that Star Colouring is polynomial-time solvable for P_4-free graphs. If $3P_1 \subseteq_i H$, then we use Lemma 11 after observing that co-bipartite graphs are $3P_1$-free. Otherwise $H = 2P_2$, but this case was excluded from the statement of the theorem. ◀

5 Injective Colouring

In this section we prove Theorem 3. We first show three lemmas.

▶ **Lemma 12.** *For every $k \geq 4$,* Injective k-Colouring *is NP-complete for C_3-free graphs.*

Proof. We reduce from Injective k-Colouring. Given an instance G of Injective k-Colouring, construct an instance G' of Injective k-Colouring for triangle-free graphs as follows. For each edge uv of G, remove the edge uv and add two vertices u'_v adjacent to u and v'_u adjacent to v. Next, place an independent set of $k-2$ vertices adjacent to both u'_v and v'_u. Then G' is triangle-free since the edge gadget described is triangle-free, any two vertices of G are now at distance at least 4 and no vertex not belonging to an edge gadget has two adjacent neighbours belonging to edge gadgets. We claim that G' has an injective k-colouring if and only if G has an injective k-colouring.

First assume that G has an injective k-colouring c. Colour the vertices of G' corresponding to vertices of G as they are coloured by c. We can extend this to an injective k-colouring c' of G' by considering the gadget corresponding to each edge uv of G. Set $c'(u'_v) = c'(v)$ and $c'(v'_u) = c'(u)$. We can now assign the remaining $k-2$ colours to the vertices of the independent sets. Clearly c' creates no bichromatic P_3 involving vertices in at most one edge gadget. Assume there exists a bichromatic P_3 involving vertices in more than one edge gadget, then this path must consist of a vertex u of G together with two gadget vertices u'_v and u'_w which are coloured alike. This is a contradiction since it implies the existence of a bichromatic path v, u, w in G.

Now assume that G' has an injective k-colouring c'. Let c be the restriction of c' to those vertices of G' which correspond to vertices of G. To see that c is an injective colouring of G, note that we must have $c'(u'_v) = c'(v)$ and $c'(v'_u) = c'(u)$ for any edge uv. Therefore, if c induces a bichromatic P_3 on u, v, w, then c' induces a bichromatic P_3 on v'_u, v, v'_w. We conclude that c is injective. ◀

In our next two results, k is part of the input.

▶ **Lemma 13.** Injective Colouring *is polynomial-time solvable for P_4-free graphs and $(P_1 + P_3)$-free graphs.*

Proof. Since connected P_4-free graphs have diameter at most 2, no two vertices can be coloured alike in an injective colouring. Hence the injective chromatic number of a P_4-free graph is equal to the number of its vertices.

We now consider $(P_1 + P_3)$-free graphs. First, note that an injective colouring of G is equivalent to a clique-covering of its complement $\overline{G}$ such that the graph induced by the vertices of the union of any two clique classes is $(P_1 + P_2)$-free (as $\overline{P_3} = P_1 + P_2$). Since G is $(P_1 + P_3)$-free, $\overline{G}$ is $\overline{P_1 + P_3}$-free. By a result of Olariu [46], each connected component of $\overline{G}$ is either triangle-free or complete multi-partite. Let $D_1, \ldots, D_p$ be the vertex sets of the connected components of $\overline{G}$ for some $p \geq 1$. Then in G, every D_i is complete to every D_j. Hence, the injective chromatic number of G is the sum of the injective chromatic numbers of the subgraphs G_i induced by D_i ($i \in \{1, \ldots, p\}$). As such, it remains to determine the injective chromatic number of each G_i, which we do below.

Let $1 \leq i \leq p$. If $\overline{G_i}$ is complete multi-partite, then G_i is a disjoint union of cliques and its injective chromatic number is equal to the size of its largest connected component. In the other case, $\overline{G_i}$ is triangle-free. Then each clique class in a clique-covering has size at most 2, and any clique class of size 2 must dominate the remaining vertices of $\overline{G_i}$ to avoid a bichromatic $P_1 + P_2$. Thus, the clique-covering is a matching, each edge of which dominates $\overline{G_i}$, together with the remaining vertices which each form clique classes of size 1. Therefore, we find an optimal $(P_1 + P_2)$-free clique-covering of $\overline{G}$ by finding a maximum matching in the graph consisting of dominating edges of $\overline{G_i}$. The injective chromatic number of G_i is then the number of vertices of G_i minus the number of edges in such a matching. ◀

▶ **Lemma 14.** Injective Colouring *is NP-complete for $6P_1$-free graphs.*

Proof. We first show that Colouring remains NP-complete given a partition of the instance G into four cliques. The Clique Covering problem is NP-complete for planar graphs [37]. A 4-colouring of a planar graph G can be found in quadratic time [47] and gives a partition of $\overline{G}$ into four cliques. Hence, given a planar instance G of clique-covering, we construct an instance $(\overline{G}, c)$ of Colouring where c is a 4-colouring of G such that the chromatic number of $\overline{G}$ is equal to the clique-covering number of G.

We now give a reduction from this problem to Injective Colouring for $6P_1$-free graphs. Given a graph G and a partition c into four cliques $C^1 \ldots C^4$, let G' be the graph obtained from G by deleting those vertices with no neighbours outside of their own clique C^i. Then G can be coloured with k colours if and only if G' can be coloured with k colours and the maximum size of a clique in the partition c of G is at most k. To see this, note that the vertices of $G \setminus G'$ then have degree at most $k-1$, hence we can greedily colour these vertices given a k-colouring of G'.

This instance (G', c) of Colouring given a partition of G' into four cliques can then be transformed in polynomial time to an instance G'' of Injective Colouring as follows. Add a fifth clique C^0 with one vertex v_e for each edge $e = xy$ in G' which has endpoints in two different cliques of c. For each such edge, replace e by two edges xv_e and yv_e. G' has a colouring with k colours if and only if G'' has an injective colouring with $k + m$ colours where m is the number of edges in G with endpoints in different cliques. To see this, note that in any injective colouring of G'', the set of colours used in C^0 is disjoint from the set of those used in the cliques $C^1 \ldots C^4$. Therefore if G'' can be injective coloured with $m + k$ colours then G' can be coloured with k colours. On the other hand, colour the vertices of

$C^1 \dots C^4$ as they are coloured in some k colouring of G' and C^0 with m further colours. This is an injective colouring of G'' since any induced P_3 contains either two vertices of C^1 or one vertex of C^0 and two vertices adjacent in G'. In either case the path must be coloured with three distinct colours. This implies that G'' has an injective colouring with $k+m$ colours if and only if G' has a colouring with k colours. ◀

We combine the above results with results of Bodlaender et al. [7] and Mahdian [44] to prove Theorem 3.

▶ **Theorem 3** (restated). *Let H be a graph. For the class of H-free graphs it holds that:*

(i) Injective Colouring *is polynomial-time solvable if $H \subseteq_i P_4$ or $H \subseteq_i P_1 + P_3$ and NP-complete if H is not a forest or $2P_2 \subseteq_i H$ or $6P_1 \subseteq_i H$.*

(ii) *For every $k \geq 4$,* Injective k-Colouring *is polynomial-time solvable if H is a linear forest and NP-complete otherwise.*

Proof. We first prove (ii). If $C_3 \subseteq_i H$, then we use Lemma 12. Now suppose $C_p \subseteq_i H$ for some $p \geq 4$. Mahdian [44] proved that for every $g \geq 4$ and $k \geq 4$, Injective k-Colouring is NP-complete for line graphs of bipartite graphs of girth at least g. These graphs may not be C_3-free but for $g \geq p+1$ they are C_p-free. Now assume H has no cycle, so H is a forest. If H contains a vertex of degree at least 3, then H contains an induced $K_{1,3}$. As every line graph is $K_{1,3}$-free, we can use the aforementioned result of Mahdian [44] again. Otherwise H is a linear forest, in which case we use Corollary 5.

We now prove (i). Due to (ii), we may assume that H is a linear forest. If $H \subseteq_i P_4$ or $H \subseteq_i P_1 + P_3$, then we use Lemma 13. Now suppose that $2P_2 \subseteq_i H$. Then the class of $(2P_2, C_4, C_5)$-free graphs (split graphs) are contained in the class of H-free graphs. Recall that Bodlaender et al. [7] proved that Injective Colouring is NP-complete for split graphs. If $6P_1 \subseteq_i H$, then we use Lemma 14. ◀

6 Conclusions

Our complexity study led to three complete and three almost complete complexity classifications (Theorems 1–3). Due to our systematic approach we were able to identify some interesting open questions for future research, which we collect below.

▶ **Open Problem 1.** *For $k \geq 4$ and $g \geq 4$, determine the complexity of* Acyclic k-Colouring *for graphs of girth at least g.*

For solving Open Problem 1 it would be helpful to have a better understanding of the structure of the critical graphs used in the proof of Lemma 6. We also aim to prove analogous results for the other two problems.

▶ **Open Problem 2.** *For every $g \geq 4$, determine the complexities of* Star Colouring *and* Injective Colouring *for graphs of girth at least g.*

Naturally we also aim to settle the remaining open cases for our three problems in Table 1. In particular, there is one case left for Star Colouring.

▶ **Open Problem 3.** *Determine the complexity of* Star Colouring *for $2P_2$-free graphs.*

Recall that the other two problems and also Colouring are all NP-complete for $2P_2$-free graphs. However, none of the hardness constructions carry over to Star Colouring. In this context, the next open problem for split graphs ($(2P_2, C_4, C_5)$-free graphs) is also interesting.

▶ **Open Problem 4.** *Determine the complexity of* Star Colouring *for split graphs.*

We proved that Injective Colouring is NP-complete for triangle-free graphs, but the following problem is still open.

▶ **Open Problem 5.** *Determine the complexity of* Injective Colouring *for bipartite graphs.*

Jin et al. [33] proved that the variant of Injective Colouring where adjacent vertices may be coloured alike is NP-complete for bipartite graphs. However, their hardness construction does not carry over to Injective Colouring.

Finally, we recall that Injective Colouring is also known as $L(1,1)$-labelling. The general distance constrained labelling problem $L(a_1,\ldots,a_p)$-Labelling is to decide if a graph G has a labelling $c : V(G) \to \{1,\ldots,k\}$ for some integer $k \geq 1$ such that for every $i \in \{1,\ldots,p\}$, $|c(u) - c(v)| \geq a_i$ whenever u and v are two vertices of distance i in G. If k is fixed, we write $L(a_1,\ldots,a_p)$-k-Labelling instead. By applying Theorem 4 we obtain the following result.

▶ **Theorem 15.** *For all $k \geq 1, a_1 \geq 1,\ldots,a_k \geq 1$, the $L(a_1,\ldots,a_p)$-k-*Labelling *problem is polynomial-time solvable for H-free graphs if H is a linear forest.*

We leave a more detailed and systematic complexity study of problems in this framework for future work (see, for example, [11, 23, 24] for some complexity results for both general graphs and special graph classes).

References

1 Michael O. Albertson, Glenn G. Chappell, Henry A. Kierstead, André Kündgen, and Radhika Ramamurthi. Coloring with no 2-colored P_4's. *Electronic Journal of Combinatorics*, 11, 2004.

2 Noga Alon, Colin McDiarmid, and Bruce A. Reed. Acyclic coloring of graphs. *Random Structures and Algorithms*, 2:277–288, 1991.

3 Noga Alon and Ayal Zaks. Algorithmic aspects of acyclic edge colorings. *Algorithmica*, 32:611–614, 2002.

4 Patrizio Angelini and Fabrizio Frati. Acyclically 3-colorable planar graphs. *Journal of Combinatorial Optimization*, 24:116–130, 2012.

5 Aistis Atminas, Vadim V. Lozin, and Igor Razgon. Linear time algorithm for computing a small biclique in graphs without long induced paths. *Proceedings of SWAT 2012, LNCS*, 7357:142–152, 2012.

6 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

7 Hans L. Bodlaender, Ton Kloks, Richard B. Tan, and Jan van Leeuwen. Approximations for lambda-colorings of graphs. *Computer Journal*, 47:193–204, 2004.

8 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex set for P_5-free graphs. *Algorithmica*, 81:1342–1369, 2019.

9 Oleg V. Borodin. On acyclic colorings of planar graphs. *Discrete Mathematics*, 25:211–236, 1979.

10 Hajo Broersma, Petr A. Golovach, Daniël Paulusma, and Jian Song. Updating the complexity status of coloring graphs without a fixed induced linear forest. *Theoretical Computer Science*, 414:9–19, 2012.

11 Tiziana Calamoneri. The $L(h,k)$-labelling problem: An updated survey and annotated bibliography. *Computer Journal*, 54:1344–1371, 2011.

12 Christine T. Cheng, Eric McDermid, and Ichiro Suzuki. Planarization and acyclic colorings of subcubic claw-free graphs. *Proc. of WG 2011, LNCS*, 6986:107–118, 2011.

13 Maria Chudnovsky, Shenwei Huang, Sophie Spirkl, and Mingxian Zhong. List-three-coloring graphs with no induced $P_6 + rP_3$. *CoRR*, abs/1806.11196, 2018. `arXiv:1806.11196`.

14 Thomas F. Coleman and Jin-Yi Cai. The cyclic coloring problem and estimation of sparse Hessian matrices. *SIAM Journal on Algebraic Discrete Methods*, 7:221–235, 1986.

15 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

16 Zdeněk Dvořák, Bojan Mohar, and Robert Šámal. Star chromatic index. *Journal of Graph Theory*, 72(3):313–326, 2013.

17 Thomas Emden-Weinert, Stefan Hougardy, and Bernd Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Combinatorics, Probability and Computing*, 7:375–386, 1998.

18 Paul Erdős. Graph theory and probability. *Canadian Journal of Mathematics*, 11:34–38, 1959.

19 Guillaume Fertin, Emmanuel Godard, and André Raspaud. Minimum feedback vertex set and acyclic coloring. *Information Processing Letters*, 84:131–139, 2002.

20 Guillaume Fertin and André Raspaud. Acyclic coloring of graphs of maximum degree five: Nine colors are enough. *Information Processing Letters*, 105:65–72, 2008.

21 Guillaume Fertin, André Raspaud, and Bruce A. Reed. Star coloring of graphs. *Journal of Graph Theory*, 47(3):163–182, 2004.

22 Jiří Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theoretical Computer Science*, 412:2513–2523, 2011.

23 Jiří Fiala, Petr A. Golovach, Jan Kratochvíl, Bernard Lidický, and Daniël Paulusma. Distance three labelings of trees. *Discrete Applied Mathematics*, 160:764–779, 2012.

24 Jiří Fiala, Ton Kloks, and Jan Kratochvíl. Fixed-parameter complexity of lambda-labelings. *Discrete Applied Mathematics*, 113:59–72, 2001.

25 Esther Galby, Paloma T. Lima, Daniël Paulusma, and Bernard Ries. Classifying k-edge colouring for H-free graphs. *Information Processing Letters*, 146:39–43, 2019.

26 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

27 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of colouring graphs with forbidden subgraphs. *Journal of Graph Theory*, 84:331–363, 2017.

28 Petr A. Golovach, Daniël Paulusma, and Jian Song. Coloring graphs without short cycles and long induced paths. *Discrete Applied Mathematics*, 167:107–120, 2014.

29 Geňa Hahn, Jan Kratochvíl, Jozef Širáň, and Dominique Sotteau. On the injective chromatic number of graphs. *Discrete Mathematics*, 256:179–192, 2002.

30 Pavol Hell, André Raspaud, and Juraj Stacho. On injective colourings of chordal graphs. *Proc. LATIN 2008, LNCS*, 4957:520–530, 2008.

31 Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10:718–720, 1981.

32 Shenwei Huang, Matthew Johnson, and Daniël Paulusma. Narrowing the complexity gap for colouring (C_s, P_t)-free graphs. *Computer Journal*, 58:3074–3088, 2015.

33 Jing Jin, Baogang Xu, and Xiaoyan Zhang. On the complexity of injective colorings and its generalizations. *Theoretical Computer Science*, 491:119–126, 2013.

34 Ross J. Kang and Tobias Müller. Frugal, acyclic and star colourings of graphs. *Discret. Appl. Math.*, 159:1806–1814, 2011.

35 T. Karthick. Star coloring of certain graph classes. *Graphs and Combinatorics*, 34:109–128, 2018.

36 Tereza Klimošová, Josef Malík, Tomáš Masařík, Jana Novotná, Daniël Paulusma, and Veronika Slívová. Colouring $(P_r + P_s)$-free graphs. *Proc. ISAAC 2018, LIPIcs*, 123:5:1–5:13, 2018.

37 Daniel Král', Jan Kratochvíl, Zsolt Tuza, and Gerhard J. Woeginger. Complexity of coloring graphs without forbidden induced subgraphs. *Proc. WG 2001, LNCS*, 2204:254–262, 2001.

38 Hui Lei, Yongtang Shi, and Zi-Xia Song. Star chromatic index of subcubic multigraphs. *Journal of Graph Theory*, 88:566–576, 2018.

39 Daniel Leven and Zvi Galil. NP-completeness of finding the chromatic index of regular graphs. *Journal of Algorithms*, 4:35–44, 1983.

40 Cláudia Linhares-Sales, Ana Karolinna Maia, Nícolas A. Martins, and Rudini Menezes Sampaio. Restricted coloring problems on graphs with few P_4's. *Annals of Operations Research*, 217:385–397, 2014.

41 Errol L. Lloyd and Subramanian Ramanathan. On the complexity of distance-2 coloring. *Proc. ICCI 1992*, pages 71–74, 1992.

42 Vadim V. Lozin and Marcin Kamiński. Coloring edges and vertices of graphs without short or long cycles. *Contributions to Discrete Mathematics*, 2(1), 2007.

43 Andrew Lyons. Acyclic and star colorings of cographs. *Discrete Applied Mathematics*, 159:1842–1850, 2011.

44 Mohammad Mahdian. On the computational complexity of strong edge coloring. *Discrete Applied Mathematics*, 118:239–248, 2002.

45 Debajyoti Mondal, Rahnuma Islam Nishat, Md. Saidur Rahman, and Sue Whitesides. Acyclic coloring with few division vertices. *Journal of Discrete Algorithms*, 23:42–53, 2013.

46 Stephan Olariu. Paw-free graphs. *Information Processing Letters*, 28:53–54, 1988.

47 Neil Robertson, Daniel P. Sanders, Paul D. Seymour, and Robin Thomas. The four-colour theorem. *Journal of Combinatorial Theory, Series B*, 70:2–44, 1997.

48 Arunabha Sen and Mark L. Huson. A new model for scheduling packet radio networks. *Wireless Networks*, 3:71–82, 1997.

49 Vadim Georgievich Vizing. On an estimate of the chromatic class of a p-graph. *Diskret Analiz*, 3:25–30, 1964.

50 David R. Wood. Acyclic, star and oriented colourings of graph subdivisions. *Discrete Mathematics and Theoretical Computer Science*, 7:37–50, 2005.

51 Xiao-Dong Zhang and Stanislaw Bylka. Disjoint triangles of a cubic line graph. *Graphs and Combinatorics*, 20:275–280, 2004.

52 Xiao Zhou, Yasuaki Kanari, and Takao Nishizeki. Generalized vertex-coloring of partial k-trees. *IEICE Transactions on Fundamentals of Electronics, Communication and Computer Sciences*, E83-A:671–678, 2000.

53 Enqiang Zhu, Zepeng Li, Zehui Shao, and Jin Xu. Acyclically 4-colorable triangulations. *Information Processing Letters*, 116:401–408, 2016.

An Algorithmic Weakening of the Erdős-Hajnal Conjecture

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Stéphan Thomassé
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Xuan Thang Tran
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Rémi Watrigant
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We study the approximability of the Maximum Independent Set (MIS) problem in H-free graphs (that is, graphs which do not admit H as an induced subgraph). As one motivation we investigate the following conjecture: for every fixed graph H, there exists a constant $\delta > 0$ such that MIS can be $n^{1-\delta}$-approximated in H-free graphs, where n denotes the number of vertices of the input graph. We first prove that a constructive version of the celebrated Erdős-Hajnal conjecture implies ours. We then prove that the set of graphs H satisfying our conjecture is closed under the so-called graph substitution. This, together with the known polynomial-time algorithms for MIS in H-free graphs (e.g. P_6-free and fork-free graphs), implies that our conjecture holds for many graphs H for which the Erdős-Hajnal conjecture is still open. We then focus on improving the constant δ for some graph classes: we prove that the classical Local Search algorithm provides an $OPT^{1-\frac{1}{t}}$-approximation in $K_{t,t}$-free graphs (hence a $\sqrt{OPT}$-approximation in C_4-free graphs), and, while there is a simple $\sqrt{n}$-approximation in triangle-free graphs, it cannot be improved to $n^{\frac{1}{4}-\varepsilon}$ for any $\varepsilon > 0$ unless $NP \subseteq BPP$. More generally, we show that there is a constant c such that MIS in graphs of girth γ cannot be $n^{\frac{c}{\gamma}}$-approximated. Up to a constant factor in the exponent, this matches the ratio of a known approximation algorithm by Monien and Speckenmeyer, and by Murphy. To the best of our knowledge, this is the first strong (i.e., $\Omega(n^\delta)$ for some $\delta > 0$) inapproximability result for Maximum Independent Set in a proper hereditary class.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation, Maximum Independent Set, H-free Graphs, Erdős-Hajnal conjecture

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.23

Funding This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX- 0007) operated by the French National Research Agency (ANR).

1 Introduction

An *independent set* of a (simple, undirected) graph is a set of pairwise non-adjacent vertices. Independent sets have been central in various research topics, both in algorithmic and structural graph theory. In structural graph theory, independent sets (and their complements, cliques) are at the core of several celebrated results, such as Kőnig's theorem, Ramsey's theorem, or Turan's theorem [8], to name only a few. Finding an independent set of maximum cardinality (called the Maximum Independent Set problem, or MIS for short) is a fundamental intractable optimization problem. Indeed, it is NP-hard to solve [21], but

also to approximate within ratio $n^{1-\varepsilon}$ for any $\varepsilon > 0$ [27, 35], where n denotes the number of vertices of the input graph (definitions related to approximation algorithms are given at the end of the section). On the positive side, MIS becomes tractable when restricted to some specific graph classes: It is polynomial-time solvable in bipartite graphs and more generally in perfect graphs [22], admits a PTAS in planar graphs [6] and in more general geometric graph classes such as pseudo-disk graphs [10], bounded genus or H-minor-free graphs [14]. Notice that all the aforementioned graph classes are closed under taking induced subgraphs. We call *hereditary* such a class, and add the qualificative *proper* if it is not the class of all graphs. A hereditary class can be defined by a (possibly infinite) set of forbidden induced subgraphs. A potentially unifying framework is to consider the complexity of MIS in H-free graphs (i.e., graphs without induced copy of H) and $\mathcal{H}$-free graphs (i.e., graphs without induced copy of any $H \in \mathcal{H}$). However, a classical reduction [2, 3] consisting of subdividing every edge of a given graph G a fixed even number of times $2c$ leads to a graph G' such that $\alpha(G') = \alpha(G) + c|E(G)|$ (where $\alpha(.)$ denotes the size of a maximum independent set of a graph). This reduction, together with the fact that MIS remains APX-hard in graphs of maximum degree at most 3 [5] (which means in particular that we may assume that $\alpha(G) = \Omega(|E(G)|)$ in the reduction) implies the following:

▶ **Theorem 1** ([2, 3] and [5]). *For any fixed connected graph H which is neither a path nor a subdivision of the claw $K_{1,3}$, MIS is APX-hard in H-free graphs.*

On the positive side, polynomial algorithms are known for P_6-free graphs [24] and fork-free graphs [4]. For paths on at least seven vertices and subdivided claws not contained in the fork, the computational complexity of MIS remains unsettled.

In this work, we start a systematic investigation of the approximability of MIS in H-free graphs. The intuition is that forbidding a fixed graph H as an induced subgraph should imply a drastic change in the structure of independent sets and cliques. This idea is at the core of the Erdős-Hajnal conjecture: while in random graphs of $\mathcal{G}(n, 1/2)$ the expected maximum of the clique number and the independence number is $O(\log n)$ [19], this value should be significantly larger for an H-free graph. More formally:

▶ **Definition 2.** *A graph H satisfies the* Erdős-Hajnal property *if there exists a constant $\delta > 0$ such that every H-free graph G with n vertices contains either a clique or an independent set of size at least n^δ.*

▶ **Conjecture 3** ([16]). *Every graph H satisfies the Erdős-Hajnal property.*

So far, the Erdős-Hajnal conjecture has been verified for only a small number of graphs, namely: all graphs on at most four vertices, the bull[1], the cliques, and every graph that can be constructed from them using the so-called *substitution* operation [12] (we describe this operation in Section 2). Interestingly, for many graphs H satisfying the Erdős-Hajnal property, MIS is known to be either polynomial or at least to admit an $n^{1-\varepsilon}$-approximation algorithm for some $\varepsilon > 0$. A typical example of this situation is when H is the clique of size $t > 1$. In that case, Ramsey's theorem can be invoked to get a $n^{\frac{t-2}{t-1}}$-approximation algorithm. Indeed a K_t-free graph always contains an independent set of size at least $n^{\frac{1}{t-1}}$, and the classical proof readily yields a polytime algorithm finding such an independent set. This leads us to define an approximation weaker version of the Erdős-Hajnal property and its companion conjecture:

[1] The bull is the graph obtained by adding a pending vertex to two different vertices of a triangle.

▶ **Definition 4.** *A graph H satisfies the* improved approximation property *if there exists a constant $\varepsilon > 0$ such that MIS admits a randomized $n^{1-\varepsilon}$-approximation polynomial algorithm on every H-free n-vertex graph G.*

Here, a *randomized* ρ-approximation algorithm is an algorithm which, given an input graph on n vertices, outputs a ρ-approximation of the problem *with high probability* (*w.h.p.* for short), that is with probability at least a function of n tending to 1 when n goes to infinity.

▶ **Conjecture 5.** *Every graph H satisfies the improved approximation property.*

We refer to Conjecture 5 as the *improved approximation conjecture*. Informally, it states that the inapproximability of MIS in general graphs can be beaten in any proper hereditary class.

Results and organization of the paper

On the one hand, there exist graphs H satisfying the improved approximation property for which the Erdős-Hajnal conjecture is still open. Indeed, as mentioned previously, MIS is polynomial-time solvable in P_6-free graphs, whereas it is still open whether P_5 satisfies the Erdős-Hajnal property. On the other hand, one may wonder if the satisfiability of the Erdős-Hajnal property for a graph H can help designing an approximation algorithm in H-free graphs, and more concretely if Conjecture 3 implies Conjecture 5. In Section 2, we prove that this is almost the case. More precisely, we prove that every graph H satisfying a *constructive* version of the Erdős-Hajnal property also satisfies the improved approximation property. We also show that the improved approximation property is preserved through the substitution operation, which is the one graph operation known to preserve the Erdős-Hajnal property.

We then try and obtain better approximation ratios for the improved approximation property: for a given H, what is the largest $\varepsilon > 0$ such that MIS admits an $O(n^{1-\varepsilon})$-approximation algorithm in H-free graphs? We investigate this question in Sections 3 and 4. More precisely, in Section 3 we describe some particular properties of graphs H as well as graph operations preserving the improved approximation property in a better way than the substitution. We also prove that the classical local search algorithm provides a $\sqrt{OPT}$ approximation ratio in C_4-free graphs and, more generally, an $O(OPT^{1-1/t})$-approximation algorithm in $K_{t,t}$-free graphs. Finally, we present in Section 4 some negative results concerning the improved approximation property: while MIS can be easily $n^{1/2}$-approximated in triangle-free graphs, we show that this ratio cannot be improved to $n^{1/4-\varepsilon}$ for any $\varepsilon > 0$, unless $NP \subseteq BPP$. We also provide a generalization of this result when we forbid all cycles of length $3, \ldots, t$ for a fixed $t \geqslant 3$.

Notations and definitions

For two positive integers $i < j$, we denote the set of integers at least i and at most j by $[i, j]$, while $[i]$ is a short-hand for $[1, i]$. All the graphs we consider are simple; they have no multiple edges nor loops. For a vertex v in a simple graph G, $N_G(v)$, or simply $N(v)$ if the graph is unambiguous, denotes the set of neighbors of v. The *closed neighborhood* of v is defined as $N[v] := N(v) \cup \{v\}$. A *universal vertex* is a vertex whose closed neighborhood is the entire set of vertices. The size of a maximum independent set of G is denoted by $\alpha(G)$. The *girth* (resp. *odd girth*) of a graph is the smallest size of an induced cycle (resp. odd cycle) in the graph. K_s, P_s, C_s respectively denotes the clique, the path, and the cycle on s

vertices, and $K_{s,t}$ is the complete bipartite graph with s vertices on one side and t on the other side (it is also called a *biclique*). The graph $K_3 = C_3$ is also called the *triangle*. The *claw* is the biclique $K_{1,3}$. The *fork* is the 5-vertex graph obtained by subdividing one edge of the claw. For a triple of integers $0 \leqslant i \leqslant j \leqslant k$, the graph $S_{i,j,k}$ is obtained by subdividing one edge of a claw $i-1$ times, a second edge, $j-1$ times, and a third edge $k-1$ times (with the convention that subdividing -1 times means removing the edge and its degree-one endpoint). Observe that with that definition, the family $\{S_{i,j,k}\}_{0 \leqslant i \leqslant j \leqslant k}$ contains the paths.

Given a non-decreasing function $\rho : \mathbb{N} \to \mathbb{N}$, a $\rho(n)$-approximation algorithm for MIS is a polynomial-time algorithm which takes as input a graph G on n vertices, and output an independent set S such that $\alpha(G) \leqslant \rho(n)|S|$. In this case, we also say that the problem can be $\rho(n)$-approximated, or that the output solution is a $\rho(n)$-approximated solution.

2 Constructive Erdős-Hajnal and the substitution operation

A graph H is said to satisfy the *constructive Erdős-Hajnal* property if there is a constant $\delta > 0$ and a polynomial-time algorithm which takes as input an H-free graph G, and outputs a clique or an independent set of size at least $|V(G)|^\delta$. We prove that the constructive Erdős-Hajnal conjecture implies Conjecture 5. To our knowledge, all the graphs H shown to satisfy the Erdős-Hajnal property so far, also satisfy its constructive version.

▶ **Theorem 6.** *Let H be a graph which satisfies the constructive Erdős-Hajnal property with constant*[2] *$0 < \delta \leqslant 1/2$. Then H satisfies the improved approximation property with constant $\delta - \delta^2 - \varepsilon$ for any fixed $\varepsilon > 0$.*

Proof. Let G be an H-free graph with $n := |V(G)|$. We assume $n \geqslant 2^{\frac{1}{1-(\delta-\delta^2)}}$, since otherwise the problem can be solved optimally in constant time. We prove that Algorithm 1 provides a $n^{1-(\delta-\delta^2)}$-approximation. In this algorithm, `Constructive-Erdős-Hajnal`(J) represents the polynomial-time algorithm which takes a graph J and outputs a set of at least $|V(J)|^\delta$ vertices of J which is either an independent set or a clique.

Algorithm 1 Approximation algorithm for MIS in H-free graphs satisfying the constructive Erdős-Hajnal property.

Input: a graph G
Output: an independent set of G

1: $V' \leftarrow V(G)$
2: **while** $|V'| \geqslant n^{1-\delta}$ **do**
3: $\quad X \leftarrow$ `Constructive-Erdős-Hajnal`$(G[V'])$
4: $\quad$ **if** X is an independent set of G **then**
5: $\quad\quad$ **return** X
6: $\quad$ **else**
7: $\quad\quad V' \leftarrow V' \setminus X$
8: **return** $\{v\}$, for an arbitrary chosen $v \in V(G)$

Let X be the independent set returned by the algorithm. If X is returned through line 5, then by the definition of the `Constructive-Erdős-Hajnal` algorithm and because of line 2, we have $|X| \geqslant n^{(1-\delta)\delta}$ which is obviously an $n^{1-(\delta-\delta^2)}$-approximate solution, since any optimal solution has size at most n.

[2] Notice that MIS in H-free graphs is trivial if H has at most 2 vertices, whereas any graph with at least three vertices cannot satisfy the Erdős-Hajnal property with a constant $\delta > 1/2$. This is the reason why we assume $0 < \delta \leqslant 1/2$.

Otherwise, X is returned through line 8 and is thus of size 1. However, in this case, observe that $V(G)$ is partitioned into cliques $C_1, \ldots, C_q$, and the last set V'. Observe that $|V'| < n^{1-\delta}$, and that $|C_i| \geqslant n^{\delta-\delta^2}$ for every $i \in \{1, \ldots, q\}$. We thus have $q \leqslant n^{1-(\delta-\delta^2)}$. But also observe that in that case:

$$\begin{aligned}
\alpha(G) &\leqslant q + |V'| \\
&\leqslant n^{1-(\delta-\delta^2)} + n^{1-\delta} \\
&\leqslant 2n^{1-(\delta-\delta^2)} \\
&\leqslant n^{1-(\delta-\delta^2)+\varepsilon} \text{ as we may assume } n \geqslant 2^{1/\varepsilon} \text{ for any fixed } \varepsilon > 0.
\end{aligned}$$

◀

It is natural to ask which kind of graph operations preserves the satisfiability of the improved approximation property. Given the previous result, natural candidates are graph operations preserving the Erdős-Hajnal property. In the following we prove that this is indeed the case concerning the substitution operation.

▶ **Definition 7.** *Let H_1, H_2 be two vertex-disjoint graphs and $v_0 \in V(H_1)$. We say that a graph H is* obtained from H_1 by substituting H_2 at v_0 *if:*

- $V(H) = (V(H_1) \setminus \{v_0\}) \cup V(H_2)$
- *For $v, v' \in V(H_1) \setminus \{v_0\}$, vv' is an edge in H if and only if it is an edge in H_1.*
- *For $v, v' \in V(H_2)$, vv' is an edge in H if and only if it is an edge in H_2.*
- *For $v \in V(H_1) \setminus \{v_0\}$, $v' \in V(H_2)$, vv' is an edge in H if and only if vv_0 is an edge in H_1.*

More generally, we say that a graph H is obtained from H_1 and H_2 by substitution *if there exists $v_0 \in V(H_1)$ such that H is obtained from H_1 by substituting H_2 at v_0.*

▶ **Theorem 8.** *Let H_1, H_2 be two fixed graphs satisfying the improved approximation property. Then every graph H obtained from H_1 and H_2 by substitution satisfies the improved approximation property.*

Let us start by sketching the idea of our algorithm. We first check whether the number of copies of H_1 in G is small. If so, then a randomly chosen subset of vertices of appropriate size will be H_1-free *w.h.p.*, and we will be able to run our approximation algorithm for H_1-free graphs. If the number of copies of H_1 is large, then we claim that we can find a large subset of vertices inducing an H_2-free graph, and we thus run our approximation algorithm for H_2-free graphs. Each time we run one of our approximation algorithms in an induced subgraph $G[X]$ which is either H_1-free or H_2-free, either it outputs a solution of size at least n^δ for some constant δ, in which case we are done, or it means that $\alpha(G[X])$ is small, in which case we keep X apart and continue the algorithm on $G[V \setminus X]$ as long as enough vertices survive. If too many vertices were kept apart along the process, it means that $\alpha(G)$ was very small at the beginning, so that any singleton $\{v\}$ is actually an approximated solution. We now prove formally the result.

Proof. Let $approx_{H_1}(G)$ (resp. $approx_{H_2}(G)$) be a polynomial-time algorithm which takes as input an H_1-free graph (resp. H_2-free graph) G on n vertices and outputs an $n^{1-\varepsilon_1}$ (resp. $n^{1-\varepsilon_2}$)-approximated solution for the MIS problem in G, for some $\varepsilon_1 > 0$ (resp. $\varepsilon_2 > 0$). For the sake of readability, we set $\varepsilon = min\{\varepsilon_1, \varepsilon_2, 0.99\}$, so that $approx_{H_1}$ and $approx_{H_2}$ are $n^{1-\varepsilon}$-approximation algorithms in H_1-free graphs and H_2-free graphs, respectively[3].

[3] Our result also holds if $approx_{H_1}$ and $approx_{H_2}$ are exact algorithms (hence with $\varepsilon_1 = \varepsilon_2 = 1$), but, for technical reasons, we view them as $n^{0.01}$-approximation algorithms.

Let H be the graph obtained by substituting H_2 at some vertex $v_0 \in V(H_1)$, and let us consider an H-free graph G. We denote by n, n_1 and n_2 the number of vertices of G, H_1 and H_2, respectively. We say that $X \subseteq V(G)$ is a set of H_1-*candidates* if there exists a set $K \subseteq V(G)$ of $n_1 - 1$ vertices such that $G[K]$ is isomorphic to $H_1 - \{v_0\}$ and, for every $x \in X$, $G[K \cup \{x\}]$ is isomorphic to H_1. Since G is H-free, $G[X]$ is H_2-free.

Let $\gamma = \frac{\varepsilon}{2n_1}$, $\eta = \min(1-\varepsilon, \gamma)$, and $\delta = \frac{\varepsilon\eta}{2+\varepsilon\eta}$. We prove that Algorithm 2 is an $O(n^{1-\delta})$-approximation algorithm for MIS in H-free graphs.

Algorithm 2 Approximation algorithm for MIS in H-free graphs, where H is the substitution of H_1 and H_2.

Input: an H-free graph G with n vertices
Output: an independent set of G

1: $i = 1$, $V_1 \leftarrow V(G)$
2: **while** $|V_i| \geqslant n^{1-\delta}$ **do**
3: **if** $G[V_i]$ contains less than $|V_i|^{n_1-\varepsilon}$ copies of H_1 **then**
4: pick a set $X_i \subseteq V_i$ of size $\lceil |V_i|^{\gamma} \rceil$ uniformly at random
5: **if** $G[X_i]$ is H_1-free **then** ▷ This condition is true *w.h.p.*
6: $W_i \leftarrow approx_{H_1}(G[X_i])$
7: **if** $|W_i| \geqslant n^{\delta}$ **then return** W_i
8: **else return** FAIL
9: **else**
10: find a set of H_1-candidates $X_i \subseteq V_i$ with $|X_i| \geqslant |V_i|^{1-\varepsilon}$
11: $W_i \leftarrow approx_{H_2}(G[X_i])$ ▷ $G[X_i]$ is H_2-free
12: **if** $|W_i| \geqslant n^{\delta}$ **then return** W_i
13: $V_{i+1} \leftarrow V_i \setminus X_i$
14: $i \leftarrow i + 1$
15: **return** $\{v\}$ for an arbitrary vertex $v \in V$

▶ **Lemma 9.** *Algorithm 2 runs in polynomial time.*

Proof. An important remark is that at every step i, the graph $G[V_i]$ is an induced subgraph of G, hence is H-free. In line 3 (resp. 5), the algorithm runs through all subsets of n_1 vertices of V_i (resp. X_i), which can be done in $O(n^{n_1})$ time.

Finally, the existence of a set of H_1-candidates in line 10 is ensured by the fact that in that case, $G[V_i]$ contains at least $|V_i|^{n_1-\varepsilon}$ copies of H_1. Hence, by the pigeonhole principle, there must exist a set V_H of $n_1 - 1$ vertices with $V_H \subseteq V_i$ such that $G[V_H]$ induces $H_1 \setminus v_0$ together with a set $X_i \subseteq V_i \setminus V_H$ of size at least $|V_i|^{1-\varepsilon}$ such that for every $x \in X_i$, $G[V_H \cup \{x\}]$ induces H_1. Finding the set V_H can be done in $O(|V_i|^{n_1-1})$ time, while finding the set X_i can be done in $O(|V_i|)$ time, since it is sufficient to find the vertices in $V_i \setminus V_H$ with the right neighborhood with respect to V_H. By the definition of H_1-candidates, $G[X_i]$ is H_2-free, which allows to run $approx_{H_2}$ on $G[X]$ in the next line of the algorithm. ◀

We now prove that, *w.h.p.*, the solution S returned by our algorithm is an $O(n^{1-\delta})$-approximation. To this end, we first prove that *w.h.p.* it does not return FAIL.

▶ **Lemma 10.** *If the number of copies of H_1 in a graph G on n vertices is less than $n^{n_1-\varepsilon}$, then any subset of vertices of size $\lceil n^{\gamma} \rceil$ picked uniformly at random induces an H_1-free graph, with high probability.*

Proof. Let n be the number of vertices of G, and P be a subset of $\lceil n^\gamma \rceil$ vertices picked uniformly at random. For any set $V_H \subseteq V$ inducing H_1, the probability that V_H is contained in P is $\frac{\binom{n-n_1}{|P|-n_1}}{\binom{n}{|P|}} < \left(\frac{|P|}{n}\right)^{n_1}$. Hence the probability that P is H_1-free is at least

$$\left(1-\left(\frac{|P|}{n}\right)^{n_1}\right)^{n^{n_1-\varepsilon}} = \left(1-\frac{1}{n^{n_1-\frac{\varepsilon}{2}}}\right)^{n^{n_1-\varepsilon}}$$

which tends to 1 when $n \to +\infty$. ◀

Next, if it returns a solution through lines 7 or 12, then this solution is an independent set of size at least n^δ, by definition. We now deal with the case in which it returns a singleton, through line 15. The aim is to prove that $\alpha(G)$ is at most $O(n^{1-\delta})$. Let $q+1$ be the largest value of i in the execution of the algorithm (i.e., $|V_{q+1}| < n^{1-\delta}$). The vertex-set V is thus partitioned into $X_1, \ldots, X_q$, and V_{q+1}. Hence we have $\alpha(G) \leqslant |V_{q+1}| + \sum_{i=1}^{q} \alpha(G[X_i])$. Since $|V_{q+1}| < n^{1-\delta}$, we only need to upper bound the second part.

▶ **Lemma 11.** *With the above definitions,* $\sum_{i=1}^{q} \alpha(G[X_i]) \leqslant n^{1-\delta}$.

Proof. Recall that we have $X_i \subseteq V_i$, where $V_i = V \setminus \bigcup_{j=1}^{i-1} X_j$, and, for each $i \in [q]$, we have constructed an independent set $W_i \subseteq X_i$. All these sets have the following properties:

1. $|V_{q+1}| < n^{1-\delta}$, by definition of q.
2. $|X_i| \geqslant |V_i|^\eta \geqslant n^{\eta(1-\delta)}$. Indeed, if X_i is defined in line 4, then it is of size at least $|V_i|^\gamma$, whereas if it is defined in line 10, it is of size at least $|V_i|^{1-\varepsilon}$, and $\eta = \min(1-\varepsilon, \gamma)$.
3. $|W_i| < n^\delta$, otherwise we would have returned it.
4. $\alpha(G[X_i]) \leqslant |W_i| \cdot |X_i|^{1-\varepsilon}$, since W_i is returned by $approx_{H_1}$ or $approx_{H_2}$, which are approximation algorithms applied to $G[X_i]$.

Now, we have the following:

$$\begin{aligned}\sum_{i=1}^{q} \alpha(G[X_i]) &\leqslant \sum_{i=1}^{q} |W_i| \cdot |X_i|^{1-\varepsilon} \\ &\leqslant n^\delta \sum_{i=1}^{q} |X_i|^{1-\varepsilon}.\end{aligned}$$

We then need the following technical lemma.

▶ **Lemma 11.1.** *Let* $(a_i)_{i=1\ldots q}$ *be some positive numbers (with* $q \in \mathbb{N}$*) such that* $\sum_{i=1}^{q} a_i = N$ *and* $a_i \geqslant k > 0$ *for all* $i \in \{1, \ldots, q\}$*. Then* $\sum_{i=1}^{q} a_i^\zeta \leqslant N k^{\zeta-1}$ *for any* $0 < \zeta < 1$.

Proof. We have:

$$\sum_{i=1}^{q} a_i^\zeta \leqslant \left(\frac{N}{q}\right)^\zeta \cdot q = N \cdot \left(\frac{N}{q}\right)^{\zeta-1} \leqslant N k^{\zeta-1}.$$ ◀

Using the above lemma together with item 2 of the previous properties in order to lower bound each $|X_i|$, we obtain:

$$\begin{aligned}\sum_{i=1}^{q}\alpha(G[X_i]) &\leqslant n^{\delta}\left(\sum_{i=1}^{q}|X_i|\right)n^{\eta(1-\delta)(1-\varepsilon-1)}\\ &\leqslant n^{1-\varepsilon\eta(1-\delta)+\delta} \quad \text{since } \sum_{i=1}^{q}|X_i|\leqslant n\\ &\leqslant n^{1-\delta} \quad \text{because } \delta=\frac{\varepsilon\eta}{2+\varepsilon\eta}\text{, hence } \varepsilon\eta(1-\delta)=2\delta\end{aligned}$$

◀

Hence, any solution of size 1 is an $O(n^{1-\delta})$-approximation in this case, which concludes the proof. ◀

3 Better approximation ratios

In this section we improve over the ratio given by Theorem 8 for some graphs H that can be built by a sequence of substitutions from graphs H' such that MIS is polynomial-time solvable in H'-free graphs. Furthermore, we present deterministic algorithms.

3.1 Adding a universal vertex

Let H^{+u} be the graph H augmented by a universal vertex, i.e., we add one vertex adjacent to all the vertices of H.

▶ **Lemma 12.** *Let $0 \leqslant \gamma < 1$ be a real number and H be a graph such that MIS admits an OPT^{γ}-approximation $\mathcal{A}$ in H-free graphs. Then it also admits an $OPT^{\frac{1}{2-\gamma}}$-approximation $\mathcal{A}^{+u}$ in H^{+u}-free graphs.*

Proof. Let G be the input graph, thus OPT $:= \alpha(G)$. The base case of the algorithm is when G does not contain any vertex, and we correctly report the empty set as optimum solution. Otherwise G has at least one vertex, say v_1. We run the approximation $\mathcal{A}$ on $G[N(v_1)]$. G being H^{+u}-free, the subgraph induced by the open neighborhood of any vertex is indeed H-free. Let S_1 be the returned solution. By assumption, $|S_1| \geqslant \alpha(G[N(v_1)])^{1-\gamma}$. For what follows, the knowledge of the value OPT would help. Unfortunately we will make some recursive calls to $\mathcal{A}^{+u}$, so exhaustively guessing this value would result in an exponential running time. Instead we will branch but the branching tree will only have at most $n := |V(G)|$ leaves. More precisely the tree will be a so-called *comb*, i.e., a path where all the vertices except one end has an additional private neighbor. We eventually output the best solution found among all the leaves.

We inductively run $\mathcal{A}^{+u}$ on $G - N[v_1]$, which produces a tree T with at most $n-1$ leaves. And we output the best solution among S and all the solutions at the leaves of T augmented by the vertex v_1. This algorithm returns an independent set since v_1 is by definition non-adjacent to any vertex of $G - N[v_1]$. The running time of our algorithm satisfies $f_{\mathcal{A}^{+u}}(n) = f_{\mathcal{A}}(n_1 - 1) + f_{\mathcal{A}^{+u}}(n - n_1) + O(1)$ (with $n_1 = |N(v_1)|$). Hence $f_{\mathcal{A}^{+u}}(n) = O(\max\{f_{\mathcal{A}}(n), n\})$, and in the likely event that $\mathcal{A}$ is *not* sublinear, $\mathcal{A}^{+u}$ has the same running time as $\mathcal{A}$ up to a multiplicative constant factor.

We shall now show that $\mathcal{A}^{+u}$ is indeed a OPT$^{\frac{1}{2-\gamma}}$-approximation. We denote by $v_1, \ldots, v_p$ with $p \leqslant n$, the vertices added along the path to the deepest leaf of T. We denote by $S_1, \ldots, S_p$ the sets returned by $\mathcal{A}$ such that S_i is computed in the graph $G'_i := G[N(v_i) \setminus \bigcup_{j<i} N[v_j]]$.

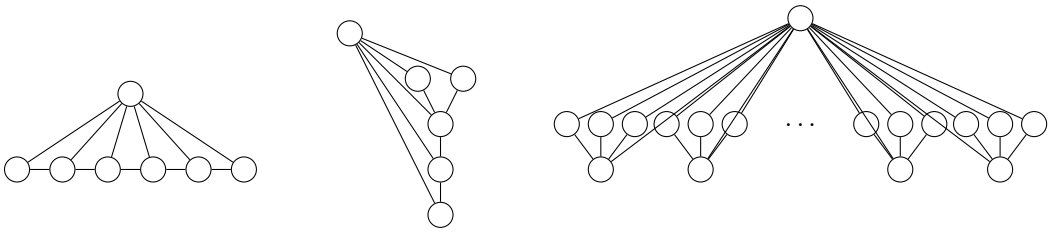


Figure 1 The three maximal locally easy graphs H constructed from P_6, the fork, and $tK_{1,3}$, respectively.

We also define $G_i := G[N[v_i] \setminus \bigcup_{j<i} N[v_j]]$, and $R_i := G - \bigcup_{j<i} N[v_j]$. Observe that $\{V(G_1), \dots, V(G_p)\}$ is a partition of $V(G)$, as well as, $\{V(G_1), \dots, V(G_i), V(R_{i+1})\}$ for every $i \in [p-1]$.

Let, if it exists, S_h be the first solution returned by $\mathcal{A}$ when called on G'_h such that $|S_h| \geqslant \alpha(R_h)^{1-\frac{1}{2-\gamma}}$. We claim that the solution output at this leaf, namely $S'_h := S_h \cup \{v_1, \dots, v_{h-1}\}$ is an $\mathrm{OPT}^{\frac{1}{2-\gamma}}$-approximation. If such an S_h does not exist, we show the same statement where $S_h = \emptyset$ and $h - 1 = p$.

We upperbound $\alpha(G_i)$ for every $i \in [h-1]$. By definition of S_h, it holds that $|S_i| < \alpha(R_i)^{1-\frac{1}{2-\gamma}}$ for any $i \in [h-1]$. Due to the approximation ratio of $\mathcal{A}$, it holds that:

$$\alpha(G_i)^{1-\gamma} \leqslant |S_i| < \alpha(R_i)^{1-\frac{1}{2-\gamma}} = \alpha(R_i)^{\frac{1-\gamma}{2-\gamma}},$$

hence $\alpha(G_i) < \alpha(R_i)^{\frac{1}{2-\gamma}} \leqslant \alpha(G)^{\frac{1}{2-\gamma}}$. Thus,

$$\mathrm{OPT} = \alpha(G) \leqslant \alpha(R_h) + \sum_{i\in[h-1]} \alpha(G_i) \leqslant |S_h|\alpha(R_h)^{\frac{1}{2-\gamma}} + \sum_{i\in[h-1]} \alpha(G)^{\frac{1}{2-\gamma}}$$

$$\leqslant |S_h|\alpha(G)^{\frac{1}{2-\gamma}} + (h-1)\alpha(G)^{\frac{1}{2-\gamma}} = (|S_h| + h - 1)\alpha(G)^{\frac{1}{2-\gamma}} = |S'_h|\alpha(G)^{\frac{1}{2-\gamma}} = |S'_h|\mathrm{OPT}^{\frac{1}{2-\gamma}}.$$

Therefore $\mathcal{A}^{+u}$ is an $\mathrm{OPT}^{\frac{1}{2-\gamma}}$-approximation for MIS in H^{+u}-free graphs. ◀

3.2 Locally easy graphs

We say that a graph H is *locally easy* if it has a universal vertex v such that there is a polynomial-time algorithm for MIS in $H - \{v\}$-free graphs. Up to now, the three maximal graphs H for which we know that MIS is polynomial-time solvable on H-free graphs are P_6 [23], the fork [4, 30], and $tK_{1,3}$ (or $t$claw) [9] (see Figure 1 for the corresponding maximal locally easy graphs).

The following is an immediate consequence of Lemma 12. Therein we recall that the constant γ may take value 0.

▶ **Theorem 13.** *For any locally easy H, MIS can be $\sqrt{OPT}$-approximated on H-free graphs.*

And in particular, there is a $\sqrt{n}$-approximation for triangle-free graphs. Lemma 12 also yields the following approximation ratio in K_{t+1}-free graphs.

▶ **Theorem 14.** *For any $t \geqslant 1$, MIS can be $OPT^{1-\frac{1}{t}}$-approximated on K_{t+1}-free graphs.*

Proof. We show this statement by induction. The base case says that we can exactly solve in polynomial time MIS in edgeless graphs, which is obviously true. We assume that the statement is true for a fixed t. As $K_{t+2} = (K_{t+1})^{+u}$, Lemma 12 implies that MIS can be $\mathrm{OPT}^{\frac{1}{2-(1-\frac{1}{t})}}$-approximated on K_{t+2}-free graphs. Furthermore, $\frac{1}{2-(1-\frac{1}{t})} = \frac{1}{1+\frac{1}{t}} = \frac{t}{t+1} = 1 - \frac{1}{t+1}$. Therefore we do obtain an $\mathrm{OPT}^{1-\frac{1}{t+1}}$-approximation on K_{t+2}-free graphs. ◀

We say that a graph H is *t-locally easy* if it has a set U of t universal vertices such that a polynomial algorithm is known for MIS in $H-U$-free graphs. Informally, these graphs are obtained by replacing universal vertices of Figure 1 by a k-clique. The previous result readily generalizes from K_{t+1}-free to H-free graphs with H $(t+1)$-locally easy, with the same proof.

▶ **Corollary 15.** *Let t be a non-negative integer. For any t-locally easy H, MIS can be $OPT^{1-\frac{1}{t+1}}$-approximated on H-free graphs.*

As we will see in Section 4.3 for forbidden graphs H containing a triangle, it is unlikely to improve the approximation ratio below $n^{1/4}$. However if H is a star, constant-approximations are achievable. It is known that MIS is polynomial-time solvable on claw-free graphs [31] (i.e., $K_{1,3}$-free graphs) while it is APX-hard on $K_{1,4}$-free graphs (see for instance [3]). The greedy algorithm (or actually any sensible algorithm) gives an s-approximation in $K_{1,s}$-free graphs. The ratio was improved to arbitrarily close to $\frac{s-1}{2}$ by Halldórsson.

▶ **Theorem 16** ([25])**.** *For every $s \geqslant 4$ and $\varepsilon > 0$, MIS is $\frac{s-1}{2}+\varepsilon$-approximable on $K_{1,s}$-free graphs.*

3.3 Local Search for $K_{t,t}$-free graphs

Here we analyze the performance of the t-Local Search algorithm in $K_{t,t}$-free graphs. Usually for the particular case of the biclique, "$K_{t,t}$-free" is intended as "no $K_{t,t}$ as a subgraph". Here we still mean "no $K_{t,t}$ as an *induced* subgraph", since our algorithm works even in this more general setting. For a fixed integer $t > 2$, t-Local Search takes as input a graph G, and construct an independent set S from a single vertex. Then, it tries to improve S in the following way: whenever there exist two sets $X \subseteq S$ (note that X can possibly be empty) and $Y \subseteq V \setminus S$ such that $0 \leqslant |X| < |Y| \leqslant t$ and $(S \setminus X) \cup Y$ is an independent set, it replaces S by $(S \setminus X) \cup Y$ (if there are several choices, it chooses an arbitrary one). When S can no longer be improved, it outputs it. Each improvement takes $O(n^{2t})$ time, and the number of such improvements is at most n, since the size of S increases by at least one at each step. Hence, the algorithm takes polynomial time. In the following theorem, we prove that this simple algorithm provides an $O(OPT^{1-1/t})$-approximation whenever the input graph is $K_{t,t}$-free. In particular, 2-Local Search is an $O(\sqrt{OPT})$-approximation in C_4-free graphs. It came to our knowledge that the same result was obtained independently by Dvořák, Feldmann, Rai, and Rzążewski [15].

▶ **Theorem 17.** *For any fixed $t \geqslant 2$, t-Local Search is an $O(OPT^{1-1/t})$-approximation in $K_{t,t}$-free graphs.*

Proof. Let S be the solution returned by the algorithm, and O be a fixed optimal solution. The objective is to bound $|O'|$ in terms of $|S'|$, where $O' := O \setminus S$ and $S' := S \setminus O$. To this end, let us consider $\mathcal{B} := G[S' \cup O']$ the bipartite graph induced by $S' \cup O'$. Let $k := |S'|$. We partition O' into D^- and D^+, where D^- are the vertices of O' whose degree within S' is at most $t-1$, and thus D^+ are the vertices of O' whose degree within S' is at least t. We now bound the sizes of D^- and D^+ separately.

- Let us partition D^- into classes $D_1^-, \ldots, D_q^-$ with respect to the equivalence relation $u \sim v$ if and only if $N_{S'}(u) = N_{S'}(v)$. By definition of D^- we have $q \leqslant \sum_{i=1}^{t-1} \binom{k}{i}$. Then, we claim that for every $i \in \{1, \ldots, q\}$, we have $|D_i^-| \leqslant t-1$. Indeed, we must have $|D_i^-| \leqslant |N_{S'}(D_i^-)|$, since otherwise the algorithm would have replaced S by $(S \setminus N_{S'}(D_i^-)) \cup D_i^-$. This proves $|D^-| \leqslant (t-1)\sum_{i=1}^{t-1} \binom{k}{i}$.

- For a set $X \subseteq S'$, let $I_X := \bigcap_{x \in X} N_{O'}(x)$. Observe that if $|X| = t$, then necessarily $I_X \subseteq D^+$, and moreover $|I_X| \leqslant t-1$, since otherwise the graph would have an induced $K_{t,t}$. Finally, we have $D^+ = \bigcup_{X \subseteq S', |X|=t} I_X$, which proves that $|D^+| \leqslant \binom{k}{t}(t-1)$.

Hence we have $|O'| \leqslant |D^-| + |D^+| \leqslant (t-1)\sum_{i=1}^{t-1}\binom{k}{i} + \binom{k}{t}(t-1) = O(k^t)$. ◀

4 Graphs without short cycles

In this section we show that the strong inapproximability of MIS in general graphs survives, albeit in a less severe form, on graphs without small cycles. More quantitatively, we show that for any positive integer γ, there is a constant $\beta = \Theta(1/\gamma)$ depending only on γ, such that an n^β-approximation of MIS in graphs with girth γ is unlikely.

4.1 Triangle-free graphs

While Lemma 12 implies an $n^{1/2}$-approximation of MIS in triangle-free graphs, a natural question is how much the ratio's exponent can be decreased. In this section we provide a lower bound for it.

The following result will be made obsolete twice. Indeed we will then generalize its statement from triangle-free, that is girth 4, to graphs with any constant girth. Then in Section 4.3 we will present a stronger inapproximability result of $\Omega(n^{1/4-\varepsilon})$. Nevertheless we choose to keep its proof as it is simpler, easier to follow, and self-contained. Furthermore, it contains all the ideas necessary to achieve the subsequent results.

▶ **Theorem 18.** *For any $\varepsilon > 0$, it is NP-hard to distinguish between triangle-free graphs G on n vertices satisfying*

- $\alpha(G) \leqslant n^{5/6-\varepsilon}$, *and*
- $\alpha(G) \geqslant n^{1-\varepsilon}$.

So for any $\varepsilon > 0$, MIS cannot be approximated within ratio $n^{1/6-\varepsilon}$ in triangle-free graphs unless NP $\subseteq$ BPP.

Proof. Let $\varepsilon > 0$ be an arbitrarily small real value, and $\varepsilon := 3\varepsilon$. We perform a randomized reduction from an infinite set of graphs H admitting the following gap: Positive instances have stable sets of size at least $|V(H)|^{1-\varepsilon}$ whereas negative instances have no stable set of size $|V(H)|^{\varepsilon}$. It is known that distinguishing between these two cases is NP-hard for randomized reductions [27], and even for deterministic ones [35].

Reduction. Given an N-vertex graph H, we construct a triangle-free graph G in the following way. We transform every vertex v of H into an independent set $I(v)$ of size $s := N^5$. For every edge $uv \in E(H)$, we put a random bipartite graph between $I(u)$ and $I(v)$: for each pair of vertices $x \in I(u)$, $y \in I(v)$, we independently add an edge xy to $E(G)$ with probability $p := N^{-4-\eta}$ with $\eta := 2\varepsilon/3$. We denote by $G_\triangle$ the graph thus obtained. A key property is that $G_\triangle$ contains only *few* triangles. For each triangle in $G_\triangle$, we remove all three vertices of it. We call that phase the *triangle removal*, and we denote by G the triangle-free graph that arises when that phase comes to an end. We further assume that N is larger than the smallest integral constant N_0 for which for every $N \geqslant N_0$, $N^{3\eta} > N^{2.5\eta} + 10N^{2\eta}\ln N$, $N^{-\varepsilon} > 6N^{-2\eta}$, $2^{17/6}N^{\eta/3} < N^{\varepsilon}$, and $N^{-\eta} < 10^{-100}$. In particular the second and third inequalities hold for sufficiently large N since $\eta < \varepsilon = 3\varepsilon < 2\eta$. The hardness of approximation [27, 35] still holds since instances with less than a constant number of vertices can be solved optimally in constant time.

▶ **Lemma 19.** *For every edge $uv \in E(H)$, the probability that there exist two sets $A \subset I(u), B \subset I(v)$ both of size $N^{4+2\eta}$ without any edge between A and B is at most $e^{-N^{4+2.5\eta}}$. Thus, with high probability, this event does not happen.*

Proof. The probability that there is no edge between two fixed sets A and B of size $N^{4+2\eta}$ is:

$$(1-p)^{|A|\cdot|B|} = (1-\frac{1}{N^{4+\eta}})^{N^{2(4+2\eta)}} \leqslant e^{-N^{4+3\eta}}.$$

By the union bound, the probability that there is at least one such pair of sets is at most:

$$\binom{N^5}{N^{4+2\eta}}^2 e^{-N^{4+3\eta}} \leqslant N^{10N^{4+2\eta}} e^{-N^{4+3\eta}} = e^{-N^{4+3\eta}+10N^{4+2\eta}\ln N} \leqslant e^{-N^{4+2.5\eta}}. \quad \blacktriangleleft$$

▶ **Lemma 20.** *The expected number of triangles in $G_\triangle$ is at most $N^{6-3\eta}$. Furthermore $|V(G_\triangle)| - |V(G)|$ is at most $3N^{6-2\eta}$ with probability at least $1 - N^{-\eta}$.*

Proof. The expected number of triangles in $G_\triangle$ is:

$$\mathbb{E}(\#(\triangle, G_\triangle)) \leqslant (sN)^3 p^3 = (N^6)^3 N^{-12-3\eta} = N^{6-3\eta}.$$

By Markov's inequality, $\mathbb{P}(\#(\triangle, G_\triangle) \geqslant N^{6-2\eta}) \leqslant N^{6-3\eta}/N^{6-2\eta} = N^{-\eta}$. $\blacktriangleleft$

Let n be the number of vertices of G (after the triangle removal). By the previous lemma $n := |V(G)| > N^6/2$, with high probability.

If H is a YES-instance, there is a stable set of size $\mathbf{n^{1-\varepsilon}}$ in G. We assume that H is a YES-instance, so there is a stable set S in H such that $|S| \geqslant N^{1-\varepsilon}$. By construction, $S_{G_\triangle} := \bigcup_{u\in S} I(u)$ is a stable set in $G_\triangle$ of size $s|S| \geqslant N^{6-\varepsilon}$. By Lemma 20, $S_{G_\triangle} \cap V(G)$ is an independent set in G of size, w.h.p., at least $N^{6-\varepsilon} - 3N^{6-2\eta} > N^{6-\varepsilon}/2 > n^{1-\varepsilon/6}/2 = n^{1-\varepsilon/2}/2 > n^{1-\varepsilon}$.

If H is a NO-instance, there is no stable set of size $\mathbf{n^{5/6+\varepsilon}}$ in G. Let S_G be an independent set of G and let $S := \{v \in V(H) \text{ such that } |I(v) \cap S_G| \geqslant N^{4+2\eta}\}$. If H is a NO-instance, then there is no stable set in H of size more than N^ε. By a union bound of applications of Lemma 19 to all pairs of vertices of S, w.h.p. S is an independent set of H, which implies that $|S| < N^\varepsilon$. Thus $|S_G| < sN^\varepsilon + N^{4+2\eta}(N - N^\varepsilon) < N^{5+\varepsilon} + N^{5+2\eta} < 2N^{5+2\eta} = 2N^{5(1+2\eta/5)} < 2^{11/6} n^{5/6+\eta/3} < n^{5/6+\varepsilon}$. $\blacktriangleleft$

4.2 Graphs with higher girth

Monien, Speckenmeyer and Murphy independently found improved approximations when the girth, actually even the odd girth, is any constant γ.

▶ **Theorem 21** ([32, 33])**.** *MIS admits a polynomial-time $n^{\frac{2}{\gamma-1}}$-approximation on graphs with odd girth γ.*

In particular, the result implies an $n^{1/2}$-approximation for triangle-free graphs, an $n^{1/3}$-approximation for $\{C_3, C_5\}$-free graphs, an $n^{1/4}$-approximation for $\{C_3, C_5, C_7\}$-free graphs, etc. On the complexity side, the construction of Theorem 18 where the probability p of having an edge between $I(u)$ and $I(v)$ with $uv \in E(H)$ is now set to $N^{-2(\gamma-1)-\eta}$ and the size s of each $I(u)$ is set to $N^{2\gamma-1}$ yields a polynomial gap on C_γ-free graphs, and even on graphs with girth $\gamma + 1$.

▶ **Theorem 22.** *For any $\varepsilon > 0$, it is NP-hard to distinguish between graphs G with n vertices and girth $\gamma + 1$ satisfying*

- $\alpha(G) \leqslant n^{\frac{2\gamma-1}{2\gamma}-\varepsilon}$, *and*
- $\alpha(G) \geqslant n^{1-\varepsilon}$.

Hence, for any $\varepsilon > 0$, MIS cannot be approximated within ratio $n^{\frac{1}{2\gamma}-\varepsilon}$ in graphs with girth $\gamma + 1$ unless NP $\subseteq$ BPP.

Proof. We do the same reduction as in Theorem 18 with the following modifications. We now set $\varepsilon := \gamma\varepsilon$, $s := N^{2\gamma-1}$, $p := N^{2(\gamma-1)-\eta}$, and $\eta := \frac{\gamma-1}{\gamma}\varepsilon$. We denote by $G_\circ$ the graph obtained before the removal step. For every cycle of length at most γ, we remove all the vertices of the cycle from the graph. When this short cycle removal ends, the graph has girth at least $\gamma + 1$. We call G the obtained graph.

▶ **Lemma 22.1.** *For every edge $uv \in E(H)$, the probability that there exist two sets $A \subset I(u), B \subset I(v)$ both of size $N^{2(\gamma-1)+2\eta}$ without any edge between A and B is at most $e^{-N^{2(\gamma-1)+2.5\eta}}$. Thus, with high probability, this event does not happen.*

Proof. The probability that there is no edge between two fixed sets A and B of size $N^{2(\gamma-1)+2\eta}$ is:

$$(1-p)^{|A|\cdot|B|} = (1 - \frac{1}{N^{2(\gamma-1)+\eta}})^{N^{2(2(\gamma-1)+2\eta)}} \leqslant e^{-N^{2(\gamma-1)+3\eta}}.$$

By the union bound, the probability that there is at least one such pair of sets is at most:

$$\binom{N^{2\gamma-1}}{N^{2(\gamma-1)+2\eta}}^2 e^{-N^{2(\gamma-1)+3\eta}} \leqslant N^{10N^{2(\gamma-1)+2\eta}} e^{-N^{2(\gamma-1)+3\eta}}$$

$$= e^{-N^{2(\gamma-1)+3\eta}+10N^{2(\gamma-1)+2\eta}\ln N} \leqslant e^{-N^{2(\gamma-1)+2.5\eta}}.$$ ◀

▶ **Lemma 22.2.** *The expected number of cycles of length at most γ in $G_\circ$ is at most $N^{(2-\eta)\gamma}$. Furthermore $|V(G_\circ)| - |V(G)|$ is at most $\gamma N^{2\gamma-\eta(\gamma-1)}$ with probability at least $1 - N^{-\eta}$.*

Proof. The expected number of cycles of length at most γ in $G_\circ$ is:

$$\mathbb{E}(\#(C_{3\to\gamma}, G_\circ)) \leqslant \gamma(sN)^\gamma p^\gamma = \gamma N^{2\gamma^2} N^{(-2(\gamma-1)-\eta)\gamma} = \gamma N^{(2-\eta)\gamma}.$$

By Markov's inequality, $\mathbb{P}(\#(C_{3\to\gamma}, G_\circ) \geqslant \gamma N^{2\gamma-\eta(\gamma-1)}) \leqslant N^{(2-\eta)\gamma}/N^{2\gamma-\eta(\gamma-1)} = N^{-\eta}$. ◀

Let n be the number of vertices of G (after the short cycle removal). By the previous lemma $n := |V(G)| > N^{2\gamma}/2$, with high probability.

If H is a YES-instance, there is a stable set of size $n^{1-\varepsilon}$ in G. We assume that H is a YES-instance, so there is a stable set S in H such that $|S| \geqslant N^{1-\varepsilon}$. By construction, $S_{G_\circ} := \bigcup_{u\in S} I(u)$ is a stable set in $G_\circ$ of size $s|S| \geqslant N^{2\gamma-\varepsilon}$. By Lemma 22.2, $S_{G_\circ} \cap V(G)$ is an independent set in G of size, w.h.p., at least $N^{2\gamma-\varepsilon} - \gamma N^{2\gamma-\eta(\gamma-1)} > N^{2\gamma-\varepsilon}/2 > n^{1-\varepsilon/2\gamma)}/2 = n^{1-\varepsilon/2}/2 > n^{1-\varepsilon}$.

If H is a NO-instance, there is no stable set of size $n^{(2\gamma-1)/(2\gamma)+\varepsilon}$ in G. Let S_G be an independent set of G and let $S := \{v \in V(H) \text{ such that } |I(v) \cap S_G| \geqslant N^{2(\gamma-1)+2\eta}\}$. If H is a NO-instance, then there is no stable set in H of size more than N^ε. By a union bound of applications of Lemma 22.1 to all pairs of vertices of S, w.h.p. S is an independent set of H, which implies that $|S| < N^\varepsilon$. Thus $|S_G| < sN^\varepsilon + N^{2(\gamma-1)+2\eta}(N - N^\varepsilon) < N^{2\gamma-1+\varepsilon} + N^{2\gamma-1+2\eta} < 2N^{2\gamma-1+2\eta} = 2N^{(2\gamma-1)(1+2\eta/(2\gamma-1))} < 2^{(4\gamma-1)/(2\gamma)} n^{(2\gamma-1)/(2\gamma)+\eta/\gamma} < n^{(2\gamma-1)/(2\gamma)+\varepsilon}$. ◀

Let us note that there is still a 4-fold multiplicative factor in the exponent between the approximation ratios of Theorem 21, namely $n^{2/\gamma}$, and the hardness ratios of $n^{1/(2\gamma)-o(1)}$ in Theorem 22. It is an interesting open question to bridge this gap.

An even hole is an induced cycle of even length at least 4. Even-hole-free graphs are $\{C_4, C_6, C_8, \ldots\}$-free graphs. The computational complexity of MIS on even-hole-free graphs is still unknown. An FPT algorithm was established recently [26]. We observe that Local Search readily gives a PTAS for that problem. We leave the existence of an EPTAS as an open problem.

▶ **Observation 23.** *MIS can be $(1+\varepsilon)$-approximated in time $n^{O(1/\varepsilon)}$ on even-hole-free graphs.*

Proof. The graph induced by the symmetric difference between any two feasible solutions is bipartite and even-hole-free, hence it is a forest. Let S be a solution obtained by local search on an input graph G, O be a fixed optimum solution, $S' := S \setminus O$, and $O' := O \setminus S$. It is known that when $G[S' \cup O']$ is planar, there is an absolute constant C such that the C/ε^2-Local Search $1+\varepsilon$-approximates the problem [34, 11], that is for a maximization problem, $|S'| \geqslant (1-\varepsilon)|O'|$, implying $|S| \geqslant (1-\varepsilon)|O|$. This gives a PTAS with running time $n^{O(1/\varepsilon^2)}$. When $G[S' \cup O']$ is even a forest, then it can be shown, and it is somewhat folklore, that a C/ε-Local Search is sufficient. ◀

4.3 Strengthening the inapproximability

There are two directions to improve the hardness-of-approximation results of Sections 4.1 and 4.2. As already mentioned, one can try to match upper and lower bounds in the approximation ratio, or at least to increase the exponent δ such that an n^δ-approximation would contradict a standard complexity-theoretic assumption. For triangle-free graphs, for instance, we do *not* expect a matching $n^{1/6}$-approximation. And a likely outcome is that, ignoring logarithmic factors, the $\sqrt{n}$-approximation is best possible. We will actually show that an $n^{1/4-\varepsilon}$-approximation is unlikely. The other direction is to derandomize our reductions. That way the inapproximability would be subject to the more (arguably the most) standard complexity assumption that P is not equal to NP. Derandomizing without degrading the quality of the gap seems challenging. We now encapsulate the reductions of Sections 4.1 and 4.2 so that both improving tasks boil down to exhibiting a randomized or deterministic family of graphs.

We say that an infinite family of graphs $\mathcal{C}$ is *non-disappearing* if there is a constant $K \in (0,1]$ such that for every positive integer n, there is a graph $G \in \mathcal{C}$ with at least Kn and at most n/K vertices. A non-disappearing family is called *efficient* if there is a polynomial-time algorithm which given an integer n (encoded in unary), outputs such a graph G. For example, a family containing at least one graph for every number of vertices is non-disappearing. We denote by $\mathcal{G}_\gamma$ the set of all graphs with girth at least γ.

▶ **Theorem 24.** *Let $\gamma > 3$ be an integer, $\delta \in (0,1)$ be a real (allowed to depend on γ), and $\mathcal{C}$ be an efficient non-disappearing family included in $\mathcal{G}_\gamma$, such that for every $G \in \mathcal{C}$ there is no disjoint pair of sets $A, B \subseteq V(G)$ satisfying both $|A| = |B| \geqslant |V(G)|^\delta$ and $E(A,B) = \emptyset$. Then MIS in $\mathcal{G}_\gamma$ cannot be $n^{\frac{1-\delta}{2}-\varepsilon}$-approximated, unless $P = NP$.*

Proof. We assume all the preconditions hold and follow the construction of Theorems 18 and 22. We again draw a graph F from graphs of size N and gap $N^{1-\varepsilon}$. We substitute every vertex v by an independent set $I(v)$ of size between $KN^{\frac{1+\delta}{1-\delta}}$ and $\frac{1}{K}N^{\frac{1+\delta}{1-\delta}}$ such that there is a $G \in \mathcal{C}$ of the same size as the obtained graph G', and the sets $I(v)$ are balanced (their size differs by at most 1). Both graphs have $\Theta(N^{\frac{1+\delta}{1-\delta}+1}) = \Theta(N^{\frac{2}{1-\delta}})$ vertices, say $cN^{\frac{2}{1-\delta}}$.

We arbitrary identify the vertices of G and G' in a one-to-one mapping. We keep an edge between two vertices u and v if uv is both an edge in G and G'. Thus we do the "intersection" of G and G'. We call J the final result.

Since G has girth at least γ, J has also girth at least γ. By assumption on $\mathcal{C}$, if there is an edge between u and v in F, then for every $A \subset I(u)$ and $B \subset I(v)$ both of size $c^\delta N^{\frac{2\delta}{1-\delta}}$, there is at least one edge in $E_J(A,B)$. We observe that $N^{\frac{2\delta}{1-\delta}} = N^{\frac{1+\delta}{1-\delta}-1}$ which is, up to constant multiplicative factors, the size of an $I(w)$ divided by N. Therefore we have the same important property as in Theorems 18 and 22. Thus we can finish the proof similarly, and conclude that distinguishing between instances with independence number at most $N^{\frac{1+\delta}{1-\delta}+\varepsilon}$ or at least $N^{\frac{2}{1-\delta}-\varepsilon}$ is NP-hard, for an arbitrary small $\varepsilon > 0$. The gap is $N^{1-\varepsilon}$ and $n := |V(J)| = \Theta(N^{\frac{2}{1-\delta}})$, hence a gap of $n^{\frac{1-\delta}{2}-\varepsilon}$. ◀

We now give a randomized counterpart of the previous theorem. For any integer $\gamma > 3$ and real $\delta \in (0,1)$, we say that a distribution of graphs $\mathcal{D}$ is (γ,δ)-*appropriate* if there is a constant $K \in (0,1]$ and a polynomial-time algorithm, that given an integer n (encoded in unary), draws a graph G of size at least Kn and at most n/K out of this distribution such that with high probability, G has girth at least γ and no disjoint pair of sets $A, B \subseteq V(G)$ satisfies both $|A| = |B| \geqslant |V(G)|^\delta$ and $E(A,B) = \emptyset$.

▶ **Theorem 25.** *Let $\gamma > 3$ be an integer, $\delta \in (0,1)$ be a real (allowed to depend on γ), and $\mathcal{D}$ be a (γ,δ)-appropriate distribution. Then MIS in $\mathcal{G}_\gamma$ cannot be $n^{\frac{1-\delta}{2}-\varepsilon}$-approximated, unless NP $\subseteq$ BPP.*

Proof. The proof is the same as Theorem 24, using a graph drawn from the distribution $\mathcal{D}$ instead of a deterministic one from $\mathcal{C}$. Therefore we need the stronger assumption that NP is not contained in BPP. ◀

There are many constructions, all randomized, of triangle-free graphs with smallest possible independence number $\tilde{O}(\sqrt{n})$ [17, 29, 18, 28, 7]. These constructions all follow a simple scheme of starting from the empty graph, ordering the edges of the clique K_n, and then inserting an edge if it does not create a triangle, either among the inserted edges or among all the previous edges. The real difficulty is in the analysis of this probabilistic experiment. The logarithmic or constant factors were improved and the proofs simplified until Kim obtained a matching bound of $O(\sqrt{n \log n})$ [28]. This can be seen as the lower bound of $\Omega(n^2/\log n)$ for the off-diagonal Ramsey number $R(3,n)$, matching the upper bound $O(n^2/\log n)$ of Ajtai et al. [1].

To apply Theorem 25, we would need to check that the triangle-free graphs built in the aforementioned papers do not contain the complement of a large biclique K_{n^δ,n^δ}. As, for our purposes, we do not need the optimal bound of Kim, we follow the original proof of Erdős [17] giving the bound of $O(\sqrt{n}\log n)$. Going through all the lemmas and replacing occurrences of K_x, where $x = O(\sqrt{n}\log n)$, by $K_{x,x}$, the desired result can be obtained. In our language, the process described in the previous paragraph yields a $(4,1/2)$-appropriate distribution. This together with Theorem 25 improves the inapproximability of Theorem 18.

▶ **Corollary 26.** *MIS in $\mathcal{G}_4$ (i.e., triangle-free graphs) cannot be $n^{\frac{1}{4}-\varepsilon}$-approximated, unless NP $\subseteq$ BPP.*

We are not aware of any explicit deterministic construction of triangle-free graphs whose complements do not contain $K_{n^{2/3},n^{2/3}}$ as a subgraph (which would derandomize Theorem 18), let alone, $K_{\sqrt{n},\sqrt{n}}$. Deterministic constructions of graphs with large girth and large chromatic number, such as Ramanujan graphs with non-constant degree, might give some lower bound

via Theorem 24, but not as good as Corollary 26. Actually, being based on a tight construction, the inapproximability of Corollary 26 can only be improved via a totally different route. One should also not completely rule out that there is an $n^{1/4}$-approximation for MIS on triangle-free graphs.

5 Concluding remarks

The Erdős-Hajnal conjecture has proven particularly difficult. For example, the cases of P_5-free or C_5-free graphs are both wide open. For the few graphs H for which a proof that the Erdős-Hajnal property holds, it appears that the proof comes with an efficient algorithm reporting a sufficiently large independent set or clique. This is what we called the constructive Erdős-Hajnal property. We proposed a first and more humble step (see Theorem 6) in proving that a graph H has the constructive Erdős-Hajnal property: show that MIS in H-free graphs can be approximated within ratio $n^{1-\varepsilon}$ for an $\varepsilon > 0$, an unachievable ratio in general graphs. As mentioned in the introduction, this is strictly simpler than Erdős-Hajnal considering the case of P_5. Yet it does not seem to us that this weaker conjecture is that much simpler now considering the graph C_5. We believe that efforts to settle the improved approximation conjecture might turn out useful to make progress on the Erdős-Hajnal conjecture. In general, a cross-fertilization between Approximability Theory and the study of favorable Ramsey properties may prove fruitful. In particular, obtaining an $n^{0.99}$-approximation algorithm for MIS in C_5-free seems like a challenging open question.

Of course, classifying the approximability of Maximum Independent Set in H-free graphs is also an interesting task by its own means. On the one hand, already known reductions rule out PTASes in most H-free graphs classes, namely for any connected H different from a path or a subdivision of a claw. On the other hand, a constant-approximation algorithm can be turned into a PTAS in many H-free classes, by running the approximation on the input graph elevated to some appropriate power (using for instance the lexicographic product). This trick, originally used to rule out approximation algorithms for Max Clique in general graphs [20], works in the setting of H-free classes when H satisfies some properties, such as being a prime graph (i.e., having no non-trivial module). Hence, although MIS admits a constant-factor approximation in $K_{1,t}$-free graphs for any $t \in \mathbb{N}$ (as mentioned in Section 3), it is not in APX when the forbidden graph is a simple tree, such as the 1-subdivision of $K_{1,4}$. Finally, another interesting consequence of the previous observation concerns P_t-free graphs: any constant-factor approximation for MIS in P_t-free graphs implies a PTAS (notice that the current "best" approximation algorithm in P_t-free graphs is a quasi-polynomial approximation scheme [13]).

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi. A note on ramsey numbers. *J. Comb. Theory, Ser. A*, 29(3):354–360, 1980. doi:10.1016/0097-3165(80)90030-8.

2 S. Alekseev. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15, issue 2:307–309, 1974.

3 V. E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982.

4 Vladimir E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discrete Applied Mathematics*, 135(1-3):3–16, 2004. doi:10.1016/S0166-218X(02)00290-1.

5 Paola Alimonti and Viggo Kann. Some apx-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1):123–134, 2000. doi:10.1016/S0304-3975(98)00158-3.

6 Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, January 1994. doi:10.1145/174644.174650.

7 Béla Bollobás. *Random Graphs, Second Edition*, volume 73 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2011. doi:10.1017/CBO9780511814068.

8 John Adrian Bondy and Uppaluri S. R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2008. doi:10.1007/978-1-84628-970-5.

9 Andreas Brandstädt and Raffaele Mosca. Maximum weight independent set for claw-free graphs in polynomial time. *Discrete Applied Mathematics*, 237:57–64, 2018. doi:10.1016/j.dam.2017.11.029.

10 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.

11 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.

12 Maria Chudnovsky. The erdös-hajnal conjecture - A survey. *Journal of Graph Theory*, 75(2):178–190, 2014. doi:10.1002/jgt.21730.

13 Maria Chudnovsky, Marcin Pilipczuk, Michal Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the maximum weight independent set problem in H-free graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2260–2278, 2020. doi:10.1137/1.9781611975994.139.

14 E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 637–646, October 2005. doi:10.1109/SFCS.2005.14.

15 Pavel Dvořák, Andreas Emil Feldmann, Ashutosh Rai, and Paweł Rzążewski. Parameterized inapproximability of independent sets in h-free graphs. In *46th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2020)*, 2020.

16 P. Erdős and A. Hajnal. Ramsey-type theorems. *Discrete Applied Mathematics*, 25(1):37–52, 1989. doi:10.1016/0166-218X(89)90045-0.

17 Paul Erdös. Graph theory and probability. ii. *Canadian Journal of Mathematics*, 13:346–352, 1961.

18 Paul Erdős, Stephen Suen, and Peter Winkler. On the size of a random maximal graph. *Random Struct. Algorithms*, 6(2/3):309–318, 1995. doi:10.1002/rsa.3240060217.

19 P. Erdős. Some remarks on the theory of graphs. *Bulletin of the American Mathematical Society*, 53(4):292–294, 1947. doi:10.1090/S0002-9904-1947-08785-1.

20 Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating Clique is Almost NP-Complete (Preliminary Version). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 2–12. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185341.

21 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

22 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

23 Andrzej Grzesik, Tereza Klimosova, Marcin Pilipczuk, and Michal Pilipczuk. Polynomial-time algorithm for maximum weight independent set on p_6-free graphs. *CoRR*, abs/1707.05491, 2017. arXiv:1707.05491.

24 Andrzej Grzesik, Tereza Klimosova, Marcin Pilipczuk, and Michal Pilipczuk. Polynomial-time algorithm for maximum weight independent set on P_6-free graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1257–1271, 2019. doi:10.1137/1.9781611975482.77.

25 Magnús M. Halldórsson. Approximating discrete collections via local improvements. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA.*, pages 160–169, 1995. URL: http://dl.acm.org/citation.cfm?id=313651.313687.

26 Edin Husić, Stéphan Thomassé, and Nicolas Trotignon. The independent set problem is FPT for even-hole-free graphs. *to appear in the proceedings of IPEC 2019*, 2019.

27 Johan Håstad. Clique is hard to approximate within n$^{1\text{-epsilon}}$. In *Acta Mathematica*, pages 627–636, 1996.

28 Jeong Han Kim. The ramsey number r (3, t) has order of magnitude t2/log t. *Random Structures & Algorithms*, 7(3):173–207, 1995.

29 Michael Krivelevich. Bounding ramsey numbers through large deviation inequalities. *Random Struct. Algorithms*, 7(2):145–156, 1995. doi:10.1002/rsa.3240070204.

30 Vadim V. Lozin and Martin Milanic. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *J. Discrete Algorithms*, 6(4):595–604, 2008. doi:10.1016/j.jda.2008.04.001.

31 George J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory, Ser. B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.

32 Burkhard Monien and Ewald Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Inf.*, 22(1):115–123, 1985. doi:10.1007/BF00290149.

33 Owen J Murphy. Computing independent sets in graphs with large girth. *Discrete Applied Mathematics*, 35(2):167–170, 1992.

34 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010. doi:10.1007/s00454-010-9285-9.

35 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.

Reconfiguration of Spanning Trees with Many or Few Leaves

Nicolas Bousquet
CNRS, LIRIS, Université de Lyon,
Université Claude Bernard Lyon 1, France
nicolas.bousquet@liris.cnrs.fr

Takehiro Ito
Graduate School of Information Sciences,
Tohoku University, Japan
takehiro@ecei.tohoku.ac.jp

Yusuke Kobayashi
Research Institute for Mathematical Sciences,
Kyoto University, Japan
yusuke@kurims.kyoto-u.ac.jp

Haruka Mizuta
Graduate School of Information Sciences,
Tohoku University, Japan
haruka.mizuta.s4@dc.tohoku.ac.jp

Paul Ouvrard
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI,
UMR5800, Talence, France
paul.ouvrard@u-bordeaux.fr

Akira Suzuki
Graduate School of Information Sciences,
Tohoku University, Japan
a.suzuki@ecei.tohoku.ac.jp

Kunihiro Wasa
Department of Computer Science and Engineering, Toyohashi University of Technology, Japan
wasa@cs.tut.ac.jp

Abstract

Let G be a graph and T_1, T_2 be two spanning trees of G. We say that T_1 can be transformed into T_2 via an edge flip if there exist two edges $e \in T_1$ and f in T_2 such that $T_2 = (T_1 \setminus e) \cup f$. Since spanning trees form a matroid, one can indeed transform a spanning tree into any other via a sequence of edge flips, as observed in [11].

We investigate the problem of determining, given two spanning trees T_1, T_2 with an additional property Π, if there exists an edge flip transformation from T_1 to T_2 keeping property Π all along.

First we show that determining if there exists a transformation from T_1 to T_2 such that all the trees of the sequence have at most k (for any fixed $k \geq 3$) leaves is PSPACE-complete.

We then prove that determining if there exists a transformation from T_1 to T_2 such that all the trees of the sequence have at least k leaves (where k is part of the input) is PSPACE-complete even restricted to split, bipartite or planar graphs. We complete this result by showing that the problem becomes polynomial for cographs, interval graphs and when $k = n - 2$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases combinatorial reconfiguration, spanning trees, PSPACE, polynomial-time algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.24

Related Version A full version of the paper is available at `https://arxiv.org/abs/2006.14309`.

Funding Partially supported by JSPS and MEAE-MESRI under the Japan-France Integrated Action Program (SAKURA).

Nicolas Bousquet: This work was supported by ANR project GrR (ANR-18-CE40-0032).

Takehiro Ito: Partially supported by JSPS KAKENHI Grant Numbers JP18H04091 and JP19K11814, Japan.

Yusuke Kobayashi: Supported by JSPS KAKENHI Grant Numbers JP17K19960, JP18H05291, and JP20K11692, Japan.

Haruka Mizuta: Partially supported by JSPS KAKENHI Grant Number JP19J10042, Japan.

Paul Ouvrard: This work was supported by ANR project GrR (ANR-18-CE40-0032).

Akira Suzuki: Partially supported by JSPS KAKENHI Grant Numbers JP18H04091 and JP20K11666, Japan.
Kunihiro Wasa: Partially supported by JST CREST Grant Numbers JPMJCR18K3 and JPMJCR1401, and JSPS KAKENHI Grant Number JP19K20350, Japan.

1 Introduction

Given an instance of some combinatorial search problem and two of its feasible solutions, a *reconfiguration problem* asks whether one solution can be transformed into the other in a step-by-step fashion, such that each intermediate solution is also feasible. Reconfiguration problems capture dynamic situations, where some solution is in place and we would like to move to a desired alternative solution without becoming infeasible. A systematic study of the complexity of reconfiguration problems was initiated in [11]. Recently the topic has gained a lot of attention in the context of constraint satisfaction problems and graph problems, such as the independent set problem, the matching problem, and the dominating set problem. Reconfiguration problems naturally arise for operational research problems but also are closely related to uniform sampling using Markov chains (see e.g. [5]) or enumeration of solutions of a problem. Reconfiguration problems received an important attention in the last few years. For an overview of recent results on reconfiguration problems, the reader is referred to the surveys of van den Heuvel [14] and Nishimura [13].

In this paper, our reference problem is the spanning tree problem. Let $G = (V, E)$ be a connected graph on n vertices. A *spanning tree* of G is a tree (chordless graph) with exactly $n - 1$ edges. Given a tree T, a vertex v is a *leaf* if its degree is one and is an *internal node* otherwise. A *branching node* is a vertex of degree at least three.

In order to define valid step-by-step transformations, an adjacency relation on the set of feasible solutions is needed. Depending on the problem, there may be different natural choices of adjacency relations. Let T_1 and T_2 be two spanning trees of G. We say that T_1 and T_2 differs by an *edge flip* if there exist $e_1 \in E(T_1)$ and $e_2 \in E(T_2)$ such that $T_2 = (T_1 \setminus e_1) \cup e_2$. Two trees T_1 and T_2 are adjacent if one can transform T_1 into T_2 via an edge flip. A *transformation* from T_s to T_t is a sequence of trees $\langle T_0 := T_\mathsf{s}, T_1, \ldots, T_r := T_\mathsf{t} \rangle$ such that two consecutive trees are adjacent. Ito et al. [11] remarked that any spanning tree can be transformed into any other via a sequence of edge flips. It easily follows from the exchange properties for matroid. Unfortunately, the problem becomes much harder when we add some restriction on the intermediate spanning trees. One can then ask the following question: does it still exist a transformation when we add some constraints on the spanning tree? If not, is it possible to decide efficiently if such a transformation exists? This problem was already studied for vertex modification between Steiner trees [12] for instance.

In this paper, we consider spanning trees with restrictions on the number of leaves. More precisely, what happens if we ask the number of leaves to be large (or small) all along the transformation? We formally consider the following problems:

Spanning Tree with Many Leaves
Input: A graph G, an integer k, two trees T_1 and T_2 with at least k leaves.
Output: yes if and only if there exists a transformation from T_1 to T_2 such that all the intermediate trees have at least k leaves.

In the Spanning Tree with At Most k Leaves problem, we instead want to find a transformation such that all the intermediate trees have at most k leaves (where k is a fixed constant).

Our results

We prove that both variants are PSPACE-complete. In other words, we show that Spanning Tree with Many Leaves and Spanning Tree with At Most k Leaves for every $k \geq 3$ are PSPACE-complete. This constrasts with many existing results on reconfiguration problems using edge flips which are polynomial such as matching reconfiguration [11], cycle, tree or clique reconfiguration [8]. As far as we know there does not exist any PSPACE-hardness proof for any problem via edge flip. We hope that our results will help to design more.

▶ **Theorem 1.** *Spanning Tree with Many Leaves is PSPACE-complete restricted to bipartite graphs, split graphs or planar graphs.*

These results are obtained from two different reductions. In both reductions, we need an arbitrarily large number of leaves in order to make the reduction work. In particular, one can ask the following question: is Spanning Tree with at least $n - k$ Leaves hard for some constant k (where n is the size of the instance)? We do not answer this question but we prove that, for the "dual" problem, the PSPACE-hardness is obtained even for $k = 3$.

▶ **Theorem 2.** *Spanning Tree with At Most k Leaves is PSPACE-complete for every $k \geq 3$.*

This proof is the most technically involved proof of this article and is based on a reduction from the decision problem of Vertex Cover to the decision problem of Hamiltonian Path. Let (G, k) be an instance of Vertex Cover. We first show that, on the graph H obtained when we apply this reduction, we can associate with any spanning tree T of H a vertex cover of G. The hard part of the proof consists in showing that *(i)* if T has at most three leaves, then the vertex cover associated with T has at most $k + 1$ vertices; and *(ii)* each edge flip consists of a modification of at most one vertex of the associated vertex cover.

One can note that for $k = 2$, the problem becomes the Hamiltonian Path Reconfiguration problem. We were not able to determine the complexity of this problem and we left it as an open problem.

We complete these results by providing some polynomial-time algorithms:

▶ **Theorem 3.** *Spanning Tree with Many Leaves can be decided in polynomial time on interval graphs, on cographs, or if the number of leaves is $n - 2$.*

We show that Spanning Tree with Many Leaves can be decided in polynomial time if the number of leaves is $n - 2$. As we already said, we left as an open question to determine if this result can be extended to any value $n - k$ for some fixed k. If such an algorithm exists, is it true that the problem is FPT parameterized by k?

We then show that in the case of cographs, the answer is always positive as long as the number of leaves is at most $n - 3$. Since there is a polynomial-time algorithm to decide the problem when $k = 2$ that completes the picture for cographs.

Since the problem is known to be PSPACE-complete for split graphs by Theorem 1 (and thus for chordal graphs), the interval graphs result is the best we can hope for in a sense. The interval graph result is based on a dynamic programming algorithm inspired by [2] where it is proved that the Independent Set Reconfiguration problem in the token sliding model is polynomial. Even if dynamic algorithms work quite well to decide combinatorial problems on interval (and even chordal) graphs, they are much harder to use in the reconfiguration setting. In particular, many reconfiguration problems become hard on chordal graphs (see e.g. [1, 9]) since the transformations can go back and forth.

Since the problem is hard on planar graphs, it would be interesting to determine its complexity on outerplanar graphs. We left this question as an open problem.

Related work

In the last few years, many graph reconfiguration problems have been studied through the lens of edge flips such as matchings [11, 4], paths or cycles [8]. None of these works provide any PSPACE-hardness results, only a NP-hardness result is obtained for (non Hamiltonian) path reconfiguration via edge flips in [8]. Even if the reachability problem is known to be polynomial in many cases, approximating the shortest transformation is often hard, see e.g. [4]. Flips are also often considered in computational geometry, for instance to measure the distance between two triangulations. In that setting, a flip of a triangulation is the modification of a diagonal of a C_4 for the other one. Usually, proving the existence of a transformation is straightforward and the main questions are about the length of a transformation which is not the problem addressed in this paper.

If, instead of "edge flips", we consider "vertex flips" the problems become much harder. For instance, the problem consisting in transforming an (induced) tree into another one (of the same size) is PSPACE-complete [8] (while the exchange property ensures that it is polynomial for the edge version). Mizuta et al. [12] also showed that the existence of vertex exchanges between two Steiner trees is PSPACE-complete. But transforming subsets of vertices with some properties is known to PSPACE-complete for a long time, for instance for independent sets or cliques [10].

Definitions

Given two sets S_1 and S_2, we denote by $S_1 \triangle S_2$ the *symmetric difference* of the sets S_1 and S_2, that is $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$.

For a spanning tree T, every vertex of degree one is a *leaf* and every vertex of degree at least two is an *internal node*. A vertex of degree at least three is called a *branching node*. Recall that the number of leaves of any tree T is equal to $(\sum_{v \in T}(\max\{0, d_T(v) - 2\})) + 2$. We denote by $in(T)$ the number of internal nodes of T. Note that if T contains n nodes, the number of leaves is indeed $n - in(T)$.

Let $G = (V, E)$ be a graph. A *vertex cover* C of G is a subset of vertices such that for every edge $e \in E$, C contains at least one endpoint of e. C is *minimum* if its cardinality is minimum among all vertex covers of G. Note that in particular, C is inclusion-wise minimal and thus for every vertex $u \in C$, there is an edge $e \in E$ which is covered only by u. We denote by $\tau(G)$ the size of a minimum vertex cover of G.

Let X, Y be two vertex covers of G. X and Y are *TAR-adjacent*[1] (resp. TJ-adjacent) if there exists a vertex x (resp. x and y) such that $X = Y \cup \{x\}$ or $Y = X \cup \{x\}$ (resp. $X = Y \setminus \{y\} \cup \{x\}$). We will consider the following problem:

Minimum TAR-Vertex Cover Reconfiguration

Input: A graph G, two minimum vertex covers X, Y of size k.

Output: yes if and only if there exists a sequence from X to Y of TAR-adjacent vertex covers, all of size at most $k + 1$.

[1] TAR stands for "Token Additional Removal".

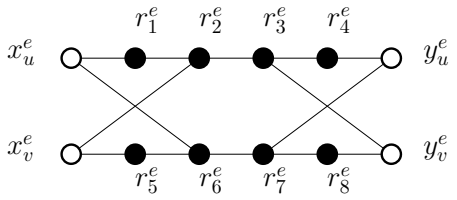


Figure 1 edge-gadget. The white vertices are the only ones connected to the outside.

Similarly, one can define the Minimum TJ-Vertex Cover Reconfiguration (MVCR for short) where we want to determine whether there exists a sequence of TJ-adjacent vertex covers from X to Y. Note that all the vertex covers must be of size $|X| = |Y| = k$.

2 Spanning trees with few leaves

▶ **Theorem 4.** *Spanning Tree with At Most three Leaves is PSPACE-complete.* [2]

In order to prove Theorem 4, we will provide a reduction from Minimum TAR-Vertex Cover Reconfiguration to Spanning Tree with At Most three Leaves.

▶ **Theorem 5** (Wrochna [15])**.** TAR-*Vertex Cover Reconfiguration is PSPACE complete even for bounded bandwidth graphs.*

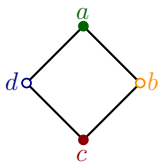
The idea of the proof of Theorem 4 consists in adapting a reduction from Vertex cover to Hamiltonian Path (for the optimization version). Let $(G = (V, E), k)$ be an instance of Vertex Cover. This reduction creates a graph $H(G)$ which contains a Hamiltonian path if and only if G admits a vertex cover of size k. The reduction is given in Section 2.1 together with some properties of the spanning trees with at most three leaves in $H(G)$. In order to adapt the proof in the reconfiguration setting, we need to prove that the proof is "robust" with respect to several meanings of the word. First, we need to show that, if we consider a spanning tree with at most three leaves in $H(G)$ then there is a "canonical" vertex cover of size at most $k + 1$ associated with it (it is the most technical part of the proof). Then, for any edge flip between two spanning trees with at most three leaves, we need to show that the corresponding vertex covers associated with them are TAR-adjacent . We will indeed also need to prove the reverse direction.

2.1 The Reduction

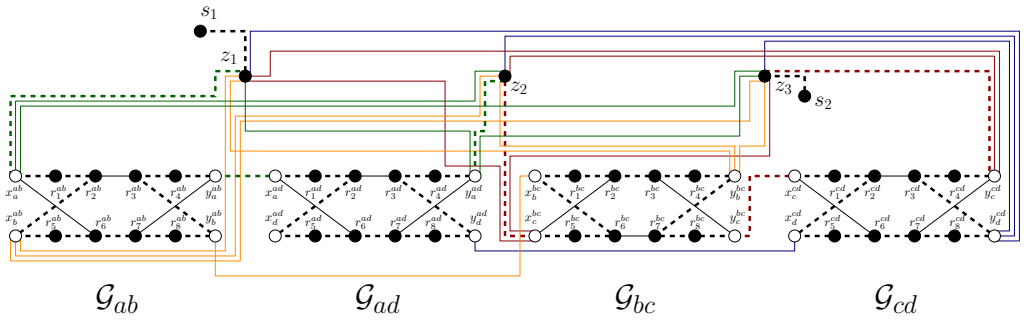
The reduction is a classical reduction (see Theorem 3.4 of [6] for a reference) from the optimization version of Vertex Cover to the optimization version of Hamiltonian Path. Let G be a graph and k be an integer. Let us construct a graph $H(G)$ (abbreviated into H when no confusion is possible) as follows:

Construction of $H(G)$. For each edge $e = uv$ of G, we create the following *edge-gadget* $\mathcal{G}_e$ represented in Figure 1. The edge-gadget $\mathcal{G}_e$ has four *special vertices* denoted by $x_u^e, x_v^e, y_u^e, y_v^e$. The vertices x_u^e and x_v^e are called the *entering vertices* and y_u^e and y_v^e the *exit vertices*. The gadget contains eight additional vertices denoted by $r_1^e, \ldots, r_8^e$. When e is clear from context, we will omit the superscript. The graph induced by these twelve vertices is represented in Figure 1. The vertices $r_1^e, \ldots, r_8^e$ are *local vertices* and their neighborhood will be included in the gadget. The only vertices connected to the rest of the graphs are the special vertices.

[2] Note that the reduction can be easily adapted to more leaves.



(a) Original instance (G, k) of Minimum Vertex Cover with a vertex cover $\{a, b\}$.



(b) Graph $H(G)$ obtained from the reduction. The ordering for the vertices of the vertex cover $\{a, b\}$ of G is the lexicographic ordering, as well as the ordering of the edges incident to each vertex. The corresponding Hamiltonian path is depicted by the thick dashed edges.

Figure 2 Illustration of the reduction of Theorem 4.

We add an independent set $Z := \{z_1, \ldots, z_{k+1}\}$ of $k+1$ new vertices to $V(H)$. And we finally add to $V(H)$ two more vertices s_1, s_2 in such a way that z_1 (resp. z_{k+1}) is the only neighbor of s_1 (resp. s_2) in $H(G)$. Since s_1 and s_2 have degree one in $H(G)$, s_1 and s_2 are leaves in any spanning tree of $H(G)$. In particular, the two endpoints of any Hamiltonian path of $H(G)$ are necessarily s_1 and s_2.

Let us now complete the description of $H(G)$ by explaining how the special vertices are connected to the other vertices of $H(G)$. Let $u \in V(G)$. Let $E' = e_1, \ldots, e_\ell$ be the set of edges incident to u in an arbitrary order. We connect $x_u^{e_1}$ and $y_u^{e_\ell}$ to all the vertices of Z. For every $1 \leq i \leq \ell - 1$, we connect $y_u^{e_i}$ to $x_u^{e_{i+1}}$. The edges $y_u^{e_i} x_u^{e_{i+1}}$ are called the *special edges of* u. The *special edges* of $H(G)$ are the union of the special edges for every $u \in V(G)$ plus the edges incident to Z but $s_1 z_1$ and $s_2 z_{k+1}$. This completes the construction of $H(G)$ (see Figure 2 for an example).

▶ Remark 6. If T is a spanning tree of $H(G)$ with at most ℓ leaves, then at most $\ell - 2$ of them are in $V(H) \setminus \{s_1, s_2\}$.

Let T be a spanning tree of $H(G)$. An edge-gadget is *irregular* if at least one of its twelve vertices is not of degree two in T. An edge-gadget is *regular* if it is not irregular. By abuse of notation we say that $e \in E(G)$ is regular (resp. irregular) if the edge-gadget of e is regular (resp. irregular). A vertex u is *regular* if every edge incident to u is regular. The vertex u is *irregular* otherwise.

Let S be a subset of vertices of $H(G)$. We denote by $\delta_T(S)$ the set of edges with exactly one endpoint in S. When there is no ambiguity, we omit the subscript T. Moreover, if S is the singleton $\{u\}$, we write $\delta_T(u)$ for $\delta_T(\{u\})$. The restriction $T(\mathcal{G}_e)$ of a spanning tree T *around an edge-gadget* $\mathcal{G}_e$ is the set of edges with both endpoints in $\mathcal{G}_e$ plus the edges of $\delta_T(\mathcal{G}_e)$ (which are considered as "semi edge" with one endpoint in $\mathcal{G}_e$).

▶ **Lemma 7.** *Let T be a spanning tree of H and $\mathcal{G}$ be a regular edge-gadget. Then the tree T around the edge-gadget $\mathcal{G}$ is one of the two graphs represented in Figure 3. Note that the graph of Figure 3(b) has to be considered up to symmetry between u and v.*

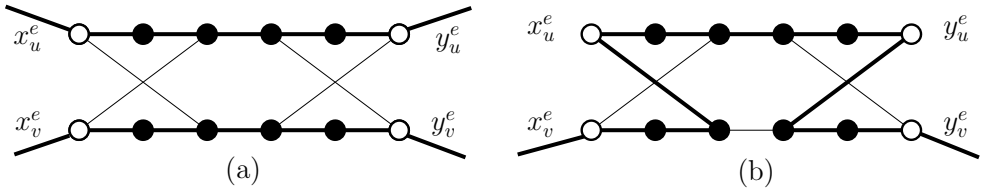


Figure 3 The two possible sub-graphs around a regular edge-gadget $\mathcal{G}$. Bold edges are edges in the tree. Edges with one endpoint in the gadget are edges of $\delta(\mathcal{G})$.

▶ **Lemma 8 (*).** *Let G be a graph, T be a spanning tree of $H(G)$, and u be a regular vertex of T. If there exists an edge $e \in E(G)$ with endpoint u such that x_u^e or y_u^e has degree one in the subgraph of T induced by the vertices of $H[\mathcal{G}_e]$, then, for every edge e' with endpoint u, $x_u^{e'}$ and $y_u^{e'}$ have degree one in the subgraph of T induced by the vertices of $H[\mathcal{G}_{e'}]$.*
In particular, there is an edge of T between Z and the first entering vertex of u and an edge between Z and the last exit vertex of u.

If, for a regular vertex u and an edge $e = uv$, x_u^e or y_u^e have degree one in $H[\mathcal{G}_e]$, then there is a path between two vertices of Z passing through all the special vertices $x_u^{e'}$ and $y_u^{e'}$ for every e' incident to u and all the vertices on this path have degree two. Note that the union of all such vertices forms a vertex cover of G.

2.2 Reconfiguration hardness

Let T be a spanning tree with at most three leaves. By Lemma 7, for every edge-gadget $\mathcal{G}_e$, if $T(\mathcal{G}_e)$ is not one of the two graphs of Figure 3, $\mathcal{G}_e$ contains a branching node or a leaf. So Remark 6 implies:

▶ Remark 9. There are at most two irregular edge-gadgets. Thus there are at most four irregular vertices.

Indeed, if T has two leaves, all the edge-gadgets are regular. If T has three leaves, the third leaf must be in an edge-gadget, creating an irregular edge-gadget. And this leaf might create a new branching node which might be in another edge-gadget than the one of the third leaf. So the number of irregular edge-gadget is at most two, and thus the number of irregular vertices is at most four (if the edges corresponding to these two edge-gadgets have pairwise distinct endpoints).

Let T be a spanning tree of $H(G)$ with at most three leaves. A vertex v is *good* if there exists an edge $e = vw$ for $w \in V(G)$ such that x_v^e or y_v^e has degree one in the subtree of T induced by the twelve vertices of the edge-gadget of e. In other words, if we simply look at the edges of T with both endpoints in $\mathcal{G}_e$, x_v^e or y_v^e has degree one (or said again differently, x_v^e or y_v^e are adjacent to exactly one local vertex). Let us denote by $S(T)$ the set of good vertices. Using the fact that every gadget contains at most one vertex of degree three and one vertex of degree one by Remark 6, we can show:

▶ **Lemma 10 (*).** *Let T be a spanning tree with at most three leaves of $H(G)$ and $e = uv$ be an edge of G. At least one special vertex of the edge-gadget $\mathcal{G}_e$ has degree one in the subgraph of T induced by the vertices of $\mathcal{G}_e$. In particular, $S(T)$ is a vertex cover.*

So, for every tree T with at most three leaves, $S(T)$ is a vertex cover. We say that $S(T)$ is *the vertex cover associated with T*.

The next two technical lemmas ensure that an edge flip transformation provides a TAR-vertex cover reconfiguration sequence.

▶ **Lemma 11 (*).** *Every spanning tree T of $H(G)$ with at most three leaves satisfies $|S(T)| \leq k+1$.*

Sketch of the proof. Assume by contradiction that $|S| \geq k+2$. By Remark 9, at least $k-2$ vertices of S are regular. By Lemma 8, for each regular vertex $w \in S$, there is an edge of T between Z and the first entering vertex of w and Z and the last exit vertex of w. So at least $2k-4$ edges of $\delta_T(Z)$ are incident to regular vertices. Moreover two edges of $\delta_T(Z)$ are incident to s_1 and s_2. So, T already has $2k-2$ edges in $\delta_T(Z)$. Since $|Z| = k+1$ and T has at most three leaves, Remark 6 ensures that $\delta_T(Z)$ has size $2k+1, 2k+2$ or $2k+3$. The main part of the proof, not included in this extended abstract, consists in proving that the edges between Z and entering or exit vertices of irregular vertices is too large. ◀

So the vertex cover $S(T)$ associated with every spanning tree T with at most three leaves has size at most $k+1$. In order to prove that a spanning tree transformation provides a vertex cover transformation for the TAR setting, we have to prove that, for every edge flip, then either S is not modified, or one vertex is added to S or one vertex is removed from S.

▶ **Lemma 12 (*).** *Let T_1 and T_2 be two adjacent trees with at most three leaves. Then the symmetric difference between the sets S associated with the two trees is at most one.*

Lemmas 11 and 12 immediately implies the following:

▶ **Lemma 13.** *If there is an edge flip reconfiguration sequence between two spanning trees T_1 and T_2, then there is a TAR-reconfiguration sequence (with threshold $k+1$) between $S(T_1)$ and $S(T_2)$.*

We refer the reader to the complete version for a proof of the converse direction.

3 Spanning tree with many leaves

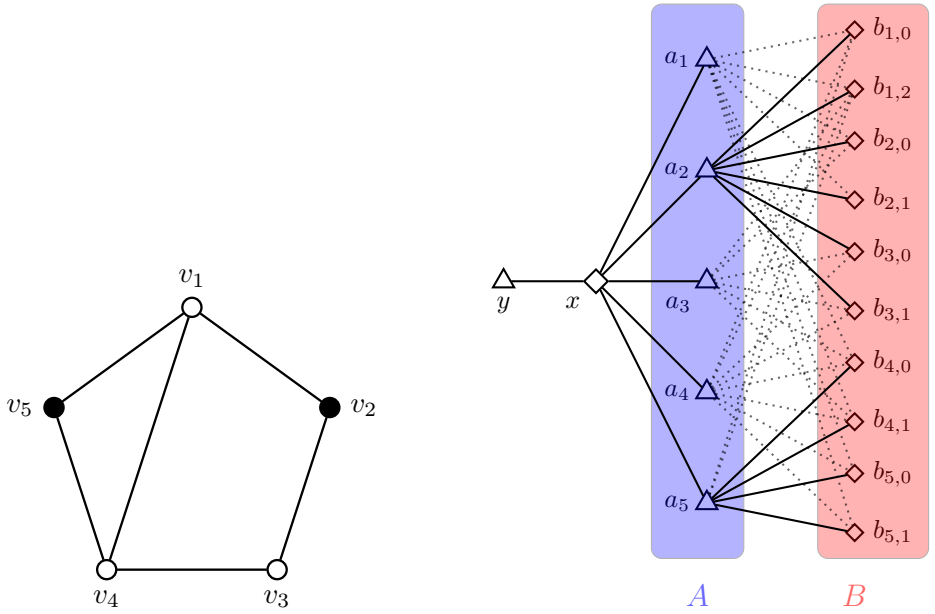
Before stating the main results of this section, let us prove the following:

▶ **Lemma 14 (*).** *Let G be a graph and T_1, T_2 be two trees. There exists a transformation from T_1 to T_2 such that every intermediate tree T satisfies $in(T) \subseteq in(T_1) \cup in(T_2)$.*
In particular, all the trees with the same set of internal nodes are in the same connected component of the reconfiguration graph.

3.1 Hardness results

▶ **Theorem 15.** *Spanning Tree with Many Leaves is PSPACE-complete even restricted to bipartite graphs or split graphs.*

Sketch of the proof. We first briefly explain the proof for bipartite graphs. We provide a polynomial-time reduction from the TAR-Dominating Set Reconfiguration problem (abbreviated in TAR-DSR problem). Haddadan et al [7]. showed that the TAR reconfiguration of dominating sets is PSPACE-complete. More precisely, they proved that given a graph G and D_s, D_t two dominating sets of G, deciding whether there is a reconfiguration sequence between D_s and D_t under the TAR$(\max(|D_s|, |D_t|) + 1)$ rule is PSPACE-complete.



(a) Original graph G. **(b)** Corresponding bipartite graph G'.

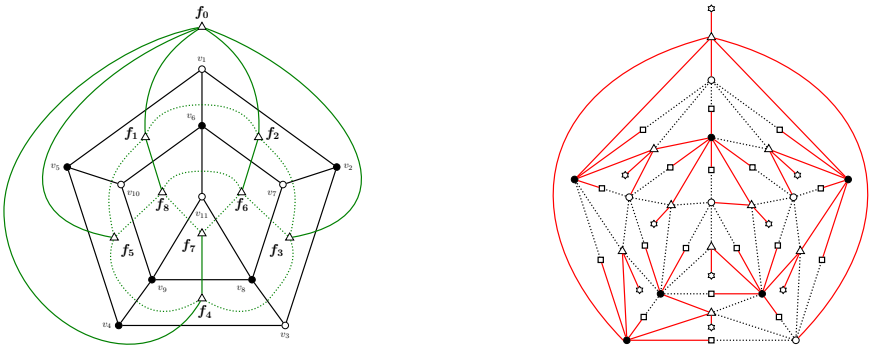
Figure 4 Example for the reduction of Theorem 15: the dominating set $D = \{v_2, v_5\}$ of G is depicted by the black vertices and the spanning tree of G' associated with D is the tree induced by the solid edges. For the split case, we add all the possible edges in $G'[A]$ so that $G'[A \cup \{x\}]$ is a clique and $G'[B \cup \{y\}]$ an independent set.

Let $G = (V, E)$ be a graph with vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$ and let D_s, D_t be two dominating sets of G. Free to add vertices to the set of smallest size, we can assume without loss of generality that D_s and D_t are both of size k. Let $(G, k+1, D_\mathsf{s}, D_\mathsf{t})$ be the corresponding instance of Dominating Set Reconfiguration under TAR, where $k+1$ is the threshold that we cannot exceed. We construct the bipartite graph G' as follows: we make a first copy $A = \{a_1, a_2, \ldots, a_n\}$ of the vertex set of G, and a second copy $B = \{b_{1,0}, b_{1,1}, b_{2,0}, b_{2,1}, \ldots, b_{n,0}, b_{n,1}\}$ where we double each vertex. We add an edge between $a_i \in A$ and $b_{j,k} \in B$ for $k \in \{0, 1\}$ if and only if $v_j \in N_G[v_i]$. Note that $N(b_{i,0}) = N(b_{i,1})$, for every $1 \leq i \leq n$. We finally add a vertex x adjacent to all the vertices in A and we attach it to a degree-one vertex y. Note that G' is bipartite since $A \cup \{y\}$ and $B \cup \{x\}$ induce two independent sets (see Figure 4 for an illustration).

▷ Claim 16 (*). For every spanning tree T of G', $in(T) \cap A$ is a dominating set of G.

▷ Claim 17 (*). For every spanning tree T of G', there exists a tree T_A in the same connected component of T in the reconfiguration graph such that $in(T_A) \subseteq in(T) \cap (A \cup \{x\})$.

Let D be a dominating set of G of size k. We can associate with D a spanning tree of G' with $k+1$ internal nodes as follows. We attach every vertex in $A \cup \{y\}$ to x. Every vertex $b_i \in B$ is a leaf adjacent to a vertex that dominates v_i in D. If v_i has more than one neighbor in D, we choose the one with the smallest index. This spanning tree is called the *spanning tree associated with* D. Due to space restrictions, the proof that $(G, k+1, D_\mathsf{s}, D_\mathsf{t})$ is yes-instance of TAR-DSR if and only if $(G', k', T_\mathsf{s}, T_\mathsf{t})$ is a yes-instance of Spanning Tree with Many Leaves is not included in this extended abstract.



(a) Original labeled planar graph G. **(b)** Corresponding planar graph G'.

Figure 5 Reduction for Theorem 18. The vertex cover C of G is depicted by the black vertices. The dual graph is the graph induced by the green edges. The spanning tree obtained from the BFS is represented by the solid edges. The face-vertices (respectively edge-vertices) of G' are depicted by triangles (resp. squares). The spanning tree T of G' associated with the vertex cover C is the tree induced by the red edges. The number of leaves of T is $2(|E(G)|+1)-|C| = 32$.

Let us now quickly explain how to adapt this proof for split graphs. We first add an edge between any two vertices in A so that $G'[A]$ is a clique. Then, observe that $G'[A \cup \{x\}]$ is a clique, and $G'[B \cup \{y\}]$ an independent set. The proof that the resulting instance is a yes-instance of Spanning Tree with Many Leaves if and only if $(G, k+1, D_s, D_t)$ is a yes-instance of TAR-DSR is similar to the one for bipartite graphs (see the full version). ◀

▶ **Theorem 18.** *Spanning Tree with Many Leaves is PSPACE-complete even restricted to planar graphs.*

The reduction. First, observe that MVCR is PSPACE-complete, even if the input graph is planar [10][3]. We use a reduction from MVCR, which is a slight adaptation of the reduction used in [12, Theorem 4]. Let $G = (V, E)$ be a planar graph and let (G, C_s, C_t) be an instance of MVCR. We can assume that G is given with a planar embedding of G since such an embedding can be found in polynomial time. Let $F(G)$ be the set of faces of G (including the outer face). We construct the corresponding instance (G', k, T_s, T_t) as follows:

We define G' from G as follows. We start from G and first subdivide every edge $uv \in E(G)$ by adding a new vertex w_{uv}. Then, for every face $f \in F(G)$, we add a new vertex w_f adjacent to all the vertices of the face f. Finally, we attach a leaf u_f to every vertex w_f. Note that G' is a planar graph and $|V(G')| = |V(G)| + |E(G)| + 2 \cdot |F(G)|$. The vertices w_{uv} for $uv \in E$ (resp. w_f for $f \in F$) are *edge-vertices* (resp. *face-vertices*). The vertices u_f for every f are called the *leaf-vertices*. Note that, for every spanning tree T, all the face-vertices are internal nodes of T and all the leaf-vertices are leaves of T. The vertices of $V(G')$ which are neither edge, face of leaf vertices are called *original vertices*. Finally, we choose an arbitrarily ordering of $V(G)$ and F. It will permit us to define later a canonical spanning tree for every vertex cover (see Figure 5 for an example).

▶ **Lemma 19 (*).** *Every spanning tree of G' has at most $2(|E(G)|+1) - \tau(G)$ leaves.*

[3] Actually, Hearn and Demaine [10] showed the PSPACE-completeness for the reconfiguration of maximum independent sets. Since the complement of a maximum independent set is a minimum vertex cover, we directly get the PSPACE-completeness of MVCR.

▶ **Lemma 20 (*).** *For any minimum vertex cover C of $G = (V, E)$, we can define a canonical tree with exactly $k := 2(|E(G)| + 1) - \tau(G)$ leaves which are all the edge-vertices, all the leaf-vertices and all the original vertices but the ones in C. Moreover, this spanning can be computed in polynomial time.*

Recall that (G, C_s, C_t) is an instance of Minimum Vertex Cover Reconfiguration. By Lemma 20, we can compute in polynomial time two spanning trees T_s and T_t from C_s and C_t with $2(|E(G)| + 1) - \tau(G)$ leaves. Finally, we set $k := 2(|E(G)| + 1) - \tau(G))$. Let (G', k, T_s, T_t) be the resulting instance of Spanning Tree with Many Leaves. It remains to prove that (G, C_s, C_t) is a yes-instance if and only (G', k, T_s, T_t) is a yes-instance. Suppose first that we have a reconfiguration sequence S between C_s and C_t. By Lemma 20, we can associate with each vertex cover C_i of S a spanning tree T_i of G'. To show that there is a reconfiguration sequence S' between T_s and T_t, we show that we can transform two consecutive spanning trees of S' without increasing the number of internal nodes. Note that we use the fact that each C_i of S is a minimum vertex cover. For the converse direction, we show that all the edge-vertices of any spanning tree in a reconfiguration sequence S from T_s to T_t is a leaf. Hence, one can directly deduce a vertex cover C_i of G from a spanning tree $T_i \in S$. Finally, we show that *(i)* each vertex cover is of size $\tau(G)$ and; *(ii)* $|C_i \triangle C_{i+1}| \in \{0, 2\}$ for any two consecutive vertex covers.

3.2 Two internal nodes and cographs

Recall that, for every tree, the number of leaves is equal to n minus the number of internal nodes. So, for convenience, our goal would consist in minimizing the number of internal nodes rather than maximizing the number of leaves.

▶ **Theorem 21.** *Let G be a graph and T_s or T_t be two spanning trees with at most two internal nodes. Then we can check in polynomial time if one can transform the other via a sequence of spanning trees with at most two internal nodes.*

Sketch of the proof. If T_s or T_t has one internal node, the problem can be easily decided. So we restrict to the case $|in(T_s)| = |in(T_t)| = 2$. Moreover, if $in(T_s) = in(T_t)$, then (G, k, T_s, T_t) is a yes-instance. So we only consider the case $in(T_s) \neq in(T_t)$.

A vertex u is a *pivot* vertex of G if $\deg u \geq n - 2$ in G ($\deg u$ being the size of the neighborhood of u, u not included). A spanning tree T of G is *frozen* if all the spanning trees in its connected component of the reconfiguration graph have the same internal nodes.

▷ Claim 22 (*). Let T be a spanning tree of G. If $in(T)$ does not contain a pivot vertex, then T is frozen.

▷ Claim 23 (*). Let u be a pivot vertex. All the trees containing u as internal vertex are in the same connected component of the reconfiguration graph.

Using these two claims, we can prove that the result follows. ◀

One can naturally wonder if this can be extended to larger values of k or if it is special for $k = 2$. We left this as an open problem. We were only interested in the case $k = 2$ since it was of particular interest for cographs. Indeed, if $k \geq 3$, one can prove that the answer is always positive for cographs. Together with Theorem 21, it implies:

▶ **Theorem 24 (*).** *Spanning Tree with Many Leaves can be decided in polynomial time on cographs.*

3.3 Interval graphs

A graph G is an *interval graph* if G can be represented as an intersection of segments on the line. More formally, each vertex can be represented with a pair (a, b) (where $a \leq b$) and vertices $u = (a, b)$ and $v = (c, d)$ are adjacent if the intervals (a, b) and (c, d) intersect. Let $u = (a, b)$ be a vertex; a is the *left extremity* of u and b the *right extremity* of u. Given an interval graph, a representation of this graph as the intersection of intervals in the plane can be found in $\mathcal{O}(|V| + |E|)$ time (see e.g. [3]). In the rest of the section we assume that a representation is given.

▶ **Theorem 25.** *Spanning Tree with Many Leaves can be decided in polynomial time on interval graphs.*

The proof techniques are inspired from [2]. The rest of this section is devoted to prove Theorem 25. Moreover, if G is a clique, then G is a cograph and then the problem can be decided in polynomial by Theorem 24. So, from now on, we can assume that G is not a clique and in particular $in(G) \geq 2$.

C-minimum spanning trees. Let k be an integer, G be a graph. We denote by $\mathcal{R}(G, k)$ the edge flip reconfiguration graph of the spanning trees of G with at most k internal nodes.

Let T, T' be two spanning trees with the same set of internal nodes. Lemma 14 ensures that T and T' are in the same connected component of $\mathcal{R}(G, k)$. So in what follows, we will often associate a tree T with its set $in(T)$ of internal nodes.

For every interval graph, we can define a spanning tree T_C called the *canonical tree* which minimizes the number of internal vertices and such that for every i, the right extremity of the i-th internal node is maximized.

A tree T is *C-minimum* if no tree T' in the connected component of T in $\mathcal{R}(G, k)$ contains fewer internal nodes than T. The goal of this part consists in showing that all the trees that are not C-minimum are in the connected component of T_C in $\mathcal{R}(G, k)$. The following lemmas follow from basic transformation on spanning trees:

▶ **Lemma 26 (*).** *Let T be a spanning tree of G and $k \geq in(T)$. If there exist two internal nodes u, v of T such that the interval of u is included in the interval of v then T is not C-minimum in $\mathcal{R}(G, k)$. Moreover a tree with internal nodes included in $in(T) \setminus \{u\}$ in the component of T can be found in polynomial time, if it exists.*

▶ **Lemma 27 (*).** *Let T be a spanning tree of G. If there exist three pairwise adjacent internal nodes u, v, w such that $N[u] \subseteq N[v] \cup N[w]$ then T is not C-minimum. Moreover a tree with internal nodes included in $in(T) \setminus \{u\}$ in the connected component of T can be found in polynomial time.*

Note that if u, v, w induce a triangle, then Lemma 26 or 27 holds. So, free to perform some pre-processing operations, we can assume that the set of internal nodes of a spanning T of G induces a path. Indeed, if an internal node x is incident to three other internal nodes u, v, w, then either at least two of them contain the left extremity (or right extremity) of x, or one interval is strictly included in the interval of x. In the first case there is a triangle and we can apply Lemma 26 or 27. In the second case, we can apply Lemma 26.

▶ **Lemma 28 (*).** *Let G be an interval graph and k be an integer. Any spanning tree T of G satisfying $in(T) < k$ is in the connected component of T_C in $\mathcal{R}(G, k)$.*

Sketch of the proof. The proof consists in showing that we can iteratively increase the number of internal nodes on which T and T_C agree without increasing the number of internal nodes at the end of the sequence (and increase it by at most one during the sequence). ◀

Full access. Let T be a tree such that $in(T)$ induces a path. Recall that the left and right extremities orderings agree. The *leftmost vertex* of T is the vertex of $in(T)$ that is minimal for both l and r. The *i-th internal node* of T is the internal node with the i-th smallest left extremity.

Let G be an interval graph and $v \in V(G)$. The *auxiliary graph* H_v of G on v is defined as follows. The vertex set of H_v is v plus the set W of vertices w which end after v and start after the beginning of v (i.e. vertices whose interval ends after v but does not contain v) plus a new vertex x, called the *artificial vertex*. The set of edges of H_v is the set of edges induced by $G[W \cup \{v\}]$ plus the edge xv.

▷ Claim 29 (*). Let G be an interval graph and v be a vertex of G. The graph H_v is an interval graph.

Let $v \in V(G)$. Every spanning tree of H_v necessarily contains v in its set of internal nodes. Indeed, by construction, the graph H_v contains a vertex x of degree one which is only incident to v. Moreover, v is the leftmost internal node of any spanning tree T of H_v.

Let G be an interval graph, $k \in \mathbb{N}$ and T be a spanning tree with internal nodes I such that $|I| = k$. Let $v \in V(G)$. The *restriction* of a spanning tree T to H_v is any spanning tree of H_v with internal nodes included in $(in(T) \cup \{v\}) \cap V(H_v)$. We denote by k'_v (or k' when no confusion is possible) the value $|(in(T) \cup \{v\}) \cap V(H_v)|$. Let T' be the restriction of T to H_v as defined above. One can easily check that the number of internal nodes of T' is at most k'.

The vertex v is *good* if the restriction of T to H_v is not C-minimum in $\mathcal{R}(H_v, k')$. The vertex v is *normal* otherwise. Let v be a normal vertex. Recall that v is the leftmost internal node of any spanning tree of H_v. Let C be the connected component of the restriction of T to H_v in $\mathcal{R}(H_v, k')$. We denote by $\ell'_v(T)$ the second internal node of a spanning tree of H_v in C that minimizes its left extremity. Similarly we denote by $r'_v(T)$ the second internal node of a spanning tree of H_v in C that maximizes its right extremity. When they do not exist[4], we set $\ell'_v(T) = -\infty$ and $r'_v(T) = +\infty$.

We say that we have *full access to* T if, for every vertex $v \in V(G)$, we have a constant time oracle saying if v is good or normal. And if v is normal, we moreover have a constant time access to $\ell'_v(T)$ and $r'_v(T)$. What remains to be proved is that *(i)* knowing this information for two spanning trees T and T' is enough to determine if they are in the same connected component of $\mathcal{R}(G, k)$, and that *(ii)* this information can be computed in polynomial time.

Dynamic programming algorithm. Let us first state the following useful lemma.

▶ **Lemma 30 (*).** *Let G be an interval graph and $k \in \mathbb{N}$. Let T be a spanning tree of G and v be an internal node of T. Let $J := in(T) \cap V(H_v)$ and $k' = |J|$. If a tree T' with internal nodes J can be transformed into a tree with internal nodes K in $\mathcal{R}(H_v, k')$ then T can be transformed into a tree with internal nodes $(in(T) \setminus J) \cup K$ in $\mathcal{R}(G, k)$.*

In particular, if T' is not C-minimum in $\mathcal{R}(H_v, k')$ then T is not C-minimum in $\mathcal{R}(G, k)$.

Let us now prove that if we have full access to H_v for any v we can determine if T is C-minimum and, if it is, the rightmost possible right extremity of the first internal node of the trees in the connected component of T in $\mathcal{R}(G, k)$.

[4] It is the case if and only if H_v is a clique.

▶ **Lemma 31 (*).** *Let G be an interval graph, $k \in \mathbb{N}$, and T be a spanning tree of G with at most k internal nodes. Assuming full access to T:*

- *We can decide in polynomial time if T is C-minimum in $\mathcal{R}(G,k)$ and,*
- *If T is C-minimum, we can compute in polynomial time the rightmost possible right extremity of the first internal node of a tree in the connected component of T in $\mathcal{R}(G,k)$.*

Sketch of the proof. Let v be the first internal node of T. Since we have full access to T, we can compute $w := \ell'_v(T)$. Lemma 30 ensures that there exists a spanning tree in the component of T in $\mathcal{R}(G,k)$ with second internal node w. We now determine how far we can move to the right the vertex v knowing this vertex. ◀

We say that we have *full access to T after v* if for every vertex $w \in V(G)$ with $w > v$, we have access in constant time to a table that permits us to know whether w is good or normal. And if w is normal, we also have access to $\ell'_w(T)$ and $r'_w(T)$. Using a proof similar to the one of Lemma 31, one can prove the following:

▶ **Lemma 32 (*).** *Let G be an interval graph, $k \in \mathbb{N}$, $v \in V(G)$ and T be a spanning tree of G with at most k internal nodes.*

- *We can decide in polynomial time if v is good if we have full access to T after v.*
- *If T is C-minimum, we can moreover compute $r'_v(T)$ and $\ell'_v(T)$ in polynomial time.*

Lemmas 32 ensures that we can, using backward induction on the ordering of the vertices, decide in polynomial time for all the vertices v of the graph if a vertex is good and if not we can compute $r'_v(T)$ and $\ell'_v(T)$. So we have full access to T in polynomial time.

▶ **Lemma 33 (*).** *Let G be an interval graph and v be a vertex of G. Let T_1, T_2 be two spanning trees of G with internal nodes I_1 and I_2 of H_v such that v is normal for both T_1 and T_2. Let $i_1 := r'_v(I_1)$ and $i_2 := r'_v(I_2)$. The trees T_1 and T_2 are in the same connected component of H_v if and only if:*

- *$i_1 = i_2$ and,*
- *Any spanning trees with internal nodes $(I_1 \setminus \{v\}) \cup \{i_1\}$ and $(I_2 \setminus \{v\}) \cup \{i_2\}$ are in the same connected component of $\mathcal{R}(H_{i_1}, k)$.*

We now have all the ingredients to prove Theorem 25.

Proof of Theorem 25. We can determine in polynomial time if the spanning trees are C-minimum by Lemma 31. If both of them are not, then both of them can be reconfigured to T_C and there exists a transformation from T_1 to T_2 by Lemma 31. If only one of them is, say T_1, we can replace T_1 by T_C (since they are in the same connected component in the reconfiguration graph). So we can assume that T_1 and T_2 are C-minimum. And the conclusion follows by Lemma 33. ◀

References

1 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 13:1–13:17, 2019. doi:10.4230/LIPIcs.STACS.2019.13.

2 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In *Graph-Theoretic Concepts in Computer Science - 43rd International Workshop, WG 2017, Eindhoven, The Netherlands, June 21-23, 2017, Revised Selected Papers*, pages 127–139, 2017. doi:10.1007/978-3-319-68705-6_10.

3 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

4 Nicolas Bousquet, Tatsuhiko Hatanaka, Takehiro Ito, and Moritz Mühlenthaler. Shortest reconfiguration of matchings. In *Graph-Theoretic Concepts in Computer Science - 45th International Workshop, WG 2019*, pages 162–174, 2019. doi:10.1007/978-3-030-30786-8_13.

5 Alan Frieze and Eric Vigoda. A survey on the use of markov chains to randomly sample colourings. *Oxford Lecture Series in Mathematics and its Applications*, 34:53, 2007.

6 M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

7 Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theor. Comput. Sci.*, 651(C):37–49, October 2016. doi:10.1016/j.tcs.2016.08.016.

8 Tesshu Hanaka, Takehiro Ito, Haruka Mizuta, Benjamin Moore, Naomi Nishimura, Vijay Subramanya, Akira Suzuki, and Krishna Vaidyanathan. Reconfiguring spanning and induced subgraphs. *Theor. Comput. Sci.*, 806:553–566, 2020. doi:10.1016/j.tcs.2019.09.018.

9 Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The coloring reconfiguration problem on specific graph classes. In *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part I*, pages 152–162, 2017. doi:10.1007/978-3-319-71150-8_15.

10 Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, October 2005. doi:10.1016/j.tcs.2005.05.008.

11 Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

12 Haruka Mizuta, Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. Reconfiguration of minimum steiner trees via vertex exchanges. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany.*, pages 79:1–79:11, 2019. doi:10.4230/LIPIcs.MFCS.2019.79.

13 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.

14 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

15 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *J. Comput. Syst. Sci.*, 93:1–10, 2018. doi:10.1016/j.jcss.2017.11.003.

When Lipschitz Walks Your Dog: Algorithm Engineering of the Discrete Fréchet Distance Under Translation

Karl Bringmann
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Insitute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
kbringma@mpi-inf.mpg.de

Marvin Künnemann
Max Planck Insitute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
marvin@mpi-inf.mpg.de

André Nusser
Saarbrücken Graduate School of Computer Science, Saarland University, Germany
Max Planck Insitute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
anusser@mpi-inf.mpg.de

Abstract

Consider the natural question of how to measure the similarity of curves in the plane by a quantity that is *invariant under translations* of the curves. Such a measure is justified whenever we aim to quantify the similarity of the curves' *shapes* rather than their positioning in the plane, e.g., to compare the similarity of handwritten characters. Perhaps the most natural such notion is the (discrete) *Fréchet distance under translation*. Unfortunately, the algorithmic literature on this problem yields a very pessimistic view: On polygonal curves with n vertices, the fastest algorithm runs in time $\mathcal{O}(n^{4.667})$ and cannot be improved below $n^{4-o(1)}$ unless the Strong Exponential Time Hypothesis fails. Can we still obtain an implementation that is efficient on realistic datasets?

Spurred by the surprising performance of recent implementations for the Fréchet distance, we perform algorithm engineering for the Fréchet distance under translation. Our solution combines fast, but inexact tools from continuous optimization (specifically, branch-and-bound algorithms for global Lipschitz optimization) with exact, but expensive algorithms from computational geometry (specifically, problem-specific algorithms based on an arrangement construction). We combine these two ingredients to obtain an *exact decision* algorithm for the Fréchet distance under translation. For the related task of computing the distance *value* up to a desired precision, we engineer and compare different methods. On a benchmark set involving handwritten characters and route trajectories, our implementation answers a typical query for either task in the range of a few milliseconds up to a second on standard desktop hardware.

We believe that our implementation will enable, for the first time, the use of the Fréchet distance under translation in applications, whereas previous algorithmic approaches would have been computationally infeasible. Furthermore, we hope that our combination of continuous optimization and computational geometry will inspire similar approaches for further algorithmic questions.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fréchet Distance, Computational Geometry, Continuous Optimization, Algorithm Engineering

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.25

Related Version `https://arxiv.org/abs/2008.07510`

Supplementary Material `https://gitlab.com/anusser/frechet_distance_under_translation`

Funding *Karl Bringmann*: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

Acknowledgements We thank Andreas Karrenbauer for helpful discussions.

1 Introduction

Consider the following natural task: Given two handwritings of (the same or different) characters, represented as polygonal curves π, σ in the plane, determine how similar they are. To measure the similarity of two such curves, several distance notions could be used, where the most popular measure in computational geometry is given by the *Fréchet distance* $d_F(\pi, \sigma)$: Intuitively, we imagine a dog walking on π and its owner walking on σ, and define $d_F(\pi, \sigma)$ as the *shortest leash length* required to connect the dog to its owner while both walk along their curves (only forward, but at arbitrarily and independently variable speeds). In this paper, we focus on the *discrete* version, in which dog and owner do not continuously walk along the curves, but jump from vertex to vertex.[1] As a fundamental curve similarity notion that takes into account the *sequence* of the points of the curves (rather than simply the set of points, as in the simpler notion of the Hausdorff distance), the discrete Fréchet distance and variants have received considerable attention from the computational geometry community, see, e.g. [4, 19, 12, 17, 3, 8, 11, 15]. While the fastest known algorithms take time $n^{2\pm o(1)}$ on polygonal curves with at most n vertices [4, 19, 3, 11] – which is best possible under the Strong Exponential Time Hypothesis [8] – a recent line of research [6, 13, 18, 10] gives fast implementations for practical input curves.

In the setting of handwritten characters, one would expect our notion of similarity to be *invariant under translations* of the curves; after all, translating one character in the plane while fixing the position of the other should not affect their similarity. In this sense, the original Fréchet distance seems inadequate, as it does not satisfy translation invariance. However, we may canonically define a translation-invariant adaptation as the minimum Fréchet distance between π and any translation of σ, yielding the *Fréchet distance under translation*. Note that beyond computing the similarity of handwritten characters, this measure is generally applicable whenever our intuitive notion of similarity is not affected by translations, such as recognition of movement patterns[2]. In some settings, we would expect our notion to additionally be scaling- or rotation-invariant; however, this is beyond the scope of this paper, as already the Fréchet distance under translation presents previously unresolved challenges.

Can we compute the Fréchet distance under translation quickly? The existing theoretical work yields a rather pessimistic outlook: For the discrete Fréchet distance under translation in the plane, the currently fastest algorithm runs in time $\mathcal{O}(n^{4.667})$, and any algorithm requires time $n^{4-o(1)}$ under the Strong Exponential Time Hypothesis [9]. These high polynomial bounds appear prohibitive in practice, and have likely impeded algorithmic uses of this similarity measure. (For the continuous analogue, the situation appears even worse, as the fastest algorithm has a significantly higher worst-case bound of $\mathcal{O}(n^8 \log n)$; we thus solely consider the discrete version in this work.) Given the surprising performance of recent Fréchet distance implementations on realistic curves [35, 10], can we still hope for faster algorithms on realistic inputs also for its translation-invariant version?

Our problem. Towards making the Fréchet distance under translation applicable for practical applications, we engineer a fast implementation and analyze it empirically on realistic input sets. Perhaps surprisingly, our fastest solution for the problem combines inexact

[1] We give a precise definition in Section 2.

[2] One may argue that the similarity of movement patterns also depends on the speed/velocity of the motion. In principle, we can also incorporate such information into any Fréchet-distance-based measure by introducing an additional dimension.

continuous optimization techniques with an exact, but expensive problem-specific approach from computational geometry to obtain an *exact decision* algorithm. We discuss our approach in Section 3 and present the details of our decision algorithm in Section 4. We develop our approach also for the related, but different task to compute the distance value up to a given precision in Section 5, and evaluate our solutions for both settings in comparison to baseline approaches in Section 6.

Further related work. Variations of the distance measure studied in this paper arise by choosing (1) the discrete or continuous Fréchet distance, (2) the dimension d of the ambient Euclidean space, and (3) a class of transformations, e.g., translations, rotations, scaling, or arbitrary linear transformations. A detailed treatment of algorithms for this class of distance measures can be found in [34]. The earliest example of a problem in this class is the continuous Fréchet distance under translations in dimension $d = 2$, which was introduced by Alt et al. [5] together with an $\mathcal{O}(n^8 \log n)$-time algorithm.

In this paper we focus on the discrete Fréchet distance under translation in the plane. This problem was first studied by Mosig and Clausen [31], who gave an $\mathcal{O}(n^4)$ algorithm for approximating the discrete Fréchet distance under rigid motions. Subsequently, Jiang et al. [29] presented an $\mathcal{O}(n^6 \log n)$-time algorithm for the exact Fréchet distance under translation. Their running time was improved by Ben Avraham et al. to $\mathcal{O}(n^5 \log n)$ [7], and then by Bringmann et al. to $\mathcal{O}(n^{4.667})$ [9]. A conditional lower bound of $n^{4-o(1)}$ can be found in [9].

Algorithm engineering efforts for the Fréchet distance were initiated by the SIGSPATIAL GIS Cup 2017 [35], where the task was to implement a nearest neighbor data structure for curves under the Fréchet distance; see [6, 13, 18] for the top three submissions. The currently fastest implementation of the Fréchet distance is due to Bringmann et al. [10]. Further recent directions of Fréchet-related algorithm engineering include k-means clustering of trajectories [14] and locality sensitive hashing of trajectories [16].

2 Preliminaries

Throughout the paper, we consider the Euclidean plane and denote the Euclidean norm by $\|\cdot\|$. A *polygonal curve* π is a sequence $\pi = (\pi_1, \dots, \pi_n)$ of vertices $\pi_i \in \mathbb{R}^2$. For any $\tau \in \mathbb{R}^2$, we write $\pi + \tau$ for the translated curve $(\pi_1 + \tau, \dots, \pi_n + \tau)$.

For any curves $\pi = (\pi_1, \dots, \pi_n), \sigma = (\sigma_1, \dots, \sigma_m)$, we define their *discrete Fréchet distance* as follows. A *traversal* is a sequence $T = ((p_1, s_1), \dots, (p_t, s_t))$ of pairs $(p_i, s_i) \in [n] \times [m]$ such that $(p_1, s_1) = (1, 1)$, $(p_t, s_t) = (n, m)$ and $(p_{i+1}, s_{i+1}) \in \{(p_i+1, s_i), (p_i, s_i+1), (p_i+1, s_i+1)\}$ for all $1 \leq i < t$. The width of a traversal is $\max_{i=1,\dots,|T|} \|\pi_{p_i} - \sigma_{s_i}\|$. The discrete Fréchet distance is then defined as the smallest width over all traversals, i.e.,

$$d_F(\pi, \sigma) := \min_{\text{traversal } T} \max_{i=1,\dots,|T|} \|\pi_{p_i} - \sigma_{s_i}\|.$$

As we only consider the discrete Fréchet distance in this paper, we drop "discrete" in the remainder. To avoid confusion, we also refer to it as the *fixed-translation* Fréchet distance.

As the canonically translation-invariant variant of the discrete Fréchet distance, we define the *discrete Fréchet distance under translation* as $d_{\text{trans-}F}(\pi, \sigma) := \min_{\tau \in \mathbb{R}^2} d_F(\pi, \sigma + \tau)$. We typically view the problem as a two-dimensional optimization problem with objective function $f(\tau) := d_F(\pi, \sigma + \tau)$. Specifically, we consider the task to decide $\min_{\tau \in \mathbb{R}^2} f(\tau) \leq \delta$? (*exact decider*) or to return a value in the range $[(1-\epsilon) \min_{\tau \in \mathbb{R}^2} f(\tau), (1+\epsilon) \min_{\tau \in \mathbb{R}^2} f(\tau)]$ (*approximate value computation*, multiplicative version). In fact, for implementation reasons

(for a discussion, see the full version of the paper), our implementation returns a value in $[\min_{\tau\in\mathbb{R}^2} f(\tau) - \epsilon, \min_{\tau\in\mathbb{R}^2} f(\tau) + \epsilon]$ (*approximate value computation*, additive version) using a straightforward adaptation of our approach.

Apart from a black-box Fréchet oracle answering decision queries $d_F(\pi, \sigma + \tau) \leq \delta$?, our algorithms only exploit the following simple properties:

▶ **Observation 1** (Lipschitz property). *The objective function f is 1-Lipschitz, i.e., $|f(\tau) - f(\tau + \tau')| \leq \|\tau'\|$.*

Proof. Note that for any $\pi_i, \sigma_j, \tau, \tau' \in \mathbb{R}^2$, we have

$$|\|\pi_i - (\sigma_j + \tau + \tau')\| - \|\pi_i - (\sigma_j + \tau)\|| \leq \|\tau'\|$$

by triangle inequality. Thus, the widths of any traversal T for $\pi, \sigma + \tau$ and $\pi, \sigma + \tau + \tau'$ differ by at most $\|\tau'\|$, which immediately yields the observation. ◀

We obtain a simple 2-approximation of the Fréchet distance under translation as follows.

▶ **Observation 2.** *Let $\tau_{\text{start}} := \pi_1 - \sigma_1$ be the translation of σ that aligns the first points of π and σ. Then $d_F(\pi, \sigma + \tau_{\text{start}}) \leq 2 \cdot d_{\textit{trans-}F}(\pi, \sigma)$.*

Analogously, for $\tau_{\text{end}} := \pi_n - \sigma_m$, we have $d_F(\pi, \sigma + \tau_{\text{end}}) \leq 2 \cdot d_{\textit{trans-}F}(\pi, \sigma)$.

Proof. Let $\delta^* := d_{\text{trans-}F}(\pi, \sigma)$ and let τ^* be such that $d_F(\pi, \sigma + \tau^*) = \delta^*$, which implies in particular that $\|\pi_1 - (\sigma_1 + \tau^*)\| \leq \delta^*$. Thus, $\|\tau_{\text{start}} - \tau^*\| = \|\pi_1 - (\sigma_1 + \tau^*)\| \leq \delta^*$. Thus by Observation 1, we obtain $d_F(\pi, \sigma + \tau_{\text{start}}) \leq d_F(\pi, \sigma + \tau^*) + \delta^* = 2\delta^*$. ◀

Note that the above observation gives a formal guarantee of a simple heuristic: translate the curves such that the start points match, and compute the corresponding fixed-translation Fréchet distance. Unfortunately, this worst-case guarantee is tight[3] – a correspondingly large discrepancy is also observed on our data sets.

3 Our Approach: Lipschitz meets Fréchet

To obtain a fast exact decider, we approach the problem from two different angles: First, we review previous problem-specific approaches to the Fréchet distance under translation, all relying on the construction of an arrangement of circles as an essential tool from computational geometry. Second, we cast the problem into the framework of global Lipschitz optimization with its rich literature on fast, numerical solutions. In isolation, both approaches are inadequate to obtain a fast, exact decider (as the arrangement can be prohibitively large even for realistic data sets, and black-box Lipschitz optimization methods cannot return an exact optimum). We then describe how to combine both approaches to obtain a fast implementation of an exact decider for the discrete Fréchet distance under translation in the plane. We evaluate our approach, including comparisons to (typically computationally infeasible) baseline approaches, on a data set that we craft from sets of handwritten character and (synthetic) GPS trajectories used in the ACM SIGSPATIAL GIS Cup 2017 [2, 1]. We believe that our approach will inspire similar combinations of fast, inexact methods from continuous optimization with expensive, but exact approaches from computational geometry also in other contexts.

[3] To see this, take any segment in the plane and let π traverse it in one direction, and σ in the other. Then the heuristic would return as estimate two times the segment length (the distance of the translated end points), while the optimal translation aligns the segments and achieves the segment length as Fréchet distance.

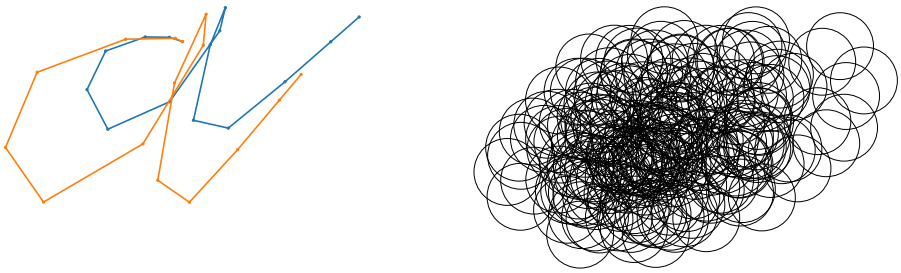


Figure 1 Example curves π, σ (left) together with their arrangement $\mathcal{A}_\delta$ (right), $\delta = d_{\text{trans-}F}(\pi, \sigma)$.

3.1 View I: Arrangement-based Algorithms

Previous algorithms for the Fréchet distance under translation in the plane work as follows. Given two polygonal curves π, σ and a decision distance δ, consider the set of circles

$$\mathcal{C} \coloneqq \{C_\delta(\pi_i - \sigma_j) \mid \pi_i \in \pi,\ \sigma_j \in \sigma\},$$

where $C_r(p)$ denotes the circle of radius $r \in \mathbb{R}$ around $p \in \mathbb{R}^2$. Define the arrangement $\mathcal{A}_\delta$ as the partition of $\mathbb{R}^2$ induced by $\mathcal{C}$. The decision of $d_F(\pi, \sigma + \tau) \leq \delta$ is then uniform among all $\tau \in \mathbb{R}$ in the same face of $\mathcal{A}_\delta$ (for a detailed explanation, we refer to [7, Section 3] or [9]). Thus, it suffices to check, for each face f of $\mathcal{A}_\delta$, an arbitrarily chosen translation $\tau_f \in f$. Specifically, the Fréchet distance under translation is bounded by δ if and only if there is some face f of $\mathcal{A}_\delta$ such that $d_F(\pi, \sigma + \tau_f) \leq \delta$. Since the arrangement $\mathcal{A}_\delta$ has size $\mathcal{O}(n^4)$ and can be constructed in time $\mathcal{O}(n^4)$ [29], using the standard $\mathcal{O}(n^2)$-time algorithm for the fixed-translation Fréchet distance [19, 4] to decide $d_F(\pi, \sigma + \tau_f) \leq \delta$ for each face f, we immediately arrive at an $\mathcal{O}(n^6)$-time algorithm.

Subsequent improvements [7, 9] speed up the decision of $d_F(\pi, \sigma + \tau_f) \leq \delta$ for all faces f by choosing an appropriate ordering of the translations τ_f and designing data structures that avoid recomputing some information for "similar" translations, leading to an $\mathcal{O}(n^{4.667})$-time algorithm. Still, these works rely on computing the arrangement $\mathcal{A}_\delta$ of worst-case size $\Theta(n^4)$, and a conditional lower bound indeed rules out $\mathcal{O}(n^{4-\epsilon})$-time algorithms [9].

Drawback: The arrangement size bottleneck. Despite the worst-case arrangement size of $\Theta(n^4)$ and the conditional lower bound in [9], which indeed constructs such large arrangements, one might hope that realistic instances often have much smaller arrangements. If so, a combination with a practical implementation of the fixed-translation Fréchet distance could already give an algorithm with reasonable running time. Unfortunately, this is not the case: our experiments in this paper exhibit typical arrangement sizes between 10^6 to 10^8 for curves of length $n \approx 200$, see Figure 5 in Section 6. Also see Figure 1 which illustrates a large arrangement already on curves with 15 vertices, subsampled from our benchmark sets of realistic curves.

This renders a purely arrangement-based approach infeasible: As existing implementations for the Fréchet distance typically answer queries within few microseconds, we would expect an average decision time between a few seconds and several minutes already for a single decision query for the Fréchet distance under translation. Thus, a reasonable approximation of the distance value via binary search would take between a minute and over an hour.

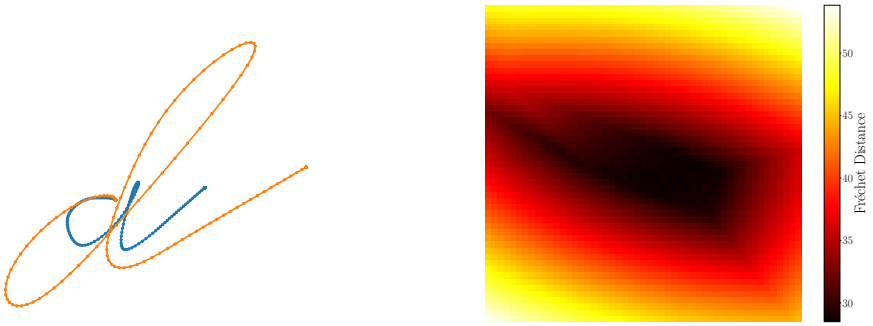


Figure 2 Example curves π, σ (left) together with a plot of the resulting non-convex objective function $f(\tau) = d_F(\pi, \sigma + \tau)$. For a closer look at the area close to the optimal translation (and highly non-convex small-scale artefacts), we refer to Figure 3.

3.2 View II: A Global Lipschitz Optimization problem

A second view on the Fréchet distance under translation results from a simple observation: For any polygonal curves π, σ and any translation $\tau \in \mathbb{R}^2$, we have $|d_F(\pi, \sigma+\tau) - d_F(\pi, \sigma)| \leq \|\tau\|_2$, see Section 2. As a consequence, the Fréchet distance under translation is the minimum of a function $f(\tau) := d_F(\pi, \sigma + \tau)$ that is 1-Lipschitz (i.e., $|f(x) - f(x + y)| \leq \|y\|_2$ for all x, y). This suggests to study the problem also from the viewpoint of the generic algorithms developed for optimizing Lipschitz functions by the continuous optimization community.

Following the terminology of [25], in an *unconstrained bivariate global Lipschitz optimization problem*, we are given an objective function $f : \mathbb{R}^2 \to \mathbb{R}$ that is 1-Lipschitz, and the aim is to minimize $f(x)$ over $x \in B := [a_1, b_1] \times [a_2, b_2]$; we can access f only by evaluating it on (as few as possible) points $x \in B$. Note that in this abstract setting, we cannot optimize f exactly, so we are additionally given an error parameter $\epsilon > 0$ and the precise task is to find a point $x \in B$ such that $f(x) \leq \min_{z \in B} f(z) + \epsilon$.

Global Lipschitz optimization techniques have been studied from an algorithmic perspective for at least half a century [32]. This suggest to explore the use of the fast algorithms developed in this context to obtain at least an *approximate* decider for the discrete Fréchet distance under translation. Indeed, our problem fits into the above framework, if we take the following considerations into account:

(1) Finite Box Domain: While we seek to minimize $f(\tau) = d_F(\pi, \sigma + \tau)$ over $\tau \in \mathbb{R}^2$, the above formulation assumes a finite box domain B. To reconcile this difference, observe that any translation τ achieving a Fréchet distance of at most δ must translate the first (last) point of σ such that the first (last) point of π is within distance at most δ. Thus, any feasible translation τ must be contained in the intersection of the two corresponding disks, and we can use any bounding box of this intersection as our box domain B.

(2) (Approximate) Decision Problem: While we seek to decide "$\min_\tau f(\tau) \leq \delta$", the above formulation solves the corresponding minimization problem. Note that approximate minimization can be used to *approximately* solve the decision problem, but *exactly* solving the decision problem is impossible in the above framework.

(3) Oracle Access to $f(\tau)$: Evaluation of $f(\tau)$ corresponds to computing the Fréchet distance of π and $\sigma + \tau$, for which we can use previous fast implementations [6, 13, 18, 10]. (Actually, these algorithms were designed to answer decision queries of the form "$f(\tau) \leq \delta$?"; we discuss this aspect at the end of this section.)

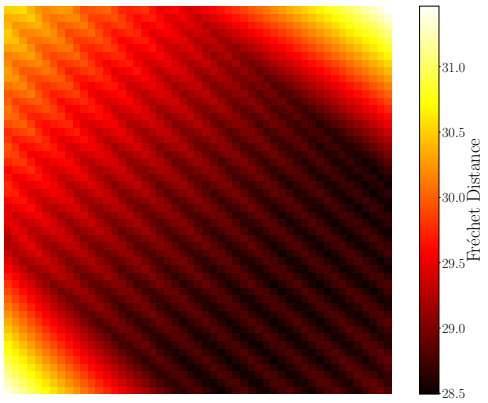


Figure 3 Highly non-convex artefacts of the objective function at a local scale, resulting particularly from the notion of traversals in the *discrete* Fréchet distance.

In Figure 2, we illustrate our view of the Fréchet distance under translation as Lipschitz optimization problem. As the figure suggests, on many realistic instances, the problem appears well-behaved (almost convex) at a global scale; using the Lipschitz property, one should be able to quickly narrow down the search space to small regions of the search space[4]. Particularly for this task, it is very natural to consider branch-and-bound approaches, as pioneered by Galperin [20, 21, 22, 23] and formalized by Horst and Tuy [26, 27, 28], since these have been applied very successfully for low-dimensional Global Lipschitz optimization (and non-convex optimization in general).

On a high level, in this approach we maintain a global upper bound $\tilde{\delta}$ and a list of search boxes $B_1, \ldots, B_b$ with lower bounds $\ell_1, \ldots, \ell_b$ (i.e., $\min_{\tau \in B_i} f(\tau) \geq \ell_i$) obtained via the Lipschitz condition. We iteratively pick some search box B_i and first try to improve the global upper bound $\tilde{\delta}$ or the local lower bound ℓ_i using a small number of queries $f(\tau)$ with $\tau \in B_i$ (and exploiting the Lipschitz property). If the local lower bound exceeds the global upper bound, i.e., $\ell_i > \tilde{\delta}$, we drop the search box B_i, otherwise, we split B_i into smaller search boxes. The procedure stops as soon as $\tilde{\delta} \leq (1+\epsilon) \min_i \ell_i$, which proves that $\tilde{\delta}$ gives a $(1+\epsilon)$-approximation to the global minimum.

Specifically, we arrive at the following branch-and-bound strategy proposed by Gourdin, Hansen and Jaumard [24]. We specify it by giving the rules with which it (i) attempts to update the global upper bound, (ii) selects the next search box from the set of current search boxes, (iii) splits a search box if it remains active after bounding, and (iv) determines the local lower bounds.[5]

- **(i) Upper Bounding Rule:** We evaluate f at the center τ_i of the current search box B_i.
- **(ii) Selection Rule:** We pick the search box with the smallest lower bound (ties are broken arbitrarily).
- **(iii) Branching Rule:** We split the current search box along its longest edge into 2 equal-sized subproblems.
- **(iv) Lower Bounding Rule:** We obtain the local lower bound ℓ_i as $f(\tau_i) - d$ where d is the half-diameter of the current box. (Since f is 1-Lipschitz, we indeed have $\min_{\tau \in B_i} f(\tau) \geq \ell_i$.)

[4] For an illustration that highly non-convex behavior may still occur at a local level, we refer to Figure 3.
[5] See [25] for a precise formalization of the generic branch-and-bound algorithm that leaves open the instantiation of these rules. In any case, we give a self-contained description of our algorithms in Section 4 and 5.

One may observe that the chosen selection rule (also known as Best-Node First) is a no-regret strategy in the sense that no other selection rule, *even with prior knowledge of the global optimum*, considers fewer search boxes (see, e.g., [36, Section 7.4]).

Drawback: Inexactness. Unfortunately, the above branch-and-bound approach for Lipschitz optimization fundamentally cannot return an exact global optimum, and thus yields only an approximate decider.

In a somewhat similar vein, in the above framework we assume that we can evaluate $f(\tau)$ quickly. Previous implementations for the fixed-translation Fréchet distance focus on the decision problem "$f(\tau) \leq \delta$?", not on determining the value $f(\tau)$. Both precise computations (via parametric search) or approximate computations (using a binary search up to a desired precision) are significantly more costly, raising the question how to make optimal use of the cheaper decision queries.

4 Contribution I: An Exact Decider by Combining Both Views

Our first main contribution is engineering an exact decider for the discrete Fréchet distance under translation by combining the two approaches. On a high level, we *globally* perform the branch-and-bound strategy described in the Lipschitz optimization view in Section 3.2, but use as a base case a *local* version of the arrangement-based algorithms of Section 3.1 once the arrangement size in a search box is sufficiently small. As each search box is thus resolved exactly, this yields an exact decider. More precisely, our final algorithm is a result of the following steps and adaptations:

(1) Fréchet Decision Oracle. We adapt the currently fastest implementation of a decider for the continuous fixed-translation Fréchet distance [10] to the discrete fixed-translation Fréchet distance. Furthermore, to handle many queries for the same curve pair *under different translations* quickly, we incorporate an implicit translation so that curves do not need to be explicitly translated for each query translation τ.

(2) Objective Function Evaluation. For our exact decider, the branch-and-bound strategy in Section 3.2 simplifies significantly: We do not maintain a global upper bound and local lower bounds ℓ_i, but for each box only test whether $f(\tau_i) \leq \delta$ (if so, we return YES) or whether $f(\tau_i) > \delta + d$ (this corresponds to updating the local lower bound beyond δ, i.e., we may drop the box completely). Therefore, we may use an arbitrary selection rule. Note that we only require decision queries to the fixed-translation Fréchet algorithm.

(3) Base Case. We implement a *local* arrangement-based algorithm: For a given search box B_i, we (essentially) construct the arrangement $\mathcal{A} \cap B_i$ using CGAL [33], and test, for each face f of $\mathcal{A} \cap B_i$, some translation $\tau' \in f$ for $f(\tau') \leq \delta$. This yields the algorithm that we may use as a base case.

(4) Base Case Criterion. For each search box, we compute an estimate of its arrangement complexity. If this estimate is smaller than a (tunable) parameter γ_{size}, or the depth of the branch-and-bound recursion for the current search box exceeds a parameter γ_{depth}, then we use the localized arrangement-based algorithm.

(5) Benchmark and Choice of Parameters. We choose the size and depth parameters $\gamma_{\text{size}}, \gamma_{\text{depth}}$ guided by a benchmark set that we create from a set of handwritten characters and synthetic GPS trajectories.

The pseudocode of the resulting algorithm is shown in Algorithm 1. For a more detailed description of our Fréchet-under-translation decider, we refer to the full version of this paper.

Algorithm 1 Algorithm for deciding the Fréchet distance under translation. We use τ_B to denote the center of the box B and d_B to denote the length of the diagonal of B.

```
1:  procedure DECIDER(π, σ, δ)
2:      decide trivial NO instances with empty initial search box quickly
3:      Q ← FIFO(initial search box)
4:      while Q ≠ ∅ do
5:          B ← extract front of search box queue Q
6:          if FRÉCHETDISTANCE(π, σ + τ_B) > δ + d_B/2 then        ▷ Lower Bounding
7:              skip B
8:          if FRÉCHETDISTANCE(π, σ + τ_B) ≤ δ then               ▷ Upper Bounding
9:              return YES
10:
11:         u ← upper bound on arrangement size inside B
12:         if u = 0 then                                         ▷ Arrangement-based Base Case
13:             skip B
14:         else if u ≤ γ_size or layer of B is γ_depth then
15:             if local arrangement-based algorithm on π, σ, δ, B returns YES then
16:                 return YES
17:             else
18:                 skip B
19:
20:         halve B along longest edge and push resulting child boxes to Q   ▷ Branching
21:     return NO
```

5 Contribution II: Computation of the Distance Value

In this section we present our second main contribution: an algorithm for computing the value of the Fréchet distance under translation. Thus, we now focus on the functional task of computing the value $d_{\text{trans-}F}(\pi,\sigma) = \min_{\tau\in\mathbb{R}^2} d_F(\pi,\sigma+\tau)$, in contrast to the previously discussed decision problem "$d_{\text{trans-}F}(\pi,\sigma) \leq \delta$?". In theory, one could use the paradigm of parametric search [30], see [7, 9] for details for the discrete case. However, it is rarely used in practice as it is non-trivial to code, and computationally costly. Instead, as in most conceivable settings an estimate with small multiplicative error $(1\pm\epsilon)$ with, e.g., $\epsilon = 10^{-7}$, suffices, we consider the problem of computing an estimate in $(1\pm\epsilon)d_{\text{trans-}F}(\pi,\sigma)$.

There are several possible approaches to obtain an approximation with multiplicative error $(1\pm\epsilon)$ for arbitrarily small $\epsilon > 0$:

1. **ϵ-approximate Set:** A natural approach underlying previous approximation algorithms [5] is to generate a set of $f(1/\epsilon)$ candidate translations T such that the best translation $\tau \in T$ gives a $(1+\epsilon)$-approximation for the Fréchet distance under translation. Unfortunately, in the plane such a set is of size $\Theta(1/\epsilon^2)$, which is prohibitively large for approximation guarantees such as $\epsilon = 10^{-7}$.
2. **Binary Search via Decision Problem:** A further canonical approach is to reduce the $(1+\epsilon)$-approximate computation task to the decision problem using a binary search.
3. **Lipschitz-only Optimization:** The main drawback of the generic Lipschitz optimization algorithms discussed in Section 3.2 was that they cannot be used to derive an exact answer. This drawback no longer applies for approximate value computation. We can thus use a pure branch-and-bound algorithm for global Lipschitz optimization. Interestingly,

Algorithm 2 Algorithm of our Lipschitz-Meets-Fréchet (LMF) algorithm for approximate value computation. We use τ_B to denote the center of the box and d_B to denote the length of the diagonal.

procedure LMF(π, σ)
  Preprocessing: build data structures for fast arrangement estimation and construction
  compute initial distance interval $[\delta_{\text{LB}}, \delta_{\text{UB}}]$ containing $d_{\text{trans-}F}(\pi, \sigma)$
  initialize global upper bound $\tilde{\delta} \leftarrow \delta_{\text{UB}}$
  $Q \leftarrow$ PriorityQueue(initial search box B_1 with local lower bound $\ell_{B_1} \leftarrow \delta_{\text{LB}}$)
  while $Q \neq \emptyset$ **do**
    $B \leftarrow$ box with smallest local lower bound ℓ_B in Q
    if $\tilde{\delta} \leq \ell_B(1+\epsilon)$ **then**
      skip B
    if FréchetDistance($\pi, \sigma + \tau_B$) $\leq \tilde{\delta}$ **then** ▷ *Upper/Lower Bounding*
      compute value $d_F(\pi, \sigma + \tau_B)$ with high precision and update $\tilde{\delta}$ and ℓ_B
    else
      if FréchetDistance($\pi, \sigma + \tau_B$) $> \tilde{\delta} + d_B/2$ **then**
        skip B
      compute value $d_F(\pi, \sigma + \tau_B)$ with coarse precision and update ℓ_B
    if $\tilde{\delta} \leq \ell_B(1+\epsilon)$ **then**
      skip B
    $u \leftarrow$ upper bound on arrangement size inside B for $\delta \in [\ell_B, \tilde{\delta}]$
    if $u = 0$ **then** ▷ *Arrangement-based Base Case*
      skip B
    else if $u \leq \gamma_{\text{size}}$ or layer of B is γ_{depth} **then**
      update $\tilde{\delta}$ via binary search over arrangement algorithm on B and $\delta \in [\ell_B, \tilde{\delta}]$
      skip B

    push child boxes of B to Q with local lower bounds set to ℓ_B ▷ *Branching*
  return $\tilde{\delta}$

experiments reveal that on many natural instances, significant time is spent branching inside very small regions. This suggests following our combined approach also for the value computation setting.

4. **Our solution, Lipschitz-meets-Fréchet:** We follow our approach of combining Lipschitz optimization with arrangement-based algorithms (described in Section 3) to compute a $(1+\epsilon)$-approximation of the distance value. As opposed to the decision algorithm, we indeed maintain a global upper bound $\tilde{\delta}$ and local lower bounds ℓ_i for each search box B_i. To update these bounds, we approximately evaluate the objective function $f(\tau)$ using a tuned binary search[6] over the fixed-translation Fréchet decider algorithm. We stop branching in a search box B_i if either the global upper bound $\tilde{\delta}$ is at most $\ell_i(1+\epsilon)$, or a base case criterion similar to the decision setting applies. As selection strategy, we employ the no-regret strategy of choosing the box with the smallest lower bound first. The base case performs a binary search using the local arrangement-based *decision* algorithm; thus, our upper bound on the arrangement size must hold for *all* δ in the search interval. The pseudocode of our solution is shown in Algorithm 2.

[6] We tune the binary search by distinguishing the precision with which we want to evaluate $f(\tau)$; intuitively, it pays off to evaluate $f(\tau)$ with high precision if this evaluation yields a better global upper bound, while for improvements of a local lower bound, a cheaper evaluation with coarser precision suffices.

For a more detailed description of the algorithm, we refer to the full version of this paper. As our experiments reveal, our solution generally outperforms the above described alternatives, see Section 6.

6 Experiments

To engineer and evaluate our approach, we provide a benchmark on the basis of the curve datasets that were used to evaluate the currently fastest fixed-translation Fréchet decider implementation in [10]. In particular, this curve set involves a set of handwritten characters (Characters, [2]) and the data set of the GIS Cup 2017 (Sigspatial, [1]). Table 1 gives statistics of these datasets.

Table 1 Information about the data sets used in the benchmarks.

Data set	Type	#Curves	Mean #vertices
Sigspatial [1]	synthetic GPS-like	20199	247.8
Characters [2]	20 handwritten characters	2858 (142.9 per character)	120.9

The aim of our evaluations is to investigate the following main questions:

1. Is our solution able to decide queries on realistic curve sets in an amount of time that is practically feasible, even when the size of the arrangement suggests infeasibility?
2. Is our combination of Lipschitz optimization and arrangement-based algorithms for value computation superior to the alternative approaches described in Section 5?

Furthermore, we aim to provide an understanding of the performance of our novel algorithms.

Decider experiments. For decision queries of the form "$d_{\text{trans-}F}(\pi,\sigma) \leq \delta$?", we generate a benchmark query set that distinguishes between how close the test distance is to the actual distance of the curves: Given a set of curves C, we sample 1000 curve pairs $\pi, \sigma \in C$ uniformly at random. Using our implementation, we determine an interval $[\delta_{\text{LB}}, \delta_{\text{UB}}]$ such that $\delta_{\text{UB}} - \delta_{\text{LB}} \leq 2 \cdot 10^{-7}$ and $d_{\text{trans-}F}(\pi,\sigma) \in [\delta_{\text{LB}}, \delta_{\text{UB}}]$. For each $\ell \in \{-10, \ldots, -1\}$, we add "$d_{\text{trans-}F}(\pi,\sigma) \leq (1-4^\ell)\delta_{\text{LB}}$?" to the query set C_ℓ^{NO}, which contains only NO instances. Similarly, for each $\ell \in \{-10, \ldots, 2\}$ we add "$d_{\text{trans-}F}(\pi,\sigma) \leq (1+4^\ell)\delta_{\text{UB}}$?" to the query set C_ℓ^{YES}, which contains only YES instances. We depict our timing results in Figure 4 on the Characters and Sigspatial data sets. Further experiments are given in the full version of the paper.

To give an insight for the speed-up achieved over the baseline arrangement-based algorithm that makes a black-box call to the fixed-translation Fréchet decider for each face of the arrangement $\mathcal{A}_\delta$, in Figure 5 we depict both the number of black-box calls to the fixed-translation Fréchet decider made by our implementation, as well as an estimate[7] for the arrangement size, and thus the number of black-box calls of the baseline approach.

[7] We only give an estimate for the arrangement size, since the size of the arrangement is too large to be evaluated exactly for all our benchmark queries within a day. Specifically, we estimate the number of vertices of the arrangement which closely corresponds to the number of faces by Euler's formula. We give the following estimate: We first determine a search box B for the given decision instance $\pi = (\pi_1, \ldots, \pi_n), \sigma = (\sigma_1, \ldots, \sigma_m), \delta$ as described for our algorithm. We then sample $S = 100000$ tuples $i_1, i_2 \in \{1, \ldots, n\}, j_1, j_2 \in \{1, \ldots, m\}$ and count the number I of intersections of the circles of radius δ around $\pi_{i_1} - \sigma_{j_1}$ and $\pi_{i_2} - \sigma_{j_2}$ inside B. The number $(I/S) \cdot (nm)^2$ is the estimated number of circle-circle intersections in B. Adding the number of circle-box intersections, which we can compute exactly, yields our estimate.

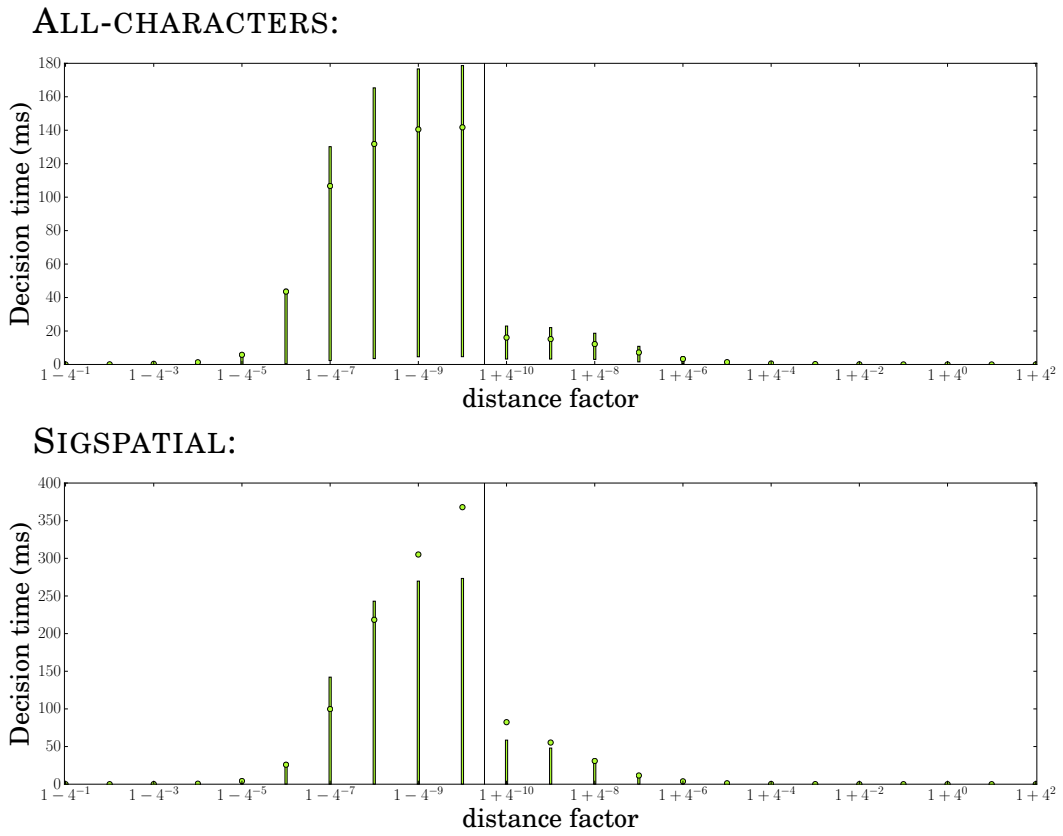


Figure 4 Running time for our decider. We plot the mean running times over 1000 NO (or YES) queries with a test distance of approximately $(1-4^{-\ell})$ (or $(1+4^{-\ell})$) times the true Fréchet distance under translation, as well as the interval between the lower and upper quartile over the queries.

We observe that on the above sets, the average decision time ranges from below 1 ms to 142 ms, deciding our CHARACTERS benchmark (involving 23,000 queries) in 628 seconds. Our estimation suggests that a naive implementation of the baseline arrangement-based algorithm would have been worse by more than *three orders of magnitude*, as for each set, the average number of black-box calls to the fixed-translation Fréchet decider is smaller by a factor of at least 1000 than our estimation of the arrangement size.

Approximate value computation experiments. We evaluate our implementation of the algorithm presented in Section 5 by computing an estimate $\tilde{\delta}$ such that $|\tilde{\delta} - d_{\text{trans-}F}(\pi,\sigma)| \leq \epsilon$ with a choice of $\epsilon = 10^{-7}$.[8] In particular, we compare the performances of the different approaches discussed in Section 5:

- **Binary Search:** Binary search using our Fréchet-under-translation decider of Section 4.
- **Lipschitz-only:** Algorithm 2 without the arrangement, i.e., without lines 18 to 23.
- **Lipschitz-meets-Fréchet (LMF):** Our implementation as detailed in Section 5.

Since simple estimates show that the ϵ-approximate sets are clearly too costly for $\epsilon = 10^{-7}$, we drop this approach from all further consideration. We took care to implement all approaches with a similar effort of low-level optimizations.

[8] Here we use additive rather than multiplicative approximation for technical reasons. Since all computed distances are within $[1.6, 120.7]$, this also yields a multiplicative $(1+\epsilon)$-approximation with $\epsilon \leq 10^{-7}$.

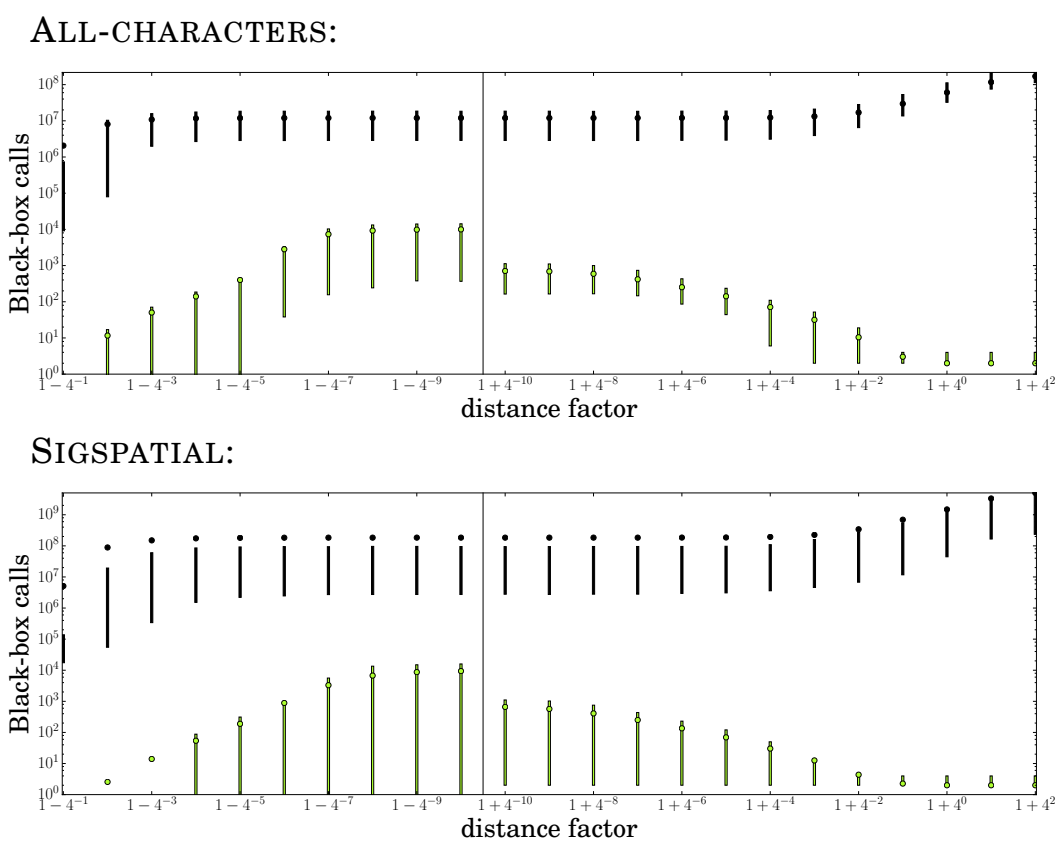


Figure 5 Number of black-box calls to the fixed-translation Fréchet decider made by our decider (below, in green), as well as an estimate of the arrangement complexity, i.e., number of calls of a naive algorithm (above, in black). We plot the mean number of calls and arrangement complexity over 1000 NO (or YES) queries with a test distance of approximately $(1 - 4^{-\ell})$ (or $(1 + 4^{-\ell})$) times the true Fréchet distance under translation, as well as the interval between the lower and upper quartile over the queries.

For our evaluation, we focus on the CHARACTERS data set which allows us to distinguish the rough shape of the curves: We subdivide the curve set into the subsets C_α for $\alpha \in \Sigma$ (where Σ is the set of 20 characters occurring in CHARACTERS). In particular for each character pair $\alpha, \beta \in \Sigma$, we create a sample of N_{samples} curve pairs (π, σ) chosen uniformly at random from $C_\alpha \times C_\beta$. For $N_{\text{samples}} = 5$, computing the value (up to $\epsilon = 10^{-7}$) for all $N_{\text{samples}} \cdot (\binom{|\Sigma|}{2} + |\Sigma|) = 1050$ sampled curve pairs gives the statistics shown in Table 2.

Since already for this example the Lipschitz-only approach is dominated by almost a factor of 30 by LMF (and by a factor of almost 8 by binary search), we perform more detailed analyses with $N_{\text{samples}} = 100$ only for LMF and binary search. The overall performance is given in Table 3. Also here LMF is more than 3 times faster than binary search. To give more insights into the relationship of their running times, we give a scatter plot of the running times of LMF and binary search on the same instances over the complete benchmark in Figure 6, showing that binary search generally outperforms LMF only on instances which are comparably easy for LMF as well. The advantage of LMF becomes particularly clear on hard instances.

We give further experiments in the full version of the paper.

Table 2 Statistics for approximate value computation for $N_{\text{samples}} = 5$. In parentheses we show the mean values averaged over a total of 1050 instances.

Approach	Time	Black-Box Calls
LMF	148,032 ms (141.0 ms per instance)	13,323,232 (12,688.8 per instance)
Binary Search	536,853 ms (511.3 ms per instance)	45,909,628 (43,723.5 per instance)
Lipschitz-only	4,204,521 ms (4,004.3 ms per instance)	820,468,224 (781,398.3 per instance)

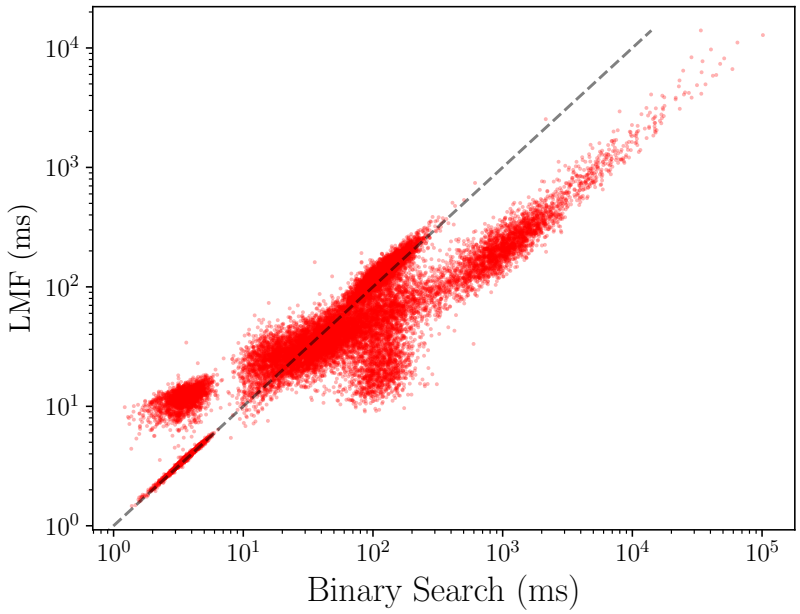


Figure 6 Running times of LMF and binary search on set of randomly sampled CHARACTERS curves.

Table 3 Statistics for approximate value computation for $N_{\text{samples}} = 100$. In parentheses, we give average values over the total of 21,000 curve pairs.

Algorithm	Time		Black-Box Calls
LMF	2,938,512 ms (140.0 ms per instance)		260,128,449 (12,387.1 per instance)
	- Preprocessing	71,728 ms	
	- Black-box calls (Lipschitz)	400,189 ms	
	- Arrangement estimation	166,479 ms	
	- Arrangement algorithm	2,250,493 ms	
	* Construction	1,537,500 ms	
	* Black-box calls	545,442 ms	
Binary Search	10,555,630 ms (502.7 ms per instance)		875,424,988 (41,686.9 per instance)

7 Conclusion

We engineer the first practical implementation for the discrete Fréchet distance under translation in the plane. While previous algorithmic solution for the problem solve it via expensive discrete methods, we introduce a new method from continuous optimization to achieve significant speed-ups on realistic inputs. This is analogous to the success of integer programming solvers which, while optimizing a discrete problem, choose to work over the reals to gain access to linear programming relaxations, cutting planes methods, and more. A novelty here is that we successfully apply such methods to obtain drastic speed-ups for a *polynomial-time problem*.

We leave as open problems to determine whether there are reasonable analogues of further ideas from integer programming, such as cutting plane methods or preconditioning, that could help to get further improved algorithms for our problem. More generally, we believe that this gives an exciting direction for algorithm engineering in general that should find wider applications. A particular direction in this vein is the use of our methods to compute rotation- or scaling-invariant versions of the Fréchet distance. Intuitively, by introducing additional dimensions in our search space, our methods can in principle also be used to optimize over such additional degrees of freedom. However, the Lipschitz condition changes significantly, and it remains subject of future work to determine the applicability of our methods in these settings.

References

1 ACM SIGSPATIAL GIS Cup 2017 Data Set. `https://www.martinwerner.de/datasets/san-francisco-shortest-path.html`. Accessed: 2018-12-03.

2 Character Trajectories Data Set. `https://archive.ics.uci.edu/ml/datasets/Character+Trajectories`. Accessed: 2018-12-03.

3 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM J. Comput.*, 43(2):429–449, 2014. `doi:10.1137/130920526`.

4 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5(1–2):78–99, 1995.

5 Helmut Alt, Christian Knauer, and Carola Wenk. Matching polygonal curves with respect to the Fréchet distance. In *Proc. 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS'01)*, pages 63–74, 2001.

6 Julian Baldus and Karl Bringmann. A fast implementation of near neighbors queries for Fréchet distance (GIS Cup). In *Proc. of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2017)*, pages 99:1–99:4. ACM, 2017. `doi:10.1145/3139958.3140062`.

7 Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. A faster algorithm for the discrete Fréchet distance under translation. *ArXiv preprint*, 2015. `arXiv:1501.03724`.

8 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th Ann. IEEE Symposium on Foundations of Computer Science (FOCS'14)*, pages 661–670, 2014.

9 Karl Bringmann, Marvin Künnemann, and André Nusser. Fréchet distance under translation: Conditional hardness and an algorithm via offline dynamic grid reachability. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 2902–2921, 2019. `doi:10.1137/1.9781611975482.180`.

10 Karl Bringmann, Marvin Künnemann, and André Nusser. Walking the dog fast in practice: Algorithm engineering of the Fréchet distance. In *Proc. 35th International Symposium on Computational Geometry (SoCG 2019)*, pages 17:1–17:21, 2019. `doi:10.4230/LIPIcs.SoCG.2019.17`.

11 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017. doi:10.1007/s00454-017-9878-7.

12 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proc. 20th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA'09)*, pages 645–654, 2009.

13 Kevin Buchin, Yago Diez, Tom van Diggelen, and Wouter Meulemans. Efficient trajectory queries under the Fréchet distance (GIS Cup). In *Proc. of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2017)*, pages 101:1–101:4. ACM, 2017. doi:10.1145/3139958.3140064.

14 Kevin Buchin, Anne Driemel, Natasja van de L'Isle, and André Nusser. klcluster: Center-based clustering of trajectories. In *Proc. of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2019)*, pages 496–499. ACM, 2019. doi:10.1145/3347146.3359111.

15 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2887–2901. SIAM, 2019. doi:10.1137/1.9781611975482.179.

16 Matteo Ceccarello, Anne Driemel, and Francesco Silvestri. FRESH: Fréchet similarity with hashing. In *Proc. of the 16th International Symposium on Algorithms and Data Structures (WADS 2019)*, volume 11646 of *LNCS*, pages 254–268. Springer, 2019. doi:10.1007/978-3-030-24766-9_19.

17 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, July 2012. doi:10.1007/s00454-012-9402-z.

18 Fabian Dütsch and Jan Vahrenhold. A filter-and-refinement-algorithm for range queries based on the Fréchet distance (GIS Cup). In *Proc. of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2017)*, pages 100:1–100:4. ACM, 2017. doi:10.1145/3139958.3140063.

19 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

20 Efim A. Galperin. The cubic algorithm. *Journal of Mathematical Analysis and Applications*, 112(2):635–640, 1985. doi:10.1016/0022-247X(85)90268-9.

21 Efim A. Galperin. Two alternatives for the cubic algorithm. *Journal of Mathematical Analysis and Applications*, 126(1):229–237, 1987. doi:10.1016/0022-247X(87)90088-6.

22 Efim A. Galperin. Precision, complexity, and computational schemes of the cubic algorithm. *Journal of Optimization Theory and Applications*, 57(2):223–238, 1988. doi:10.1007/BF00938537.

23 Efim A. Galperin. The fast cubic algorithm. *Computers & Mathematics with Applications*, 25(10):147–160, 1993. doi:10.1016/0898-1221(93)90289-8.

24 E. Gourdin, P. Hansen, and B. Jaumard. Global optimization of multivariate lipschitz functions: Survey and computational comparison, 1994.

25 Pierre Hansen and Brigitte Jaumard. Lipschitz optimization. In Reiner Horst and Panos M. Pardalos, editors, *Handbook of Global Optimization*, pages 407–493. Springer US, Boston, MA, 1995. doi:10.1007/978-1-4615-2025-2_9.

26 Reiner Horst. A general class of branch-and-bound methods in global optimization with some new approaches for concave minimization. *Journal of Optimization Theory and Applications*, 51:271–291, 1986. doi:10.1007/BF00939825.

27 Reiner Horst and Hoang Tuy. On the convergence of global methods in multiextremal optimization. *Journal of Optimization Theory and Applications*, 54:253–271, 1987. doi:10.1007/BF00939434.

28 Reiner Horst and Hoang Tuy. *Global Optimization – Deterministic Approaches*. Springer Berlin Heidelberg, 3rd edition, 1996.

29 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure–structure alignment with discrete Fréchet distance. *J. Bioinformatics and Computational Biology*, 6(01):51–64, 2008.

30 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983. doi:10.1145/2157.322410.

31 Axel Mosig and Michael Clausen. Approximately matching polygonal curves with respect to the fréchet distance. *Computational Geometry*, 30(2):113–127, 2005. Special Issue on the 19th European Workshop on Computational Geometry. doi:10.1016/j.comgeo.2004.05.004.

32 S.A. Piyavskii. An algorithm for finding the absolute extremum of a function. *USSR Computational Mathematics and Mathematical Physics*, 12(4):57–67, 1972. doi:10.1016/0041-5553(72)90115-2.

33 Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2D arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020. URL: https://doc.cgal.org/5.0.2/Manual/packages.html#PkgArrangementOnSurface2.

34 Carola Wenk. *Shape matching in higher dimensions*. PhD thesis, Freie Universität Berlin, 2002. PhD Thesis.

35 Martin Werner and Dev Oliver. ACM SIGSPATIAL GIS cup 2017: Range queries under Fréchet distance. *SIGSPATIAL Special*, 10(1):24–27, 2018.

36 L.A. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.

Improved Algorithms for Alternating Matrix Space Isometry: From Theory to Practice

Peter A. Brooksbank
Department of Mathematics, Bucknell University, Lewisburg, PA, USA
pbrooksb@bucknell.edu

Yinan Li
CWI and QuSoft, Amsterdam, The Netherlands
liyinan9252@gmail.com

Youming Qiao
Centre for Quantum Software and Information, University of Technology Sydney, Ultimo, Australia
Youming.Qiao@uts.edu.au

James B. Wilson
Department of Mathematics, Colorado State University, Fort Collins, CO, USA
James.Wilson@ColoState.Edu

Abstract

Motivated by testing isomorphism of p-groups, we study the alternating matrix space isometry problem (AltMatSpIso), which asks to decide whether two m-dimensional subspaces of $n \times n$ alternating (skew-symmetric if the field is not of characteristic 2) matrices are the same up to a change of basis. Over a finite field $\mathbb{F}_p$ with some prime $p \neq 2$, solving AltMatSpIso in time $p^{O(n+m)}$ is equivalent to testing isomorphism of p-groups of class 2 and exponent p in time polynomial in the group order. The latter problem has long been considered a bottleneck case for the group isomorphism problem.

Recently, Li and Qiao presented an average-case algorithm for AltMatSpIso in time $p^{O(n)}$ when n and m are linearly related (FOCS '17). In this paper, we present an average-case algorithm for AltMatSpIso in time $p^{O(n+m)}$. Besides removing the restriction on the relation between n and m, our algorithm is considerably simpler, and the average-case analysis is stronger. We then implement our algorithm, with suitable modifications, in Magma. Our experiments indicate that it improves significantly over default (brute-force) algorithms for this problem.

2012 ACM Subject Classification Computing methodologies → Algebraic algorithms; Computing methodologies → Combinatorial algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases Alternating Matrix Spaces, Average-case Algorithm, p-groups of Class 2and Exponent p, Magma

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.26

Related Version The main results of this submission come from [5] (`https://arxiv.org/abs/1905.02518`), which will be subdivided into several papers.

Funding The authors would like to acknowledge the financial support by NSF grant DMS-1750319. Additionally,
Peter A. Brooksbank: Partially supported by NSF grant DMS-1620362.
Yinan Li: Partially supported by ERC Consolidator Grant 615307-QPROGRESS.
Youming Qiao: Partially supported by the Australian Research Council DE150100720 and DP200100950.
James B. Wilson: Partially supported by NSF grant DMS-1620454.

Acknowledgements The authors would like to acknowledge László Babai and Xiaorui Sun for discussions, and Joshua A. Grochow for discussions and comments on improving the presentation. The authors acknowledge the Hausdorff Institute for Mathematics, the University of Auckland, the Santa Fe Institute, and the TACA workshop at the University of Colorado, Boulder and Colorado State University, where this research was carried out by (subsets of) the authors.

1 Introduction

Motivated by testing isomorphism of p-groups, we study the Alternating Matrix Space Isometry (AltMatSpIso) problem. To state this problem, we set up some notation and definitions. Let $[n] = \{1, \ldots, n\}$, and let $\mathrm{M}(n, q)$ be the linear space of $n \times n$ matrices over the finite field $\mathbb{F}_q$ (q is a prime power). A matrix $A \in \mathrm{M}(n, q)$ is *alternating* if for any vector $v \in \mathbb{F}_q^n$, $v^{\mathrm{t}} A v = 0$, where v^{t} denote the transpose of v. (When q is odd, A is alternating if and only if A is *skew-symmetric*, i.e. $A^{\mathrm{t}} = -A$.) Let $\Lambda(n, q)$ be the linear space of all alternating matrices in $\mathrm{M}(n, \mathbb{F}_q)$. Let $\mathcal{A}, \mathcal{B}$ be subspaces of $\Lambda(n, q)$. We say that $\mathcal{A}$ and $\mathcal{B}$ are *isometric*, if there exists an invertible matrix $T \in \mathrm{GL}(n, q)$, such that $T^{\mathrm{t}}\mathcal{A}T := \{T^{\mathrm{t}} A T : A \in \mathcal{A}\}$ and $\mathcal{B}$ are the same subspace. The AltMatSpIso problem then asks, given linear bases of two alternating matrix spaces $\mathcal{A}$ and $\mathcal{B}$, decide whether $\mathcal{A}$ and $\mathcal{B}$ are isometric.

As we will elaborate in detail in Subsection 1.1, the AltMatSpIso problem has been studied for decades. Indeed, it lies at the heart of the Group Isomorphism (GpI) problem, and has an intimate relationship with many other isomorphism problems, including Graph Isomorphism (GI). As a problem in $\mathsf{NP} \cap \mathsf{coAM}$, its worst-case time complexity has barely been improved over the brute-force algorithm, which enumerates all invertible matrices in $\mathrm{GL}(n, q)$ to search for some $T \in \mathrm{GL}(n, q)$ satisfying $T^{\mathrm{t}}\mathcal{A}T = \mathcal{B}$. The brute-force algorithm takes time $q^{\Theta(n^2)} \cdot \mathrm{poly}(n, m, \log q)$ in the worst case. To obtain an algorithm in time $q^{O(n+m)}$ would lead to an algorithm testing isomorphism of *p-groups of class 2 and exponent p* in time polynomial *in the group order*, which is a long-standing open problem.

Recently, Li and Qiao developed an *average-case* algorithm to tackle the AltMatSpIso problem, *when m and n are linearly related* [21]. The algorithm takes time $q^{O(n)}$ and tests isometry between all but at most $q^{-\Omega(n)}$ fraction of $\mathcal{A}$ in $\Lambda(n, q)$, and an arbitrary $\mathcal{B}$. Here a random m-dimensional $n \times n$ alternating matrix space $\mathcal{A}$ is chosen with probability $\begin{bmatrix} \binom{n}{2} \\ m \end{bmatrix}_q^{-1}$, where $\begin{bmatrix} \cdot \\ \cdot \end{bmatrix}_q$ denotes the q-Gaussian binomial coefficient and $\begin{bmatrix} \binom{n}{2} \\ m \end{bmatrix}_q$ is the total number of m-dimensional alternating matrix spaces in $\Lambda(n, q)$. This may be viewed as a linear algebraic analogue of the *Erdős-Rényi model* for graphs [10]. A key idea behind their algorithm is to view the AltMatSpIso problem as a linear algebraic analogue of the GI problem [21], which leads to adapting the individualisation and refinement technique for random graph isomorphism as used by Babai, Erdős, and Selkow in [2]. We summarise these algorithms in Subsection 1.2. For convenience, we shall refer to the algorithm from [21] as the Li-Qiao algorithm.

In this paper, we present another average-case algorithm for AltMatSpIso.

▶ **Theorem 1.** *Suppose $m \geqslant 20$. There is an algorithm that, for all but at most $q^{-\Omega(nm)}$ fraction of m-dimensional alternating matrix spaces $\mathcal{A}$ in $\Lambda(n, q)$, tests the isometry of $\mathcal{A}$ to an arbitrary m-dimensional alternating matrix space $\mathcal{B}$, in time $q^{O(n+m)}$.*

The algorithm in Theorem 1 significantly improves over the Li-Qiao algorithm:

- First, it removes the linear dependence of m on n. The Li-Qiao algorithm inherently requires this linear dependence. That is, for general m, the Li-Qiao algorithm does *not* run in time $q^{O(n+m)}$. Indeed, some of its techniques do not work when m is even $\Omega(n^{1+\epsilon})$ or $O(n^{1-\epsilon})$ with some $0 < \epsilon < 1$.
- Second, even in the case of $m = \Theta(n)$, the average-case analysis of our algorithm is better: it works for all but $q^{-\Omega(n^2)}$ fraction of $\mathcal{A}$, while the Li-Qiao algorithm works for all but $q^{-\Omega(n)}$ fraction.
- Third, our algorithm is considerably simpler than the Li-Qiao algorithm, as the reader may compare in the descriptions of these algorithms in Subsection 1.2 and Subsection 1.3.

Partly because of the simplicity of our algorithm, we implement our algorithm, with suitable modifications and some heuristic shortcuts, in MAGMA [6]. Our experiments indicate that it improves significantly over default (brute-force) algorithms for this problem.

An immediate open problem is to examine whether our isometry testing algorithm can be strengthened to a canonical form algorithm for random alternating matrix spaces. For graph isomorphism, efficient canonical form algorithms for random graphs have long been known [2, 3]. However, to the best of authors' knowledge, there have been no canonical form algorithms for random alternating matrix spaces even in time $q^{o(n^2)}$.

1.1 Motivation and related works

Group isomorphism problem. The main motivation for us to study the AltMatSpIso problem is to understand the complexity of the Group Isomorphism (GpI) problem: Deciding whether two finite groups of order N are isomorphic. The complexities of GpI depend on how groups are represented in algorithms. When groups are specified by their multiplication (Cayley) tables, GpI reduces to the Graph Isomorphism problem (GI); cf. [24, Section 10]. When groups are represented by generators as permutations, matrices, or black-box groups, GI reduces to GpI; cf. [14, 23, 24, 13]. In either input model, the state-of-the-art algorithm runs in time *quasi-polynomial* in the group order [11, 26].

For our purpose, we shall mostly focus on the Cayley table model, since we do not even know an $N^{o(\log N)}$-time algorithm [29] (log to the base 2), despite that a simple $N^{\log N+O(1)}$-time algorithm has been known for decades [11, 26]. We note that Rosenbaum presented an algorithm in time $N^{\frac{1}{2}\log N+O(1)}$ [27]. Moreover, following Babai's breakthrough proof that GI is in quasi-polynomial time [1, 15], GpI in Cayley table model is now a key bottleneck to put GI in P, as Babai himself pointed out [1, Sec. 13.2 & 13.4 in arXiv version]. The past few years have witnessed a resurgence of activity on algorithms for this problem with worst-case analyses in terms of the group order. We refer the reader to [12] which contains a survey of these algorithms.

A natural approach to tackle GpI is to assume our given groups lie in a certain group class. For instance, for Abelian groups, one can test their isomorphism in linear time [19]. However, moving out Abelian a little bit, p-groups of class 2 and exponent p, the next natural group class beyond Abelian groups, pose great difficulty. Recall that a group is of exponent p if any nontrivial element has order p, and a group is of class 2 if its commutator subgroup is contained in its centre. Recent works [20, 9, 7, 21] solved some nontrivial subclasses, and lead to substantial improvement in practical algorithms. But their methods do not lead to any improvement for the worst-case time complexity of the general class.

***p*-groups of class 2 and exponent *p*, and alternating matrix spaces.** Alternating matrix spaces naturally appear in the study of p-groups of class 2 and exponent p via Baer's correspondence [4] for a prime $p > 2$. In fact, because of this correspondence, most recent works on this class of groups study alternating matrix spaces or alternating bilinear maps [20, 9, 7, 21]. We review this correspondence briefly. Given such a p-group, by taking the commutator map, one obtains an alternating bilinear map. Conversely, given an alternating bilinear map, one can construct such a p-group using an explicit formula (see e.g. [12, Fact 4.3]). Given an alternating bilinear map $\mathbb{F}_p^n \times \mathbb{F}_p^n \to \mathbb{F}_p^m$, one can obtain m $n \times n$ alternating matrices representing it, and take the linear span to get an alternating matrix space. Given an alternating matrix space, by taking a linear basis, one can obtain an alternating bilinear map. The above procedures preserve and respect isomorphism types of groups and isometry types of alternating matrix spaces. Moreover, it is easy to verify that the above procedures

are computationally efficient in the Cayley table model [1]. Therefore, testing isomorphism of p-groups of class 2 and exponent p in time polynomial in the group order reduces to solving AltMatSpIso over $\mathbb{F}_p$ in time $p^{O(n+m)}$. Because of the current status of GpI, we see that solving AltMatSpIso in $q^{O(n+m)}$ is already very difficult. Recently, it has been shown that a large number of (hard) isomorphism problems can be reduced to AltMatSpIso in polynomial time; we refer the readers to [13] for more details. As an application, in [18], one can also build post-quantum cryptography schemes based on AltMatSpIso.

Current status of AltMatSpIso. For the AltMatSpIso problem, the brute-force algorithm takes time $q^{\Theta(n^2)} \cdot \mathrm{poly}(n, m, \log q)$. There are two slightly improved worst-case algorithms within certain range of parameters: the $N^{\frac{1}{4}\log_p N+O(1)}$-time algorithm for p-group isomorphism by Rosenbaum [27] and Rosenbaum and Wagner [28] translates to a $p^{\frac{1}{4}(n+m)^2+O(n+m)}$-time algorithm for AltMatSpIso over $\mathbb{F}_p$. Li and Qiao [21] adapted Luks' dynamic programming technique for GI [25] to obtain a $q^{\frac{1}{4}(n^2+m^2)+O(n+m)}$-time algorithm for AltMatSpIso. Note that both algorithms only improve the worst-case time-complexity when $m \leqslant n$; in fact, if $m = \Omega(n^2)$, then the brute-force algorithm already runs in time $q^{O(n+m)}$.

Another extreme case is when $m = O(1)$, where AltMatSpIso can be solved in $\mathrm{poly}(n, q)$ time for odd q. The algorithm is based on reducing AltMatSpIso to the Alternating Matrix Tuple Isometry (AltMatTupIso) problem, which asks the following: Given two m-tuples of alternating matrices $\mathbf{A}, \mathbf{B} \in \Lambda(n, q)^m$, decide whether there is an invertible matrix $T \in \mathrm{GL}(n, q)$ such that $T^{\mathrm{t}}\mathbf{A}T = (T^{\mathrm{t}}A_1T, \ldots, T^{\mathrm{t}}A_mT) = (B_1, \ldots, B_m) = \mathbf{B}$. Unlike AltMatSpIso, the AltMatTupIso problem can be solved in time $\mathrm{poly}(n, m, \log q)$ for *odd* q [17]. The reduction from AltMatSpIso to AltMatTupIso is by fixing a tuple of alternating matrices $\mathbf{A}$ in $\mathcal{A}$ as basis, and enumerate all possible m-tuples of alternating matrices $\mathbf{B}$ in $\mathcal{B}^m$. Then one can invoke the algorithm for AltMatTupIso to test isometry between $\mathbf{A}$ and $\mathbf{B}$ efficiently.

1.2 Average-case algorithms for GI and AltMatSpIso

In this subsection, we review the average-case algorithms for GI [2] and for AltMatSpIso [21].

To obtain average-case algorithms for GI (resp. AltMatSpIso), one needs to identify a property which is satisfied by almost all graphs (resp. alternating matrix spaces) chosen from a certain random model. Then for those graphs (resp. AltMatSpIso) satisfying the property, we test its isomorphism (resp. isometry) with an arbitrary graph (resp. alternating matrix space). The efficiency of the algorithm is guaranteed by both the efficiency of testing the property and the efficiency of the isomorphism (resp. isometry) testing.

We clarify the random models. In the graph setting, a natural choice is the celebrated Erdős-Rényi model [10], in which an n-vertex m-edge graph is endowed with probability $\binom{\binom{n}{2}}{m}^{-1}$. In the alternating matrix space setting, Li and Qiao defined a linear algebraic analogue of the Erdős-Rényi model, where each m-dimensional alternating matrix space in $\Lambda(n, q)$ will be chosen with probability $\begin{bmatrix}\binom{n}{2}\\ m\end{bmatrix}_q^{-1}$; see also Definition 2.

Average-case algorithm for GI. We first define a property $\mathcal{P}$ for graphs, which is a variant used in [2]. Let $G = ([n], E)$ be a simple undirected graph. Let $r \leqslant n$ be a positive integer, $S = [r]$ and $S' = [n] \setminus [r]$. Let $B = (S \cup S', F)$ be the bipartite graph induced by the cut (S, S') in G, where $F = \{\{i, j\} \in E : \ i \in S, \ j \in S'\}$. We label each $j \in S'$ by its adjacency

[1] The procedures are even efficient in matrix groups over finite fields [13, Lemma 7.5].

relations with those vertices in S. That is, assign an r-bit string $f_j \in \{0,1\}^r$ to each $j \in S'$ such that $f_j(i) = 1$ if and only if $\{i,j\} \in F$. We say a graph G satisfies property $\mathcal{P}$ if f_j's are distinct over $j \in S'$. It is easy to verify that, choosing $r = \lceil 3 \log n \rceil$, all but at most $n^{-\Omega(1)}$ fraction of graphs in the Erdős-Rényi model satisfy the property $\mathcal{P}$.

Here is an algorithm which tests isomorphism between graph G (satisfying property $\mathcal{P}$) and $H = ([n], E')$ (an arbitrary one), based on the well-known *individualisation and refinement* procedure. Let St_G be the set of r-bit strings obtained from the property $\mathcal{P}$. Note that $|\mathrm{St}_G| = n - r$. In the individualisation step, we enumerate all r-tuples of vertices in H. For each r-tuple $(i_1, \ldots, i_r) \in [n]^r$, we perform the refinement step: assign the remaining vertices in H r-bit strings according to their adjacency relations with the r-tuple $(i_1, \ldots, i_r)$ as before, to obtain another set of bit strings St_H. If $\mathrm{St}_G \neq \mathrm{St}_H$ we neglect this r-tuple; otherwise we obtained a bijective map from $[n]$ to $[n]$ by mapping j to i_j when $j \in [r]$ and the rest according to the labels. The last step is to check whether this bijective map induces an isomorphism between G and H.

The above algorithm runs in time $n^{O(\log n)}$ if G satisfies property $\mathcal{P}$. To recover the algorithm in [2], one can canonicalise the choice of the r-tuples by choosing the one with largest r degrees for both G and H, assuming that the largest r degrees are distinct. (This is another property which is satisfied by most graphs.)

Average-case algorithm for AltMatSpIso. Li and Qiao generalised the aforementioned graph property and the individualisation and refinement procedure to the alternating matrix space setting. It is helpful to think of alternating matrix spaces as a linear algebraic analogue of graphs. That is, we view vectors in $\mathbb{F}_q^n$ as a linear algebraic analogue of vertices. We then viewing alternating matrices as a linear algebraic analogue of edges. This is because we can think of edges as binary relations and alternating matrices as alternating bilinear forms on vectors.

We first define a property $\mathcal{Q}$ for alternating matrix spaces, in light of the one defined for graphs. Let $\mathcal{A}$ be an m-dimensional alternating matrix space in $\Lambda(n,q)$, and let $r \leqslant n$ be a positive integer. Let U_0 and V_0 be the $n \times r$ and $n \times (n-r)$ matrices over $\mathbb{F}_q$, whose columns are the first r and last $(n-r)$ standard basis, respectively. Let $\mathcal{A}_{U_0,V_0} = {U_0}^{\mathrm{t}}\mathcal{A}V_0 = \mathrm{span}\{{U_0}^{\mathrm{t}}AV_0 : \ A \in \mathcal{A}\}$, which is a subspace of $\mathrm{M}(r \times (n-r), q)$. The role of $\mathcal{A}_{U_0,V_0}$ is similar to the role of the bipartite graph $B = (S \cup S', F)$ in the graph setting.[2] Recall that in the graph setting, the hope was to label vertices in S' *uniquely*, meaning that they should have different adjacency relations with vertices in S. An equivalent way of saying this is that the right automorphism group of this bipartite graph B – those permutations of the right-hand-side vertices preserving the graph structure – is trivial. Inspired by this, Li and Qiao define the property $\mathcal{Q}$ on $\mathcal{A}$ as $\mathcal{R} := \{R \in \mathrm{GL}(n-r,q) : \ \mathcal{A}_{U_0,V_0}R = \mathcal{A}_{U_0,V_0}\}$ has size at most q^n, where $\mathcal{A}_{U_0,V_0}R = \{BR : B \in \mathcal{A}_{U_0,V_0}\}$. Here, $\mathcal{R}$ can be thought of as the corresponding to the right automorphism group in the graph setting. In [21], it is shown that when m and n are linearly related ($m = \Theta(n)$) and r is a constant (depending on the ratio m/n), $|\mathcal{R}| \leqslant q^n$ for all but at most $q^{-\Omega(n)}$ fractions of alternating matrix spaces.

Now we outline the Li-Qiao algorithm for isometry testing. Let $\mathcal{B}$ be another m-dimensional alternating matrix space in $\Lambda(n,q)$ for testing isometry with $\mathcal{A}$ (satisfying the property $\mathcal{Q}$). In the individualisation step, we enumerate all $n \times r$ matrices $U \in \mathrm{M}(n \times r, q)$ whose columns are linearly independent. Denote the columns space of U by C_U. We also need to enumerate all $(n-r)$-dimensional subspaces C_V such that $C_U \oplus C_V = \mathbb{F}_q^n$. C_V is specified

[2] Matrix spaces of the form $\mathcal{A}_{U_0,V_0}$ are studied as a linear algebraic analogue of cuts on graphs in [22].

by an $n \times (n-r)$ matrix V whose columns span C_V. Let $\mathcal{B}_{U,V} = \{U^{\mathrm{t}}BV : B \in \mathcal{B}\}$, which is a subspace of $\mathrm{M}(n \times (n-r), q)$. For each (U, V) with $\mathcal{B}_{U,V}$ we perform the refinement step. That is, we enumerate all $R \in \mathrm{GL}(n-r, q)$ such that $\mathcal{A}_{U_0,V_0}R = \mathcal{B}_{U,V}$, which can be done using algorithms from [8, 16][3]. If no such R exists we neglect the pair (U, V). Otherwise, we can recover an invertible $T \in \mathrm{GL}(n, q)$ from the information of (U, V) and R. Finally, check whether T is an isometry between $\mathcal{A}$ and $\mathcal{B}$.

The above algorithm runs in time $q^{O(n)}$, if $\mathcal{A}$ satisfies property $\mathcal{Q}$ (recall that n and m are linearly related and $r = O(1)$). In particular, the two enumerations in the individualisation step take time at most $q^{rn+r(n-r)} = q^{O(n)}$. In the refinement step, since $\mathcal{A}$ satisfies property $\mathcal{Q}$, $|\{R \in \mathrm{GL}(n-r, q) : \mathcal{A}_{U_0,V_0}R = \mathcal{B}_{U,V}\}|$ can be bounded above by q^n, as the set $\{R \in \mathrm{GL}(n-r, q) : \mathcal{A}_{U_0,V_0}R = \mathcal{B}_{U,V}\}$ is either empty, or a coset of the group H, whose order is upper bounded by q^n as $\mathcal{A}$ satisfies property $\mathcal{Q}$ we imposed. Thus the enumeration cost in the refinement step is at most q^n. All the other steps can be carried out in time $\mathrm{poly}(n, m, \log q)$; we refer the readers to [21] for more technical details on how to verify whether $\mathcal{A}$ satisfies the property $\mathcal{Q}$ and how to enumerate elements in $\{R \in \mathrm{GL}(n-r, q) : \mathcal{A}_{U_0,V_0}R = \mathcal{B}_{U,V}\}$ and get the invertible matrix T from R and (U, V).

1.3 A simplified algorithm and its implementation

Although it is a nice linear algebraic analogue of the algorithm in [2], the algorithm in [21] has several drawbacks. First, the algorithm only works when m and n are linear related. Second, the algorithm is still somewhat tricky, which makes it difficult to put into actual implementation. In this subsection, we describe a *simpler* average-case algorithm for AltMatSpIso, which works for all m and n (but only for odd q) and achieves the same performance as the one of Li–Qiao. A detailed description can be found in Subsection 4.1. This simplified algorithm already captures the essence of the strategy. The main algorithm for Theorem 1 in Subsection 4.3 further works for any q and achieves better average-case analysis.

The key idea behind our algorithm. Let $\mathcal{A} \leqslant \Lambda(n, q)$. The key idea behind our algorithm is to replace individualising r-dimensional subspaces of $\mathbb{F}_q^n$ by individualising r-dimensional subspaces of $\mathcal{A}$. This is inspired by the notion of *genus* of p-groups of class 2 and exponent p in [7].

Roughly speaking, genus-r p-groups of class 2 and exponent p correspond to r-dimensional alternating matrix spaces over $\mathbb{F}_p$. In [7], it is shown that even genus-2 p-groups of class 2 and exponent p demonstrate interesting behaviours. That is to say, constant-genus p-groups are already non-trivial objects. This leads us to consider individualising constant-dimensional subspaces of $\mathcal{A}$. In the graph setting, this would correspond to individualising r edges, which does not differ much from individualising $2r$ vertices, as each edge connects to two vertices. In the alternating matrix space setting, it turns out that individualising r-dimensional subspaces of $\mathcal{A}$ could impose severe constraints on the possible isometries, if this subspace satisfies certain generic conditions. This is not so surprising, as one full-rank alternating matrix is much more "powerful" than a single edge. Reflecting back, the combination of individualisation and refinement from graphs and the genus concept from p-groups reveals a nice interaction between graph-theoretic techniques and group-theoretic notions. We would

[3] In fact, the uses of [16, 8] are not necessary, as one can relax the property $\mathcal{Q}$ and then only use certain linear algebra computations; see [21].

also like to add that, the reason for the algorithm in [21] to individualising r-dimensional subspaces of $\mathbb{F}_q^n$ is because it was a close analogy of the average-case graph isomorphism algorithm of Babai, Erdős, and Selkow [2].

Algorithm outline. We first state another property $\mathcal{Q}'$ on a m-dimensional $\mathcal{A} \leqslant \Lambda(n,q)$. Let $\mathbf{A} = (A_1, \ldots, A_m)$ be a tuple of ordered linear basis of $\mathcal{A}$. For $c \in \mathbb{N}$, set $\mathbf{A}_c = (A_1, \ldots, A_c)$. We say $\mathcal{A}$ satisfies the property $\mathcal{Q}'$ if the group $\mathrm{Aut}(\mathbf{A}_c) := \{T \in \mathrm{GL}(n,q) : \ T^{\mathrm{t}}\mathbf{A}_c T = \mathbf{A}_c\}$ has order at most q^n.[4] When c is a large enough constant, all but at most $q^{-\Omega(n)}$ fraction of alternating matrix spaces satisfy property $\mathcal{Q}$, as shown in Theorem 5. Furthermore by algorithms in [9] (cf. Theorem 8), a generating set of $\mathrm{Aut}(\mathbf{A}_c)$ can be computed in time $\mathrm{poly}(n, \log q)$.

Our algorithm can be now stated as follows. Assume we would like to test isometry for $\mathcal{A}$ (satisfying the property $\mathcal{Q}'$) with an arbitrary $\mathcal{B}$, specified as two m-tuples of alternating matrices $\mathbf{A} = (A_1, \ldots, A_m)$ and $\mathbf{B} = (B_1, \ldots, B_m)$ from $\Lambda(n,q)^m$ for sufficiently large m and odd q. In the individualisation step, enumerate all c-tuples of alternating matrices $\mathbf{B}_c = (B_1, \ldots, B_c) \in \mathcal{B}^c$. For each $\mathbf{B}_c$ we perform the refinement step: Test isometry for alternating matrix tuples $\mathbf{A}_c$ and $\mathbf{B}_c$, using the $\mathrm{poly}(n, c, \log q)$-time algorithm in [17] (cf. Theorem 8). If they are not isometric, we neglect $\mathbf{B}_c$. Otherwise, the algorithm from [17] computes a specific isometry. Because we have computed $\mathrm{Aut}(\mathbf{A}_c)$, we can enumerate over all isometry T such that $T^{\mathrm{t}}\mathbf{A}_c T = \mathbf{B}_c$, and check whether $T^{\mathrm{t}}\mathcal{A}T = \mathcal{B}$.

The correctness lies in the simple fact that every isometry of $\mathcal{A}$ and $\mathcal{B}$ maps $\mathbf{A}_c$ to a c-tuple $\mathbf{B}_c$ in $\mathcal{B}^c$. The running time of the above procedure is $q^{O(n+m)}$ if $\mathcal{A}$ satisfies property $\mathcal{Q}'$. In particular, the enumeration cost in the individualisation step is at most q^{cm}. In the refinement step, enumerating all invertible matrix T such that $T^{\mathrm{t}}\mathbf{A}_c T = \mathbf{B}_c$ can be done in time $q^{O(n)} \cdot \mathrm{poly}(n, m, \log q)$ when q is odd, since $\{T \in \mathrm{GL}(n,q) : \ T^{\mathrm{t}}\mathbf{A}_c T = \mathbf{B}_c\}$ is a coset of the group $\{T \in \mathrm{GL}(n,q) : \ T^{\mathrm{t}}\mathbf{A}_c T = \mathbf{A}_c\}$, whose size is upper bounded by q^n by property $\mathcal{Q}'$. This algorithm is clearly simpler than the Li-Qiao algorithm, as it does not need to compute $\mathcal{B}_{U,V}$ etc..

Implementation and Performance. A bonus is our algorithm is more suitable to implement. We do so in Magma with some key adjustments, as detailed in Subsection 4.2. The implementation is publicly available on GitHub as part of a comprehensive collection of tools – developed and maintained by P. A. Brooksbank, J. Maglione, J. B. Wilson and their collaborators – to compute with groups, algebras, and multilinear functions [6].

The implementation is more convenient to describe using alternating bilinear maps (as the default algorithms do). A bilinear map $\alpha : V \times V \to W$ is alternating if for any $v \in V$, $\alpha(v, v) = 0$. Two alternating bilinear maps $\alpha, \beta : V \times V \to W$ are pseudo-isometric, if they are the same up to the natural action of $\mathrm{GL}(V) \times \mathrm{GL}(W)$. Let $V \cong \mathbb{F}_q^n$ and $W \cong \mathbb{F}_q^m$. An alternating bilinear map α can be specified by an m-tuple of alternating matrices from $\Lambda(n,q)$. Let α and β be represented by the alternating matrix tuples $\mathbf{A}$ and $\mathbf{B}$, respectively and $\mathcal{A}$ and $\mathcal{B}$ be the corresponding alternating matrix spaces of $\mathbf{A}$ and $\mathbf{B}$, respectively. It is readily verified that α and β are pseudo-isometric if and only if $\mathcal{A}$ and $\mathcal{B}$ are isometric.

Absent additional characteristic structure that can be exploited, the traditional (brute force) approach to deciding pseudo-isometry between alternating bilinear maps $\alpha, \beta \colon V \times V \to W$ is as follows. Let $\hat{\alpha}, \hat{\beta} \colon V \wedge V \to W$ denote the linear maps induced by α, β ($V \wedge V$ is the wedge product of a vector space V with itself). Compute the natural (diagonal) action of $\mathrm{GL}(V)$ on $V \wedge V$, and decide if $\ker \hat{\alpha}$ and $\ker \hat{\beta}$ – each of codimension $\dim W$ in $V \wedge V$ –

[4] The alert reader would note that the property defined here depends on the choice of bases of $\mathcal{A}$. This is not an essential problem due to the discussion in Section 3.

belong to the same orbit. An alternative version of brute force is to enumerate $\mathrm{GL}(W)$ and check if one of these transformations lifts to a pseudo-isometry from α to β. Which of these two brute-force options represents the best choice depends on the dimensions of V and W.

Our implementation is typically an improvement over both options. For example, in a preliminary experiment, our implementation readily decides pseudo-isometry between randomly selected alternating bilinear maps $\mathbb{F}_3^5 \times \mathbb{F}_3^5 \to \mathbb{F}_3^4$, while both brute-force options failed to complete. Note that the worst-case for all methods should be when α, β are *not* isometric, since in that case one must exhaust the entire enumerated list (or orbit) to confirm non-equivalence. However, the modifications we made tend to detect non-equivalence rather easily, since other (easily computed) invariants typically do not align in this case. We were therefore careful to also run tests with equivalent inputs, so as to ensure a fair comparison with default methods.

2 Preliminaries

Let $[m] = \{1, \ldots, m\}$ for $m \in \mathbb{N}$. Let $\begin{bmatrix} n \\ d \end{bmatrix}_q = \frac{(1-q^n)\cdot\ldots\cdot(1-q^{n-d+1})}{(1-q^d)\cdot\ldots\cdot(1-q)}$ denote the Gaussian binomial coefficient with parameters n, d and base q. Note that $\begin{bmatrix} n \\ d \end{bmatrix}_q$ counts the number of d-dimensional subspaces of $\mathbb{F}_q^n$. The bound $\begin{bmatrix} n \\ d \end{bmatrix}_q \leqslant q^{nd}$ is useful.

Let $\mathrm{M}(n \times n', \mathbb{F})$ (resp. $\mathrm{M}(n, \mathbb{F})$) be the linear space of all $n \times n'$ (resp. $n \times n$) matrices over $\mathbb{F}$. For a matrix $A \in \mathrm{M}(n \times n', \mathbb{F})$, $A^{\mathrm{t}} \in \mathrm{M}(n' \times n, \mathbb{F})$ denotes the transpose of A. We use $A(i, j)$ to denotes the (i, j)th entry of the matrix A. The general linear group of degree n over $\mathbb{F}$ is denoted by $\mathrm{GL}(n, \mathbb{F})$. When $\mathbb{F} = \mathbb{F}_q$ for some prime power q, we write simply $\mathrm{M}(n, q)$ and $\mathrm{GL}(n, q)$ in place of $\mathrm{M}(n, \mathbb{F}_q)$ and $\mathrm{GL}(n, \mathbb{F}_q)$. An $n \times n$ matrix A over $\mathbb{F}$ is alternating if for every $v \in \mathbb{F}^n$, $v^{\mathrm{t}} A v = 0$. When $\mathbb{F}$ is not of characteristic 2, this is equivalent to the skew-symmetric condition. Let $\Lambda(n, \mathbb{F})$ be the linear space of all $n \times n$ alternating matrices over $\mathbb{F}$ (and $\Lambda(n, q)$ when $\mathbb{F} = \mathbb{F}_q$). We denote $\mathbf{A} = (A_1, \ldots, A_m) \in \Lambda(n, q)^m$ to be an alternating matrix tuple and $\mathcal{A} = \mathrm{span}\{A_1, \ldots, A_m\} \leqslant \Lambda(n, q)$ be an (the corresponding) alternating matrix space ($\leqslant$ denote the subspace notation).

We say two alternating matrix tuples $\mathbf{A}, \mathbf{B} \in \Lambda(n, \mathbb{F})^m$ are *isometric* if there exists $T \in \mathrm{GL}(n, \mathbb{F})$ such that

$$T^{\mathrm{t}}\mathbf{A}T := (T^{\mathrm{t}}A_1T, \ldots, T^{\mathrm{t}}A_mT) = (B_1, \ldots, B_m) = \mathbf{B}.$$

The set of isometries between $\mathbf{A}$ and $\mathbf{B}$ is denoted as

$$\mathrm{Isom}(\mathbf{A}, \mathbf{B}) = \{T \in \mathrm{GL}(n, \mathbb{F}) : T^{\mathrm{t}}\mathbf{A}T = \mathbf{B}\};$$

the group of *autometries* (or self-isometries) of $\mathbf{A}$ is denoted as $\mathrm{Aut}(\mathbf{A}) = \mathrm{Isom}(\mathbf{A}, \mathbf{A})$. We say two alternating matrix tuples $\mathbf{A}, \mathbf{B} \in \Lambda(n, \mathbb{F})^m$ are *pseudo-isometric* if there exist $T \in \mathrm{GL}(n, \mathbb{F})$ and $R \in \mathrm{GL}(m, \mathbb{F})$ such that

$$T^{\mathrm{t}}\mathbf{A}T = (T^{\mathrm{t}}A_1T, \ldots, T^{\mathrm{t}}A_mT) = (\sum_{j=1}^{m} R(1, j)B_j, \ldots, \sum_{j=1}^{m} R(m, j)B_j) =: \mathbf{B}^R.$$

The set of pseudo-isometries between $\mathbf{A}$ and $\mathbf{B}$ is defined as

$$\Psi\mathrm{Isom}(\mathbf{A}, \mathbf{B}) = \{T \in \mathrm{GL}(n, \mathbb{F}) : \exists\ R \in \mathrm{GL}(m, q),\ T^{\mathrm{t}}\mathbf{A}T = \mathbf{B}^R\}.$$

The group of *pseudo-autometries* (or self-pseudo-isometries) of $\mathbf{A}$ is denoted as $\Psi\mathrm{Aut}(\mathbf{A}) = \Psi\mathrm{Isom}(\mathbf{A}, \mathbf{A})$. It is straightforward to see that $\mathrm{Isom}(\mathbf{A}, \mathbf{B})$ (resp. $\Psi\mathrm{Isom}(\mathbf{A}, \mathbf{B})$) is a (possibly empty) coset of $\mathrm{Aut}(\mathbf{A})$ (resp. $\Psi\mathrm{Aut}(\mathbf{A})$).

We say two alternating matrix spaces $\mathcal{A}, \mathcal{B} \leqslant \Lambda(n, \mathbb{F})$ are *isometric*, if there exists $T \in \mathrm{GL}(n, \mathbb{F})$ such that $T^{\mathrm{t}}\mathcal{A}T := \{T^{\mathrm{t}}AT : A \in \mathcal{A}\} = \mathcal{B}$. We can define the coset of isometries from $\mathcal{A}$ to $\mathcal{B}$ $\mathrm{Isom}(\mathcal{A}, \mathcal{B})$, and the group of autometries $\mathrm{Aut}(\mathcal{A})$, for alternating matrix spaces. If $\mathcal{A}$ and $\mathcal{B}$ are the corresponding alternating matrix spaces spanned by $\mathbf{A}$ and $\mathbf{B}$, respectively, then $\mathcal{A}$ and $\mathcal{B}$ are isometric if and only if $\mathbf{A}$ and $\mathbf{B}$ are pseudo-isometric, i.e. $\mathrm{Isom}(\mathcal{A}, \mathcal{B}) = \Psi\mathrm{Isom}(\mathbf{A}, \mathbf{B})$.

For two tuples of alternating matrices $\mathbf{A}, \mathbf{B} \in \Lambda(n, \mathbb{F})^m$, the the *adjoint space* from $\mathbf{A}$ to $\mathbf{B}$ is defined as $\mathrm{Adj}(\mathbf{A}, \mathbf{B}) = \{(T, T') \in \mathrm{M}(n, \mathbb{F}) \oplus \mathrm{M}(n, \mathbb{F}) : T\mathbf{A} = \mathbf{B}T'\}$. The *adjoint algebra* of $\mathbf{A}$ is defined as $\mathrm{Adj}(\mathbf{A}) = \{(T, T') \in \mathrm{M}(n, \mathbb{F}) \oplus \mathrm{M}(n, \mathbb{F}) : T\mathbf{A} = \mathbf{A}T'\}$. Clearly, if $T \in \mathrm{Aut}(\mathbf{A})$, then $(T^{\mathrm{t}}, T^{-1}) \in \mathrm{Adj}(\mathbf{A})$. Furthermore, if $\mathbf{A}$ and $\mathbf{B}$ are isometric, then $|\mathrm{Adj}(\mathbf{A}, \mathbf{B})| = |\mathrm{Adj}(\mathbf{A})|$.

3 Random models and average-case properties

We now formally define the linear algebraic analogue of the Erdős-Rényi model, which has been mentioned frequently in Section 1.

▶ **Definition 2** (The linear algebraic analogue of the Erdős-Rényi model)**.** *The linear algebraic analogue of the Erdős-Rényi model,* $\mathrm{LinER}(n, m, q)$*, is the uniform probability distribution over the set of* m*-dimensional subspaces of* $\Lambda(n, q)$*, that is, each subspace is endowed with probability* $\begin{bmatrix} \binom{n}{2} \\ m \end{bmatrix}_q^{-1}$*.*

We also recall a random model for alternating matrix tuples, introduced in [21].

▶ **Definition 3** (The naive model for alternating matrix tuples)**.** *The naive model for alternating matrix tuples,* $\mathrm{NaiT}(n, m, q)$*, is the probability distribution over the set of all* m*-tuples of* $n \times n$ *alternating matrices over* $\mathbb{F}_q$*, where each tuple is endowed with probability* $q^{-\binom{n}{2}m}$*.*

A useful fact is that if we would like to show a certain property holds with high probability for alternating matrix spaces in $\mathrm{LinER}(n, m, q)$, we can in turn show that a corresponding property holds with high probability for alternating matrix tuples in $\mathrm{NaiT}(n, m, q)$. The statement can be quantified as follows: Suppose we have $\mathcal{P}(n, m, q)$, a property of m-dimensional alternating matrix spaces in $\Lambda(n, q)$, and wish to show that $\mathcal{P}(n, m, q)$ holds with high probability in $\mathrm{LinER}(n, m, q)$. $\mathcal{P}(n, m, q)$ naturally induces $\mathcal{P}'(n, m, q)$, a property of alternating matrix tuples in $\Lambda(n, q)^m$ that span m-dimensional alternating matrix spaces. Let $\mathcal{Q}(n, m, q)$ be a property of all alternating matrix tuples in $\Lambda(n, q)^m$, so that $\mathcal{Q}(n, m, q)$ and $\mathcal{P}'(n, m, q)$ coincide when restricting to those alternating tuples spanning m matrix spaces. The following is proved in [21].

▶ **Proposition 4** ([21, Proposition 13 in arXiv version])**.** *Let* $\mathcal{P}(n, m, q)$ *and* $\mathcal{Q}(n, m, q)$ *be as above. Suppose in* $\mathrm{NaiT}(n, m, q)$*,* $\mathcal{Q}(n, m, q)$ *happens with probability at least* $1 - f(n, m, q)$ *for* $0 \leqslant f(n, m, q) < 1$*. Then in* $\mathrm{LinER}(n, m, q)$*,* $\mathcal{P}(n, m, q)$ *happens with probability at least* $1 - 4f(n, m, q)$*.*

In the rest of this paper, we assume a random alternating matrix space is chosen from the linear algebraic analogue of the Erdős-Rényi model and a random alternating matrix tuple is chosen from the naive model. To prove Theorem 1, it is sufficient to work with alternating matrix tuples and the naive model.

We now present the average-case property, which will be used in our algorithm. Recall that in Subsection 1.3, the desired property is to show that, for a random alternating matrix tuple $\mathbf{A} = (A_1, \ldots, A_m) \in \Lambda(n, q)^m$ and some $c = O(1)$, the tuple $\mathbf{A}_c = (A_1, \ldots, A_c)$ has autometry group $\mathrm{Aut}(\mathbf{A}_c)$ of size at most q^n with probability $1 - q^{-\Omega(n)}$. We prove a stronger statement.

▶ **Theorem 5.** *Let $c = 20$. For all but at most $q^{-\Omega(n)}$ fraction of $\mathbf{A}_c = (A_1, \ldots, A_c) \in \Lambda(n,q)^c$, we have $|\operatorname{Adj}(\mathbf{A}_c)| \leqslant q^n$.*

Note that $|\operatorname{Aut}(\mathbf{A}_c)| \leqslant |\operatorname{Adj}(\mathbf{A}_c)|$ for any $\mathbf{A}_c$. To prove Theorem 5, we need the following from [21]. Given a tuple of matrices $\mathbf{A} = (A_1, \ldots, A_r) \in \mathrm{M}(n,q)^r$, define the image of $U \leqslant \mathbb{F}_q^n$ under $\mathbf{A}$ as $\mathbf{A}(U) := \operatorname{span}\{A_i u : \ i \in [r], \ u \in U\}$.

▶ **Definition 6.** We say $\mathbf{A} = (A_1, \ldots, A_r) \in \mathrm{M}(n,q)^r$ is *stable*, if for any nonempty proper subspace $U \leqslant \mathbb{F}_q^n$, we have $\dim(\mathbf{A}(U)) > \dim(U)$.

▶ **Proposition 7** ([21, Proposition 10 in arXiv version])**.** *If $\mathbf{A} \in \mathrm{M}(n,q)^r$ is stable, then $|\operatorname{Adj}(\mathbf{A})| \leqslant q^n$.*

Thus, to prove Theorem 5, we need to upper bound the probability of a random alternating matrix tuple being *not stable*. The proof is somewhat similar to the one in [21], with one interesting observation: We can use some random alternating matrices in $\Lambda(n,q)$ to "mimick" a random matrix $\mathbb{M}(n,q)$. The detail of the proof can be found, e.g. in [5, Section 6.3].

4 Average-case algorithms for AltMatSpIso

4.1 The simplified main algorithm

As we have mentioned in Subsection 1.3, our algorithm invokes the algorithm for testing isometry for alternating matrix tuples as subroutines, which is formally state it here.

▶ **Theorem 8** ([9, 17])**.** *Let $\mathbf{A}, \mathbf{B} \in \Lambda(n,q)^m$ for some odd q. There exists a $\operatorname{poly}(n, m, \log q)$-time algorithm which takes $\mathbf{A}$ and $\mathbf{B}$ as inputs and outputs $\operatorname{Isom}(\mathbf{A}, \mathbf{B})$, specified by (if nonempty) a generating set of $\operatorname{Aut}(\mathbf{A})$ (by the algorithm in [9]) and a coset representative $T \in \operatorname{Isom}(\mathbf{A}, \mathbf{B})$ (by the algorithm in [17]).*

We also need the following observation to enumerate elements in $\operatorname{Aut}(\mathbf{A})$, which follows easily by computing the closure of the given generating set.

▶ **Observation 9.** *Let $C_1, \ldots, C_t \in \mathrm{GL}(n,q)$, and let G be the group generated by C_i's. Let $s \in \mathbb{N}$. Then there exists an algorithm that either reports that $|G| > s$, or lists all elements in G, in time $\operatorname{poly}(s, n, \log q)$.*

Now we formally describe the simplified main algorithm for AltMatSpIso stated in Subsection 1.3, that is Algorithm 1. Note that we are given alternating matrix tuples $\mathbf{A}$ and $\mathbf{B}$, which span $\mathcal{A}$ and $\mathcal{B}$, respectively. By the discussion in Section 2, we can equivalently decide the pseudo-isometry between $\mathbf{A}$ and $\mathbf{B}$.

▶ **Proposition 10.** *Algorithm 1 runs in time $\operatorname{poly}(q^{cm}, s, n)$.*

Proof. If Algorithm 1 outputs $|\operatorname{Aut}(\mathbf{A}_c)| > s$, then its running time is determined by Theorem 8 and Observation 9, which together require $\operatorname{poly}(s, n, \log q)$.

If $|\operatorname{Aut}(\mathbf{A}_c)| \leqslant s$, we analyse the two For-loops at Step 4 and Step 4c, respectively. The first loop adds a multiplicative factor of q^{cm}, since enumerating all matrices in $\mathcal{B}$ costs q^m. The second loop adds a multiplicative factor of $\operatorname{poly}(n, s, \log q)$, due to the fact that $|\operatorname{Isom}(\mathbf{A}_c, \mathbf{B}_c)| = |\operatorname{Aut}(\mathbf{A}_c)| \leqslant s$ (as $\operatorname{Isom}(\mathbf{A}_c, \mathbf{B}_c)$ is a coset of $\operatorname{Aut}(\mathbf{A}_c)$). Other steps can be carried out in time $\operatorname{poly}(n, \log q)$. Therefore the overall running time is upper bounded by $\operatorname{poly}(q^{cm}, s, n)$. ◀

We prove the correctness of Algorithm 1, if it does not report $|\operatorname{Aut}(\mathbf{A}_c)| > s$.

Algorithm 1 The first average-case algorithm for AltMatSpIso.

Input: $\mathbf{A} = (A_1, \ldots, A_m), \mathbf{B} = (B_1, \ldots, B_m) \in \Lambda(n, q)^m$, $c, s \in \mathbb{N}$, and q is odd.
Output: Either (1) "$|\operatorname{Aut}(\mathbf{A}_c)| > s$.", or (2) $\Psi\mathrm{Isom}(\mathbf{A}, \mathbf{B})$ as a set L, which may be empty.
Algorithm procedure:

1. Set $L \leftarrow \{\}$. Set $\mathbf{A}_c = (A_1, \ldots, A_c)$.
2. Use Theorem 8 to compute a generating set for $\operatorname{Aut}(\mathbf{A}_c)$.
3. Use Observation 9 with input s and the generating set of $\operatorname{Aut}(\mathbf{A}_c)$. (If $|\operatorname{Aut}(\mathbf{A}_c)| > s$, we terminate the algorithm and report that "$|\operatorname{Aut}(\mathbf{A}_c)| > s$.")
4. Set $\mathcal{B} = \operatorname{span}\{B_1, \ldots, B_m\}$; for every $\mathbf{B}_c = (B'_1, \ldots, B'_c) \in \mathcal{B}^c$, do the following.
   (a) Use Theorem 8 to decide whether $\mathbf{A}_c$ and $\mathbf{B}_c$ are isometric.
   (b) If not, go to the next $\mathbf{B}_c$. Otherwise, we get the non-empty coset $\mathrm{Isom}(\mathbf{A}_c, \mathbf{B}_c)$.
   (c) For every $T \in \mathrm{Isom}(\mathbf{A}_c, \mathbf{B}_c)$, do the following.
   Test whether the linear spans of $T^{\mathrm{t}}\mathbf{A}T$ and $\mathbf{B}$ are the same. If not, go to the next T. If so, add T into L.
5. Output L.

▶ **Proposition 11.** *If Algorithm 1 does not report $|\operatorname{Aut}(\mathbf{A}_c)| > s$, then it lists the set of pseudo-isometries (resp. isometries) between $\mathbf{A}$ (resp. $\mathcal{A}$) and $\mathbf{B}$ (resp. $\mathcal{B}$). In particular, $|\mathrm{Isom}(\mathcal{A}, \mathcal{B})| = |\Psi\mathrm{Isom}(\mathbf{A}, \mathbf{B})| \leqslant q^{cm} \cdot s$.*

Proof. By Step 4c, every T added to L is a pseudo-isometry. We are left to show that L contains all the pseudo-isometries. For this, take any pseudo-isometry T. Since $T^{\mathrm{t}}\mathcal{A}T = \mathcal{B}$, we know $T^{\mathrm{t}}\mathbf{A}_cT$ is equal to some $\mathbf{B}_c \in \mathcal{B}^c$. So when enumerating these $\mathbf{B}_c$ in Step 4, T will pass all the tests in the following, and then be added to L. ◀

It remains to specify the choices of c and s in Algorithm 1. This can be done by Theorem 5.

▶ **Proposition 12.** *Let $c = 20$ and $s = q^n$. For all but at most $q^{-\Omega(n)}$ fraction of $\mathbf{A}_c = (A_1, \ldots, A_c) \in \Lambda(n, q)^c$, we have $|\operatorname{Aut}(\mathbf{A}_c)| \leqslant s$.*

Proof. This is because, if $T \in \operatorname{Aut}(\mathbf{A}_c)$, then $(T^{\mathrm{t}}, T^{-1}) \in \operatorname{Adj}(\mathbf{A}_c)$. So $|\operatorname{Aut}(\mathbf{A}_c)| \leqslant |\operatorname{Adj}(\mathbf{A}_c)|$. ◀

Combining Propositions 10 to 12, we have the following theorem.

▶ **Theorem 13.** *Let $m \geqslant 20$, and let $\mathbb{F}_q$ be a finite field of odd size. For all but at most $q^{-\Omega(n)}$ fraction of $\mathbf{A} = (A_1, \ldots, A_m) \in \Lambda(n, q)^m$, Algorithm 1 tests the isometry of $\mathbf{A}$ with an arbitrary $\mathbf{B} \in \Lambda(n, q)^m$ in time $q^{O(n+m)}$.*

4.2 Magma implementation of Algorithm 1

To make this algorithm suitable for practical purposes, recall that the algorithm's running time is dominated by the two For-loops which give multiplicative factors of q^{cm} and s, respectively. For the average-case analysis we used $c = 20$, but having this standing on the exponent is too expensive. In practice, actually using $c = 3$ already imposes a severe restriction on s, the order of $\operatorname{Aut}(\mathbf{A}_c)$. So we use $c = 3$ in the implementation which gives a reasonable performance.

But having q^{3m} in the For-loop is still too demanding. Indeed, in practice the tolerable enumeration is around 5^{10}, namely $q = 5$ and 10 on the exponent. So with $c = 3$, the range of m is still severely limited. (Interestingly, the algorithm seems to have a better dependence on n.) It is most desirable if we could let $c = 1$, namely simply q^m.

To achieve that we use the following heuristic. Note that if $A_1, \ldots, A_c$ are low-rank matrices, then we will only need to match them with the low-rank matrices from $\mathcal{B}$. Our experiement shows that, for a random m-tuple of alternating matrices $\mathbf{A}$ over $\mathbb{F}_q$, when q is a small constant, the number of low-rank (i.e. non-full-rank) matrices in the linear span of $\mathbf{A}$ is expected to be small (i.e. much smaller than q^m) and non-zero (i.e. no less than 3) at the same time. Note that the set of all low-rank matrices can be computed in time $q^m \cdot \mathrm{poly}(n, \log q)$-time. We then choose 3 low-rank matrices from the linear span of $\mathbf{A}$. Then use $q^m \cdot \mathrm{poly}(n, \log q)$-time to compute the set of low-rank matrices from $\mathcal{B}$, denoted as $\mathcal{B}_l$. We can then replace enumerating $\mathcal{B}^c$ with $\mathcal{B}_l^c$, which in general is much smaller.

4.3 The main algorithm and proof of Theorem 1

We now introduce the algorithm (see Algorithm 2) that supports Theorem 1, which differs from Algorithm 1 in two places.

1. The first and major difference is to replace the uses of $\mathrm{Aut}(\mathbf{A}_c)$ and $\mathrm{Isom}(\mathbf{A}_c, \mathbf{B}_c)$ with the adjoint algebra $\mathrm{Adj}(\mathbf{A}_c)$ and adjoint space $\mathrm{Adj}(\mathbf{A}_c, \mathbf{B}_c)$, thereby avoiding using Theorem 8. Since $\mathrm{Adj}(\mathbf{A}_c)$ and $\mathrm{Adj}(\mathbf{A}_c, \mathbf{B}_c)$ are easy to compute over any field, this removes the odd q issue. On the other hand, although $\mathrm{Adj}(\mathbf{A}_c)$ and $\mathrm{Adj}(\mathbf{A}_c, \mathbf{B}_c)$ are also easier to analyse, $\mathrm{Adj}(\mathbf{A}_c)$ and $\mathrm{Adj}(\mathbf{A}_c, \mathbf{B}_c)$ could be larger than $\mathrm{Aut}(\mathbf{A}_c)$ and $\mathrm{Isom}(\mathbf{A}_c, \mathbf{B}_c)$, so they are less useful from the practical viewpoint.
2. The second place is step 2 in Algorithm 2: instead of just using the first c matrices as in the algorithm presented in Algorithm 1, Algorithm 2 slices the m matrices of $\mathbf{A}$ into $\lfloor m/c \rfloor$ segments of c-tuples of matrices, and tries each segment until it finds one segment with a small adjoint algebra. This step helps in improving the average-case analysis, and can be applied to the algorithm presented in Algorithm 1 as well.

▶ **Proposition 14.** *Algorithm 2 runs in time* $\mathrm{poly}(q^{cm}, s, n)$.

Proof. If Algorithm 2 outputs "$\mathbf{A}$ does not satisfy the generic condition.", then it just executes the For-loop in Step 2, which together runs in time $\mathrm{poly}(m, n, \log q)$.

Otherwise, there are two For-loops at Step 4 and Step 4c, which add multiplicative factors q^{cm} and s, respectively. Other steps can be carried out in time $\mathrm{poly}(n, \log q)$. Therefore the whole algorithm runs in time $\mathrm{poly}(q^{cm}, s, n)$. ◀

We prove the correctness of Algorithm 2 in the case that it does not report "$\mathbf{A}$ does not satisfy the generic condition."

▶ **Proposition 15.** *Suppose that Algorithm 2 does not report "*$\mathbf{A}$ *does not satisfy the generic condition." Then the algorithm lists the (possibly empty) set of pseudo-isometries between* $\mathbf{A}$ *and* $\mathbf{B}$. *In particular,* $|\Psi\mathrm{Isom}(\mathbf{A}, \mathbf{B})| \leqslant q^{cm} \cdot s$.

Proof. By Step 4c, every T^{t} added to L is a pseudo-isometry. So we are left to show that L contains all the pseudo-isometries. For this, take an arbitrary pseudo-isometry T. Then T sends $\mathbf{A}_c$ to some $\mathbf{B}_c \in \mathcal{B}^c$, i.e., $T^{\mathrm{t}}\mathbf{A}_c T = \mathbf{B}_c$. In particular, $(T^{\mathrm{t}}, T^{-1}) \in \mathrm{Adj}(\mathbf{A}_c, \mathbf{B}_c)$. So when enumerating this $\mathbf{B}_c \in \mathcal{B}^c$, (T^{t}, T^{-1}) will pass all the tests in the following, and T will be be added to L. ◀

Algorithm 2 The second average-case algorithm for AltMatSpIso.

Input: $\mathbf{A} = (A_1, \ldots, A_m), \mathbf{B} = (B_1, \ldots, B_m) \in \Lambda(n,q)^m$ and $c, s \in \mathbb{N}$.

Output: Either (1) "$\mathbf{A}$ does not satisfy the generic condition."; or (2) $\Psi\mathrm{Isom}(\mathbf{A}, \mathbf{B})$ as a set L, which may be empty.

Algorithm procedure:

1. Set $L \leftarrow \{\}$. Set $F \leftarrow$ `false`.
2. For $i = 1, \ldots, \lfloor m/c \rfloor$, do the following.
   (a) Set $\mathbf{A}_c = (A_{c(i-1)+1}, \ldots, A_{ci})$.
   (b) Compute a linear basis of $\mathrm{Adj}(\mathbf{A}_c) \subseteq \mathrm{M}(n,q) \oplus \mathrm{M}(n,q)$.
   (c) If $|\mathrm{Adj}(\mathbf{A}_c)| \leqslant s$, set F to be `true`, and break the For-loop.
3. If $F =$ `false`, return "$\mathbf{A}$ does not satisfy the generic condition." and terminate. Otherwise,
4. Set $\mathcal{B} = \mathrm{span}\{B_1, \ldots, B_m\}$; for every $\mathbf{B}_c = (B_1, \ldots, B_c) \in \mathcal{B}^c$, do the following.
   (a) Compute a linear basis for $\mathrm{Adj}(\mathbf{A}_c, \mathbf{B}_c) \subseteq \mathrm{M}(n,q) \oplus \mathrm{M}(n,q)$.
   (b) If $|\mathrm{Adj}(\mathbf{A}_c, \mathbf{B}_c)| > s$, go to the next $\mathbf{B}_c$.
   (c) For every $(T, T') \in \mathrm{Adj}(\mathbf{A}_c, \mathbf{B}_c)$, do the following.
   If T and T' are invertible and $(T')^{-1} = T^{\mathrm{t}}$, test whether the linear spans of $T\mathbf{A}T^{\mathrm{t}}$ and $\mathbf{B}$ are the same. If not, go to the next (T, T'). If so, add T^{t} into L.
5. Output L.

Now we specify the choice of $c = 20$ and $s = q^n$, based on Theorem 5.

▶ **Proposition 16.** *Let $m \geqslant c = 20$, and let $\ell = \lfloor m/c \rfloor \in \mathbb{N}$. For all but at most $q^{-\Omega(n\cdot\ell)} = q^{-\Omega(nm)}$ fraction of $\mathbf{A} = (A_1, \ldots, A_m) \in \Lambda(n,q)^m$, there exists some $i \in [\ell]$, such that, letting $\mathbf{A}_{c,i} = (A_{c(i-1)+1}, \ldots, A_{ci})$, we have $|\mathrm{Adj}(\mathbf{A}_{c,i})| \leqslant q^n$.*

Proof. We slice $\mathbf{A}$ into $\ell = \lfloor m/c \rfloor$ segments, where each segment consists of c random alternating matrices. Each segment is some $\mathbf{A}_{c,i} \in \Lambda(n,q)^c$, with $\Pr[|\mathrm{Adj}(\mathbf{A}_{c,i})| > q^n] \leqslant q^{-\Omega(n)}$ by Theorem 5. Since $A_1, \ldots, A_m$ are chosen independently and uniformly at random, the probability of every $\mathbf{A}_{c,i} = (A_{c(i-1)+1}, \ldots, A_{ci})$, $i \in [\ell]$, with $|\mathrm{Adj}(\mathbf{A}_{c,i})| > q^n$, is upper bounded by $(q^{-\Omega(n)})^\ell = q^{-\Omega(nm)}$. ◀

Theorem 1 then follows from Propositions 14 to 16.

References

1 László Babai. Graph Isomorphism in Quasipolynomial Time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 684–697, 2016. `arXiv:1512.03547v2`. `doi:10.1145/2897518.2897542`.

2 László Babai, Paul Erdős, and Stanley M. Selkow. Random Graph Isomorphism. *SIAM Journal on Computing*, 9(3):628–635, 1980. `doi:10.1137/0209047`.

3 László Babai and Ludek Kucera. Canonical Labelling of Graphs in Linear Average Time. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 39–46, 1979. `doi:10.1109/SFCS.1979.8`.

4 Reinhold Baer. Groups with Abelian Central Quotient Group. *Transactions of the American Mathematical Society*, 44(3):357–386, 1938. `doi:10.2307/1989886`.

5 Peter A. Brooksbank, Joshua A. Grochow, Yinan Li, Youming Qiao, and James B. Wilson. Incorporating Weisfeiler-Leman into Algorithms for Group Isomorphism, 2019. arXiv. `arXiv:1905.02518`.

6 Peter A. Brooksbank, Joshua Maglione, and James B. Wilson. TheTensor.Space. https://github.com/thetensor-space/.

7 Peter A. Brooksbank, Joshua Maglione, and James B. Wilson. A Fast Isomorphism Test for Groups whose Lie Algebra has Genus 2. *Journal of Algebra*, 473:545–590, 2017. doi:10.1016/j.jalgebra.2016.12.007.

8 Peter A. Brooksbank and Eamonn A. O'Brien. Constructing the Group Preserving a System of Forms. *International Journal of Algebra and Computation*, 18(02):227–241, 2008. doi:10.1142/S021819670800441X.

9 Peter A. Brooksbank and James B. Wilson. Computing Isometry Groups of Hermitian Maps. *Transactions of the American Mathematical Society*, 364(4):1975–1996, 2012. doi:10.2307/41524909.

10 Paul Erdős and Alfréd Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

11 Volkmar Felsch and Joachim Neubüser. On a Programme for the Determination of the Automorphism Group of a Finite Group. In *Computational Problems in Abstract Algebra*, pages 59–60. Pergamon, 1970. doi:10.1016/B978-0-08-012975-4.50011-4.

12 Joshua A. Grochow and Youming Qiao. Algorithms for Group Isomorphism via Group Extensions and Cohomology. *SIAM Journal on Computing*, 46(4):1153–1216, 2017. doi:10.1137/15M1009767.

13 Joshua A. Grochow and Youming Qiao. Isomorphism Problems for Tensors, Groups, and Cubic Forms: Completeness and Reductions, 2019. arXiv. arXiv:1907.00309.

14 Hermann Heineken and Hans Liebeck. The Occurrence of Finite Groups in the Automorphism Group of Nilpotent Groups of Class 2. *Archiv der Mathematik*, 25:8–16, 1974. doi:10.1007/BF01238631.

15 Harald Andrés Helfgott, Jitendra Bajpai, and Daniele Dona. Graph Isomorphisms in Quasi-polynomial Time, 2017. arXiv. arXiv:1710.04574.

16 Gábor Ivanyos, Marek Karpinski, and Nitin Saxena. Deterministic polynomial time algorithms for matrix completion problems. *SIAM Journal on Computing*, 39(8):3736–3751, 2010. doi:10.1137/090781231.

17 Gábor Ivanyos and Youming Qiao. Algorithms Based on $*$-Algebras, and Their Applications to Isomorphism of Polynomials with One Secret, Group Isomorphism, and Polynomial Identity Testing. *SIAM Journal on Computing*, 48(3):926–963, 2019. doi:10.1137/18M1165682.

18 Zhengfeng Ji, Youming Qiao, Fang Song, and Aaram Yun. General Linear Group Action on Tensors: A Candidate for Post-quantum Cryptography. In *Theory of Cryptography*, pages 251–281, 2019. doi:10.1007/978-3-030-36030-6_11.

19 Telikepalli Kavitha. Linear Time Algorithms for Abelian Group Isomorphism and Related Problems. *Journal of Computer and System Sciences*, 73(6):986–996, 2007. doi:10.1016/j.jcss.2007.03.013.

20 Mark L. Lewis and James B. Wilson. Isomorphism in Expanding Families of Indistinguishable Groups. *Groups Complexity Cryptology*, 4(1):73–110, 2012. doi:10.1515/gcc-2012-0008.

21 Yinan Li and Youming Qiao. Linear Algebraic Analogues of the Graph Isomorphism Problem and the Erdős-Rényi Model. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 463–474, 2017. arXiv:1708.04501v2. doi:10.1109/FOCS.2017.49.

22 Yinan Li and Youming Qiao. Group-theoretic Generalisations of Vertex and Edge Connectivities, 2019. arXiv. arXiv:1906.07948.

23 Ruvim Lipyanski and Natalia Vanetik. On Borel Complexity of the Isomorphism Problems for Graph related Classes of Lie Algebras and Finite p-groups. *Journal of Algebra and its Applications*, 14(5):1550078, 15, 2015. doi:10.1142/S0219498815500784.

24 Eugene M. Luks. Permutation Groups and Polynomial-time Computation. In *Groups and Computation*, volume 11 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, 1993.

25 Eugene M. Luks. Hypergraph Isomorphism and Structural Equivalence of Boolean Functions. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing STOC*, pages 652–658. ACM, 1999. doi:10.1145/301250.301427.

26 Gary L. Miller. On the $n \log n$ Isomorphism Technique (A Preliminary Report). In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, page 51–58, New York, NY, USA, 1978. Association for Computing Machinery. doi:10.1145/800133.804331.

27 David J. Rosenbaum. Bidirectional Collision Detection and Faster Deterministic Isomorphism Testing, 2013. arXiv. arXiv:1304.3935.

28 David J. Rosenbaum and Fabian Wagner. Beating the Generator-enumeration Bound for p-group Isomorphism. *Theoretical Computer Science*, 593:16–25, 2015. doi:10.1016/j.tcs.2015.05.036.

29 James B. Wilson. 2014 conference on *Groups, Computation, and Geometry* at Colorado State University, co-organized by P. Brooksbank, A. Hulpke, T. Penttila, J. Wilson, and W. Kantor. Personal communication, 2014.

Sometimes Reliable Spanners of Almost Linear Size

Kevin Buchin
Department of Mathematics and Computing Science, TU Eindhoven, The Netherlands
k.a.buchin@tue.nl

Sariel Har-Peled
Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA
sariel@illinois.edu

Dániel Oláh
Department of Mathematics and Computing Science, TU Eindhoven, The Netherlands
d.olah@tue.nl

Abstract

Reliable spanners can withstand huge failures, even when a linear number of vertices are deleted from the network. In case of failures, some of the remaining vertices of a reliable spanner may no longer admit the spanner property, but this collateral damage is bounded by a fraction of the size of the attack. It is known that $\Omega(n \log n)$ edges are needed to achieve this strong property, where n is the number of vertices in the network, even in one dimension. Constructions of reliable geometric $(1+\varepsilon)$-spanners, for n points in $\mathbb{R}^d$, are known, where the resulting graph has $\mathcal{O}(n \log n \operatorname{loglog}^6 n)$ edges.

Here, we show randomized constructions of smaller size spanners that have the desired reliability property in expectation or with good probability. The new construction is simple, and potentially practical – replacing a hierarchical usage of expanders (which renders the previous constructions impractical) by a simple skip list like construction. This results in a 1-spanner, on the line, that has linear number of edges. Using this, we present a construction of a reliable spanner in $\mathbb{R}^d$ with $\mathcal{O}(n \operatorname{loglog}^2 n \operatorname{logloglog} n)$ edges.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Geometric spanners, vertex failures, reliability

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.27

Funding *Sariel Har-Peled*: Work on this paper was partially supported by a NSF AF awards CCF-1421231 and CCF-1907400.
Dániel Oláh: Supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

1 Introduction

Geometric graphs are such that their vertices are points in the d-dimensional Euclidean space $\mathbb{R}^d$ and edges are straight line segments. The quality or efficiency of a geometric graph is often measured in terms of the ratio of shortest path distances and geometric distances between its vertices. Let $G = (P, E)$ be a geometric graph, where $P \subset \mathbb{R}^d$ is a set of n points and E is the set of edges. The shortest path distance between two points $p, q \in P$ in the graph G is denoted by $\mathsf{d}_G(p, q)$ (or just $\mathsf{d}(p, q)$). The graph G is a t-spanner for some constant $t \geq 1$, if $\mathsf{d}(p, q) \leq t \cdot \|p - q\|$ holds for all pairs of points $p, q \in P$, where $\|p - q\|$ stands for the Euclidean distance of p and q. The spanning ratio, stretch factor, or dilation of a graph G is the minimum number $t \geq 1$ for which G is a t-spanner. A path between p and q is a *t-path* if its length is at most $t \cdot \|p - q\|$.

Table 1.1 Comparison of the size of constructions of reliable spanners and reliable spanners in expectation. The reliability parameter is $\vartheta > 0$, and, for dimensions $d \geq 2$, the graphs are $(1+\varepsilon)$-spanners for $\varepsilon > 0$.

	dim	# edges	constants
Reliable spanners			
Buchin et al. [4]	$d = 1$	$\mathcal{O}(n \log n)$	$\mathcal{O}\left(\vartheta^{-6}\right)$
	$d \geq 2$	$\mathcal{O}\left(n \log n \log\log^6 n\right)$	$\mathcal{O}\left(\varepsilon^{-7d}\vartheta^{-6}\log^7 \varepsilon^{-1}\right)$
Bose et al. [2]	$d \geq 1$	$\mathcal{O}\left(n \log^2 n \log\log n\right)$	?
Reliable spanners *in expectation*			
New results	$d = 1$	$\mathcal{O}(n)$	$\mathcal{O}\left(\vartheta^{-1}\log \vartheta^{-1}\right)$
	$d \geq 2$	$\mathcal{O}(n \log\log^2 n \log\log\log n)$	$\mathcal{O}\left(\varepsilon^{-2d}\vartheta^{-1}\log^3 \varepsilon^{-1}\log \vartheta^{-1}\right)$

We focus our attention to construct spanners that can survive massive failures of vertices. The most studied notion is fault tolerance [5, 7, 8, 9, 10], which provides a properly functioning residual graph if there are no more failures than a predefined parameter k. It is clear, that a k-fault tolerant spanner must have $\Omega(kn)$ edges to avoid small degree nodes, which can be isolated by deleting their neighbors. Therefore, fault tolerant spanners must have quadratic size to be able to survive a failure of a constant fraction of vertices. Another notion is robustness [3], which gives more flexibility by allowing the loss of some additional nodes by not guaranteeing t-paths for them. For a function $f : \mathbb{N} \to \mathbb{R}^+$ a t-spanner G is f-robust, if for any set of failed points B there is an extended set B^+ with size at most $f(|B|)$ such that the residual graph $G \setminus B$ has a t-path for any pair of points $p, q \in P \setminus B^+$. The function f controls the robustness of the graph - the slower the function grows the more robust the graph is. The benefit of robustness is that a near linear number of edges are enough to achieve it, even for the case when f is linear, there are constructions with nearly $\mathcal{O}(n \log n)$ edges. For $\vartheta \in (0, 1)$, a spanner that is f-robust with $f(k) = (1 + \vartheta)k$ is a ϑ-*reliable* spanner [4]. This is the strongest form of robustness, since the dilation can increase beyond t only for a tiny additional fraction of points. The fraction is relative to the number of failed vertices and controlled by the parameter ϑ.

Recently, Buchin et al. [4] showed a construction of reliable 1-spanners of size $\mathcal{O}(n \log n)$ in one dimension, and of reliable $(1+\varepsilon)$-spanners of size $\mathcal{O}\left(n \log n \log\log^6 n\right)$ in higher dimensions (the constant in the $\mathcal{O}$ depends on the dimension, ε, and the reliability parameter). An alternative construction, with slightly worse bounds, was given by Bose et al. [2].

Limitations of previous constructions. The construction of Buchin et al. [4] (and also the construction of Bose et al. [2]) relies on using expanders to get a 1-spanner for points on the line, and then extending it to higher dimensions. The spanner (in one dimension) has $\mathcal{O}(n \log n)$ edges. Unfortunately, even in one dimension, such a reliable spanner requires $\Omega(n \log n)$ edges, as shown by Bose et al. [3]. Furthermore, the constants involved in these constructions [2, 4] are quite bad, because of the usage of expanders. See Table 1.1 for a summary of the sizes of different constructions (together with the new results).

The problem. As such, the question is whether one can come up with simple and practical constructions of spanners that have linear or near linear size, while still possessing some reliability guarantee – either in expectation or with good probability.

Some definitions. Given a graph G, an *attack* $B \subseteq V(G)$ is a set of vertices that are being removed. The *damaged set* B^+, is the set of all the vertices which are no longer connected to the rest the graph, or are badly connected to the rest of the graph – that is, these vertices no longer have the desired spanning property. The *loss* caused by B, is the quantity $|B^+ \setminus B|$ (where we take the minimal damaged set). The *loss rate* of B is $\lambda(G, B) = |B^+ \setminus B| / |B|$. A graph G is *ϑ-reliable* if for any attack B, the loss rate $\lambda(G, B)$ is at most ϑ.

Randomness and obliviousness. As mentioned above, reliable spanners must have size $\Omega(n \log n)$. A natural way to get a smaller spanner, is to consider randomized constructions, and require that the reliability holds in expectation (or with good probability). Randomized constructions are (usually) still sensitive to adversarial attacks, if the adversary is allowed to pick the attack set after the construction is completed (and it is allowed to inspect it). A natural way to deal with this issue is to restrict the attacks to be *oblivious* – that is, the attack set is chosen before the graph is constructed (or without any knowledge of the edges).

In such an oblivious model, the loss rate is a random variable (for a fixed attack B). It is thus natural to construct the graph G randomly, in such a way that $\mathbb{E}[\lambda(G, B)] \leq \vartheta$, or alternatively, that the probability $\mathbb{P}[\lambda(G, B) \geq \vartheta]$ is small.

1-spanner. Surprisingly, the one-dimensional problem is the key for building reliable spanners. Here, the graph G is constructed over the set of vertices $[n] = \{1, \ldots, n\}$. An attack is a subset $B \subseteq [n]$. Given an attack B, the requirement is that for all $i, j \in [n] \setminus B^+$, such that $i < j$, there is a monotonically increasing path from i to j in $G \setminus B$ – here, the length of the path between i and j is exactly $j - i$. Since there is no distortion in the length of the path, such graphs are 1-spanners.

Our results. We give a randomized construction of a 1-spanner in one dimension, that is ϑ-reliable in expectation, and has size $\mathcal{O}(n)$. Formally, the construction has the property that $\mathbb{E}[\lambda(G, B)] \leq \vartheta$. This construction can also be modified so that $\lambda(G, B) \leq \vartheta$ holds with some desired probability. This is the main technical contribution of this work.

Next, following in the footsteps of the construction of reliable spanners, we use the one-dimensional construction to get $(1 + \varepsilon)$-spanners that are ϑ-reliable either in expectation or with good probability. The new constructions have size roughly $\mathcal{O}\big(n \operatorname{loglog}^2 n\big)$.

Main idea. We borrow the notion of shadow from the work of Buchin et al. [4]. A point p is in the α-shadow if there is a neighborhood of p, such that an α-fraction of it belongs to the attack set. One can think about the maximum α such that p is in the α-shadow of B as the *depth* of p (here, the depth is in the range $[0, 1]$). A point with depth close to one, are intuitively surrounded by failed points, and have little hope of remaining well connected. Fortunately, only a few points have depth truly close to one 1. The flip side is that the attack has little impact on shallow points (i.e., points with depth close to 0). Similar to people, shallow points are surrounded by shallow points. As such, only a small fraction of the shallow points needs to be strongly connected to other points in the graph, as paths from (shallow) points around them can then travel via these hub points.

To this end, similar in spirit to skip lists, we define a random gradation of the points $P = P_0 \supseteq P_1 \supseteq \ldots \supseteq P_{\log n}$, where $|P_i| = n/2^i$ – this is done via a random tournament tree. In each level, each point of P_i is connected to all its neighbors within a certain distance (which increases as i increases). Intuitively, because of the improved connectivity, the probability that a point is well-connected (after the attack) increases if they belong to higher level of the

gradation. Thus, the probability of a shallow point to remain well connected is, intuitively, good. Specifically, we can quantify the probability of a vertex to lose its connectivity as a function of its depth. Combining this with bounds on the number of points of certain depths, results in bounds on the expected size of the damaged set.

Reliable skip lists. Our construction can be interpreted as a reliable construction of skip lists. Here, an attack removes certain cells in the skip list, which are no longer available. This can happen, for example, if the skip list is stored in a distributed fashion in a network, and certain nodes of the network are down. Our construction implies that one can withstand an attack with small expected loss. The previous work on skip graphs [1], or [4], presented constructions of variants of skip lists with somewhat similar properties, but using $\mathcal{O}(n \log n)$ pointers. The current construction requires only $\mathcal{O}(n)$ pointers.

Comparison to previous work. While we borrow some components of Buchin et al. [4], the basic scheme in the one-dimensional case, is new, and significantly different – the previous construction used expanders in a hierarchical way. The new construction requires different analysis and ideas. The extension to higher dimension is relatively straightforward and follows the ideas of Buchin et al. [4], although some modifications and care are necessary.

Paper organization. We review some necessary machinery in Section 2. The one-dimensional construction is described in Section 3. We describe the extension to higher dimensions in Section 4.

2 Preliminaries

Let $G = (P, E)$ be a t-spanner for some $t \geq 1$. An ***attack*** on G is a set of vertices B that fail, and no longer can be used. An attack is ***oblivious***, if the set B is picked without any knowledge of E.

▶ **Definition 1** (Reliable spanner)**.** *Let $G = (P, E)$ be a t-spanner for some $t \geq 1$ constructed by a (possibly) randomized algorithm. Given an attack B, its* **damaged set** *B^+ is a smallest set, such that for any pair of vertices $u, v \in P \setminus B^+$, we have*

$$\mathsf{d}_{G \setminus B}(u, v) \leq t \cdot \|u - v\|,$$

that is, t-paths are preserved for all pairs of points not contained in B^+. The quantity $|B^+ \setminus B|$ is the **loss** *of G under the attack B. The* **loss rate** *of G is $\lambda(G, B) = |B^+ \setminus B| / |B|$. For $\vartheta \in (0, 1)$, the graph G is* **ϑ-reliable** *if $\lambda(G, B) \leq \vartheta$ holds for any attack $B \subseteq P$.*

Further, we say that the random graph G is **ϑ-reliable in expectation** *if $\mathbb{E}[\lambda(G, B)] \leq \vartheta$ holds for any oblivious attack $B \subseteq P$. For $\vartheta, \rho \in (0, 1)$, we say that the graph G is* **ϑ-reliable with probability** *$1 - \rho$ if $\mathbb{P}[\lambda(G, B) \leq \vartheta] \geq 1 - \rho$ holds for any oblivious attack $B \subseteq P$.*

▶ Remark 2. We emphasize that in the latter case the graph is random and the expectation and the probability is taken with respect to the distribution of graphs.

Another remark is that the set B^+ is not unique, since one can (possibly) choose the point to include in B^+ for a pair that does not have a t-path in $G \setminus B$. However, this does not cause a problem in defining the loss rate.

▶ **Definition 3.** *Let $[n]$ denote the* interval *$\{1, \ldots, n\}$. Similarly, for x and y, let $[x \ldots y]$ denote the interval $\{x, x + 1, \ldots, y\}$.*

We use the shadow notion as it was introduced by Buchin et al. [4].

▶ **Definition 4.** *Consider an arbitrary set $B \subseteq [n]$ and a parameter $\alpha \in (0,1)$. A number i is in the* **left α-shadow** *of B, if and only if there exists an integer $j \geq i$, such that $\left|[i \ldots j] \cap B\right| \geq \alpha \left|[i \ldots j]\right|$. Similarly, i is in the* **right α-shadow** *of B, if and only if there exists an integer h, such that $h \leq i$ and $|[h \ldots i] \cap B| \geq \alpha |[h \ldots i]|$. The left and right α-shadow of B is denoted by $\mathcal{S}_{\rightarrow}(\alpha, B)$ and $\mathcal{S}_{\leftarrow}(\alpha, B)$, respectively. The combined shadow is denoted by $\mathcal{S}(\alpha, B) = \mathcal{S}_{\rightarrow}(\alpha, B) \cup \mathcal{S}_{\leftarrow}(\alpha, B)$.*

▶ **Lemma 5** ([4]). *For any set $B \subseteq [n]$, and $\alpha \in (0,1)$, we have that $|\mathcal{S}(\alpha, B)| \leq \left(1 + 2\lceil 1/\alpha \rceil\right)|B|$. Further, if $\alpha \in (2/3, 1)$, we have that $|\mathcal{S}(\alpha, B)| \leq |B| / (2\alpha - 1)$.*

▶ **Definition 6.** *Given a graph G over $[n]$, a* **monotone path** *between $i, j \in [n]$, such that $i < j$, is a sequence of vertices $i = i_1 < i_2 < \cdots < i_k = j$, such that $i_{\ell-1} i_\ell \in E(G)$, for $\ell = 2, \ldots, k$.*

A monotone path between i and j has length $|j - i|$. Throughout the paper we use $\log x$ and $\ln x$ to denote the base 2 and natural base logarithm of x, respectively. For any set $A \subseteq P$, let $A^c = P \setminus A$ denote the complement of A. For two integers $x, y > 0$, let $x_{\uparrow y} = \lceil x/y \rceil \, y$.

3 Reliable spanners in one dimension

We show how to build a random graph on $[n]$ that still has monotone paths almost for all vertices that are not directly attacked. First, in Section 3.2, we show that our construction is ϑ-reliable in expectation. Then, in Section 3.3, we show how to modify the construction to obtain a 1-spanner that is ϑ-reliable with probability $1 - \rho$.

3.1 Construction

The input consists of a parameter $\vartheta > 0$ and the point set $P = [n] = \{1, \ldots, n\}$. The backbone of the construction is a random elimination tournament. We assume that n is a power of 2 as otherwise one can construct the graph for the next power of two, and then throw away the unneeded vertices.

The tournament is a full binary tree, with the leafs storing the values from 1 to n, say from left to right. The value of a node is computed randomly and recursively. For a node, once the values of the nodes were computed for both children, it randomly copies the value of one of its children, with equal probability to choose either child. Let P_i be the values stored in the ith bottom level of the tree. As such, $P_0 = P$, and $P_{\log n}$ is a singleton. Each set P_i can be interpreted as an ordered set (from left to right, or equivalently, by value).

Let

$$\alpha = 1 - \frac{\vartheta}{8} \qquad \text{and} \qquad \varepsilon = \frac{8(1-\alpha)}{c \ln \vartheta^{-1}} = \frac{\vartheta}{c \ln \vartheta^{-1}}, \tag{3.1}$$

where $c > 1$ is a sufficiently large constant. Let M be the smallest integer for which $|P_M| \leq 2^{M/2}/\varepsilon$ holds (i.e., $M = \lceil (2/3) \log(\varepsilon n) \rceil$). For $i = 0, 1, \ldots, M$, and for all $p \in P_i$ connect p with the first

$$\ell(i) = \left\lceil \frac{2^{i/2}}{\varepsilon} \right\rceil \tag{3.2}$$

successors (and hence predecessors) of p in P_i. Let E_i be the set of all edges in level i. The graph G on P is defined as the union of all edges over all levels – that is, $E(G) = \cup_{i=0}^{M} E_i$. Note, that top level of the graph G is a clique.

▶ Remark 7. Before dwelling on the correctness of the construction, note that the obliviousness of the attack is critical. Indeed, it is quite easy to design an attack if the structure of G is known. To this end, let B_i be the set of $\ell(M) = O(n^{1/3}/\varepsilon)$ values of P_i closest to $n/2$ – namely, we are taking out the middle-part of the graph, that belongs to the ith level. Consider the attack $B = \cup B_i$. It is easy to verify that this attack breaks G into at least two disconnected graphs, each of size at least $n/2 - O(n^{1/3}\varepsilon^{-1}\log n)$.

3.2 Analysis

▶ **Lemma 8.** *The graph G has $\mathcal{O}\left(n\vartheta^{-1}\log\vartheta^{-1}\right)$ edges.*

Proof. The number of edges contributed by a point in P_i is at most $\ell(i)$ at level i, and $|P_i| = n/2^i$. Thus, we have

$$|E(G)| \leq \sum_{i=0}^{M} |P_i| \cdot \ell(i) \leq \sum_{i=0}^{M} \frac{n}{2^i} \cdot \left\lceil \frac{2^{i/2}}{\varepsilon} \right\rceil \leq \sum_{i=0}^{M} \frac{n}{2^i} \cdot \frac{2 \cdot 2^{i/2}}{\varepsilon} \leq \frac{n}{\varepsilon} \cdot \sum_{i=0}^{\infty} \frac{2}{2^{i/2}} = \mathcal{O}\left(\frac{n}{\varepsilon}\right). \quad \blacktriangleleft$$

Fix an attack $B \subseteq P$. The high-level idea is to show that if a point $p \in P \setminus B$ is far enough from the faulty set, then, with high probability, there exist monotone paths reaching far from p in both directions. For two points $p < q$, we show that if both p and q have far reaching monotone paths, then the path going to the right from p, and the path going to the left from q must cross each other, which in turn implies, that there is a monotone path between p and q. Therefore, it is enough to bound the number of points that does not have far reaching monotone paths.

▶ **Definition 9** (Stairway)**.** *Let $p \in P$ be an arbitrary point. The path $p = p_0, p_1, \ldots, p_j$ is a right (resp., left)* **stairway** *of p to level j, if*

- **(i)** *$p = p_0 \leq p_1 \leq \cdots \leq p_j$ (resp., $p \geq p_1 \geq \cdots \geq p_j$),*
- **(ii)** *if $p_i \neq p_{i+1}$, then $p_i p_{i+1} \in E_i$, for $i = 0, 1, \ldots, j-1$,*
- **(iii)** *$p_i \in P_i$, for $i = 1, \ldots, j$.*

Furthermore, a stairway is **safe** *if none of its points are in the attack set B. A right (resp., left) stairway is* **usable**, *if $[p_j \ldots n] \cap P_j$ (resp., $[1 \ldots p_j] \cap P_j$) forms a clique in G. Let $T \subseteq P$ denote the set of points that have a safe and usable stairway to both directions.*

Let $\alpha_k = \alpha/2^k$, for $k = 0, 1, \ldots, \log n$. Let $\mathcal{S}_k = \mathcal{S}(\alpha_k, B)$ be the α_k-shadow of B, for $k = 0, 1, \ldots, \log n$. Observe that $\mathcal{S}_0 \subseteq \mathcal{S}_1 \subseteq \cdots \subseteq \mathcal{S}_{\log n}$, and there is an index j such that $\mathcal{S}_j = P$, if $B \neq \emptyset$. A point is classified according to when it gets "buried" in the shadow. A point p, for $k \geq 1$, is a ***k*th round** point, if $p \in \mathcal{S}_k \setminus \mathcal{S}_{k-1}$. Intuitively, a kth round point is more likely to have a safe stairway the larger the value of k is.

▶ **Definition 10.** *A point is* **bad** *if it belongs to B, or it does not have a right or left stairway that is safe and usable. Formally, a point $p \in P$ is bad, if and only if $p \in P \setminus T$.*

▶ **Lemma 11.** *For any two points $p, q \in T$ that are not bad, there is a monotone path connecting p and q in the residual graph $G \setminus B$.*

Proof. Suppose we have $p < q$. Let $(p, p_1, \ldots, p_{j(p)})$ be a safe usable right stairway starting from p and $(q, q_1, \ldots, q_{j(q)})$ be a safe usable left stairway from q. These stairways exist, since $p, q \in T$. Let $j = \min(j(p), j(q))$ and consider the stairways $(p, p_1, \ldots, p_j)$ and $(q, q_1, \ldots, q_j)$. Notice that both are safe and at least one of them is usable.

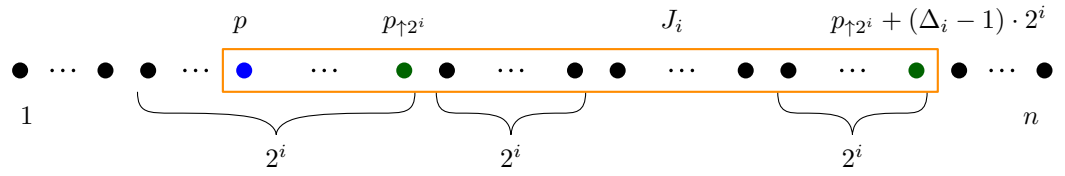


Figure 3.1 The interval $J_i = \left[p \dots p_{\uparrow 2^i} + (\Delta_i - 1) \cdot 2^i\right]$.

Let i be the first index such that $p_i \geq q_i$, if there is any. We distinguish two cases based on whether $p_i < q_{i-1}$ holds or not. In case $p_i < q_{i-1}$, the path $(p, p_1, \dots, p_{i-1}, p_i, q_{i-1}, \dots, q_1, q)$ is a monotone path from p to q, since $q_i q_{i-1} \in E_{i-1}$ implies $p_i q_{i-1} \in E_{i-1}$. On the other hand, if we have $p_i \geq q_{i-1}$, the path $(p, p_1, \dots, p_{i-1}, q_{i-1}, \dots, q_1, q)$ is a monotone path between p and q, since $p_{i-1} p_i \in E_{i-1}$ implies $p_{i-1} q_{i-1} \in E_{i-1}$.

Finally, if $p_i < q_i$ holds for all $i = 1, \dots, j$, then the path $(p, p_1, \dots, p_j, q_j, \dots, q_1, q)$ is a monotone path between p and q. We have $p_j q_j \in E_j$, since at least one of the stairways is usable. This concludes the proof that there is a monotone path from p to q. ◀

▶ **Lemma 12.** *For a fixed set $Q \subseteq [n]$, we have that $\mathbb{P}[Q \cap P_i = \emptyset] \leq \exp(-|Q|/2^i)$.*

Proof. Let $Q = \{q_1, \dots, q_r\}$, and observe that knowing that certain points of Q are not in P_i, increases the probability of another point to be in P_i. That is, $\mathbb{P}[q_j \in P_i \mid q_1, \dots, q_{j-1} \notin P_i] \geq \mathbb{P}[q_j \in P_i] = 1/2^i$. As such, we have

$$\begin{aligned}\mathbb{P}\left[Q \cap P_i = \emptyset\right] &= \mathbb{P}\left[\bigcap_j (q_j \notin P_i)\right] = \prod_{j=1}^{r} \mathbb{P}[q_j \notin P_i \mid q_1, \dots, q_{j-1} \notin P_j] \\ &\leq \left(1 - 1/2^i\right)^r \leq \exp(-r/2^i).\end{aligned}$$

◀

▶ **Lemma 13.** *Assume that $\vartheta \in (0, 1/2)$ and let $p \in \mathcal{S}_k \setminus \mathcal{S}_{k-1}$ be a kth round point for some $k \geq 1$. The probability that p is bad is at most $(\vartheta/2)^k/32$.*

Proof. For any integer $i \geq 1$, let $\Delta_i = \left\lfloor 2^{(i-1)/2}/(2\varepsilon) \right\rfloor$ and let $J_i = \left[p \dots p_{\uparrow 2^i} + (\Delta_i - 1) \cdot 2^i\right]$, see Figure 3.1. Recall that $p \in [n]$, so $p_{\uparrow 2^i} = \left\lceil p/2^i \right\rceil 2^i$ is the next multiple of 2^i. Let ξ be the largest integer such that $J_\xi \subseteq P$. For $i = 0, \dots, \xi$, the points of $J_{i+1} \cap P_i$ form a clique in G, since

$$|J_{i+1} \cap P_i| \leq \left\lceil |J_{i+1}|/2^i \right\rceil \leq \left\lceil 2^{i+1} \Delta_{i+1}/2^i \right\rceil = 2\Delta_{i+1} \leq \left\lceil 2^{i/2}/\varepsilon \right\rceil = \ell(i).$$

Indeed, any two vertices of P_i with distance at most $\ell(i)$ are connected by an edge of E_i. As such, it is enough to prove that there is a right safe stairway from p, that climbs on the levels to level ξ. Since $J_{\xi+1} \cap P_\xi$ forms a clique, it follows that such a stairway would be usable.

Let $\mathcal{E}_i$ be the event that $(J_i \setminus B) \cap P_i$ is empty, for $i = 1, \dots, \xi$. Since $p \notin \mathcal{S}_{k-1}$, we have that $|J_i \cap B| < \alpha_{k-1} |J_i| \leq 2^i \alpha_{k-1} \Delta_i$. On the other hand, we have $|J_i \cap P_i| \geq 2^i(\Delta_i - 1)/2^i = \Delta_i - 1$. As such, if $|J_i \cap B| < |J_i \cap P_i|$ then $\mathbb{P}[\mathcal{E}_i] = 0$. This happens if $2^i \alpha_{k-1} \Delta_i \leq \Delta_i - 1 \iff 2^{i-k+1} \alpha \leq (\Delta_i - 1)/\Delta_i$, which happens if $i \leq k - 2$, given that $\Delta_i \geq 2$. Notice that $\Delta_i \geq 2$ holds for all $i \geq 1$, if $\varepsilon \leq \frac{1}{4}$.

So assume that $i \geq k - 1$. Let $q_1, \dots, q_r$ be all points of $J_i \setminus B$, which are the possible candidates to be contained in $(J_i \setminus B) \cap P_i$. By Eq. (3.1), there are at least

$$r = |J_i| - |J_i \cap B| \geq (1 - \alpha_{k-1}) |J_i| \geq (1 - \alpha_{k-1}) 2^i (\Delta_i - 1)$$

$$\geq (1-\alpha_{k-1})2^i\Big(\frac{2^{(i-1)/2}}{2\varepsilon}-2\Big) = \frac{c(1-\alpha_{k-1})\ln\vartheta^{-1}}{16(1-\alpha)}2^{3i/2-1/2} - (1-\alpha_{k-1})2^{i+1}$$
$$\geq c2^{3i/2-9/2}\ln\vartheta^{-1} - 2^{i+1}$$

such points. Observe, that by the structure of the construction, a point is more likely to be contained in P_i conditioned on the event there are some other points which are not contained in P_i. Therefore, by Lemma 12, we have $\mathbb{P}\big[\mathcal{E}_i\big] \leq \exp\big(-r/2^i\big) \leq \tau_i$, for $\tau_i = \exp\big(2 - c2^{i/2-9/2}\ln\vartheta^{-1}\big)$. The sequence τ_i has a fast decay in i, since

$$\frac{\tau_{i+1}}{\tau_i} = \exp\Big(-(\sqrt{2}-1)c2^{i/2-9/2}\ln\vartheta^{-1}\Big) \leq \exp\big(-c2^{-6}\ln 2\big) = 2^{-c2^{-6}} \leq \frac{1}{2},$$

if $c \geq 2^6$ holds. Thus, we have

$$\mathbb{P}\Big[\cup_{i=1}^{\xi}\mathcal{E}_i\Big] \leq \sum_{i=1}^{\xi}\mathbb{P}\big[\mathcal{E}_i\big] \leq \sum_{i=k-1}^{\xi}\tau_i \leq 2\tau_{k-1} = 2\exp\Big(2 - c2^{(k-1)/2-9/2}\ln\vartheta^{-1}\Big)$$
$$\leq 16\exp\Big(-\frac{c}{32}2^{k/2}\ln\vartheta^{-1}\Big) = 16\cdot\vartheta^{\frac{c}{32}\cdot 2^{k/2}} \leq 2^4\cdot\vartheta^{\frac{c}{2^6}\cdot k}$$
$$\leq 2^4\cdot\left(\frac{1}{2}\right)^{\frac{c}{2^7}\cdot k}\cdot\left(\vartheta^{\frac{c}{2^7}}\right)^k \leq \frac{(\vartheta/2)^k}{64}$$

for $c \geq 2^{11}$, using the conditions $0 < \vartheta \leq \frac{1}{2}$, $k \geq 1$ and the fact that $x \leq 2^x$.

Let p_i be the leftmost point in $(J_i \setminus B) \cap P_i$, for $i \geq 0$. Since $P_i \subseteq P_{i-1}$, for all i, it follows that $p = p_0 \leq p_1 \leq \cdots \leq p_\xi$. Furthermore, since $J_{i+1} \cap P_i$ is a clique in level i of G, and $p_i, p_{i+1} \in J_{i+1} \cap P_i$, it follows that $p_ip_{i+1} \in E_i$, if $p_i \neq p_{i+1}$, for all i. We conclude that $p, p_1, \ldots, p_\xi$ is a safe and usable right stairway in G.

The bound now follows by applying the same argument symmetrically for the left stairway. Indeed, using the union bound, we obtain $\mathbb{P}[p \text{ is bad}] \leq 2(\vartheta/2)^k/64 = (\vartheta/2)^k/32$. ◀

▶ **Lemma 14.** *Let $\vartheta \in (0, 1/2)$ and $B \subseteq P$ be an oblivious attack. Recall, that T^c is the set of bad points. Then, we have $\mathbb{E}[|T^c|] \leq (1+\vartheta)\,|B|$.*

Proof. We may assume that all the points of $\mathcal{S}_0$ are bad. Fortunately, by Lemma 5, we have $|\mathcal{S}_0| \leq |B|/(2\alpha-1) = |B|/(1-\vartheta/4) \leq (1+\vartheta/2)\,|B|$, since $\alpha = 1-\vartheta/8$ and $1/(1-x/4) \leq 1+x/2$ for $0 \leq x \leq 2$. Again, using Lemma 5, we have

$$|\mathcal{S}_k \setminus \mathcal{S}_{k-1}| \leq |\mathcal{S}_k| \leq \big(1 + 2\lceil 2^k/\alpha\rceil\big)\,|B| \leq \left(3 + \frac{2^{k+1}}{\alpha}\right)|B| \leq 2^{k+3}\,|B|\,.$$

For $k \geq 1$, we have, by Lemma 13, that

$$b_k = \mathbb{E}[|(\mathcal{S}_k \setminus \mathcal{S}_{k-1}) \cap T^c|] \leq \sum_{p\in\mathcal{S}_k\setminus\mathcal{S}_{k-1}}\mathbb{P}[p \text{ is bad}] \leq 2^{k+3}\,|B|\cdot\frac{(\vartheta/2)^k}{32} \leq \frac{\vartheta^k}{4}\,|B|\,.$$

Since, $T^c = (\mathcal{S}_0 \cap T^c) \cup \bigcup_{k\geq 1}\big[(\mathcal{S}_k \setminus \mathcal{S}_{k-1}) \cap T^c\big]$, we have, by linearity of expectation, that

$$\frac{\mathbb{E}\Big[|T^c|\Big]}{|B|} \leq \frac{1}{|B|}\left(|\mathcal{S}_0| + \sum_{k=1}^{\infty}b_k\right) \leq 1 + \frac{\vartheta}{2} + \sum_{k=1}^{\infty}\frac{\vartheta^k}{4} \leq 1 + \frac{\vartheta}{2} + \frac{\vartheta}{4(1-\vartheta)} \leq (1+\vartheta),$$

since $\vartheta < 1/2$. ◀

▶ **Theorem 15.** *Let $\vartheta \in (0, 1/2)$ and $P = [n]$ be fixed. The graph G, constructed in Section 3.1, has $\mathcal{O}\big(n\vartheta^{-1}\log\vartheta^{-1}\big)$ edges, and it is a ϑ-reliable 1-spanner of P in expectation. Formally, for any oblivious attack B, we have $\mathbb{E}[\lambda(G, B)] \leq \vartheta$.*

Proof. By Lemma 8 the size of the construction is $|E(G)| = \mathcal{O}\big(n\vartheta^{-1}\log\vartheta^{-1}\big)$. Let $B \subseteq P$ be an oblivious attack and consider the bad set $P \setminus T$. By Lemma 11, for any two points outside the bad set, there is a monotone path connecting them. Further, by Lemma 14, we have $\mathbb{E}[|P \setminus T|] \leq (1 + \vartheta)\,|B|$ for any oblivious attack. Therefore, we obtain $\mathbb{E}[\lambda(G, B)] \leq \mathbb{E}[|T^c \setminus B| \,/\, |B|] \leq \vartheta$. ◀

3.3 Probabilistic bound

One can replace the guarantee, in Theorem 15, on the bound of the loss rate (which holds in expectation), by an upper bound that holds with probability at least $1 - \rho$, for some prespecified $\rho > 0$. A straightforward application of Markov's inequality implies that taking the union of $\log \rho^{-1}$ independent copies (G') of the construction of Theorem 15 with parameter $\vartheta/2$, results in a graph with the desired property. Indeed, we have

$$\mathbb{P}[\lambda(G, B) > \vartheta] \leq \mathbb{P}[\lambda(G', B) > \vartheta]^{\log \rho^{-1}} \leq \left(\frac{\mathbb{E}[\lambda(G', B)]}{\vartheta}\right)^{\log \rho^{-1}} \leq \left(\frac{1}{2}\right)^{\log \rho^{-1}} = \rho.$$

Here we show how one can do better to avoid the multiplicative factor $\log \rho^{-1}$.

Construction. The input consists of two parameters $\vartheta, \rho > 0$ and the set $P = [n]$. Let G be the graph constructed in Section 3.1 with parameters

$$\alpha = 1 - \frac{\vartheta}{8} \qquad \text{and} \qquad \varepsilon = \frac{8(1-\alpha)}{c(\ln\vartheta^{-1} + \ln\rho^{-1})} = \frac{\vartheta}{c(\ln\vartheta^{-1} + \ln\rho^{-1})},$$

where $c > 1$ is a sufficiently large constant. First, we need a variant of Lemma 13 to bound the probability of a kth round point being bad, using the new value of ε.

▶ **Lemma 16.** *Assume that $\vartheta \in (0, 1/2)$, $\rho \in (0, 1)$ and let $p \in \mathcal{S}_k \setminus \mathcal{S}_{k-1}$ be a kth round point for some $k \geq 1$. The probability that p is bad is at most $\vartheta \cdot \rho/2^{3k+4}$.*

Proof. The proof is the same as the proof of Lemma 13. The only difference is due to the new value of ε, which results in $\tau_i = \exp\big(2 - c2^{i/2-9/2}(\ln\vartheta^{-1} + \ln\rho^{-1})\big)$, using the same notation. Therefore, we have

$$\begin{aligned}
\mathbb{P}[p \in \mathcal{S}_k \setminus \mathcal{S}_{k-1} \text{ is bad}] &\leq 4\tau_{k-1} = 4\exp\Big(2 - c2^{k/2-5}(\ln\vartheta^{-1} + \ln\rho^{-1})\Big) \\
&\leq 2^5 \exp\Big(-\frac{c}{2^6}k(\ln\vartheta^{-1} + \ln\rho^{-1})\Big) = 2^5 \cdot \vartheta^{\frac{c}{2^6}k} \cdot \rho^{\frac{c}{2^6}k} \\
&\leq 2^5 \cdot \left(\frac{1}{2}\right)^{\frac{c}{2^6}k-1} \cdot \vartheta \cdot \rho = 2^{-\frac{c}{2^6}k+6} \cdot \vartheta \cdot \rho \leq 2^{-3k-4} \cdot \vartheta \cdot \rho,
\end{aligned}$$

for $c \geq 2^{10}$. See Lemma 13 for a complete proof. ◀

▶ **Lemma 17.** *Let $\vartheta \in (0, 1/2)$, $\rho \in (0, 1)$ be fixed and $B \subseteq P$ be an oblivious attack. Then, with probability $\geq 1 - \rho$, the number of bad points is at most $(1 + \vartheta)\,|B|$. That is, we have $\mathbb{P}[|T^c| \leq (1 + \vartheta)\,|B|] \geq 1 - \rho$.*

Proof. The idea is to give bounds on the number of bad kth round points for all $k \geq 1$. Let $\mathcal{E}_k$ be the event that $|(\mathcal{S}_k \setminus \mathcal{S}_{k-1}) \cap T^c| > \frac{\vartheta}{2^{k+1}} |B|$ happens, for $k \geq 1$. Recall, by the choice of α, we have $|\mathcal{S}_0 \cap T^c| \leq |\mathcal{S}_0| \leq \left(1 + \frac{\vartheta}{2}\right) |B|$. Notice, that at least one of the events $\mathcal{E}_k$ must happen, for $k \geq 1$, in order to have $|T^c| > (1+\vartheta)|B|$, since

$$|T^c| = |\mathcal{S}_0 \cap T^c| + \sum_{k=1}^{\infty} |(\mathcal{S}_k \setminus \mathcal{S}_{k-1}) \cap T^c| \leq \left(1 + \frac{\vartheta}{2}\right) |B| + \sum_{k=1}^{\infty} \frac{\vartheta}{2^{k+1}} |B| = (1+\vartheta)|B|.$$

Using Markov's inequality and Lemma 16 we get

$$\mathbb{P}[\mathcal{E}_k] \leq \frac{\mathbb{E}[|(\mathcal{S}_k \setminus \mathcal{S}_{k-1}) \cap T^c|]}{\frac{\vartheta}{2^{k+1}}|B|} \leq \frac{|\mathcal{S}_k| \cdot \mathbb{P}[p \in \mathcal{S}_k \setminus \mathcal{S}_{k-1} \text{ is bad}]}{\frac{\vartheta}{2^{k+1}}|B|} \leq \frac{2^{k+3}|B| \cdot \frac{\vartheta \cdot \rho}{2^{3k+4}}}{\frac{\vartheta}{2^{k+1}}|B|} = \frac{\rho}{2^k}.$$

Therefore, we obtain

$$\mathbb{P}[|T^c| > (1+\vartheta)|B|] \leq \mathbb{P}[\cup_{k \geq 1} \mathcal{E}_k] \leq \sum_{k=1}^{\infty} \mathbb{P}[\mathcal{E}_k] \leq \sum_{k=1}^{\infty} \frac{\rho}{2^k} \leq \rho,$$

which is equivalent to $\mathbb{P}[|T^c| \leq (1+\vartheta)|B|] \geq 1 - \rho$. ◀

▶ **Theorem 18.** *Let $\vartheta \in (0, 1/2)$, $\rho \in (0,1)$ and $P = [n]$ be fixed. The graph G, constructed above, is a ϑ-reliable 1-spanner of P, with probability at least $1 - \rho$. Formally, we have $\mathbb{P}[\lambda(G, B) \leq \vartheta] \geq 1 - \rho$ for any oblivious attack B. Furthermore, the graph G has $\mathcal{O}\big(n\vartheta^{-1}(\log \vartheta^{-1} + \log \rho^{-1})\big)$ edges.*

Proof. The bound on the size follows directly from Lemma 8. Let $B \subseteq P$ be an oblivious attack and consider the bad set $P \setminus T$. By Lemma 11, for any two points outside the bad set, there is a monotone path connecting them. Further, by Lemma 17, we have $\mathbb{P}[\lambda(G, B) \leq \vartheta] \geq \mathbb{P}[|T^c| \leq (1+\vartheta)|B|] \geq 1 - \rho$ for any oblivious attack. ◀

4 Reliable spanners in higher dimensions

Now we turn to the higher-dimensional setting, and show that one can construct spanners with near linear size that are reliable in expectation or with some fixed probability (which can be provided as part of the input). We use the same technique as Buchin et al. [4], that is, we use our one-dimensional construction as a black box in combination with a result of Chan et al. [6]. Let the dimension $d > 1$ be fixed. In the following we assume $P \subset [0,1)^d$, which can be achieved by an appropriate scaling and translation of the d-dimensional Euclidean space $\mathbb{R}^d$. For an ordering σ of $[0,1)^d$, and two points $p, q \in [0,1)^d$, such that $p \prec q$, let $(p,q)_\sigma = \left\{z \in [0,1)^d \;\middle|\; p \prec z \prec q\right\}$ be the set of points between p and q in the order σ.

▶ **Theorem 19** ([6])**.** *For $\varsigma \in (0,1)$, there is a set $\Pi^+(\varsigma)$ of $M(\varsigma) = \mathcal{O}(\varsigma^{-d} \log \varsigma^{-1})$ orderings of $[0,1)^d$, such that for any two (distinct) points $p, q \in [0,1)^d$, with $\ell = \|p - q\|$, there is an ordering $\sigma \in \Pi^+$, and a point $z \in [0,1)^d$, such that*

- **(i)** $p \prec_\sigma q$,
- **(ii)** $(p,z)_\sigma \subseteq \mathrm{ball}\big(p, \varsigma\ell\big)$,
- **(iii)** $(z,q)_\sigma \subseteq \mathrm{ball}\big(q, \varsigma\ell\big)$, *and*
- **(iv)** $z \in \mathrm{ball}\big(p, \varsigma\ell\big)$ *or* $z \in \mathrm{ball}\big(q, \varsigma\ell\big)$.

Furthermore, given such an ordering σ, and two points p, q, one can compute their ordering, according to σ, using $\mathcal{O}(d \log \varsigma^{-1})$ arithmetic and bitwise-logical operations.

The above theorem ensures that it is enough to maintain only a "few" linear orderings, and for any pair of points $p, q \in P$ there exists an ordering where all points that lie between p and q are either very close to p or q. It is natural to build the one-dimensional construction for each of these orderings with some carefully chosen parameter. Then, since there is a reliable path in the one-dimensional construction, there is an edge $p'q'$ along the path between p and q that connects the locality of p and the locality of q. We fix the edge $p'q'$ and apply recursion on the subpaths from p to p' and q to q'.

4.1 Construction

Let $\vartheta, \varepsilon \in (0,1)$ be fixed parameters and $P \subseteq [0,1)^d$ be a set of n points. Set $\varsigma = \varepsilon/16$ in Theorem 19 and let $\Pi^+ = \Pi^+(\varsigma)$ be the set of $M = M(\varsigma)$ orderings that fulfills the conditions of the theorem. We define $\vartheta' = \frac{\vartheta}{3MN}$, where $N = \lceil \log\log n \rceil$. Now, for each ordering $\sigma \in \Pi^+$, we build N independent spanners $G^1_\sigma, \ldots, G^N_\sigma$, using the construction in Section 3.1 with parameter ϑ'. The (random) graph G is defined as the union of graphs G^i_σ for all $\sigma \in \Pi^+$ and $i \in [N]$, that is, $E(G) = \cup_{\sigma\in\Pi^+, i\in[N]} E\big(G^i_\sigma\big)$.

4.2 Analysis

▶ **Lemma 20.** *The graph G, constructed above, has $\mathcal{O}\big(c\, n \log\log^2 n \log\log\log n\big)$ edges, where the $\mathcal{O}$ hides constant that depends on the dimension d, and $c = \mathcal{O}(\varepsilon^{-2d}\vartheta^{-1}\log^3\varepsilon^{-1}\log\vartheta^{-1})$.*

Proof. There are $M = \mathcal{O}\big(\varepsilon^{-d}\log\varepsilon^{-1}\big)$ orderings, and for each ordering there are N copies, for which we build the one-dimensional construction with parameter ϑ'. The size of the one-dimensional construction is $\mathcal{O}\big(n\cdot\vartheta'^{-1}\cdot\log\vartheta'^{-1}\big)$, by Lemma 8. Therefore, G has size

$$\begin{aligned}
|E(G)| = \big|\cup_{\sigma\in\Pi^+, i\in[N]} E\big(G^i_\sigma\big)\big| &\le \sum_{\sigma\in\Pi^+, i\in[N]} |E(G'_\sigma)| \le NM\cdot\mathcal{O}\big(n\cdot\vartheta'^{-1}\cdot\log\vartheta'^{-1}\big)\\
&= \mathcal{O}\big(n\cdot N^2M^2\vartheta^{-1}\cdot\big(\log\vartheta^{-1}+\log N+\log M\big)\big)\\
&= \mathcal{O}\big(n\cdot\log\log^2 n\cdot\varepsilon^{-2d}\log^2\varepsilon^{-1}\cdot\vartheta^{-1}\cdot(\log\vartheta^{-1}+\\
&\qquad + \log\log\log n + d\log\varepsilon^{-1} + \log\log\varepsilon^{-1})\big)\\
&= \mathcal{O}\big(c\, n\log\log^2 n\log\log\log n\big), \qquad \text{where } c = \mathcal{O}(\varepsilon^{-2d}\vartheta^{-1}\log^3\varepsilon^{-1}\log\vartheta^{-1}).
\end{aligned}$$

◀

Fix an attack set $B \subseteq P$. In order to bound $\lambda(G, B)$ in expectation, we define a sequence of sets $B_0 \subseteq B_1 \subseteq \cdots \subseteq B_N$ as follows. First, we set $B_0 = B$. Then, for $i = 1, \ldots, N$, we define B^σ_i for each $\sigma \in \Pi^+$ to contain all points that do not have a right or left stairway in G^i_σ that is safe and usable with respect to B_{i-1}, that is, B^σ_i contains the bad points with respect to B_{i-1}. We set $B_i = \cup_{\sigma\in\Pi^+} B^\sigma_i$. Our goal is to show that the expected size of B_N is small, and there is a $(1+\varepsilon)$-path for all pairs of points outside of B_N.

▶ **Lemma 21.** *Let B be an oblivious attack and let $B_0 \subseteq B_1 \subseteq \cdots \subseteq B_N$ be the sequence defined above. Then, for $i = 1, \ldots, N$, we have $\mathbb{E}[|B^\sigma_i| \mid B_{i-1}] \le (1+\vartheta')\,|B_{i-1}|$, for all $\sigma \in \Pi^+$.*

Proof. The set B_{i-1} has information only about graphs G^j_σ for $j \le i-1$. Thus, the attack B_{i-1} on the graph G^i_σ is oblivious and we have $\mathbb{E}[|B^\sigma_i| \mid B_{i-1}] \le (1+\vartheta')\,|B_{i-1}|$ by Lemma 14. ◀

▶ **Lemma 22.** *Let B_N be the set defined above. For any oblivious attack B, the expected size of B_N is at most $(1+\vartheta)\cdot|B|$.*

Proof. By Lemma 21 we have $\mathbb{E}[|B_i^\sigma| \mid B_{i-1}] \leq (1+\vartheta')\,|B_{i-1}|$ for all $\sigma \in \Pi^+$. Therefore,

$$\mathbb{E}[|B_i| \mid B_{i-1}] \leq \big((1+\vartheta')\,|B_{i-1}| - |B_{i-1}|\big) \cdot M + |B_{i-1}| = \left(1 + \frac{\vartheta}{3N}\right)|B_{i-1}|$$

holds, for $i = 1, \ldots, N$, which gives

$$\begin{aligned}\mathbb{E}[|B_N|] \leq \mathbb{E}[\mathbb{E}[|B_N| \mid B_{N-1}]] &\leq \left(1 + \frac{\vartheta}{3N}\right) \cdot \mathbb{E}[|B_{N-1}|] \\ &\leq \left(1 + \frac{\vartheta}{3N}\right)^N \cdot \mathbb{E}[|B_0|] = \left(1 + \frac{\vartheta}{3N}\right)^N \cdot |B|\,.\end{aligned}$$

Using $1 + x \leq e^x \leq 1 + 3x$, for $x \in [0,1]$, we obtain

$$\mathbb{E}[|B_N|] \leq \left(1 + \frac{\vartheta}{3N}\right)^N \cdot |B| \leq \exp\left(N\frac{\vartheta}{3N}\right) \cdot |B| = e^{\frac{\vartheta}{3}} \cdot |B| \leq (1+\vartheta) \cdot |B|\,. \qquad \blacktriangleleft$$

▶ **Lemma 23.** *Let B_N be the set defined above. Then, for any two points $p, q \in P \setminus B_N$, there is a $(1+\varepsilon)$-path in the graph $G \setminus B$.*

Proof. The proof is essentially the same as the proof of Theorem 15 in [4].

Let $p, q \in P \setminus B_N$ be fixed. According to Theorem 19, there is an ordering $\sigma \in \Pi^+$, such that all the points $z \in (p,q)_\sigma$ lie in one of the balls of radius $\varsigma\,\|p-q\|$ around p and q. Recall that the graph G contains G_σ^N as a subgraph. Since $p, q \notin B_N$ and G_σ^N is reliable, there is a path connecting p and q that is monotone with respect to σ and avoids any point in B_{N-1} by Theorem 15. Therefore, there is a unique edge $p'q'$ along this path such that p' is in the close neighborhood of p and q' is in the close neighborhood of q. Furthermore, we also have that $p', q' \in P \setminus B_{N-1}$. We fix the edge $p'q'$ in path π and find subpaths between the pairs pp' and qq' in a recursive manner. The bounds on the distances are

- **(i)** $\|p'-q'\| \leq (1+2\varsigma)\,\|p-q\|\,,$
- **(ii)** $\|p-p'\| \leq \varsigma\,\|p-q\|$ and similarly $\|q-q'\| \leq \varsigma\,\|p-q\|$.

We repeat this process $N-1$ times. Let Q_i be the set of pairs that needs to be connected in the ith round, that is, $Q_0 = \{pq\}, Q_1 = \{pp', qq'\}$ and so on. There are at most 2^i pairs in Q_i and for any pair $xy \in Q_i$ we have $x, y \in P \setminus B_{N-i}$. For each pair $xy \in Q_i$, there is an ordering σ such that the argument above can be repeated. That is, there is a monotone path in the graph $G_\sigma^{N-i} \setminus B_{N-i-1}$ according to σ and there is an edge $x'y'$ along this path such that

- **(i)** $\|x'-y'\| \leq (1+2\varsigma)\,\|x-y\| \leq (1+2\varsigma)\varsigma^i\,\|p-q\|\,,$
- **(ii)** $\|x-x'\| \leq \varsigma\,\|x-y\| \leq \varsigma^{i+1}\,\|p-q\|$ and similarly $\|y-y'\| \leq \varsigma^{i+1}\,\|p-q\|$.

The edge $x'y'$ is added to path π and the pairs xx' and yy' are added to Q_{i+1}, unless they are trivial (i.e., $x = x'$ or $y = y'$). After $N-1$ rounds, Q_{N-1} is the set of active pairs that still needs to be connected. Notice that $x, y \in P \setminus B_1$ holds for any pair $xy \in Q_{N-1}$. Again, for each pair in Q_{N-1}, we apply Theorem 19 and Theorem 15 to obtain a monotone path according to some ordering σ in the graph G_σ^1. None of these paths use any points in B. In order to complete the path π we add the whole paths obtained in the last step. It is not hard to see that the number of edges of each of the paths added in the last step is at most $2\log n$. Indeed, it is clear from the analysis of our one-dimensional construction that a path using the stairways can have at most two points per level. Since the number of levels in the construction is fewer than $\log n$, we get the bound $2\log n$.

Now, that we have a path π that connects the points p and q without using any points in the failed set B, we give an upper bound on the length of π. First, we calculate the total length added in the last step. There are $|Q_{N-1}| \leq 2^{N-1}$ pairs in the last step and for each pair $xy \in Q_{N-1}$ we have $\|x-y\| \leq \|p-q\| \varsigma^{N-1}$. Thus, we obtain

$$\begin{aligned}\sum_{\{x,y\}\in Q_{N-1}} \text{length}(\pi[x,y]) &\leq 2^{N-1}\big((1+2\varsigma)\|p-q\|\varsigma^{N-1} + 2\log n \|p-q\|\varsigma^N\big)\\ &\leq 2\cdot 2\varsigma\|p-q\| + (2\varsigma)^N \log n \|p-q\| = \Big(4\varsigma + (2\varsigma)^N \log n\Big)\|p-q\|\\ &\leq \left(\frac{\varepsilon}{4} + \left(\frac{\varepsilon}{8}\right)^{\log\log n}\log n\right)\|p-q\|\\ &\leq \left(\frac{\varepsilon}{4} + \frac{\varepsilon}{4}\cdot\left(\frac{1}{2}\right)^{\log\log n}\log n\right)\|p-q\| = \frac{\varepsilon}{2}\|p-q\|,\end{aligned}$$

where we simply use $2\varsigma \leq 1$ in the second line and $\varsigma = \varepsilon/16$ and $N = \lceil \log\log n \rceil$ in the third line. Second, we bound the total length of the edges that were added to path π in any round except the last. This contributes at most

$$\begin{aligned}\sum_{i=0}^{N-2} 2^i\cdot(1+2\varsigma)\varsigma^i\|p-q\| &\leq (1+2\varsigma)\|p-q\|\cdot\sum_{i=0}^{\infty}(2\varsigma)^i = (1+2\varsigma)\|p-q\|\cdot\frac{1}{1-2\varsigma}\\ &= \left(1+\frac{4\varsigma}{1-2\varsigma}\right)\|p-q\| = \left(1+\frac{\varepsilon/4}{1-\varepsilon/8}\right)\|p-q\| \leq \left(1+\frac{\varepsilon}{2}\right)\|p-q\|\end{aligned}$$

to the length of π. Therefore the total length of the path π connecting p and q, without using any points of B, is at most $(1+\varepsilon)\|p-q\|$. ◀

▶ **Theorem 24.** *Let $\vartheta, \varepsilon \in (0,1)$ be fixed parameters and $P \subseteq [0,1)^d$ be a set of n points. The graph G, constructed in Section 4.1, is a ϑ-reliable $(1+\varepsilon)$-spanner of P in expectation and has size $\mathcal{O}\big(c\,n\log\log^2 n\log\log\log n\big)$, where $\mathcal{O}$ hides constant that depends on the dimension d, and $c = \mathcal{O}(\varepsilon^{-2d}\vartheta^{-1}\log^3\varepsilon^{-1}\log\vartheta^{-1})$.*

Proof. The size of the construction is proved in Lemma 20. Let B_N be the set defined above. By Lemma 22, the expected size of B_N is at most $(1+\vartheta)|B|$. By Lemma 23, for any two points $p, q \in P \setminus B_N$, there is a $(1+\varepsilon)$-path between p and q in the graph $G \setminus B$. Thus, we have $\mathbb{E}[\lambda(G,B)] \leq \vartheta$. ◀

4.3 Probabilistic bound

The same construction, as we used in Section 4.1, can be applied to construct spanners with near linear edges that are reliable with probability $1-\rho$. The idea is to use the probabilistic version of the one-dimensional construction with parameters $\rho' = \frac{\rho}{MN}$ and $\vartheta' = \frac{\vartheta}{3MN}$. Then, similarly to Lemma 22, it is not hard to show that $|B_N| \leq (1+\vartheta)|B|$ holds with probability $1-\rho$.

▶ **Lemma 25.** *Let B_N be the set defined in Section 4.2. The probability that the size of B_N is larger than $(1+\vartheta)\cdot|B|$ is at most ρ.*

Proof. By Lemma 17, and since all attacks are oblivious, we have $\mathbb{P}[|B_i^\sigma| > (1+\vartheta')\,|B_{i-1}|] \leq \rho'$ for all $\sigma \in \Pi^+$ and $i \geq 1$. Therefore,

$$\begin{aligned}\mathbb{P}[|B_i| > (1+M\vartheta')\,|B_{i-1}|] &= \mathbb{P}[|B_i \setminus B_{i-1}| > M\vartheta'\,|B_{i-1}|] \\ &\leq \mathbb{P}[\cup_{\sigma\in\Pi^+} |B_i^\sigma \setminus B_{i-1}| > \vartheta'\,|B_{i-1}|] \leq \sum_{\sigma\in\Pi^+} \mathbb{P}[|B_i^\sigma \setminus B_{i-1}| > \vartheta'\,|B_{i-1}|] \\ &= \sum_{\sigma\in\Pi^+} \mathbb{P}[|B_i^\sigma| > (1+\vartheta')\,|B_{i-1}|] \leq M\rho'\end{aligned}$$

holds for $i = 1, \ldots, N$. Since $\left(1+\frac{\vartheta}{3N}\right)^N \leq \left(e^{\frac{\vartheta}{3N}}\right)^N \leq 1+\vartheta$, we get

$$\begin{aligned}\mathbb{P}[|B_N| > (1+\vartheta)\,|B|] &\leq \mathbb{P}\left[|B_N| > \left(1+\frac{\vartheta}{3N}\right)^N |B|\right] \leq \mathbb{P}\left[\bigcup_{i=1}^N |B_i| > \left(1+\frac{\vartheta}{3N}\right)|B_{i-1}|\right] \\ &\leq \sum_{i=1}^N \mathbb{P}\left[|B_i| > \left(1+\frac{\vartheta}{3N}\right)|B_{i-1}|\right] = \sum_{i=1}^N \mathbb{P}\left[|B_i| > \left(1+M\vartheta'\right)|B_{i-1}|\right] \leq NM\rho' = \rho.\end{aligned}$$

◀

Therefore, using the same argument as for Theorem 24, we obtain the following result, which gives a slight improvement in the constants, compared to the trivial multiplicative factor $\mathcal{O}\big(\log \rho^{-1}\big)$ by simply repeating the construction of Section 4.1.

▶ **Theorem 26.** *Let $\vartheta, \varepsilon, \rho \in (0,1)$ be fixed parameters and $P \subseteq [0,1)^d$ be a set of n points. The graph described above is a ϑ-reliable $(1+\varepsilon)$-spanner of P with probability $1-\rho$. Furthermore, the size of the construction is $\mathcal{O}\big(c\, n \operatorname{loglog}^2 n \operatorname{logloglog} n\big)$, where $\mathcal{O}$ hides constant that depends on the dimension d, and $c = \mathcal{O}\big(\varepsilon^{-2d}\vartheta^{-1}\log^3\varepsilon^{-1}(\log\vartheta^{-1} + \log\rho^{-1})\big)$.*

5 Conclusions

Reliable spanners require $\Omega(n \log n)$ edges. In this paper, we showed that fewer edges are sufficient, if the spanner only has to be reliable against oblivious attacks (in expectation or with a certain probability). Our new construction avoids the use of expanders, and as a result has much smaller constants than previous constructions, making it potentially practical. The number of edges in the new spanner is significantly smaller – it is linear in one dimension, and roughly $\mathcal{O}(n \operatorname{loglog}^2 n)$ in higher dimensions. An open problem is whether these loglog-factors in higher dimensions can be avoided. Furthermore, similar results for reliable spanners for general metrics would be of interest.

References

1 J. Aspnes and G. Shah. Skip graphs. *ACM Transactions on Algorithms*, 3(4):37, November 2007.

2 P. Bose, P. Carmi, V. Dujmović, and P. Morin. Near-optimal O(k)-robust geometric spanners. *CoRR*, abs/1812.09913, 2018. `arXiv:1812.09913`.

3 P. Bose, V. Dujmović, P. Morin, and M. Smid. Robust geometric spanners. *SIAM Journal on Computing*, 42(4):1720–1736, 2013. `doi:10.1137/120874473`.

4 K. Buchin, S. Har-Peled, and D. Oláh. A spanner for the day after. In *Proc. 35th Int. Annu. Sympos. Comput. Geom.* (SoCG), pages 19:1–19:15, 2019. `doi:10.4230/LIPIcs.SoCG.2019.19`.

5 T.-H. H. Chan, M. Li, L. Ning, and S. Solomon. New doubling spanners: Better and simpler. *SIAM Journal on Computing*, 44(1):37–53, 2015. `doi:10.1137/130930984`.

6 T. M. Chan, S. Har-Peled, and M. Jones. On Locality-Sensitive Orderings and Their Applications. In *Proc. 10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, pages 21:1–21:17, 2018. doi:10.4230/LIPIcs.ITCS.2019.21.

7 C. Levcopoulos, G. Narasimhan, and M. Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proc. 30th Annu. ACM Sympos. Theory Comput.* (STOC), pages 186–195, 1998. doi:10.1145/276698.276734.

8 C. Levcopoulos, G. Narasimhan, and M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32(1):144–156, 2002. doi:10.1007/s00453-001-0075-x.

9 T. Lukovszki. New results of fault tolerant geometric spanners. In *Proc. 6th Workshop Algorithms Data Struct.* (WADS), pages 193–204, 1999. doi:10.1007/3-540-48447-7_20.

10 S. Solomon. From hierarchical partitions to hierarchical covers: Optimal fault-tolerant spanners for doubling metrics. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 363–372, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591864.

New Binary Search Tree Bounds via Geometric Inversions

Parinya Chalermsook
Aalto University, Finland
chalermsook@gmail.com

Wanchote Po Jiamjitrak
Aalto University, Finland
wanchotej@gmail.com

Abstract

The long-standing dynamic optimality conjecture postulates the existence of a dynamic binary search tree (BST) that is $O(1)$-competitive to all other dynamic BSTs. Despite attempts from many groups of researchers, we believe the conjecture is still far-fetched. One of the main reasons is the lack of the "right" potential functions for the problem: existing results that prove various consequences of dynamic optimality rely on very different potential function techniques, while proving dynamic optimality requires a single potential function that can be used to derive all these consequences. In this paper, we propose a new potential function, that we call *extended (geometric) inversion*. Inversion is arguably the most natural potential function principle that has been used in competitive analysis but has never been used in the context of BSTs. We use our potential function to derive new results, as well as streamlining/strengthening existing results.

First, we show that a broad class of BST algorithms (including Greedy and Splay) are $O(1)$-competitive to Move-to-Root algorithm and therefore have simulation embedding property – a new BST property that was recently introduced and studied by Levy and Tarjan (SODA 2019). This result, besides substantially expanding the list of BST algorithms having this property, gives the first potential function proof of the simulation embedding property for BSTs (thus unifying apparently different kinds of results). Moreover, our analysis is the first where the costs of two dynamic binary search trees are compared against each other directly and systematically. Secondly, we use our new potential function to unify and strengthen known BST bounds, e.g., showing that *Greedy* satisfies the weighted dynamic finger property within a multiplicative factor of $(5 + o(1))$.

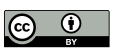
2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Binary Search Tree, Potential Function, Inversion, Data Structures, Online Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.28

Funding *Parinya Chalermsook*: Part of this work was done while Parinya was visiting the Simons Institute for the Theory of Computing. It was partially supported by the DIMACS/Simons Collaboration on Bridging Continuous and Discrete Optimization through NSF grant #CCF-1740425. This project has received funding from European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759557). Parinya is also funded by Academy of Finland Research Fellowship, under grant number 310415.

Acknowledgements We would like to thank Thatchaphol Saranurak for his contributions in the early stage of this paper and for many insightful discussions. We also thank anonymous reviewers for many detailed comments and suggestions.



1 Introduction

The dynamic optimality conjecture [23] is among the most fundamental problems in data structures. The conjecture postulates the existence of an online[1] and dynamic[2] binary search tree (BST) that is $O(1)$-competitive (or simply *competitive*) to the optimal offline BST algorithm. So far, the main candidates for being dynamically optimal are Splay and Greedy[17, 12, 18] (a.k.a. *Greedy Future* and *Greedy ASS*) since they possess many desirable properties that are consequences of dynamic optimality [5, 14, 11, 10, 19], although the best known competitive ratio of $O(\log \log n)$ is given by Tango trees [13]. Despite various attempts from many groups of researchers, the conjecture remains elusive. There are several observed reasons that make the conjecture long-standing, and one such reason (that this work tries to address) is the difficulty of comparing the behavior of two dynamic BSTs (see, for instance, [16]), which can be attributed to the lack of "generic" and "intuitive" potential function in this context: The simplest bound, static optimality, was derived via sum-of-logs [23, 6], and Splay's dynamic finger (which extends static optimality) uses sophisticated potential function that in some way extends sum-of-logs [10, 11]; Greedy's weighted dynamic finger [14] relies on very different potential function that neither seems related to sum-of-logs nor Splay's dynamic finger; the recent simulation embedding properties [16, 21] as well as best known bound for Splay's deque property [19, 20] do not even use potential function. Finally, none of these techniques was used to prove that Greedy or Splay is $o(\log n)$-competitive. Given this state of the art, it is relatively unclear which potential function should be used/extended for proving dynamic optimality. This paper is inspired by the following question:

> Is there a natural, generic potential function technique that allows us to compare the cost of two dynamic BSTs in a modular way?

1.1 Our Contribution

Our main conceptual contribution is a new potential function that, in our opinion, seems to be the right way to handle binary search trees. Our idea is inspired by *inversions*[3], which are arguably the most natural potential function for the purpose of analyzing online algorithms (see, for instance, [2, 22, 1] in the context of list update and [9] in the context of the k-server problem). We illustrate the power of our techniques in two directions.

Let us first introduce some notation before stating our results. We consider keys in $[n] = \{1, 2, \ldots, n\}$ and access sequence $X = (x_1, x_2, \ldots, x_m) \in [n]^m$. For an online dynamic BST algorithm $\mathcal{A}$, denoted by $cost_{\mathcal{A}}(X)$ the cost of serving sequence X using algorithm $\mathcal{A}$. We illustrate the power of our new concept in two ways.

First contribution: MTR-Competitiveness and Simulation Embedding

Recently, Levy and Tarjan [16] (and independently Russo [21]) noted the difficulty of comparing two dynamic BSTs and proposed an alternative path towards dynamic optimality. They rephrase dynamic optimality as two intrinsic properties (which they call, *simulation*

[1] A BST is online if an input sequence is revealed one at a time, i.e. the request for x_t appears at time t.
[2] A binary search tree is dynamic if it is allowed to change its form after each access, paying the cost of pointer movements.
[3] In general, the inversion potential function (or its generalization to "distance potential") measures the difference between the algorithm and the optimal, so it is suitable for analyzing the case when the optimal can change.

embeddings and *approximately monotone*) of an algorithm. Roughly speaking, a BST has simulation embeddings if it can "simulate"[4] any other BST algorithm, and it is approximately monotone if the cost of running the algorithm on an input sequence X is asymptotically at least the cost of running it on an arbitrary subsequence of X. An algorithm is $O(1)$-competitive if and only if it has both properties. They argue that Splay trees satisfy simulation embeddings and outlined a plan to prove that Splay trees are approximately monotone.

In this paper, we show that a broad class of algorithms (as defined in [6]) in fact satisfies simulation embeddings. This result is derived as a corollary to the following theorem.

▶ **Theorem 1.** *All BST algorithms (including Greedy and Splay) described in [6] are $O(1)$-competitive to Move-to-Root.*

Move-to-Root (MTR) is a classical BST algorithm that always rotates the requested key up until it becomes the root of the tree. It is known to be sub-optimal but not subsumed by any existing BST bounds (such as working set [23], lazy finger [14], pattern avoidance [5], or multiple fingers [8]). See discussion in the full version. Interestingly, so far no dynamic BSTs have been shown to be competitive even to this simplest dynamic algorithm.

▶ **Corollary 2.** *Let $\mathcal{A}$ be any BST algorithm according to [6]. Then, $\mathcal{A}$ has simulation embedding property. Therefore, it is dynamically optimal if and only if $\mathcal{A}$ is approximately monotone.*

Corollary 2 gives the first potential function proof of simulation property of any BST algorithm, therefore unifying the classical potential function techniques with the new attempt by Levy, Tarjan and Russo. The fact that infinitely many BST algorithms (that have simple description) have simulation embedding property can be interpreted in many ways. For an optimist, this could give us an access to a large design toolbox for studying the Levy-Tarjan approach: Instead of focusing on Splay or Greedy, we have the freedom to seek an algorithm that is approximately monotone by fine-tuning.

Another interesting aspect of MTR-competitiveness is perhaps a conceptual resemblance between Move-to-Root and the second Wilber bound [24] which is believed to be stronger than the first Wilber bound but so far no algorithm has ever exploited such bound[5]. Being able to charge the cost of an algorithm to Move-to-Root is a very first step towards this direction. (Informally, the second Wilber bound is equal to *crossing* Move-to-Root, see [16] for a more detailed discussion).

Second contribution: Streamlining known bounds

Now, we discuss how to use our potential function to streamline the BST bounds. Our second main result is an improved bound for the lazy finger property of Greedy. For a sequence $X = (x_1, \ldots, x_m)$ and a fixed BST R, denoted by $\mathsf{LF}_R(X) = \sum_t d_R(x_t, x_{t+1})$ where $d_R(a, b)$ denotes the number of edges on the unique path in R from a to b. Let $\mathsf{LF}(X) := \min_R \mathsf{LF}_R(X)$, we say that an algorithm $\mathcal{A}$ has lazy finger property if $cost_{\mathcal{A}}(X) \leqslant O(\mathsf{LF}(X))$. For any sequence X, denote by $G(X)$ the cost of Greedy in the geometric view[6]. Iacono and Langerman showed that $G(X) \leqslant C' \cdot (\mathsf{LF}(X) + m)$ where C' is around 50 (A quick glance at

[4] The definition of simulation is quite technical and we will only discuss this formally later.

[5] The best known competitive ratio is due to Tango Trees and its variants [13] which charge the cost to the first Wilber bound, which is provably insufficient for dynamic optimality [15, 4]

[6] Recall that, when turning the Greedy algorithm into a standard BST view, there is a constant factor blowup in the cost, that is, $cost_{Greedy}(X)$ (in the tree view) is at most $O(G(X))$.

their paper would show that the value of C' is 24, but there is also another multiplicative factor hidden in converting the result from "leaf-oriented" tree[7] setting to the BST setting). We show the following.

▶ **Theorem 3.** *For each access sequence X, $G(X) \leqslant 5 \cdot \mathsf{LF}(X) + O(m+n)$.*

In particular, when the input sequence is sufficiently costly, this bound converges to a multiplicative factor of 5 in front of the lazy finger term. We highlight that the interesting aspect of this result is not the improvement of constant but rather (1) the fact that our lazy finger proof directly and intuitively extends the proof of static optimality, and (2) the fact that we are able to charge the cost directly to the reference BST R instead of a leaf-oriented tree, in contrast to [14]. We hope that these two points open up natural new paths to adapt our techniques to handle stronger BST bounds where leaf-oriented settings become unnatural (e.g. BST rotation is less natural in the leaf-oriented setting).

Conclusion & Open Problems. We introduce a new potential function that allows us to, for the first time, compare two dynamic BSTs directly in a systematic way. Moreover, to our knowledge, ours is the first potential function in the BST context that uses a natural concept of inversions. We show many applications of our potential function. We note that, once the potential function is formally defined, the proofs of our results do not require any ground-breaking ideas but rather a careful adaptation of existing proofs [23, 6, 14] to our potential function.

Though the most intriguing open question is to prove dynamic optimality, we feel that it is still very far from the current understanding. We list here some open questions that we believe they are not overly far-fetched.

- **(i)** Can we use our method to analyze weighted dynamic finger (or even just dynamic finger) for Splay trees?
- **(ii)** Can we show that Greedy satisfies multi-finger properties as defined in [8]? We find this non-trivial even when there are two fingers.
- **(iii)** The property of being $O(\sqrt{\log n})$-competitive BST can be cast as a "local" BST property, e.g. see a survey paper [7]. Can we show that Greedy satisfies this property?

Further Related Work. Most prior works in this area focus on proving consequential properties of dynamic optimality. In particular, dynamic BSTs satisfy various forms of locality of reference properties that allow efficient access when the input sequence is in some way "local". For instance, the dynamic finger bound allows an efficient access when the access is close to the previous one (on average). It is often not so difficult to prove that these locality properties are satisfied by an optimal offline BST, so a candidate for optimal online BST must satisfy them as well. Proving that an online BST algorithm satisfies such properties has, however, been very challenging (for instance, Cole's proof [11, 10] of Splay's dynamic finger property spans 80 pages in total). Recently, a much stronger locality of reference bound, called Lazy Finger [3], was proved in a breakthrough result of Iacono and Langerman [14]. One way to view these locality of reference properties is as a "mildly dynamic" BST algorithm, i.e. each such property can be described by a restricted way of using the power of dynamic BST algorithms. Therefore, proving these bounds has naturally been seen as intermediate steps to study the dynamic optimality conjecture.

[7] A leaf-oriented binary search tree is one where all keys in $[n]$ are maintained as leafs, and each internal BST node is auxiliary (corresponding to a subset of leafs in the subtree under it).

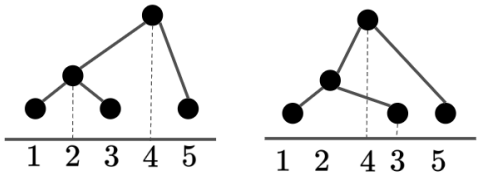


Figure 1 Examples of good (left) and bad (right) drawing of the same binary search tree.

2 Overview of Techniques

Let us recall the BST search model. The algorithm $\mathcal{A}$ maintains a binary search tree T on keys $[n]$, and when access $x_t \in [n]$ comes at time t, the cost incurred is equal to the depth of x_t in T. Elements on the search path of x_t (including x_t) are said to be *touched* by T. After that, the algorithm may adjust the shape of the tree, paying the cost which is equal to the number of keys that are involved in the adjustment. The total cost is then equal to the sum of search cost and adjustment cost. In all algorithms we consider, it suffices to analyze only the search cost (in particular, for BST algorithms that only change the search path, the update cost is at most a constant factor of the length of the search path. Therefore, such cost can be charged to the search cost). We follow this standard practice.

For convenience, we will abuse notation and use $\mathcal{A}$ to stand for both the BST algorithm and the state of BST at certain time.

2.1 Interval geometry for BST & Extended Inversion

Our potential function is defined based on geometry that requires correct drawing of BSTs. When drawing a BST, one should always use two rules: (1) imagine placing node containing key z on the plane at point p where $p.x = z$ (the x-coordinate is z). (2) If u is a parent of v, always draw u higher than v (See Figure 1)

Let $\mathcal{A}$ be a BST. For each key $z \in [n]$, let $I_{\mathcal{A}}(z)$ be the largest open interval in $(0, n+1)$ containing only all keys in the subtree of $\mathcal{A}$ rooted at z (in Figure 1 we have $I_{\mathcal{A}}(2) = (0, 4)$). More formally, $I_{\mathcal{A}}(z) = (\mathsf{left}_{\mathcal{A}}(z), \mathsf{right}_{\mathcal{A}}(z))$ where $\mathsf{left}_{\mathcal{A}}(z)$ is the nearest left ancestor of z (0 if not exist) and $\mathsf{right}_{\mathcal{A}}(z)$ is the nearest right ancestor of z ($n+1$ if not exist). When drawing the BST correctly, we have that the intervals $\{I_{\mathcal{A}}(z)\}_{z\in[n]}$ form a laminar family (that is, each pair of intervals is either disjoint or nested); see Figure 2.

▶ **Observation 4.** *If z' is a left child of z, then $I_{\mathcal{A}}(z') = (\mathit{left}_{\mathcal{A}}(z'), \mathit{right}_{\mathcal{A}}(z')) = (\mathit{left}_{\mathcal{A}}(z), z)$. If z' is a right child of z, then $I_{\mathcal{A}}(z') = (\mathit{left}_{\mathcal{A}}(z'), \mathit{right}_{\mathcal{A}}(z')) = (z, \mathit{right}_{\mathcal{A}}(z))$. Also, for any keys $y, z \in [n]$, we have $y \in I_{\mathcal{A}}(z)$ iff y is in the subtree rooted at z.*

Let $\mathcal{A}$ be an algorithm we want to analyze and $\mathcal{O}$ be an optimal algorithm. In this paper, we only consider $\mathcal{A}$ that changes only search path (this includes Greedy and Splay). Both $\mathcal{A}$ and $\mathcal{O}$ store the keys in $[n]$. We want to have a potential function that captures the "difference" between $\mathcal{A}$ and $\mathcal{O}$. The most natural scheme (which does not always work) often used in the context of online algorithm for this purpose is *inversion*.

▶ **Definition 5** (Inversion). *Let $z, \alpha \in [n]$. We say that z forms an* inversion *with α if $z \in I_{\mathcal{O}}(\alpha)$ and $\alpha \in I_{\mathcal{A}}(z)$.*

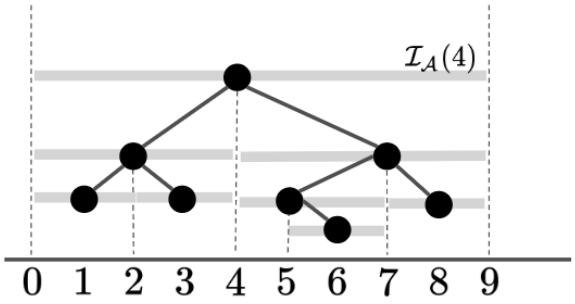


Figure 2 Example of intervals (in grey) for all keys. In this figure, $I_{\mathcal{A}}(4) = (0, 9)$ and $I_{\mathcal{A}}(5) = (4, 7)$.

Notice that an inversion is an indicator of α being in the subtree of z in one BST, but z is in the subtree of α in the other. See Figure 3. Unfortunately, we are unable to use this natural and intuitive scheme to prove any meaningful result. We will use *extended inversion* instead.

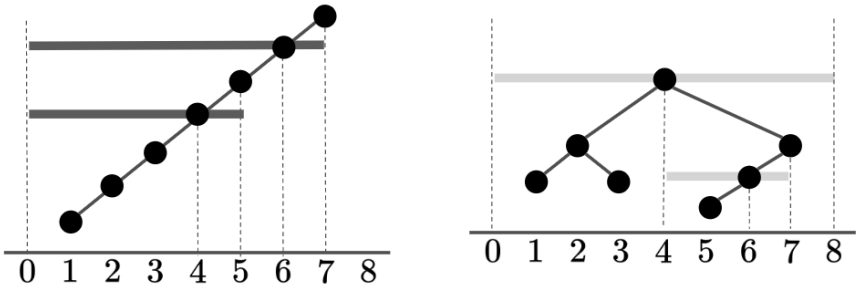


Figure 3 An example of inversion between the keys 4 and 6. The intervals of $\mathcal{A}$ are shown in dark grey and intervals of $\mathcal{O}$ in light grey. The left and right BSTs are $\mathcal{A}$ and $\mathcal{O}$ respectively.

For each interval in BST $\mathcal{O}$, $I_{\mathcal{O}}(\alpha)$, we define three **important points** of α in $\mathcal{O}$ as $\mathcal{P}_{\mathcal{O}}(\alpha) = \{\alpha, \mathsf{left}_{\mathcal{O}}(\alpha), \mathsf{right}_{\mathcal{O}}(\alpha)\}$.

▶ **Definition 6** (Extended Inversion). *Let $z, \alpha \in [n]$. We say that z forms an extended inversion with α if and only if $z \in I_{\mathcal{O}}(\alpha)$ and $\exists \beta \in \mathcal{P}_{\mathcal{O}}(\alpha)(\beta \in I_{\mathcal{A}}(z))$. See Figure 4.*

Unlike inversion, the notion of extended inversion is not symmetric. Clearly, if z forms an inversion with α, then z also forms an extended inversion with α.

Figure 4 The darker intervals are those intervals $I_{\mathcal{A}}(z)$, and the lighter ones are $I_{\mathcal{O}}(z)$. In the first figure from left, z forms an inversion with α. In the second, z forms an extended inversion with α but not an inversion. In the 3rd figure, there is no extended inversion, since $I_{\mathcal{A}}(z)$ does not contain any point in $\mathcal{P}_{\mathcal{O}}(\alpha)$. In the 4th figure, there is no extended inversion because $z \notin I_{\mathcal{O}}(\alpha)$.

2.2 Our potential function and its basic properties

All proofs in this paper build upon the following base function.

▶ **Definition 7** (Base Potential function). *Define the potential $\Phi = \Phi_{\mathcal{A},\mathcal{O}}$ at any state of execution of our algorithm $\mathcal{A}$ and optimal $\mathcal{O}$ as follows. Let $\Phi(z,\alpha) = 1$ if z forms an extended inversion with α; otherwise, $\Phi(z,\alpha) = 0$. The potential is defined as $||\Phi|| = \sum_{z,\alpha} \Phi(z,\alpha)$. This is the total number of extended inversions.*

We "visualize" our potential function value as a collection of "coins". Whenever $\Phi(z,\alpha) = 1$, one can imagine there is a coin of label z (or z-coin) placed at node α in $\mathcal{O}$ whenever z forms an extended inversion with α (see Figure 5). By definition, z always forms an extended inversion with itself, so there is always a z-coin at z. We use $\Phi(\bullet,\alpha) = \sum_{z\in[n]} \Phi(z,\alpha)$ to denote the total number of coins at α, and $\Phi(z,\bullet) = \sum_{\alpha\in[n]} \Phi(z,\alpha)$ the total number of z-coins. The coin interpretation will be used crucially in our analysis.

The main properties we would need are the following:

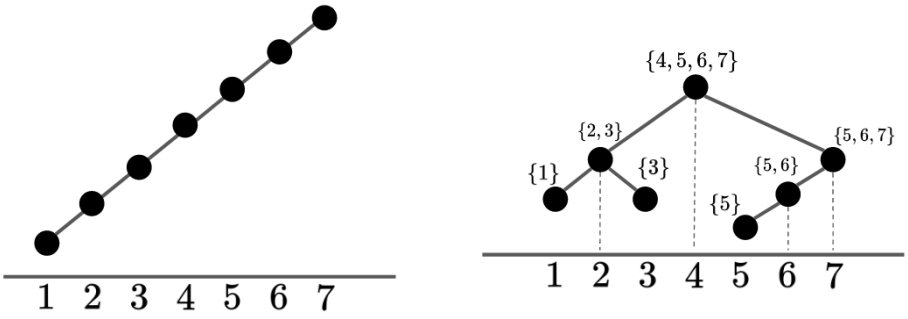


Figure 5 The left and right BSTs are $\mathcal{A}$ and $\mathcal{O}$ respectively. The set notation shown at each node is a collection of coins placed at that node. For instance, there are 5- and 6-coins at node 6. The path of 3-coins (according to Lemma 8) is a path containing nodes 3 and 2.

▶ **Lemma 8.** *[Upward path property] For each $z \in [n]$, the set of nodes having z-coin (that is, $Q_z = \{\alpha : \Phi(z,\alpha) = 1\}$) is a contiguous subpath of the path from z to the root of $\mathcal{O}$.*

Proof. Let $\alpha_1, \alpha_2 \in Q_z$. Since $z \in I_{\mathcal{O}}(\alpha_1) \cap I_{\mathcal{O}}(\alpha_2)$, we must have that $I_{\mathcal{O}}(\alpha_1) \subseteq I_{\mathcal{O}}(\alpha_2)$ (from laminar property). Since $z \in Q_z$, we have that Q_z is a subset of path from z to the root. Next, we argue that Q_z is connected. Suppose α' is on the path from z to the root such that $\alpha' \notin Q_z$. See Figure 6. We argue that the parent of α' (say α, where α' is the right child of α) is also not in Q_z: From Observation 4, $I_{\mathcal{O}}(\alpha') = (\alpha, \mathsf{right}_{\mathcal{A}}(\alpha))$. Recall that $\mathcal{P}_{\mathcal{O}}(\alpha) = \{\alpha, \mathsf{left}_{\mathcal{A}}(\alpha), \mathsf{right}_{\mathcal{A}}(\alpha)\}$. Since $\alpha' \notin Q_z$, we have that $I_{\mathcal{A}}(z)$ is completely contained in $(\alpha, \mathsf{right}_{\mathcal{O}}(\alpha))$. This means that it does not contain any point in $\mathcal{P}_{\mathcal{O}}(\alpha)$. ◀

By the above lemma, we view the potential function $\Phi(z,\bullet)$ as the coins on an upward path in $\mathcal{O}$ and Φ as a collection of upward paths.

▶ **Lemma 9.** *Consider an access to key $x \in [n]$. Let z be a key on the search path $\mathcal{S}_{\mathcal{A}}(x)$ of our algorithm. Then x forms an inversion with $LCA_{\mathcal{O}}(x,z)$. Hence, there is a z-coin at node $LCA_{\mathcal{O}}(x,z)$.*

Proof. Fix $z \in \mathcal{S}_{\mathcal{A}}(x)$. Recall that $x \in I_{\mathcal{A}}(z)$. Denote by $\ell = LCA_{\mathcal{O}}(x,z)$. Since ℓ is between x and z, $\ell \in I_{\mathcal{A}}(z)$. Since ℓ is an ancestor of z in $\mathcal{O}$, $z \in I_{\mathcal{O}}(\ell)$, and thus an inversion occurs between z and ℓ. ◀

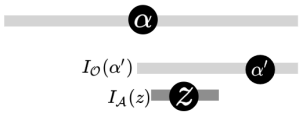


Figure 6 Illustration of the proof of Lemma 8.

The above lemma is the main advantage of the inversion principle: Each $z \in \mathcal{S}_{\mathcal{A}}(x)$ contributes $+1$ to the cost of $\mathcal{A}$ when accessing x. The lemma shows that such cost can be "deducted" from the z-coin at $LCA_{\mathcal{O}}(x, z)$.

▶ **Definition 10** (Canonical payment function)**.** *Consider algorithm $\mathcal{A}$ and optimal $\mathcal{O}$. The* ***canonical payment function*** *for accessing x is a function $\mathcal{CP} = \mathcal{CP}_{\mathcal{A},\mathcal{O},x}$ such that for each $z \in \mathcal{S}_{\mathcal{A}}(x)$, we have $\mathcal{CP}(z, LCA_{\mathcal{O}}(z, x)) = 1$. The function is 0 everywhere else (Figure 7).*

We will simply write $\mathcal{CP}$ instead of $\mathcal{CP}_{\mathcal{A},\mathcal{O},x}$ when it is clear from the context. Notice that, for each $z \in \mathcal{S}_{\mathcal{A}}(x), \mathcal{CP}(z, \alpha) = 1$ only if α is on the search path $\mathcal{S}_{\mathcal{O}}(x)$ of the optimal. From Lemma 9, we have that $\mathcal{CP}(z, \alpha) \leqslant \Phi(z, \alpha)$ for all $z, \alpha \in [n]$.

▶ **Observation 11.** *We have $\sum_{z,\alpha} \mathcal{CP}(z, \alpha) = |\mathcal{S}_{\mathcal{A}}(x)|$.*

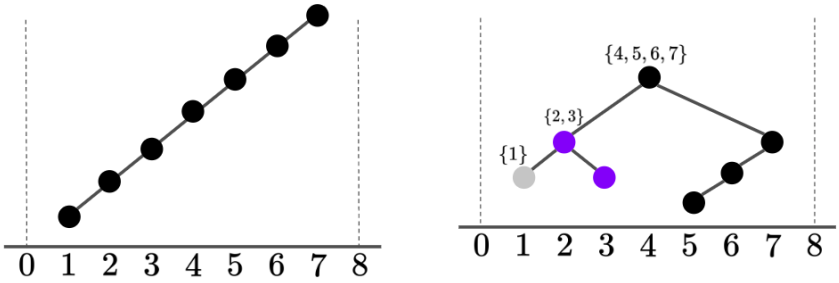


Figure 7 An example of coins in the support of $\mathcal{CP}$. In this example, when access 1, we touch every key in $\mathcal{A}$, but we touch only {1,2,4} in $\mathcal{O}$. Each $z \in \{4, 5, 6, 7\}$ has its coin at node 4 in the support of $\mathcal{CP}$. Each z in $\{2, 3\}$ has its coin at node 2. Finally, 1 has a coin at node 1.

2.3 Overview of our proofs

We first recall the basic ideas of potential function proofs.

▶ **Definition 12.** *We say that potential function Φ* ***proves*** *$\mathcal{A} \leqslant_\eta \mathcal{O}$ if for any state $\mathcal{A}$ and $\mathcal{O}$ of execution, when an access $x \in [n]$ arrives, we have*

$$(||\Phi_{\mathcal{A},\mathcal{O}}|| - ||\Phi_{\mathcal{A}',\mathcal{O}'}||) + \eta|\mathcal{S}_{\mathcal{O}}(x)| \geqslant |\mathcal{S}_{\mathcal{A}}(x)|$$

where $\mathcal{A}'$ and $\mathcal{O}'$ are the BSTs after the algorithm and the optimal update their trees.

The following claim follows from a standard potential function analysis. See, for instance, [23] for the formal proof.

▶ **Proposition 13.** *If potential function Φ proves $\mathcal{A} \leqslant_\eta \mathcal{O}$, then for any input $X \in [n]^m$, we have $cost_{\mathcal{A}}(X) \leqslant \eta \cdot cost_{\mathcal{O}}(X) + \Phi_{\max}$ where $\Phi_{\max}$ is the maximum value of $||\Phi_{\mathcal{A},\mathcal{O}}||$ over all possible states of execution of $\mathcal{A}$ and $\mathcal{O}$.*

Finally, since the size of support of $\mathcal{CP}$ is exactly $|\mathcal{S}_{\mathcal{A}}(x)|$, one can rewrite the condition in Definition 12 as $||\Phi_{\mathcal{A}',\mathcal{O}'}|| \leqslant ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| + \eta|\mathcal{S}_{\mathcal{O}}(x)|$. All our proofs follow this high-level idea: (i) Upper bound $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}||$, and (ii) upper bound $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}||$. The extended inversion potential function allows us to analyze Step (i) of various kinds of BST bounds in a modular way. The following is a key property.

▶ **Proposition 14.** *Consider algorithm $\mathcal{A}$ that, after access x arrives, updates the trees into $\mathcal{A}'$, while $\mathcal{O}$ does not change. Let $\widehat{\Phi} = \Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}$.*

(i) *For each $z \notin \mathcal{S}_{\mathcal{A}}(x)$ and $\alpha \in [n]$, we have $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leqslant \widehat{\Phi}(z,\alpha)$.*

(ii) *For each $\alpha \notin \mathcal{S}_{\mathcal{O}}(x)$ and $z \in [n]$, we have $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leqslant \widehat{\Phi}(z,\alpha)$.*

These imply that the potential change $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\widehat{\Phi}||$ is upper bounded by the "change on search paths", that is, $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| \leqslant \sum_{z\in\mathcal{S}_{\mathcal{A}}(x)} \sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)} \left(\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Phi}(z,\alpha)\right)$

Proof. First, for each such $z \notin \mathcal{S}_{\mathcal{A}}(x)$, the intervals $I_{\mathcal{A}}(z) = I_{\mathcal{A}'}(z)$, so the number of extended inversions for them remains the same. Since $\mathcal{CP}(z,\alpha) = 0$, $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leqslant \widehat{\Phi}(z,\alpha) = \Phi_{\mathcal{A},\mathcal{O}}(z,\alpha)$.

Now we prove the second item. Fix $\alpha \notin \mathcal{S}_{\mathcal{O}}(x)$. Due to the previous observation, only $z \in \mathcal{S}_{\mathcal{A}}(x)$ may create a new inversion. By Lemma 9, $\Phi_{\mathcal{A},\mathcal{O}}(z, LCA_{\mathcal{O}}(x,z)) = 1$. Since $LCA_{\mathcal{O}}(x,z)$ is an ancestor of α, by Lemma 8, $\Phi_{\mathcal{A},\mathcal{O}}(z,\alpha) = 1$. This means that there was already a z-coin at α. Since $\mathcal{CP}(z,\alpha) = 0$, $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leqslant \widehat{\Phi}(z,\alpha) = \Phi_{\mathcal{A},\mathcal{O}}(z,\alpha)$. ◀

All our proofs rely on this proposition to upper bound the change from $\mathcal{A}$ to $\mathcal{A}'$.

2.4 The First Showcase: Splay's Ziz-zig

We consider the Splay tree currently maintaining a path with n as a root and 1 as a leaf; and $\mathcal{O}$ does not change (that is, $\mathcal{O} = \mathcal{O}'$). Define the potential $\Psi(z,\alpha) = 2\Phi(z,\alpha)$ for all $z,\alpha \in [n]$ (there are 2 coins for each extended inversion). Let us consider the situation when an access $x = 1$ arrives. We will upper bound the change of potential function Ψ as follows:

$$||\Psi_{\mathcal{A}',\mathcal{O}}|| \leqslant ||\Psi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| + 6|\mathcal{S}_{\mathcal{O}}(x)|$$

Deducting $\mathcal{CP}$ from Ψ can be seen as removing some coins from the search path $\mathcal{S}_{\mathcal{O}}(x)$. Recall that in $\Psi - \mathcal{CP}$, for each z on the search path of $\mathcal{A}$, we had removed one z-coin at $LCA(z,x)$. The proof has two steps. In the first step, we move the coins around from "bad" to "good" keys. Refer to Figure 8 for the shape of $\mathcal{A}'$. For simplicity assume that n is odd. Except for the root, we pair i with $i+1$ for $i = 2,4,\ldots,n-1$. We say that the even keys are good and the odd keys are bad. For each i, we turn one $(i+1)$-coin at $LCA(i+1,x)$ into an i-coin at $LCA(i,x)$. Let $\widehat{\Psi}$ be the function after moving the coins, so $||\widehat{\Psi}|| = ||\Psi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}||$.

▶ **Lemma 15.** *We have $||\Psi_{\mathcal{A}',\mathcal{O}}|| - ||\widehat{\Psi}|| \leqslant O(|\mathcal{S}_{\mathcal{O}}(x)|)$. (In other words, we can add at most $O(|\mathcal{S}_{\mathcal{O}}|)$ to achieve the coin level of $\Psi_{\mathcal{A}',\mathcal{O}}$.)*

The rest of this section is devoted to proving the lemma. Denote $\Psi_{\mathcal{A}',\mathcal{O}}$ by Ψ' for convenience. From Proposition 14, we have $||\Psi_{\mathcal{A}',\mathcal{O}}|| - ||\widehat{\Psi}|| \leqslant \sum_{z\in\mathcal{S}_{\mathcal{A}}(x)} \sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)} \left(\Psi'(z,\alpha) - \widehat{\Psi}(z,\alpha)\right)$. The following claim therefore finishes the proof.

▷ Claim 16. We have $\sum_{z\in\mathcal{S}_{\mathcal{A}}(x)} \sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)} \left(\Psi'(z,\alpha) - \widehat{\Psi}(z,\alpha)\right) \leqslant 14|\mathcal{S}_{\mathcal{O}}(x)|$.

Proof. Refer to Figure 8 that illustrate the shape of $\mathcal{A}'$ (Splay) after the access of x. For $z = 1$, we use the crude bound of $\sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)} \left(\Psi'(1,\alpha) - \widehat{\Psi}(1,\alpha)\right) \leqslant 2|\mathcal{S}_{\mathcal{O}}(x)|$. For $z \geqslant 2$, we divide the analysis into two cases:

(i) new extended inversions, i.e. $\Psi'(z,\alpha) = 1$ and $\widehat{\Psi}(z,\alpha) = \Psi_{\mathcal{A},\mathcal{O}}(z,\alpha) = 0$

(ii) existing extended inversions that pay their coins out, i.e. $\Psi'(z,\alpha) = 1, \widehat{\Psi}(z,\alpha) = 0$ and $\Psi(z,\alpha) = 1$.

We upper bound the potential increase by the notion of *witness intervals*. For the first case, a new extended inversion can appear in $\Psi'(z,\alpha)$ due to the fact that the interval $I_{\mathcal{A}'}(z)$ becomes larger than $I_{\mathcal{A}}(z)$. In this case, we define $J(z,\alpha) = I_{\mathcal{A}'}(z) \backslash I_{\mathcal{A}}(z)$ as a **witness** for the new inversion (z,α). From Figure 8, these witness intervals (drawn in red) are non-overlapping. Since there are 3 important points for each $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$ and each point can be in at most one witness interval, there are at most $3|\mathcal{S}_{\mathcal{O}}(x)|$ witness intervals. Each witness interval requires 2 coins, so we need at most $6|\mathcal{S}_{\mathcal{O}}(x)|$ coins in total.

For the second case, this happened due to z being a bad key that pays its coin out of canonical payment (the good ones received their coins back from the transfer in the first step). In this case, we say that $J(z,\alpha) = I_{\mathcal{A}'}(z)$ is a witness for the inversion (z,α). From Figure 8, the witness intervals of this type (drawn in blue) are disjoint. Therefore, with the same argument as in the first case, we need at most $6|\mathcal{S}_{\mathcal{O}}(x)|$ coins in total. ◁

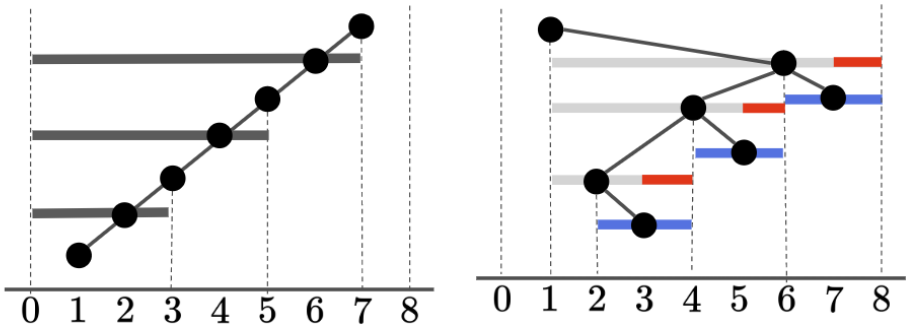


Figure 8 A BST before (left) and after (right) splaying $x = 1$. After splaying, intervals of odd keys (except access key) are disjoint. The red and blue intervals are the witnesses as in Claim 16.

There are two remarks about this proof. First, this proof only gives an analysis of a zig-zig case in Splay trees. A full Splay trees analysis can be found as a special case of our Theorem 1 whose proof is deferred to the full version. Second, if $\mathcal{O}$ is not a static tree, we add one more step to the analysis, that is, upper bounding the change due to updating $\mathcal{O}$; the second step for MTR-competitiveness results is simple, while for lazy finger, the second step is more involved, relying on a modified concept of extended inversions and on some ideas from [14].

2.5 Geometric Inversion for Geometric BST Algorithms

In order to use our scheme, all algorithms must be described in terms of *intervals*. In the previous section, we already explained how BST algorithms $\mathcal{A}$ in the tree view can be described as intervals $I_{\mathcal{A}}(z)$. In this section, we explain how to do the same for BST algorithms in the geometric view. As discussed in [12], the tree and geometric views are equivalent up to constant factor in the competitive ratios.

Let $[n]$ be the set of keys. An access sequence of binary search trees (BST) is specified as searching for key $x_t \in [n]$ at time $t = 1, \ldots, m$. Given an access sequence $X = (x_1, \ldots, x_m) \in [n]^m$, we view X as points $\mathcal{Q}_X = \{(x_t, t) : t = 1, \ldots, m\}$ in the plane where t-th access appears on row t (starting from bottom). We abuse the notation and simply write $\mathcal{Q}_X$ as X.

For $\mathbf{p},\mathbf{q} \in \mathbb{R}^2$, define $\square_{\mathbf{p},\mathbf{q}}$ as the set of integral points in the minimally closed rectangular area defined by $\mathbf{p}$ and $\mathbf{q}$. Let $Z \subseteq \mathbb{R}^2$, we say that such the rectangle $\square_{\mathbf{p},\mathbf{q}}$ is *empty* in Z (or Z-empty) if $Z \cap \square_{\mathbf{p},\mathbf{q}}$ contains no other point than $\mathbf{p}$ or $\mathbf{q}$. Also, the point set Z is *arborally satisfied* if for every pair $\mathbf{p}, \mathbf{q} \in Z$, the rectangle $\square_{\mathbf{p},\mathbf{q}}$ is not Z-empty.

Geometric BST problem (MinASS)

Let $X_{\leqslant t}$ denote the set of points in X in rows t and below. In the MinASS problem, given an input X of points, we are interested in computing a superset $Y \supseteq X$ such that Y is arborally satisfied, while minimizing $|Y|$. In the online version, at any time $t = 1, 2, \ldots, m$, a point in X arrives on the t-th row, and the algorithm must produces a feasible solution $Y_{\leqslant t}$ at each time t by adding points on the t^{th} row. Points in Y are said to be *touched* by the algorithm $\mathcal{A}$. We denote a set of touched keys at time t by $\mathcal{S}_{\mathcal{A}}(x_t)$ (same notation as search path of x_t in the tree view).

▶ **Theorem 17** (Demaine et al. [12]). *Let $\mathcal{A}$ be an algorithm for the MinASS problem. Then for each sequence X, there is a BST algorithm $\mathcal{A}'$ whose total cost on X is at most $O(cost_{\mathcal{A}}(X))$.*

Greedy

For each time t and key $z \in [n]$, let $\tau(z,t)$ denote the last touched time of z on or before time t. At time t, Greedy touches key $z \in [n]$ if and only if $(z, \tau(z,t))$ forms an empty rectangle with (x_t, t). The final output of Greedy on input X is defined as $G(X)$; denote the first t rows of such output by $G_{\leqslant t}(X)$. See Figure 9 for an illustration.

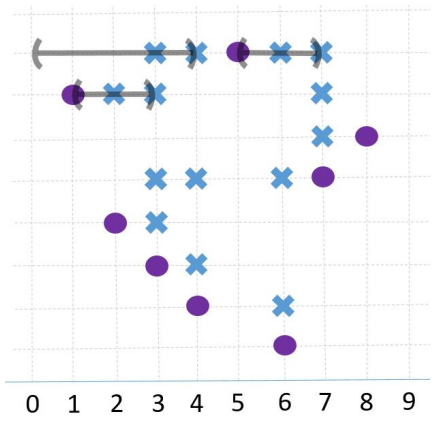


Figure 9 An example of intervals of Greedy. Here, circles represent access keys X and crosses represent touched points $Y \backslash X$. Interval of each key is defined by the top point (i.e. the last touched point) of such key. Intervals of 2,3, and 6 are shown in this figure.

Intervals for Geometric BSTs

Since the intervals of BST algorithms in a tree view depend on sub-trees, for Greedy, we need an analogue of sub-trees in geometric view. Let $z \in [n]$ be a key. The interval $I_G(z)$ is defined as $(\mathsf{left}_G(z), \mathsf{right}_G(z))$ where $\mathsf{left}_G(a)$ is the maximum key less than z that is touched at the same time as when a was last touched (0 if it does not exist). Similarly, we define $\mathsf{right}_G(z)$ as the minimum key greater than $a = z$, touched at the last touched time of z

$(n+1$ if it does not exist). So the interval $I_G(z)$ only change whenever z is touched. See Figure 9. Given an optimal tree $\mathcal{O}$, Lemma 8, Lemma 9 and Proposition 14 still hold for these geometric intervals. The following observations are important. (See Figure 10a)

▶ **Observation 18.** *Suppose key x_t is accessed. Let $\{I_G(b)\}_{b\in[n]}$ and $\{I'_G(b)\}_{b\in[n]}$ be the intervals before and after Greedy updates. Let $b_1 < b_2 < \ldots < b_k$ be the touched keys, $b_0 = 0$ and $b_{k+1} = n+1$. Then we have that $I'_G(b_i) = (b_{i-1}, b_{i+1})$ for all $i = 1, \ldots, k$.*

▶ **Corollary 19.** *For each point $p \in \mathbb{R}$, there are at most two touched keys z for which $p \in I'_G(z)$. In other words, the clique size of this interval graph is 2 and therefore, the touched intervals $I'_G(z)$ can be decomposed into two disjoint sets of intervals.*

2.6 The Second Showcase: Greedy

We show the following static optimality bound for Greedy using our potential.

▶ **Theorem 20.** *For each X and each static tree $\mathcal{O}$, $G(X) \leqslant 4 \cdot cost_{\mathcal{O}}(X) - m + n^2$.*

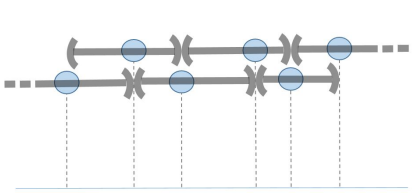
We note that the term n^2 is from the number of initial coins. Later, we will show that these initial coins can be bounded by $cost_{\mathcal{O}}(X)$, therefore we have $G(X) \leqslant 5 \cdot cost_{\mathcal{O}}(X)$.

The potential function we use is simply the extended inversion Φ. Let $\mathcal{O}$ be any tree. After accessing x, the Greedy intervals change from $\{I_G(z)\}_{z\in[n]}$ to $\{I'_G(z)\}_{z\in[n]}$, and the potential changes from Φ to Φ'. We will prove that $||\Phi'|| \leqslant ||\Phi - \mathcal{C}P|| + 4|\mathcal{S}_{\mathcal{O}}(x)| - 1$.

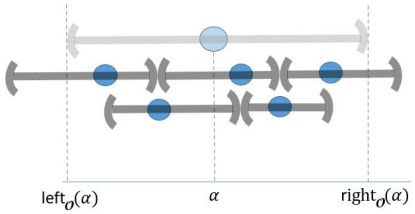
Summing over all accesses over a sequence X of length m will give us the desired bound. Denote $\Phi - \mathcal{C}P$ by $\widehat{\Phi}$. Due to Proposition 14, the change of potential function is only due to the inversions (z, α) where $z \in \mathcal{S}_G(x)$ and $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$.

▶ **Lemma 21.** *For each $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$, we have $||\Phi'(\bullet, \alpha)|| - ||\widehat{\Phi}(\bullet, \alpha)|| \leqslant 4$. Moreover, $||\Phi'(\bullet, x)|| - ||\widehat{\Phi}(\bullet, x)|| \leqslant 3$.*

Proof. Fix $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$. Notice that the term $\Phi'(z, \alpha) - \widehat{\Phi}(z, \alpha) = 1$ only if $z \in \mathcal{S}_G(x)$ and $\Phi'(z, \alpha) = 1$. To have $\Phi'(z, \alpha) = 1$, the interval $I'_G(z)$ must contain a point in $\mathcal{P}_{\mathcal{O}}(\alpha) = \{l, \alpha, r\}$ (where $l = \mathsf{left}_{\mathcal{O}}(\alpha)$ and $r = \mathsf{right}_{\mathcal{O}}(\alpha)$) and point z must be inside $I_{\mathcal{O}}(\alpha)$. From the fact that z is inside $I_O(\alpha)$ and the geometry of Greedy (Observation 18), only one interval can contain l, only one interval can contain r, and at most two intervals can contain α (this is all illustrated in Figure 10b); moreover, when $\alpha = x$, only one interval may contain α ($I_G(x)$ itself). This concludes the proof. ◀



(a) Intervals of keys that are touched at the same time in Greedy. Think of all of them as in the same height, we perturb them in order to show their intervals clearly.



(b) Intervals in G are represented in black, while interval in $\mathcal{O}$ is represented in grey.

Figure 10

To recap, the framework is as follows. Let $\mathcal{A}$ be an algorithm and $\mathcal{O}$ be an optimal algorithm. Suppose, after accesssing key $x \in [n]$, $\mathcal{A}$ changes to $\mathcal{A}'$ and $\mathcal{O}$ to $\mathcal{O}'$. We wan to upper bound $|\mathcal{S}_{\mathcal{A}}(x)| + ||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A},\mathcal{O}}|| \leqslant O(|\mathcal{S}_{\mathcal{O}}(x)|)$. We upper bound this in two steps:

- The change of $\mathcal{A}$ to $\mathcal{A}'$ after subtracting the cost: $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{C}P||$
- And then the change of $\mathcal{O}$ to $\mathcal{O}'$: $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}||$

We have shown how to bound the term $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{C}P||$ for Greedy and (a special case of) Splay. This is the "systematic" part of our framework that allows us to prove MTR-competitiveness for a family of BSTs. In the next subsection, we outlined how to upper bound $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}||$ for Splay and Greedy, thus implying their $O(1)$-competitiveness to MTR. The proof for weighted dynamic finger is omitted due to the lack of space. Full details appear in the full version.

2.7 MTR-Competitiveness

In this section, we present the proof that Greedy and Splay have simulation embeddings. Due limitation of space, we defer the rest of the proofs to the full version.

First, we present necessary preliminaries.

▶ **Definition 22** (Static Optimality). *A BST algorithm $\mathcal{A}$ is statically optimal if, for all input X and static (reference) tree R, $cost_{\mathcal{A}}(X) \leqslant O(cost_R(X))$.*

Remark that a more precise definition of static optimality involves "initial tree" (the initial state of algorithm $\mathcal{A}$). In the context of our potential function, it is not difficult to see that the initial tree only affects the cost by at most an additive factor of $cost_R(X)$. Hence we omit the reference to initial trees for brevity. Denote the move-to-root algorithm by MTR.

▶ **Definition 23** (MTR-Competiveness). *A BST algorithm $\mathcal{A}$ is MTR-competitive if, for all input X, $cost_{\mathcal{A}}(X) \leqslant O(cost_{MTR}(X))$.*

▶ **Definition 24** (Simulation Embeddings). *A BST algorithm $\mathcal{A}$ has simulation embeddings if, for all access sequence X and algorithm $\mathcal{O}$, there exists a supersequence $Y \supseteq X$ such that $cost_{\mathcal{A}}(Y) \leqslant O(cost_{\mathcal{O}}(X))$.*

Levy and Tarjan [16] show that Splay has simulation embeddings by constructing a transition graph G_4 of Splay. Transition graph G_k is a directed graph where each node represents an instance of k-node BST. There is an edge (u, v) if a tree instance u can be changed to a tree instance v using one access. Also, they show that, in order for BST $\mathcal{A}$ to have simulation embeddings, it suffices to show that G_k of $\mathcal{A}$ is strongly connected for some constant $k \geqslant 3$.

▷ Claim 25. The transition graph G_3 of MTR is strongly connected. Hence, MTR has simulation embeddings. (See Figure 11)

This implies the following theorem.

▶ **Theorem 26.** *If a BST algorithm $\mathcal{A}$ is $O(1)$-competitive to MTR, then $\mathcal{A}$ has simulation embeddings.*

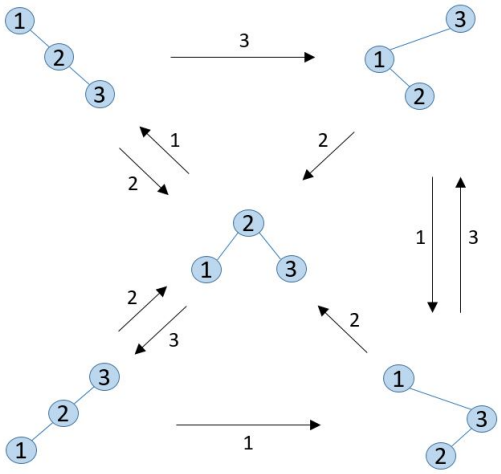


Figure 11 G_3 of MTR. The number on each arrow represents the key that MTR has to access in order to change its tree to the specific structure.

Proof. Suppose $\mathcal{A}$ is c-competitive to MTR. Now fix an input sequence X and a BST algorithm $\mathcal{O}$. From Claim 25, let d be the factor in the simulation embeddings of MTR, so we know that there exists a super-sequence X' such that $cost_{\mathsf{MTR}}(X') \leqslant d \cdot cost_{\mathcal{O}}(X)$. Since algorithm $\mathcal{A}$ is c-competitive to MTR, we have

$$cost_{\mathcal{A}}(X') \leqslant c \cdot cost_{\mathsf{MTR}}(X') \leqslant (cd) \cdot cost_{\mathcal{O}}(X)$$

This implies that $\mathcal{A}$ has simulation embeddings. ◀

Our potential function is particularly suitable for proving MTR-competitiveness. In fact, to show that $\mathcal{A}$ is $O(1)$-competitive to MTR, it suffices to only prove static optimality for $\mathcal{A}$ using extended inversions (together with a function that depends only on n.)

▶ **Lemma 27.** *A BST algorithm that (1) satisfies static optimality via potential function Ψ that depends linearly on extended inversions (i.e. $||\Psi|| = c_1||\Phi|| + c_2 n$), and (2) moves the accessed key to the root, must be $O(1)$-competitive to MTR.*

Proof. If $\mathcal{A}$ satisfies static optimality via Ψ, we have that:

$$||\Psi_{\mathcal{A}',\mathcal{O}}|| - ||\Psi_{\mathcal{A},\mathcal{O}} - \mathcal{C}P|| \leqslant C \cdot |\mathcal{S}_{\mathcal{O}}(x)|$$

Now, since the second term of potential is $c_2 n$ (only depending on n), to upper bound $||\Psi_{\mathcal{A}',\mathcal{O}'}|| - ||\Psi_{\mathcal{A}',\mathcal{O}}||$, it suffices to show that, we have $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}|| \leqslant 0$.

For key $z = x$, $||\Phi_{\mathcal{A}',\mathcal{O}'}(z,x)|| = ||\Phi_{\mathcal{A}',\mathcal{O}}(z,x)|| = c_1$. Now, we consider keys $z \neq x$. Observe that intervals in $\mathcal{O}'$ change as follow:

- **(i)** $I_{\mathcal{O}'}(x) = (0, n+1)$.
- **(ii)** $I_{\mathcal{O}'}(y) = (\mathsf{left}_{\mathcal{O}}(y), x)$ for all y such that $y < x$ and $y \in \mathcal{S}_{\mathcal{O}}(x)$.
- **(iii)** $I_{\mathcal{O}'}(y) = (x, \mathsf{right}_{\mathcal{O}}(y))$ for all y such that $y > x$ and $y \in \mathcal{S}_{\mathcal{O}}(x)$.
- **(iv)** $I_{\mathcal{O}'}(y) = I_{\mathcal{O}}(y)$ for all $y \notin S_{\mathcal{O}}(x)$.

In other words, the only new important point in $\mathcal{O}'$ is x. Since $\mathcal{A}'$ also has x as the root, no $I'_{\mathcal{A}}(z)$ can contain x. Since x is the only new ancestor for z, this means $||\Phi_{\mathcal{A}',\mathcal{O}'}(z,x)|| = 0$. ◀

Since Splays and Greedy satisfy static optimality via our potential function, Lemma 27 implies that they are MTR-competitive, and hence have simulation embeddings.

References

1 Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998.

2 Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

3 Presenjit Bose, Karim Douïeb, John Iacono, and Stefan Langerman. The power and limitations of static binary search trees with lazy finger. In *International Symposium on Algorithms and Computation*, pages 181–192. Springer, 2014.

4 Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak. Pinning down the strong wilber 1 bound for binary search trees. *APPROX*, 2020.

5 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 410–423. IEEE, 2015.

6 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Self-adjusting binary search trees: What makes them tick? In *Algorithms-ESA 2015*, pages 300–312. Springer, 2015.

7 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. The landscape of bounds for binary search trees. *arXiv preprint*, 2016. `arXiv:1603.04892`.

8 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Multi-finger binary search trees. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

9 Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991. `doi:10.1137/0220008`.

10 Richard Cole. On the dynamic finger conjecture for splay trees. part ii: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000.

11 Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. On the dynamic finger conjecture for splay trees. part i: Splay sorting log n-block sequences. *SIAM Journal on Computing*, 30(1):1–43, 2000.

12 Erik D Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Patraşcu. The geometry of binary search trees. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. SIAM, 2009.

13 Erik D Demaine, Dion Harmon, John Iacono, and Mihai Patraşcu. Dynamic optimality—almost. *SIAM Journal on Computing*, 37(1):240–251, 2007.

14 John Iacono and Stefan Langerman. Weighted dynamic finger in binary search trees. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 672–691. SIAM, 2016.

15 Victor Lecomte and Omri Weinstein. Settling the relationship between wilber's bounds for dynamic optimality. *arXiv preprint*, 2019. `arXiv:1912.02858`.

16 Caleb Levy and Robert Tarjan. A new path from splay to dynamic optimality. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1311–1330. Society for Industrial and Applied Mathematics, 2019.

17 Joan Marie Lucas. *Canonical forms for competitive binary search tree algorithms*. Rutgers University, Department of Computer Science, Laboratory for Computer . . . , 1988.

18 J Ian Munro. On the competitiveness of linear search. In *European Symposium on Algorithms*, pages 338–345. Springer, 2000.

19 Seth Pettie. Splay trees, davenport-schinzel sequences, and the deque conjecture. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1115–1124. Society for Industrial and Applied Mathematics, 2008.

20 Seth Pettie. Applications of forbidden 0–1 matrices to search tree and path compression-based data structures. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, pages 1457–1467. SIAM, 2010.

21 Luís MS Russo. A study on splay trees. *Theoretical Computer Science*, 2018.

22 Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

23 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985.

24 Robert Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM journal on Computing*, 18(1):56–67, 1989.

More on Change-Making and Related Problems

Timothy M. Chan
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
tmc@illinois.edu

Qizheng He
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
qizheng6@illinois.edu

Abstract

Given a set of n integer-valued coin types and a target value t, the well-known *change-making* problem asks for the minimum number of coins that sum to t, assuming an unlimited number of coins in each type. In the more general *all-targets* version of the problem, we want the minimum number of coins summing to j, for every $j = 0, \ldots, t$. For example, the textbook dynamic programming algorithms can solve the all-targets problem in $O(nt)$ time. Recently, Chan and He (SOSA'20) described a number of $O(t \operatorname{polylog} t)$-time algorithms for the original (single-target) version of the change-making problem, but not the all-targets version.

In this paper, we obtain a number of new results on change-making and related problems:

- We present a new algorithm for the all-targets change-making problem with running time $\tilde{O}(t^{4/3})$, improving a previous $\tilde{O}(t^{3/2})$-time algorithm.
- We present a very simple $\tilde{O}(u^2 + t)$-time algorithm for the all-targets change-making problem, where u denotes the maximum coin value. The analysis of the algorithm uses a theorem of Erdős and Graham (1972) on the Frobenius problem. This algorithm can be extended to solve the all-capacities version of the *unbounded knapsack* problem (for integer item weights bounded by u).
- For the original (single-target) coin changing problem, we describe a simple modification of one of Chan and He's algorithms that runs in $\tilde{O}(u)$ time (instead of $\tilde{O}(t)$).
- For the original (single-capacity) unbounded knapsack problem, we describe a simple algorithm that runs in $\tilde{O}(nu)$ time, improving previous near-u^2-time algorithms.
- We also observe how one of our ideas implies a new result on the *minimum word break* problem, an optimization version of a string problem studied by Bringmann et al. (FOCS'17), generalizing change-making (which corresponds to the unary special case).

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Coin changing, knapsack, dynamic programming, Frobenius problem, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.29

Funding *Timothy M. Chan*: Supported in part by NSF Grant CCF-1814026.

Acknowledgements We thank Adam Polak and Chao Xu for discussion and, in particular, for bringing the minimum word break problem to our attention.

1 Introduction

In the *change-making* problem (also known as *coin changing*), a set of n positive-integer-valued coin types is given, and the cashier wants to use the minimum number of coins to sum to a target value t exactly, where the number of coins in each type can be used an unlimited number of times. This is a well-known textbook problem, which is weakly NP-hard [20], and standard solutions using dynamic programming [27] have $O(nt)$ running time.

Change-making is closely related to another textbook problem, *subset sum* (the differences are that in subset sum, each item may be used at most once and there is no objective function to minimize). A series of work in the last few years [5, 16, 15, 17] have given improved algorithms for subset sum, using convolution (FFT). Very recently, at SOSA'20, Chan and He [8] revisited the change-making problem and described a number of $O(t \operatorname{polylog} t)$-time algorithms, using FFT; their fastest deterministic and randomized algorithms have $O(t \log t \log\log t)$ and $O(t \log t)$ running time respectively.

All-targets change-making. In this paper, we consider a more general, *all-targets* version of the change-making problem: the aim is to compute, for each target value $j = 0, \ldots, t$, the minimum number of coins that can be used to sum to j exactly. This version of the problem is equally natural. For instance, the standard $O(nt)$-time dynamic programming algorithms are actually designed to solve this more general version. Some of the newer subset-sum algorithms [5, 16, 17, 15] also solved the analogous all-targets version of subset sum, but in contrast, Chan and He's algorithms for change-making do *not* work for the all-targets version.

The best previous result for the all-targets change-making problem that we are aware of was an $\tilde{O}(t^{3/2})$-time[1] algorithm by Karl Bringmann and Tomasz Kociumaka (2019), cited as a personal communication (and briefly sketched) in a very recent paper by Lincoln, Polak, and Vassilevska Williams (ITCS'20) [19]. Lincoln et al.'s paper gave a web of fine-grained reductions connecting a variety of problems, including a reduction from all-targets change-making to the "monochromatic convolution" problem, the latter of which is shown to have near $n^{3/2}$ time complexity iff 3SUM has near quadratic time complexity. Their work implicitly hints at the possibility that the all-targets change-making problem might have near $t^{3/2}$ complexity as well, but the reduction is in the opposite direction.

Our first result is an $\tilde{O}(t^{4/3})$-time algorithm for the all-targets change-making problem, interestingly beating $t^{3/2}$ and placing the problem in a different category than monochromatic convolution and all its surrounding problems. Our algorithm is conceptually simple, exploiting an easy lemma on a binary special case of $(\min, +)$-convolution (using FFTs).

All-targets change-making in terms of u. Next, we consider the complexity of the all-targets change-making problem in terms of some other natural parameters besides n and t: specifically,

- the largest coin value, denoted by u;
- the sum of the n given coin values, denoted by σ.

Some prior works have analyzed algorithms in terms of u and σ for the subset sum problem [22, 16]. A few recent papers have also analyzed algorithms in terms of u for the *0-1 knapsack* and the *unbounded knapsack* problem [2, 4, 12, 14, 24]. The unbounded knapsack problem is particularly relevant: given integer weights $w_1, \ldots, w_n$ and profits $p_1, \ldots, p_n$ and capacity value t, find nonnegative integers $m_1, \ldots, m_n$ to maximize $\sum_i m_i p_i$ such that $\sum_i m_i w_i \leq t$. Change-making is a special case, for example, by setting $w_i = v_i$ and $p_i = Mv_i - 1$ for a sufficiently large M. Improving some previous algorithms [4, 24], Axiotis and Tzamos (ICALP'19) [2] and Jansen and Rohwedder (ITCS'19) [14] independently described algorithms[2] for unbounded knapsack running in $\tilde{O}(u^2)$ time with $u := \max_i w_i$ (the

[1] The $\tilde{O}$ notation hides polylogarithmic factors.

[2] We found that an $\tilde{O}(u^2)$ algorithm (basically the same as Axiotis and Tzamos') appeared earlier in a commentary on a 2016 programming contest problem by Arthur Nascimento, solved by Yan Soares Couto; see Problem L of `https://www.ime.usp.br/~maratona/assets/seletivas/2016/comentarios.pdf`.

time bound can be reduced slightly to $O(u^2/2^{\Theta(\sqrt{\log u})})$ by using known slightly subquadratic algorithms for $(\min,+)$-convolution [26]). However, these algorithms do not solve the all-targets or all-capacities version (computing the optimal profit for every capacity $j = 0, \ldots, t$).[3]

For the all-targets version of change-making, it is not difficult to obtain an $O(u^3 + t)$-time algorithm, based on a known observation that when the target is sufficiently large, it is always advantageous to use the largest coin. We describe a new algorithm that improves the running time to $O(u^2 \log u + t)$. Note that the algorithm is optimal for large $t \gg u^2 \log u$, since the output size for the all-targets problem is $\Omega(t)$.

The new algorithm is remarkably simple – just a slight variation of one of the standard dynamic programming solutions, with a 3-line pseudocode! (See page 8.) It is easily implementable and does not require FFT. However, the correctness argument is far from obvious, and requires a nice application of a number-theoretic theorem by Erdős and Graham [13] on the *Frobenius problem* (about the smallest target value that cannot be represented by a coin system). Arguably, algorithms that are simple but nontrivial to analyze are the most interesting kinds of algorithms.

All-capacities unbounded knapsack in terms of u. Our algorithm can be easily modified to solve the unbounded knapsack problem in the all-capacities version, with the same $O(u^2 \log u + t)$ time bound. This also implies an $O(u^2 \log u)$-time algorithm for the single-capacity version, which is a bit simpler than the previous $\tilde{O}(u^2)$ algorithms [2, 14] (in addition to extending it to all-capacities). For unbounded knapsack, a nearly matching conditional lower bound is known [9, 18]: more precisely, if single-capacity unbounded knapsack could be solved in truly subquadratic time for instances with $t, u = \Theta(n)$, then so could $(\min,+)$-convolution.

In terms of σ. We describe a variant of our algorithm with time bound $\tilde{O}((t\sigma)^{2/3} + t)$ for the all-targets change-making or all-capacities unbounded knapsack problem. Note that if $\sigma \ll t$, this is better than our earlier $\tilde{O}(t^{4/3})$ bound for the all-targets change-making.

Single-target change-making. For the single-target (original) change-making problem, we also describe how to improve the running time of one of Chan and He's FFT-based algorithms [8] from $\tilde{O}(t)$ to $\tilde{O}(u)$, which is faster than applying the previous $\tilde{O}(u^2)$-time algorithms [2, 14] for single-capacity unbounded knapsack.

Single-capacity unbounded knapsack. For the single-capacity (original) unbounded knapsack problem, we also describe a simple algorithm with running time $\tilde{O}(nu)$, which (ignoring $n^{o(1)}$ factors) simultaneously improves the standard $O(nt)$-time dynamic programming algorithm and the previous $\tilde{O}(u^2)$-time algorithms [2, 14] (since $u \le t$ without loss of generality, and $n \le u$ after pruning unnecessary items). There was a previous $O(nu)$-time algorithm by Pisinger [22] for subset sum, but not for unbounded knapsack.

Minimum word break. Finally, we consider a generalization of the problem for strings, known as the *minimum word break* problem: Given a string s with length n and a set D of strings (a "dictionary" of "words") with total length m, express s as a concatenation of words

[3] Cygan et al. [9] refered to the all-capacities version as Unbounded-Knapsack$^+$; Kunnemann et al. [18] called it the *output-intensive* version.

from D, using the smallest number of words, where a word may be used multiple times. It is easy to see that if the alphabet is unary, then the problem is the same as change-making (the single-target version, with n and m corresponding to t and σ). A straightforward dynamic programming algorithm runs in $\tilde{O}(nd + m)$ time, where d denotes the number of distinct lengths among the words in D, by using randomized fingerprints [3] (which can be made deterministic [28]). Because $m \geq \frac{d(d+1)}{2}$, the bound is $\tilde{O}(n\sqrt{m} + m)$.

The decision version of the problem – deciding whether a solution exists, without minimizing the number of words – was considered by Bringmann, Grønlund, and Larsen (FOCS'17) [6], who gave an $\tilde{O}(nm^{1/3} + m)$-time algorithm, using FFT (improving a previous algorithm by Backurs and Indyk [3] with running time $\tilde{O}(nm^{1/2-1/18} + m)$). Bringmann et al. also proved a nearly matching conditional lower bound for combinatorial algorithms, assuming the conjecture that k-clique requires near n^k time for combinatorial algorithms. However, they did not obtain results on the *minimum* word break problem: part of the difficulty is that for the optimization problem, the various convolution operations needed change to $(\min, +)$-convolutions, which appear to be more expensive.

Nevertheless, we note that Bringmann et al.'s algorithm can still be adapted to solve the minimum word break problem. In fact, the time bound $\tilde{O}(nm^{1/3} + m)$ remains the same. This shows that surprisingly the optimization problem is not harder but has the same fine-grained complexity as the decision problem (at least for combinatorial algorithms, assuming the k-clique conjecture). The only new ingredient in our adaptation of Bringmann et al.'s algorithm is the same lemma on $(\min, +)$-convolutions that we have used in our $\tilde{O}(t^{4/3})$ algorithm for change-making.

2 Preliminaries

The all-targets version of the change-making problem can be formally defined as follows:

▶ **Problem 1** (All-Targets Change-Making). *Given a set $V = \{v_1, \ldots, v_n\}$ of n positive integers (coin values) and an integer t, for each $j = 0, \ldots, t$, find the size of the smallest multiset S (duplicates allowed) of coin values from V such that S sums to exactly j, i.e., find the minimum m_j^* of $\sum_{i=1}^n m_i$ subject to the constraint that $\sum_{i=1}^n m_i v_i = j$, where $m_i \in \mathbb{N}$.*

Besides n (the number of coin values) and t (the maximum target value), we introduce two more parameters: let $u = \max_{i=1}^n v_i$ denote the maximum coin value, and $\sigma = \sum_{i=1}^n v_i$ denote the sum of input coin values. Simple observation reveals some inequalities relating the parameters: we have $n = O(\sqrt{\sigma})$ (because the distinctness of the v_i's implies $\sigma \geq \frac{n(n+1)}{2}$), $n \leq u$, $u \leq t$ (without loss of generality), and $\sigma \leq nu$. Note that unlike in the subset sum problem, t may be smaller or larger than σ.

Boolean convolution. The Boolean convolution $A \circ B$ of two Boolean arrays $A[0, \ldots, t_1]$ and $B[0, \ldots, t_2]$ is a Boolean array with $t_1 + t_2 + 1$ elements, where $(A \circ B)[j] = \bigvee_{j'=0}^{t_1} (A[j'] \wedge B[j - j'])$ (we assume out-of-range values are 0).

Change-making is closely related with Boolean convolution. For any integer k, let $C_V^{(k)}[0, \ldots, t]$ denote the Boolean array where

$C_V^{(k)}[j] = 1$ iff there exist k coins from V with their sum being j.

Then $C_V^{(k)}$ can be obtained from the first $t + 1$ elements of $C_V^{(k_1)} \circ C_V^{(k_2)}$, for any $k_1, k_2 > 0$ where $k = k_1 + k_2$.

The Boolean convolution of two arrays of size $O(t)$ can be computed in $O(t \log t)$ time by FFT.

$(\min,+)$-convolution. The $(\min,+)$-convolution $A \star B$ of two arrays $A[0,\ldots,t_1]$ and $B[0,\ldots,t_2]$ is an array with t_1+t_2+1 elements, where $(A\star B)[j] = \min_{j'=0}^{t_1}(A[j']+B[j-j'])$ (we assume out-of-range values are ∞).

Change-making is also related to $(\min,+)$-convolution. For a set V of coin values, let $D_V[0,\ldots,t]$ denote the array where

$$D_V[j] = \text{the minimum number of coins from } V \text{ needed to sum to } j$$

(if no solution exists, $D_V[j]=\infty$). Then $D_{V_1\cup V_2}$ can be obtained from the first $t+1$ elements of $D_{V_1} \star D_{V_2}$.

It has been conjectured by some researchers that $(\min,+)$-convolution cannot be solved in truly subquadratic time (e.g., see [9, 18]). However, the following lemma shows that a subquadratic algorithm is possible for the special case of $(\min,+)$-convolution where the second array is "binary", i.e., all entries of B are in $\{1,\infty\}$. This "trick" is not new and is known before, for example, in the context of matrix multiplication (for computing the $(\min,+)$-product when one of the matrices is binary [25, 11, 7]), with the basic idea tracing back to Matoušek's dominance algorithm [21].

▶ **Lemma 1.** *Given two arrays $A[0,\ldots,t]$ and $B[0,\ldots,t]$ where all entries of B are in $\{1,\infty\}$, we can compute the $(\min,+)$-convolution of A and B in $\tilde{O}(t^{3/2})$ time.*

Furthermore, if we just want t' user-specified entries of the $(\min,+)$-convolution, the time bound may be reduced to $\tilde{O}(t\sqrt{t'})$.

Proof. By sorting and replacing elements by their ranks, we may assume the values of A are in $[t]$, and are distinct (without loss of generality). Divide the range $[t]$ into $\sqrt{t'}$ subintervals of length $t/\sqrt{t'}$. For each such subinterval I, define a Boolean array A'_I with $A'_I[j]=1$ iff $A[j]\in I$, and define a Boolean array B' with $B'[j]=1$ iff $B[j]\neq\infty$; compute the Boolean convolution between A'_I and B'; this requires $\sqrt{t'}$ FFTs and takes $\tilde{O}(t\sqrt{t'})$ time. Then for each index j for which we want to compute the output entry, we can now identify which subinterval contains the minimum answer (namely, the smallest subinterval I such that $(A'_I \circ B')[j]$ is true) in $O(\sqrt{t'})$ time, so we can do a brute-force search in $O(t/\sqrt{t'})$ time; the total time for t' output entries is $O(t'\cdot(\sqrt{t'}+t/\sqrt{t'})) = O(t\sqrt{t'})$. ◀

3 $\tilde{O}(t^{4/3})$ Algorithm

Previous algorithm. Before presenting the new algorithm, we first give a sketch on the previous $\tilde{O}(t^{3/2})$-time algorithm by Bringmann and Kociumaka (as mentioned in [19]). Let ℓ_0 be a parameter to be chosen later. Let $H=\{v_i : v_i > \ell_0\}$ be the set of all *heavy* coin values, and let $L=\{v_i : v_i \le \ell_0\}$ be the set of all *light* coin values. Because the coin values are distinct, $|L|\le \ell_0$. To sum to any value $j\le t$, we can use at most t/ℓ_0 heavy coins. We use Boolean convolution to compute the array $C_H^{(k)}$ from $C_H^{(k-1)}$ for each $k=1,\ldots,\lfloor t/\ell_0\rfloor$. The total time for these $\lfloor t/\ell_0\rfloor$ convolutions is $\tilde{O}(t^2/\ell_0)$. We can thus obtain $D_H[j]$ by taking the minimum $k\le t/\ell_0$ such that $C_H^{(k)}[j]>0$. To finish, we use the classical dynamic programming algorithm to add the light coins. Namely, for each $j=1,\ldots,t$, we set $D_V[j]=\min\{D_H[j], \min_{v_i\in L} D_V[j-v_i]+1\}$. This step takes $O(\ell_0 t)$ time. The overall running time is

$$\tilde{O}\left(\frac{t^2}{\ell_0}+\ell_0 t\right).$$

To balance cost, we choose $\ell_0=\sqrt{t}$ and obtain a time bound of $\tilde{O}(t^{3/2})$.

New algorithm. To improve the running time, we describe a more efficient way to add the light coins, by using $(\min,+)$-convolution. As before, we first compute D_H for the heavy coins in $\tilde{O}(t^2/\ell_0)$ time. Initialize S to H.

Now, consider a fixed value $\ell \le \ell_0/2$, and consider the subset of light coins $L_\ell = \{v_i : v_i \in (\ell, 2\ell]\}$. In order to add L_ℓ to S, we need to compute $D_{S\cup L_\ell}$ from D_S. Naively, one could perform a single $(\min,+)$-convolution of D_S with D_{L_ℓ}, but this is expensive, and D_{L_ℓ} is not known yet (and is not binary). A better approach is to do *multiple* $(\min,+)$-convolutions by dividing the array into smaller blocks of size $O(\ell)$, as follows:

For each $i = 0, \ldots, t/\ell$, we compute $D_{S\cup L_\ell}[\ell i, \ldots, \ell(i{+}1)]$ by taking a $(\min,+)$-convolution D' of $D_{S\cup L_\ell}[\ell(i-2), \ldots, \ell i]$ with a binary array $B[\ell, \ldots, 2\ell]$ using Lemma 1, where $B[j] = 1$ if $j \in L_\ell$, and $B[j] = \infty$ otherwise. Then $D_{S\cup L_\ell}[\ell i, \ldots, \ell(i+1)]$ is the entry-wise minimum of $D'[\ell i, \ldots, \ell(i+1)]$ and $D_S[\ell i, \ldots, \ell(i+1)]$. Each of the above $O(t/\ell)$ $(\min,+)$-convolutions is done to arrays of size $O(\ell)$ (after shifting indices). Thus, the total running time is $\tilde{O}((t/\ell)\cdot \ell^{3/2}) = \tilde{O}(\sqrt{\ell} t)$.

We repeat the above steps for all ℓ's that are powers of 2 and smaller than ℓ_0, until all coin values are added to S. This requires $O(\log \ell_0)$ rounds, and the total running time forms a geometric series bounded by $\tilde{O}(\sqrt{\ell_0} t)$. The overall running time is

$$\tilde{O}\left(\frac{t^2}{\ell_0} + \sqrt{\ell_0} t\right).$$

To balance cost, we choose $\ell_0 \approx t^{2/3}$ and obtain a time bound of $\tilde{O}(t^{4/3})$.

▶ **Theorem 2.** *The all-targets change-making problem can be solved in $\tilde{O}(t^{4/3})$ time.*

▶ Remark. If we choose $\ell_0 = u$ instead, the heavy coin case can be ignored and we obtain an $\tilde{O}(t\sqrt{u})$-time algorithm, which is faster for small u. We will give still faster algorithms for small u in the next section.

4 $O(u^2 \log u + t)$ Algorithm

We now explore more algorithms with running time sensitive to u.

Warm-up. We first observe that there is a simple algorithm with $O(u^3 + t)$ running time. We use the following lemma, which is "folklore":[4]

▶ **Lemma 3.** *For any target value $j \ge u^2$, any optimal solution to the change-making problem must use the largest coin value u.*

Proof. Suppose that an optimal solution X for a target value j does not use the coin value u.

A simple argument shows that $j < u^3$: If X uses a coin value v_i at least u times, we can replace u copies of v_i with v_i copies of u, and the number of coins in X would decrease: a contradiction. Thus, each of the at most u coin values is used fewer than u times, and so the sum of X must be less than u^3.

[4] Bateni et al. [4, Lemma 7.2] gave a proof for the (more general) unbounded knapsack problem, using the pigeonhole principle, similar to what we give here (Eisenbrand and Weismantel [12] also proved a similar statement for higher-dimensional unbounded knapsack). But it was known much earlier: we personally learned of the pigeonhole proof for coin changing from comments by Bruce Merry in 2006 on a US Olympiad question (`https://contest.usaco.org/TESTDATA/DEC06.fewcoins.htm`), and the same pigeonhole proof for unbounded knapsack from a Chinese web post in 2016 (`https://www.zhihu.com/question/27547892/answer/133582594`).

We give a better argument showing $j < u^2$ by using the pigeonhole principle: Let $\langle x_1, \ldots, x_h \rangle$ be the sequence of coins used in X, with duplicates included, in an arbitrary order. Define the prefix sum $s_i = x_1 + \cdots + x_i$. Suppose $h \geq u$. By the pigeonhole principle, there must exist $0 \leq i < j \leq h$ with $s_i \equiv s_j \pmod{u}$. Then the subsequence $x_{i+1}, \ldots, x_j$ sums to a number divisible by u. We can replace this subsequence with some number of copies of u, and the number of coins in X would decrease (since u is the largest coin value): a contradiction. Thus $h < u$, and so the sum of X is less than u^2. ◀

The above lemma ensures that it is sufficient to compute $D_V[j]$ for all $j < u^2$; by the naive dynamic programming algorithm, this step takes $O(nu^2) \leq O(u^3)$ time. Afterwards, for $j = u^2, \ldots, t$, we can simply set $D_V[j] = D_V[j-u] + 1$; this step takes $O(t)$ time. We thus get the time bound $O(u^3 + t)$.

If in the first part we instead use the $\tilde{O}(t\sqrt{u})$-time algorithm in the remark after Theorem 2 (with t replaced by u^2), then the first part takes $\tilde{O}(u^2\sqrt{u})$ time. The total time is then reduced to $O(u^{2.5} \operatorname{polylog} u + t)$. (This requires FFT, however.)

New algorithm. To improve the running time further, we use number-theoretic results on the *Frobenius problem*, which has received much attention from mathematicians: given k positive integer coin values $v_1 > \cdots > v_k$ with $\gcd(v_1, \ldots, v_k) = 1$, what is the largest number that cannot be represented? For $k = 2$, classical results show that the number is exactly $v_1 v_2 - v_1 - v_2$. For $k \geq 3$, the problem becomes much more challenging, for which there are no closed-form formulas. In 1972, Erdős and Graham [13] proved an upper bound of $2\left\lfloor \frac{v_1}{k} \right\rfloor v_2 - v_1$, which will be useful in our algorithmic application:

▶ **Lemma 4** (Erdős–Graham). *Given integers $v_1 > \ldots > v_k > 0$ ($k \geq 2$) with $\gcd(v_1, \ldots, v_k) = 1$, any integer greater than $2\left\lfloor \frac{v_1}{k} \right\rfloor v_2 - v_1$ can be expressed as a nonnegative integer linear combination of $v_1, \ldots, v_k$.*

In terms of $u = \max_i v_i$, Erdős and Graham's bound is $O(u^2/k)$, which is known to be tight in the worst case, within a constant factor (see [10] for improvements on the constant factor). For constant k, the bound remains quadratic, as in the 2-coins case. In our algorithmic application, we will consider non-constant k – here, the k in the denominator will prove crucial.

First, let us restate the bound more generally without assuming $\gcd(v_1, \ldots, v_k) = 1$:

▶ **Corollary 5.** *Given integers $v_1 > \cdots > v_k > 0$ ($k \geq 2$) with $\gcd(v_1, \ldots, v_k) = d$, any integer that is greater than $2\left\lfloor \frac{v_1}{dk} \right\rfloor v_2 - v_1$ and is divisible by d can be expressed as a nonnegative integer linear combination of $v_1, \ldots, v_k$.*

Proof. Apply Lemma 4 to the numbers $v_1/d, \ldots, v_k/d$. The bound becomes $\left(2\left\lfloor \frac{v_1/d}{k} \right\rfloor v_2/d - v_1/d\right) \cdot d$. ◀

We use Corollary 5 to prove a more refined version of Lemma 3, which takes into account the k largest coin values instead of just the largest value:

▶ **Lemma 6.** *Let $v_1, \ldots, v_k \leq u$ be the k largest input coin values. For any target value $j \geq 2u^2/k$, any optimal solution to the change-making problem must use at least one coin from $\{v_1, \ldots, v_k\}$.*

Proof. We may assume $k \geq 2$ (because of Lemma 3). Let $d = \gcd(v_1, \ldots, v_k)$. Suppose that an optimal solution X for a target value j does not use any coins from $\{v_1, \ldots, v_k\}$.

Consider the sequence of coins used in X, with duplicates included, in an arbitrary order. Divide the sequence into subsequences $X_1, \ldots, X_h$, each of which has sum in $(\frac{2u^2}{dk} - u, \frac{2u^2}{dk}]$, except that the last has sum at most $\frac{2u^2}{dk} - u$. Suppose $h > d$. Define s_i to be the sum of the concatenation of $X_1, \ldots, X_i$. By the pigeonhole principle, there exist $0 \le i < j < h$ with $s_i \equiv s_j \pmod{d}$. Then the subsequence formed by concatenating $X_{i+1}, \ldots, X_j$ sums to a number divisible by d and greater than $\frac{2u^2}{dk} - u$. By Corollary 5, we can replace this subsequence with coins from the set $\{v_1, \ldots, v_k\}$, and the number of coins in X would decrease (since $v_1, \ldots, v_k$ have larger values): a contradiction. Thus $h \le d$, and so the sum of X is less than $d \cdot \frac{2u^2}{dk} = 2u^2/k$. ◀

Thus, the optimal solution for target value j must use at least one coin value which is among the $\lceil 2u^2/j \rceil$ largest. This leads to the following extremely simple algorithm, which is just a small modification to the standard dynamic programming algorithm (no FFT required):

Algorithm 1 All-targets change-making.

1: Sort $v_1, \ldots, v_n$ in decreasing order, and set $D_V[0] = 0$.
2: **for** $j = 1, \ldots, t$ **do**
3: Set $D_V[j] = \min_{1 \le i \le \lceil 2u^2/j \rceil : v_i \le j} D_V[j - v_i] + 1$.

The total running time is bounded by a Harmonic series:

$$O\left(\sum_{j=1}^{t} \left(\frac{u^2}{j} + 1\right)\right) = O(u^2 \log u + t).$$

▶ **Theorem 7.** *The all-targets change-making problem can be solved in $O(u^2 \log u + t)$ time.*

As a corollary of the above algorithm, we can also obtain an algorithm with running time sensitive to σ, the total sum of the input coin values: Define the heavy coins H and light coins L as before, with respect to a parameter ℓ_0 to be chosen later. We first compute D_L for the light coins by the above algorithm in $\tilde{O}(\ell_0^2 + t)$ time. Then we add the heavy coins by dynamic programming: $D_V[j] = \min\{D_L[j], \min_{v_i \in H} D_V[j - v_i] + 1\}$. Since there are at most σ/ℓ_0 heavy coins, this step takes $O(\sigma/\ell_0 \cdot t)$ time. The overall running time is

$$\tilde{O}\left(\ell_0^2 + \frac{t\sigma}{\ell_0} + t\right).$$

To balance cost, we choose $\ell_0 = (t\sigma)^{1/3}$ and obtain the time bound $\tilde{O}((t\sigma)^{2/3} + t)$. (Again, no FFT is required.)

▶ **Corollary 8.** *The all-targets change-making problem can be solved in $\tilde{O}((t\sigma)^{2/3} + t)$ time.*

▶ Remark. The $O(t)$ term can be eliminated in Theorem 7 (and thus Corollary 8) if we are fine with an *implicit* representation of the output (i.e., a structure that allows us to return the answer for any given target in constant time), since by Lemma 3, we can first reduce the target j to below u^2 by using some number (i.e., $\max\{\lceil (j - u^2)/u \rceil, 0\}$) of copies of the largest coin value u.

5 All-Capacities Unbounded Knapsack

We note that the algorithm in the preceding section can be extended to solve the all-capacities version of the unbounded knapsack problem, defined as follows:

▶ **Problem 2** (All-Capacities Unbounded Knapsack). *Given n items where the i-th item has a positive integer weight w_i and a positive profit p_i, and given an integer t, for each $j = 0, \ldots, t$, find the maximum total profit of a multiset of items such that the total weight is at most j, i.e., find the maximum of $\sum_{i=1}^{n} m_i p_i$ subject to the constraint that $\sum_{i=1}^{n} m_i w_i \le j$, where $m_i \in \mathbb{N}$.*

Like before, let $u = \max_{i=1}^{n} w_i$ and $\sigma = \sum_{i=1}^{n} w_i$. We may assume that the weights are distinct (since if there are two items with the same weight, we may remove the one with the smaller profit).

We use the following analog to Lemma 6:

▶ **Lemma 9.** *Suppose items $1, \ldots, k$ have the k largest profit-to-weight ratios. For any capacity value $j \ge 3u^2/k$, any optimal solution to the unbounded knapsack problem must use at least one item from $\{1, \ldots, k\}$.*

Proof. Similar to the proof of Lemma 6, since replacing a subsequence with items that have larger profit-to-weight ratios while maintaining the same total weight would increase the total profit. One difference in the unbounded knapsack problem is that the total weight in the optimal solution may not be exactly j. But it must be at least $j - u$ (otherwise, we could add one more item to get a better solution). When $j \ge 3u^2/k$, we have $j - u \ge 2u^2/k$. ◀

The same analysis shows correctness of the following very simple algorithm, which runs in $O(u^2 \log u + t)$ time:

Algorithm 2 All-capacities unbounded knapsack.

```
Sort the items in decreasing order of p_i/w_i.
for j = 0, ..., t do
    Set D[j] = max{0, max_{1≤i≤⌈3u²/j⌉: w_i≤j} (D[j − w_i] + p_i)}.
```

The $\tilde{O}((t\sigma)^{2/3} + t)$ algorithm can be extended as well.

▶ **Corollary 10.** *The all-capacities unbounded knapsack problem can be solved in $O(u^2 \log u + t)$ or $\tilde{O}((t\sigma)^{2/3} + t)$ time.*

▶ Remark. As before, the $O(t)$ term can be eliminated with an implicit representation of the output (since by an analog to Lemma 3, we can first reduce the capacity to below u^2 by using some number of copies of the item with the largest profit-to-weight ratio). In particular, for the single-capacity version, we obtain a very simple $O(u^2 \log u)$-time algorithm.

The algorithm works even when the profits are reals but the weights are integers. Alternatively, a variant of the algorithm works when the weights are reals but the profits are integers: the same time bound $O(u^2 \log u)$ holds but with $u = \max_{i=1}^{n} p_i$. Here, we recast the problem as minimizing $\sum_{i=1}^{n} m_i w_i$ subject to the constraint that $\sum_{i=1}^{n} m_i p_i \ge j$, and modify the algorithm appropriately (applying Erdős–Graham to the profits instead of the weights). From the implicitly represented output, we can determine the answer for any given capacity by predecessor search.

6 $\tilde{O}(u)$ Algorithm for Single-Target Change-Making

In this section, we present an $\tilde{O}(u)$-time algorithm for the single-target change-making problem. It is obtained by modifying the third algorithm of Chan and He [8], which originally ran in $O(t \log^2 t)$ time. They first solved the decision problem: deciding whether we can sum to t using at most m coins for a given value m. By adding 0 to the input set of coin values, "at most m" can be changed to "exactly m".

Their decision algorithm relies on the following partition lemma, which shows the multiset of coins S can be almost evenly partitioned simultaneously in terms of cardinality *and* the total value (see [8] for a short self-contained proof for the even case; the odd case is similar):

▶ **Lemma 11** (Partition Lemma). *Suppose S is a multiset with $|S| = m$ and $\sigma(S) = t$. If m is odd, then there exists a partition of S into three parts S_1, S_2 and a singleton $\{s_0\}$, such that $|S_1| = |S_2| = \frac{m-1}{2}$ and $\sigma(S_1), \sigma(S_2) \le \frac{t}{2}$.*

If m is even, then there exists a partition of S into three parts S_1, S_2 and two singletons $\{s_0, s_1\}$, such that $|S_1| = |S_2| = \frac{m}{2} - 1$, and $\sigma(S_1), \sigma(S_2) \le \frac{t}{2}$.

Notice that since the maximum coin value is u, we also have $\sigma(S_1), \sigma(S_2) \ge \frac{t}{2} - 2u$ (as we take out one or two coins).

The Partition Lemma suggests a simple recursive algorithm to compute $C_V^{(m)}[0, \dots, t]$: we just take the first $t+1$ entries of

$$\begin{cases} C_V^{(\frac{m-1}{2})}[0, \dots, \frac{t}{2}] \circ C_V^{(\frac{m-1}{2})}[0, \dots, \frac{t}{2}] \circ C_V^{(1)}[0, \dots, t] & \text{if } m \text{ is odd,} \\ C_V^{(\frac{m}{2}-1)}[0, \dots, \frac{t}{2}] \circ C_V^{(\frac{m}{2}-1)}[0, \dots, \frac{t}{2}] \circ C_V^{(1)}[0, \dots, t] \circ C_V^{(1)}[0, \dots, t] & \text{if } m \text{ is even.} \end{cases}$$

That was essentially Chan and He's previous algorithm.

We describe a more efficient recursive algorithm to compute a smaller subarray $C_V^{(m)}[t - 4u, \dots, t]$: we just take the relevant entries of

$$\begin{cases} C_V^{(\frac{m-1}{2})}[\frac{t-4u}{2} - 2u, \dots, \frac{t}{2}] \circ C_V^{(\frac{m-1}{2})}[\frac{t-4u}{2} - 2u, \dots, \frac{t}{2}] \circ C_V^{(1)}[0, \dots, u] & \\ \qquad\qquad \text{if } m \text{ is odd,} & \\ C_V^{(\frac{m}{2}-1)}[\frac{t-4u}{2} - 2u, \dots, \frac{t}{2}] \circ C_V^{(\frac{m}{2}-1)}[\frac{t-4u}{2} - 2u, \dots, \frac{t}{2}] \circ C_V^{(1)}[0, \dots, u] \circ C_V^{(1)}[0, \dots, u] & \\ \qquad\qquad \text{if } m \text{ is even.} & \end{cases}$$

Each of the above Boolean convolutions is done to arrays of size $O(u)$ (after shifting indices), and thus takes $O(u \log u)$ time. The subarrays $C_V^{(\frac{m-1}{2})}[\frac{t-4u}{2} - 2u, \dots, \frac{t}{2}] = C_V^{(\frac{m-1}{2})}[\frac{t}{2} - 4u, \dots, \frac{t}{2}]$ and $C_V^{(\frac{m}{2}-1)}[\frac{t-4u}{2} - 2u, \dots, \frac{t}{2}] = C_V^{(\frac{m}{2}-1)}[\frac{t}{2} - 4u, \dots, \frac{t}{2}]$ can be computed by recursion. Thus, the running time satisfies the recurrence

$$T(m, t) = T(\lfloor \tfrac{m-1}{2} \rfloor, \tfrac{t}{2}) + O(u \log u),$$

which solves to $T(m, t) = O(u \log u \log t)$.

The decision problem can now be solved by inspecting the entry $C_V^{(m)}[t]$. We can find the optimal number of coins by binary search with $O(\log t)$ calls to the decision algorithm. By Lemma 3, we can first reduce t to below u^2 by repeatedly using the largest coin value. Therefore, the total running time is $O(u \log u \log^2 t) \le O(u \log^3 u)$.

▶ **Theorem 12.** *The single-target change-making problem can be solved in $O(u \log^3 u)$ time.*

▶ Remark. The above algorithm shares some similarity with the $\tilde{O}(u^2)$ algorithm by Axiotis and Tzamos [2] for unbounded knapsack, which also involves logarithmically many convolutions on subarrays of size $O(u)$, except that they used $(\min, +)$-convolutions and a more naive parititioning that approximately halves t, but not m. In contrast, the above Partition Lemma is crucial to our faster algorithm for change-making.

7 $\tilde{O}(nu)$ Algorithm for Single-Capacity Unbounded Knapsack

In this section, we revisit the standard (single-capacity) version of the unbounded knapsack problem and present a new $\tilde{O}(nu)$-time algorithm (recall that $u = \max_i w_i$). This algorithm is simple (no FFT needed), and is based on the following combinatorial lemma, which is obtained by another pigeonhole argument:

▶ **Lemma 13.** *For the unbounded knapsack problem for a given capacity $j < t_0$, there exists an optimal solution that uses at most $\log t_0$ different types of items.*

In particular, in some optimal solution, there exists an item i that is used at least $\frac{j}{w_i \log t_0}$ times.

Proof. Consider an optimal solution that uses the minimum number of types of items. Let S be the set of items used in this solution, excluding multiplicities. If $|S| > \log t_0$, by the pigeonhole principle there must exist two different subsets S_1 and S_2 of S with the same total weight, multiplicities included (since there are $2^{|S|}$ subsets and t_0 integers between 0 and $t_0 - 1$). We can replace the items in $S_2 \setminus S_1$ with $S_1 \setminus S_2$, or vice versa (depending which of the two has smaller total value), and get a new solution that has the same total weight but has larger or equal total value. And if it has equal total value, the new solution uses a smaller number of types of items (since $S_2 \setminus S_1$ and $S_1 \setminus S_2$ are nonempty): a contradiction.

Thus, $|S| \le \log t_0$. This also implies that some item contributes at least $\frac{j}{\log t_0}$ to the total weight. ◀

Let $b := \lceil \log t_0 \rceil$. Let $D[j]$ be the maximum profit for the unbounded knapsack problem with capacity j. The above lemma implies the following recursive formula for all $j < t_0$:

$$D[j] \;=\; \max\left\{0,\; \max_{i=1}^{n}(D[j - w_i x_{ij}] + p_i x_{ij})\right\} \qquad \text{where } x_{ij} := \left\lfloor \tfrac{j}{w_i b} \right\rfloor.$$

Note that $j - w_i x_{ij} \in \left[(1 - \frac{1}{b})j,\; (1 - \frac{1}{b})j + u\right]$.

The above formula allows us to compute the subarray $D[t, \ldots, t + bu]$ from the subarray $D\left[(1 - \frac{1}{b})t, \ldots, (1 - \frac{1}{b})(t + bu) + u\right] = D\left[(1 - \frac{1}{b})t, \ldots, (1 - \frac{1}{b})t + bu\right]$ in $O(bu \cdot n)$ time. The latter subarray can be computed recursively.

Let $T(t)$ denote the time for computing $D[t, \ldots, t + bu]$. We thus obtain the following recurrence:

$$T(t) \;=\; T((1 - \tfrac{1}{b})t) \;+\; O(bnu).$$

For the base case, we have $T(0) = O(bnu)$ by the standard dynamic programming algorithm (which computes $D[0, \ldots, j]$ in $O(nj)$ time). The number of levels of recursion is $O(b \log t)$. So, $T(t) = O(b^2 nu \log t) = O(nu \log^2 t_0 \log t)$. We can set $t_0 = (t + u)^{O(1)}$. As before, we can initially reduce the capacity t to below u^2 by repeatedly using the item with the largest profit-to-weight ratio. This yields the following result:

▶ **Theorem 14.** *The single-capacity unbounded knapsack problem can be solved in $O(nu \log^3 u)$ time.*

▶ Remark. There are alternative ways to exploit the above lemma to get $\tilde{O}(nu)$ algorithms (by computing $D[j]$ for a different choice of $\tilde{O}(bu)$ indices j), and the polylogarithmic factor is likely improvable.

8 Minimum Word Break

Bringmann, Grønlund, and Larsen [6] studied the decision version of the word break problem, and gave an algorithm with $\tilde{O}(nm^{1/3} + m)$ running time (with a matching conditional lower bound for combinatorial algorithms).

We consider the optimization version of the problem (with unit weight), defined as follows:

▶ **Problem 3** (Minimum Word Break)**.** *Given a string s with length n and a dictionary D with total length m, find the minimum number t^* such that s can be split into t^* words in D (duplicates are allowed).*

The single-target change-making problem can be viewed as a special case of this problem, by representing each coin with value v_i as a string with length v_i over a unary alphabet.

In this section, we briefly note that Bringmann et al.'s algorithm can be modified to solve the minimum word break problem without increasing the running time (ignoring polylogarithmic factors), by using our Lemma 1.

The modification. Since much of the solution proceeds as in Bringmann et al.'s paper [6], we will only describe the difference and assume the reader is already familiar with the previous paper. In particular, we will use the same notation.

Upon close inspection of their paper, we see that most parts of Bringmann et al.'s method require no (or straightforward) changes. Their "first algorithm" is no longer required, and the main change lies in their "second algorithm", specifically, the "query algorithm" in [6, Section 4.2 (arXiv version)]. Instead of computing $S + S_B$ using FFT, we now need to compute the $(\min, +)$-convolution between S and S_B, where $S_B[i] = 1$ if u_i is marked, and $S_B[i] = \infty$ otherwise. We are only interested in q_B entries in the output array, where $\sum_{B\in\mathcal{B}} q_B = O(q)$ and $|\mathcal{B}| \le \frac{m}{q\cdot\lambda_q}$. (Note that $|\mathcal{B}| = 0$ if $\lambda_q > m/q$.) By using the output-sensitive bound from Lemma 1, we can perform the $(\min, +)$-convolution in $\tilde{O}(q\sqrt{q_B})$ time. By the Cauchy–Schwarz inequality, the sum of the cost over all $B \in \mathcal{B}$ is

$$\tilde{O}\left(\sum_{B\in\mathcal{B}} q\sqrt{q_B}\right) = \tilde{O}\left(q\sqrt{q|\mathcal{B}|}\right) = \tilde{O}\left(q\sqrt{\frac{m}{\lambda_q}}\right).$$

The other parts of the query algorithm requires $\tilde{O}(q \cdot \lambda_q)$ time. The total time is

$$\tilde{O}\left(q\lambda_q + q\sqrt{\frac{m}{\lambda_q}}\right).$$

To balance cost, we choose $\lambda_q = m^{1/3}$, and as a result, the query time in [6, Lemma 2 (arXiv version)] becomes $\tilde{O}(qm^{1/3})$, instead of $\tilde{O}(\min\{q^2, \sqrt{qm}\})$.

The final running time as analyzed in [6, page 10] was

$$\tilde{O}\left(\sum_{q=2^\ell} \frac{n}{q} \cdot \min\{q^2, \sqrt{qm}\}\right) = \tilde{O}(nm^{1/3}),$$

plus $\tilde{O}(n + m)$ for preprocessing. With the new query time bound, the sum changes to

$$\tilde{O}\left(\sum_{q=2^\ell} \frac{n}{q} \cdot qm^{1/3}\right) = \tilde{O}(nm^{1/3}),$$

which luckily gives the same result.

▶ **Theorem 15.** *The minimum word break problem can be solved in $\tilde{O}(nm^{1/3} + m)$ time.*

▶ Remark. The algorithm can actually solve an extension of the problem: compute the minimum number of breaks for every prefix of the input string. In particular, when the alphabet is unary, this implies an $\tilde{O}(t\sigma^{1/3} + \sigma)$-time algorithm for the all-targets change-making problem. However, this bound is not as good as those from Theorem 2 and Corollary 8 ($\tilde{O}(\min\{t^{4/3}, (t\sigma)^{2/3} + t\})$).

9 Concluding Remarks

Our change-making algorithms can be modified to compute not just the minimum number of coins but also a representation of the minimum multiset of coins for every target value. For the FFT-based algorithms, we need standard techniques for witness finding [1, 23] (which only increases the running time by polylogarithmic factors).

Although Erdős and Graham's $\Theta(u^2/k)$ bound on the Frobenius problem is asymptotically tight in the worst case (one bad coin set is $\{x, 2x, \ldots, (k-1)x, (k-1)x-1\}$ with $x = \lceil \frac{u}{k-1} \rceil$), the Frobenius number tends to be smaller for "many" k-tuples of coin values (it is usually subquadratic even for $k = 3$). This suggests that our $\tilde{O}(u^2 + t)$-time algorithm for all-targets coin changing might be improvable for many input sets of coins. However, obtaining an improvement in the worst case remains intriguingly open (this might require new results on the Frobenius problem – the interplay between combinatorial and algorithmic results seems worthy of further study).

References

1 Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. Witnesses for Boolean matrix multiplication and for shortest paths. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 417–426, 1992. doi:10.1109/SFCS.1992.267748.

2 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 19:1–19:13, 2019. doi:10.4230/LIPIcs.ICALP.2019.19.

3 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 457–466, 2016.

4 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1269–1282, 2018. doi:10.1145/3188745.3188876.

5 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017.

6 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 307–318, 2017. arXiv:1611.00918.

7 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

8 Timothy M. Chan and Qizheng He. On the change-making problem. In *Proceedings of the 4th ACM-SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 38–42, 2020.

9 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Transactions on Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465.

10 Jacques Dixmier. Proof of a conjecture by Erdős and Graham concerning the problem of Frobenius. *Journal of Number Theory*, 34(2):198–209, 1990.

11 Ran Duan and Seth Pettie. Fast algorithms for (max,min)-matrix multiplication and bottleneck shortest paths. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–391, 2009.

12 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. *ACM Transactions on Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322.

13 Paul Erdős and Ronald L Graham. On a linear diophantine problem of Frobenius. *Acta Arithmetica*, 21(1):399–408, 1972.

14 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 43:1–43:17, 2019. doi:10.4230/LIPIcs.ITCS.2019.43.

15 Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms (SOSA)*, volume 69, pages 17:1–17:6, 2019.

16 Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1062–1072, 2017.

17 Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Transactions on Algorithms*, 15(3):1–20, 2019.

18 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 21:1–21:15, 2017.

19 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 53:1–53:18, 2020.

20 George S. Lueker. Two NP-complete problems in nonnegative integer programming. Technical report, Princeton University. Department of Electrical Engineering, 1975.

21 Jiří Matoušek. Computing dominances in E^n. *Information Processing Letters*, 38(5):277–278, 1991. doi:10.1016/0020-0190(91)90071-O.

22 David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1–14, 1999.

23 Raimund Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 745–749, 1992. doi:10.1145/129712.129784.

24 Arie Tamir. New pseudopolynomial complexity bounds for the bounded and other integer knapsack related problems. *Operations Research Letters*, 37(5):303–306, 2009. doi:10.1016/j.orl.2009.05.003.

25 Virginia Vassilevska, R. Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(1):173–189, 2009.

26 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

27 J. W. Wright. The change-making problem. *Journal of the ACM*, 22(1):125–128, 1975.

28 Chao Xu. Word break with cost. https://chaoxuprime.com/posts/2019-09-19-word-break-with-cost.html, 2019.

The Maximum Binary Tree Problem

Karthekeyan Chandrasekaran
Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
karthe@illinois.edu

Elena Grigorescu
Purdue University, West Lafayette, IN, USA
elena-g@purdue.edu

Gabriel Istrate
West University of Timişoara, Romania
e-Austria Research Institute, Timişoara, Romania
gabrielistrate@acm.org

Shubhang Kulkarni
Purdue University, West Lafayette, IN, USA
kulkar17@purdue.edu

Young-San Lin
Purdue University, West Lafayette, IN, USA
lin532@purdue.edu

Minshen Zhu
Purdue University, West Lafayette, IN, USA
zhu628@purdue.edu

Abstract

We introduce and investigate the approximability of the *maximum binary tree problem* (MBT) in directed and undirected graphs. The goal in MBT is to find a maximum-sized binary tree in a given graph. MBT is a natural variant of the well-studied longest path problem, since both can be viewed as finding a maximum-sized tree of bounded degree in a given graph.

The connection to longest path motivates the study of MBT in directed acyclic graphs (DAGs), since the longest path problem is solvable efficiently in DAGs. In contrast, we show that MBT in DAGs is in fact hard: it has no efficient $\exp(-O(\log n/\log\log n))$-approximation algorithm under the exponential time hypothesis, where n is the number of vertices in the input graph. In undirected graphs, we show that MBT has no efficient $\exp(-O(\log^{0.63} n))$-approximation under the exponential time hypothesis. Our inapproximability results rely on self-improving reductions and structural properties of binary trees. We also show constant-factor inapproximability assuming $\mathbf{P} \neq \mathbf{NP}$.

In addition to inapproximability results, we present algorithmic results along two different flavors: (1) We design a randomized algorithm to verify if a given directed graph on n vertices contains a binary tree of size k in $2^k\mathsf{poly}(n)$ time. (2) Motivated by the longest heapable subsequence problem, introduced by Byers, Heeringa, Mitzenmacher, and Zervas, *ANALCO 2011*, which is equivalent to MBT in *permutation DAGs*, we design efficient algorithms for MBT in bipartite permutation graphs.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases maximum binary tree, heapability, inapproximability, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.30

Related Version `https://arxiv.org/abs/1909.07915`

Funding *Karthekeyan Chandrasekaran*: Supported by NSF CCF-1814613 and NSF CCF-1907937.
Elena Grigorescu: Supported by NSF CCF-1910659 and NSF CCF-1910411.
Gabriel Istrate: Supported by a grant of Ministry of Research and Innovation, CNCS - UEFISCDI project number PN-III-P4-ID-PCE-2016-0842, within PNCDI III.
Young-San Lin: Supported by NSF CCF-1910659 and NSF CCF-1910411.
Minshen Zhu: Supported by NSF CCF-1910659 and NSF CCF-1910411.

1 Introduction

A general degree-constrained subgraph problem asks for an optimal subgraph of a given graph with specified properties while also satisfying degree constraints on all vertices. Degree-constrained subgraph problems have numerous applications in the field of network design and consequently, have been studied extensively in the algorithms and approximation literature [1, 15–17, 29, 31, 32]. In this work, we introduce and study the *maximum binary tree problem* in directed and undirected graphs. In the maximum binary tree problem (MBT), we are given an input graph G and the goal is to find a *binary tree* in G with maximum number of vertices.

Our first motivation for studying MBT arises from the viewpoint that it is a variant of the longest path problem: In the longest path problem, the goal is to find a maximum-sized tree in which every vertex has degree at most 2. In MBT, the goal is to find a maximum-sized tree in which every vertex has degree at most 3. Certainly, one may generalize both these problems to finding a *maximum-sized degree-constrained tree* in a given graph. In this work we focus on binary trees; however, all our results extend to the maximum-sized degree-constrained tree problem for *constant* degree bound.

Our second motivation for studying MBT is its connection to the *longest heapable subsequence problem* introduced by Byers, Heeringa, Mitzenmacher, and Zervas [10]. Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ be a permutation on n elements. Byers et al. define a subsequence (not necessarily contiguous) of σ to be *heapable* if the elements of the subsequence can be sequentially inserted to form a binary *min-heap* data structure. Namely, insertions subsequent to the first element, which takes the root position, happen below previously placed elements. The longest heapable subsequence problem asks for a maximum-length heapable subsequence of a given sequence. This generalizes the well-known longest increasing subsequence problem. Porfilio [30] showed that the longest heapable subsequence problem is equivalent to MBT in permutation directed acyclic graphs (abbreviated *permutation DAGs*): a permutation DAG associated with the sequence σ is obtained by introducing a vertex u_i for every sequence element σ_i, and arcs (u_i, u_j) for every pair (i, j) such that $i > j$ and $\sigma_i \geq \sigma_j$. On the other hand, for sequences of intervals the maximum binary problem is easily solvable by a greedy algorithm [6] (see also [21] for further results and open problems on the heapability of partial orders). These results motivate the study of MBT in restricted graph families.

We now formally define MBT in undirected graphs, which we denote as UndirMaxBinaryTree. A *binary tree* of an *undirected* graph G is a subgraph T of G that is connected and acyclic with the degree of u in T being at most 3 for every vertex u in T. In UndirMaxBinaryTree, the input is an undirected graph G and the goal is to find a binary tree in G with maximum number of vertices. In the rooted variant of this problem, the input is an undirected graph G along with a specified root vertex r and the goal is to find a binary tree containing r in G with maximum number of vertices such that the degree of r in the tree is at most 2. We focus on the unrooted variant of the problem and mention that it reduces to the rooted variant. We emphasize that a binary tree T of G is not necessarily spanning (i.e., may not contain all vertices of the given graph). The problem of verifying whether a given undirected graph has a *spanning* binary tree is **NP**-complete. This follows by a reduction from the Hamiltonian path problem: Given an undirected graph $G = (V, E)$, create a pendant vertex v' adjacent to v for every vertex $v \in V$. The resulting graph has a spanning binary tree if and only if G has a Hamiltonian path.

Next, we formally define MBT in directed graphs. A *tree of a directed graph* G is a subgraph T of G such that T is acyclic and has a unique vertex, termed as the root, with the property that every vertex v in T has a unique directed path *to the root* in T. A *binary tree*

of a directed graph G is a tree T such that the incoming-degree of every vertex u in T is at most 2 while the outgoing-degree of every vertex u in T is at most 1. In the rooted variant of the maximum binary tree problem for directed graphs, the input is a directed graph G along with a specified root r and the goal is to find an r-rooted binary tree T in G with maximum number of vertices. The problem of verifying whether a given directed graph has a *spanning* binary tree is **NP**-complete (by a similar reduction as that for undirected graphs).

The connection to the longest path problem as well as the longest heapable subsequence problem motivates the study of the maximum binary tree problem in directed acyclic graphs (DAGs). In contrast to directed graphs, the longest path problem in DAGs can be solved in polynomial-time (e.g., using dynamic programming or LP-based techniques). Moreover, verifying whether a given DAG contains a spanning binary tree is solvable in polynomial-time using the following characterization: a given DAG on vertex set V contains a spanning binary tree if and only if the partition matroid corresponding to the in-degree of every vertex being at most two and the partition matroid corresponding to the out-degree of every vertex being at most one have a common independent set of size $|V| - 1$. These observations raise the intriguing possibility of solving the maximum binary tree problem in DAGs in polynomial-time. For this reason, we focus on DAGs within the family of directed graphs in this work. We denote the maximum binary tree problem in DAGs as DAGMaxBinaryTree.

The rooted and the unrooted variants of the maximum binary tree problem in DAGs are polynomial-time equivalent by simple transformations. Indeed, the unrooted variant can be solved by solving the rooted variant for every choice of the root. To see the other direction, suppose we would like to find a maximum r-rooted binary tree in a given DAG $G = (V, E)$. Then, we discard from G all outgoing arcs from r and all vertices that cannot reach r (i.e., we consider the sub-DAG induced by the descendents of r) and find an unrooted maximum binary tree in the resulting DAG. If this binary tree is rooted at a vertex $r' \neq r$, then it can be extended to an r-rooted binary tree by including an arbitrary $r' \to r$ path P – since the graph is a DAG, any such path P will not visit a vertex that is already in the tree (apart from r'). The equivalence is also approximation preserving. For this reason, we only study the rooted variant of the problem in DAGs.

We present inapproximability results for MBT in DAGs and undirected graphs. On the algorithmic side, we show that MBT in directed graphs is fixed-parameter tractable when parameterized by the solution size. We observe that the equivalence of the longest heapable subsequence to MBT in permutation DAGs motivates the study of MBT even in restricted graph families. As a first step towards understanding MBT in permutation DAGs, we design an algorithm for bipartite permutation graphs. We use a variety of tools including self-improving and gadget reductions for our inapproximability results, and algebraic and structural techniques for our algorithmic results.

1.1 Related work

Degree-constrained subgraph problems appeared as early as 1978 in the textbook of Garey and Johnson [18] and have garnered plenty of attention in the approximation community [1, 15–17, 23, 29, 31, 32]. A rich line of works have addressed the minimum degree spanning tree problem as well as the minimum cost degree-constrained spanning tree problem leading to powerful rounding techniques and a deep understanding of the spanning tree polytope [12, 13, 16, 19, 25, 28, 32]. Approximation and bicriteria approximation algorithms for the counterparts of these problems in directed graphs, namely degree-constrained arborescence and min-cost degree-constrained arborescence, have also been studied in the literature [7].

In the *maximum-edge* degree-constrained connected subgraph problem, the goal is to find a connected degree-constrained subgraph of a given graph with maximum number of edges. This problem does not admit a PTAS [3] and has been studied from the perspective of fixed-parameter tractability [4]. MBT could be viewed as a *maximum-vertex* degree-constrained connected subgraph problem, where the goal is to maximize the number of *vertices* as opposed to the number of *edges* – the degree-constrained connected subgraph maximizing the number of vertices may be assumed to be acyclic and hence, a tree. It is believed that the *connectivity constraint* makes the maximum-edge degree-constrained connected subgraph problem to become extremely difficult to approximate. Our results formalize this belief when the objective is to maximize the number of vertices.

Switching the objective with the constraint in the maximum-vertex degree-constrained connected subgraph problem leads to the minimum-degree k-tree problem: here the goal is to find a minimum degree subgraph that is a tree with at least k vertices. Minimum degree k-tree admits a $O(\sqrt{(k/\Delta^*)\log k})$-approximation, where Δ^* is the optimal degree and does not admit a $o(\log n)$-approximation [23]. We note that the hardness reduction here (from set cover) crucially requires the optimal solution value Δ^* to grow with the number n of vertices in the input instance, and hence, does not imply any hardness result for input instances in which Δ^* is a constant. Moreover, the approximation result implies that a tree of degree $O(\sqrt{k\log k})$ containing k vertices can be found in polynomial time if the input graph contains a constant-degree tree with k vertices.

We consider the maximum binary tree problem to be a generalization of the longest path problem as both can be viewed as asking for a maximum-sized degree-constrained connected acyclic subgraph. The longest path problem in undirected graphs admits an $\Omega\left((\log n/\log\log n)^2/n\right)$-approximation [9], but it is APX-hard and does not admit a $2^{-O\left(\log^{1-\varepsilon} n\right)}$-approximation for any constant $\varepsilon > 0$ unless $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{\log^{O(1/\varepsilon)} n}\right)$ [22]. Our hardness results for the max binary tree problem in undirected graphs bolsters this connection. The longest path problem in directed graphs is much harder: For every $\varepsilon > 0$ it cannot be approximated to within a factor of $1/n^{1-\varepsilon}$ unless $\mathbf{P} = \mathbf{NP}$, and it cannot be approximated to within a factor of $(\log^{2+\varepsilon} n)/n$ under the Exponential Time Hypothesis [9]. However, the longest path problem in DAGs is solvable in polynomial time. Our hardness results for the max binary tree problem in DAGs are in stark contrast to the polynomial-time solvability of the longest path problem in DAGs.

On the algorithmic side, the color-coding technique introduced by Alon, Yuster, and Zwick [2] can be used to decide whether an undirected graph $G = (V, E)$ contains a copy of a bounded treewidth pattern graph $H = (V_H, E_H)$ where $|V_H| = O(\log |V|)$, and if so, then find one in polynomial time. The idea here is to randomly color the vertices of G by $O(\log |V|)$ colors and to find a maximum colorful copy of H using dynamic programming. We note that the same dynamic programming approach can be modified to find a maximum colorful binary tree. This algorithm can be derandomized, thus leading to a deterministic $\Omega((1/n)\log n)$-approximation to UndirMaxBinaryTree.

In parameterized complexity, designing algorithms with running time $\beta^k\mathsf{poly}(n)$ ($\beta > 1$ is a constant) for problems like k-Path and k-Tree is a central topic. For k-Path, the color-coding technique mentioned above already implies a $(2e)^k\mathsf{poly}(n)$-time algorithm. Koutis [26] noticed that k-Path can be reduced to detecting whether a given polynomial contains a multilinear term. Using algebraic methods for the latter problem, Koutis obtained a $2^{1.5k}\mathsf{poly}(n)$ time algorithm for k-Path. This was later improved by Williams [36] to $2^k\mathsf{poly}(n)$. The current state-of-art algorithm is due to Björklund, Husfeldt, Kaski and Koivisto [8], which is also an algebraic algorithm with running time $1.66^k\mathsf{poly}(n)$. All of

these algorithms are randomized. Our study of the k-BinaryTree problem, which is the problem of deciding whether a given graph G contains a binary tree of size at least k, is inspired by this line of results.

Several **NP**-hard problems are known to be solvable in specific families of graphs. Bipartite permutation graphs is one such family which is known to exhibit this behaviour [24, 33–35]. Our polynomial-time solvability result for these families of graphs crucially identifies the existence of structured optimal solutions to reduce the search space and solves the problem over this reduced search space.

1.2 Our contributions

1.2.1 Inapproximability results

Directed graphs. We first focus on directed graphs and in particular, on directed acyclic graphs. It is well-known that the longest path problem in DAGs is solvable in polynomial-time. In contrast, we show that DAGMaxBinaryTree does not even admit a constant-factor approximation. Furthermore, if DAGMaxBinaryTree admitted a polynomial-time $\exp\left(-O\left(\log n/\log\log n\right)\right)$-approximation algorithm then the Exponential Time Hypothesis would be violated.

▶ **Theorem 1.** *We have the following inapproximability results for DAGMaxBinaryTree on n-vertex input graphs:*

1. *DAGMaxBinaryTree does not admit a polynomial-time constant-factor approximation assuming* $\mathbf{P} \neq \mathbf{NP}$.
2. *If DAGMaxBinaryTree admits a polynomial-time* $\exp\left(-O\left(\log n/\log\log n\right)\right)$*-approximation, then* $\mathbf{NP} \subseteq \mathbf{DTIME}\left(\exp\left(O\left(\sqrt{n}\right)\right)\right)$*, refuting the Exponential Time Hypothesis.*
3. *For any* $\varepsilon > 0$*, if DAGMaxBinaryTree admits a quasi-polynomial time* $\exp\left(-O\left(\log^{1-\varepsilon} n\right)\right)$*-approximation, then* $\mathbf{NP} \subseteq \mathbf{DTIME}\left(\exp\left(\log^{O(1/\varepsilon)} n\right)\right)$*, thus refuting the Exponential Time Hypothesis.*

Remark. The longest path problem in DAGs can be solved using a linear program (LP) based on cut constraints. Based on this connection, an integer program (IP) based on cut constraints can be formulated for DAGMaxBinaryTree. In the full version of this work [11], we show that the LP-relaxation of this cut-constraints-based-IP has an integrality gap of $\Omega(n^{1/3})$ in n-vertex DAGs.

Undirected graphs. Next, we turn to undirected graphs. We show that UndirMaxBinaryTree does not have a constant-factor approximation and does not admit a quasi-polynomial-time $\exp(-O(\log^{0.63} n))$-approximation under the Exponential Time Hypothesis.

▶ **Theorem 2.** *We have the following inapproximability results for UndirMaxBinaryTree on n-vertex input graphs:*

1. *UndirMaxBinaryTree does not admit a polynomial-time constant-factor approximation assuming* $\mathbf{P} \neq \mathbf{NP}$.
2. *For* $c = \log_3 2$ *and any* $\varepsilon > 0$*, if UndirMaxBinaryTree admits a quasi-polynomial time* $\exp\left(-O\left(\log^{c-\varepsilon} n\right)\right)$*-approximation, then* $\mathbf{NP} \subseteq \mathbf{DTIME}\left(\exp\left(\log^{O(1/\varepsilon)} n\right)\right)$*, thus refuting the Exponential Time Hypothesis.*

We summarize our hardness results for MBT on various graph families in Table 1 and contrast them with the corresponding known hardness results for the longest path problem on those families.

Table 1 Summary of inapproximability results. Here, n refers to the number of vertices in the input graph and ϵ is any positive constant. We include the known results for longest path for comparison. Text in gray refer to known results while text in black refer to our contributions.

Family	Assumption	Max Binary Tree	Longest Path
DAGs	$\mathbf{P \neq NP}$	No poly-time $\Omega(1)$-apx (Thm 1)	Poly-time solvable
	ETH	No poly-time $\exp(-O(\frac{\log n}{\log\log n}))$-apx No quasi-poly-time $\exp(-O(\log^{1-\varepsilon} n))$-apx (Thm 1)	Poly-time solvable
Directed	$\mathbf{P \neq NP}$	Same as DAGs	No poly-time $\frac{1}{n^{1-\varepsilon}}$-apx [9]
	ETH	Same as DAGs	Same as $\mathbf{P \neq NP}$
Undirected	$\mathbf{P \neq NP}$	No poly-time $\Omega(1)$-apx (Thm 2)	No poly-time $\Omega(1)$-apx [22]
	ETH	No quasi-poly-time $\exp(-O(\log^{0.63-\varepsilon} n))$-apx (Thm 2)	No quasi-poly-time $\exp(-O(\log^{1-\varepsilon} n))$-apx [22]

1.2.2 Algorithmic results

Fixed-parameter tractability. We denote the decision variant of MBT as k-BinaryTree– here the goal is to verify if a given graph contains a binary tree with at least k vertices. Since k-BinaryTree is **NP**-hard when k is part of the input, it is desirable to have an algorithm that runs in time $f(k)\mathsf{poly}(n)$ (i.e., a fixed parameter algorithm parameterized by the solution size). Our first algorithmic result achieves precisely this goal. Our algorithm is based on algebraic techniques.

▶ **Theorem 3.** *There exists a randomized algorithm that takes a directed graph $G = (V, E)$, a positive integer k, and a real value $\delta \in (0, 1)$ as input, runs in time $2^k\mathsf{poly}(|V|)\log(1/\delta)$ and*

1. *outputs "no" if G does not contain a binary tree of size k;*
2. *outputs a binary tree of size k with probability $1 - \delta$ if G contains one.*

Bipartite permutation graphs. Next, motivated by its connection to the max heapable subsequence problem, we study MBT in *bipartite permutation graphs*. A bipartite permutation graph is a permutation graph (undirected) which is also bipartite. We show that bipartite permutation graphs admit an efficient algorithm for MBT. Our algorithm exploits structural properties of bipartite permutation graphs. We believe that these structural properties could be helpful in solving MBT in permutation graphs which, in turn, could provide key insights towards solving MBT in permutation DAGs.

▶ **Theorem 4.** *There exists an algorithm to solve UndirMaxBinaryTree in n-vertex bipartite permutation graphs that runs in time $O(n^3)$.*

We summarize our algorithmic results for MBT in Table 2 and contrast them with the corresponding best known bounds for the longest path problem.

Table 2 Summary of algorithmic results. Here, n refers to the number of vertices in the input graph. We include the known results for longest path for comparison. Text in gray refer to known results while text in black refer to our contributions.

Problem	Max Binary Tree	Longest Path
FPT parameterized by solution size (Dir.)	$2^k\mathsf{poly}(n)$-time (Thm 3)	$1.66^k\mathsf{poly}(n)$-time [8]
Bipartite permutation graphs (Undir.)	$O(n^3)$-time (Thm 4)	$O(n)$-time [35]

We remark again that our inapproximability as well as algorithmic results are also applicable to the maximum degree-constrained tree problem for larger, but constant degree constraint. We focus on the degree constraint corresponding to binary trees for the sake of simplicity in exposition.

1.3 Proof techniques

In this section, we outline the techniques underlying our results.

1.3.1 Inapproximability results

At a very high level, our inapproximability results for MBT rely on the proof strategy for hardness of longest path due to Karger, Motwani, and Ramkumar [22], which has two main steps: (1) a *self-improving* reduction whose amplification implies that a constant-factor approximation immediately leads to a PTAS, and (2) a proof that there is no PTAS. However, we achieve both these steps in a completely different manner compared to the approach of Karger, Motwani, and Ramkumar. Both their steps are tailored for the longest path problem, but fail for the maximum degree-constrained tree problem. Our results for MBT require several novel ideas, as described next.

Karger, Motwani and Ramkumar's self-improving reduction for the longest path proceeds as follows: given an undirected graph G, they obtain a squared graph G^2 by replacing each edge $\{u, v\}$ of G with a copy of G by adding edges from u and v to all vertices in that edge copy. Let $OPT(G)$ be the length of the longest path in G. They make the following two observations: Obs (i) $OPT(G^2) \geq OPT(G)^2$ and Obs (ii) a path in G^2 of length at least $\alpha OPT(G^2)$ can be used to recover a path in G of length at least $\sqrt{\alpha}OPT(G)$. The first observation is because we can extend any path P in G into a path of length $|E(P)|^2$ by traversing each edge copy also along P. The second observation is because for any path P_2 in G^2 either P_2 restricted to some edge copy of G is a path of length at least $\sqrt{|E(P_2)|}$ or projecting P_2 to G (i.e., replacing each sub-path of P_2 in each edge copy by a single edge) gives a path of length at least $\sqrt{|E(P_2)|}$. We note that a similar construction of the squared graph for directed graphs also has the above mentioned observations: replace each directed arc (u, v) of G with a copy of G by adding arcs from u to all vertices in that edge copy and from all vertices in that edge copy to v.

In order to obtain inapproximability results for the maximum binary tree problem, we first introduce different constructions for the squared graph in the self-improving reduction compared to the ones by Karger et al. Moreover, our constructions of the squared graph differ substantially between undirected and directed graphs. Interestingly, our constructions also generalize naturally to the max degree-constrained tree problem. Secondly, although our reduction for showing the lack of PTAS in undirected graphs for MBT is also from TSP$(1, 2)$, it is completely different from that of Karger et al. and, once again, generalizes to the max degree-constrained tree problem. Thirdly, we show the lack of PTAS in DAGs for MBT by reducing from the max 3-coloring problem. This reduction is altogether new – the reader might recall that the longest path problem in DAGs is solvable in polynomial-time, so there cannot be a counterpart of this step (i.e., lack of PTAS in DAGs) for longest path. We next present further details underlying our proofs.

Self-improving reduction for directed graphs. We focus on the rooted variant of MBT in directed graphs. We first assume that the given graph G contains a source (if not, adding such a source vertex with arcs to all the vertices changes the optimum only by one). In

contrast to the squared graph described above (i.e., instead of adding edge copies), we replace every vertex in G by a copy of G (that we call as a vertex copy) and for every arc (u, v) in G, we add an arc from the root node of the vertex copy corresponding to u to the source node of the vertex copy corresponding to v. Finally, we declare the root node of the root vertex copy to be the root node of G^2. Let $\alpha \in (0, 1]$ and $OPT(G)$ be the number of vertices in the maximum binary tree in G. With this construction of the squared graph, we show that (1) $OPT(G^2) \geq OPT(G)^2$ and (2) an α-approximate rooted binary tree T_2 in G^2 can be used to recover a rooted binary tree T_1 in G which is a $\sqrt{\alpha}$-approximation. We emphasize that if G is a DAG, then the graph G^2 obtained by this construction is also a DAG.

Inapproximability for DAGs. In order to show the constant-factor inapproximability result for DAGs, it suffices to show that there is no PTAS (due to the self-improving reduction for directed graphs described above). We show the lack of a PTAS in DAGs by reducing from the max 3-coloring problem in 3-colorable graphs. It is known that this problem is APX-hard – in particular, there is no polynomial-time algorithm to find a coloring that colors at least 32/33-fraction of the edges properly [20]. Our reduction encodes the coloring problem into a DAGMaxBinaryTree instance in a way that recovers a consistent coloring for the vertices while also being proper for a large fraction of the edges. Our ETH-based inapproximability result is also a consequence of this reduction in conjunction with the self-improving reduction. We again emphasize that there is no counterpart of APX-hardness in DAGs for max binary tree in the longest path literature.

Self-improving reduction for undirected graphs. For UndirMaxBinaryTree, the self-improving reduction is more involved. Our above-mentioned reduction for DirMaxBinaryTree heavily exploits the directed nature of the graph (e.g., uses source vertices) and hence, is not applicable for undirected graphs. Moreover, the same choice of squared graph G^2 as Karger et al. [22] fails since Obs (ii) does not hold any more: the tree T_2 restricted to each edge copy may not be a tree (but it will be a forest). However, we observe that T_2 restricted to each edge copy may result in a forest with up to four binary trees in it. This observation and a more careful projection can be used to recover a tree of size at least $\sqrt{|V(T_2)|}/4$ (let us call this weakened Obs (ii)). Yet, weakened Obs (ii) is insufficient for a self-improving reduction. One approach to fix this would be to construct a *different squared graph* $G^{\boxtimes 2}$ that strengthens Obs (i) to guarantee that $OPT(G^{\boxtimes 2}) \geq 16OPT(G)^2$ while still allowing us to recover a binary tree of size $\sqrt{|V(T_2)|}/4$ in G from a binary tree T_2 in $G^{\boxtimes 2}$. Such a strengthened Obs (i) coupled with weakened Obs (ii) would complete the self-improving reduction. Our reduction is a variant of this approach: we introduce a construction of the squared-graph that strengthens Obs (i) by a factor of 2 while also weakening Obs (ii) only by a factor of 2. We prove these two properties of the construction by relying on a handshake-like property of binary trees which is a relationship between the number of nodes of each degree and the total number of nodes in the binary tree.

Inapproximability for undirected graphs. In order to show the constant-factor inapproximability result, it suffices to show that there is no PTAS (due to the self-improving reduction). We show the lack of a PTAS by reducing from TSP$(1, 2)$. We mention that Karger, Motwani, and Ramkumar [22] also show the lack of a PTAS for the longest path problem by reducing from TSP$(1, 2)$. However, our reduction is much different from their reduction. Our reduction mainly relies on the fact that if we add a pendant node to each vertex of a graph G and obtain a binary tree T that has a large number of such pendants, then the binary tree restricted to G cannot have too many nodes of degree three. Our ETH-based inapproximability result is also a consequence of this reduction in conjunction with the self-improving reduction.

1.3.2 Algorithmic results

A 2^kpoly(n) time algorithm for k-BinaryTree. The proof of this result is inspired by the algebrization technique introduced in [26, 27, 36] for designing randomized algorithms for k-Path and k-Tree – in k-Path, the goal is to recover a path of length k in the given graph while k-Tree asks to recover a *given tree* on k vertices in the given graph. Their idea is to encode a path (or the given tree) as a multilinear monomial term in a carefully constructed polynomial, which is efficiently computable using an arithmetic circuit. Then, a result due to Williams [36] is used to verify if the constructed polynomial contains a multilinear term – Williams' result gives an efficient randomized algorithm, which on input a small circuit that computes the polynomial, outputs "yes" if a multilinear term exists in the sum of products representation of the input polynomial, and "no" otherwise. The subgraph that is sought may then be extracted using an additional pass over the graph. Our main technical contribution is the construction of a polynomial P_G whose multilinear terms correspond to binary trees of size k in G and which is efficiently computable by an arithmetic circuit. We remark that the polynomial constructions in previous results do not readily generalize for our problem. Our key contribution is the construction of a suitable polynomial, based on a carefully designed recursion.

Efficient algorithm for bipartite permutation graphs. Our main structural insight for bipartite permutation graphs is that there exists a maximum binary tree which is *crossing-free* with respect to the so-called *strong ordering* of the vertices. With this insight, MBT in bipartite permutation graphs reduces to finding a maximum crossing-free binary tree. We solve this latter problem by dynamic programming.

Organization. We present the 2^kpoly(n) time algorithm for k-BinaryTree in Section 2. We present our hardness results for DAGs in Section 3. We conclude with a few open problems in Section 4. Due to page limits, we formulate an IP for DAGs and discuss its integrality gap, show our hardness results for undirected graphs, and design an efficient algorithm for bipartite permutation graphs in the full version [11].

1.4 Preliminaries

MBT in directed graphs. Given a directed graph $G = (V, E)$ and a vertex $r \in V$, we say that a subgraph T where $V(T) \subseteq V$ and $E(T) \subseteq E$, is an *r-rooted tree* in G if T is acyclic and every vertex v in T has a *unique* directed path (in T) to r. If the in-degree of each vertex in T is at most 2, then T is an *r-rooted binary tree*. The problem of interest in directed graphs is rooted-DirMaxBinaryTree: Given a directed graph $G = (V, E)$ and a root $r \in V$, the goal is to find an r-rooted binary tree in G with maximum number of vertices. The problem DAGMaxBinaryTree is a special case of rooted-DirMaxBinaryTree in which the input directed graph is a DAG. We recall that the rooted and unrooted variants of the maximum binary tree problem in DAGs are equivalent.

MBT in undirected graphs. Given an undirected graph $G = (V, E)$, we say that a subgraph T, where $V(T) \subseteq V$ and $E(T) \subseteq E$, is a *binary tree* in G if T is connected, acyclic, and $\deg_T(v) \leq 3$ for every vertex $v \in V(T)$. We will focus on the unrooted variant, i.e., UndirMaxBinaryTree, since the inapproximability results for the rooted variant are implied by inapproximability results for the unrooted variant. Here, we are given an undirected graph G and the goal is to find a binary tree in G with maximum number of vertices.

2 A $2^k\mathsf{poly}(n)$ time algorithm for k-BinaryTree

In this section, we present a randomized algorithm that solves k-BinaryTree exactly and runs in time $2^k\mathsf{poly}(n)$ where n is the number of vertices in the input graph. We recall that k-BinaryTree is the problem of deciding whether a given directed graph contains a binary tree of size k. Our algorithm is inspired by an algebraic approach for solving the k-Path problem – the algebraic approach relies on efficient detection of multilinear terms in a given polynomial.

k-Path, polynomials and multilinear terms. We begin with a recap of the algebraic approach to solve k-Path– here, the goal is to verify if a given (directed or undirected) graph G contains a path of length at least k. There has been a rich line of research dedicated to designing algorithms for k-Path with running time $\beta^k\mathsf{poly}(n)$ where $\beta > 1$ is a constant and n is the number of vertices in G (cf. [2, 8, 26, 36]). In particular, the algorithms in [26] and [36] are based on detecting multilinear terms in a polynomial.

We now recall the problem of detecting multilinear terms in a polynomial. Here, we are given a polynomial with coefficients in a finite field $\mathbb{F}_q$ and the goal is to verify if it has a multilinear term. We emphasize that the input polynomial is given *implicitly* by an arithmetic circuit consisting of additive and multiplicative gates. In other words, the algorithm is allowed to evaluate the polynomial at any point but does not have direct access to the sum-of-products expansion of the polynomial. We recall that a multilinear term in a polynomial $p \in \mathbb{F}_q[x_1, x_2, \cdots, x_m]$ is a monomial in the sum-of-products expansion of p consisting of only degree-1 variables. For example, in the following polynomial

$$p(x_1, x_2, x_3) = x_1^2x_2 + x_3 + x_1x_2x_3,$$

the monomials x_3 and $x_1x_2x_3$ are multilinear terms, whereas $x_1^2x_2$ is not a multilinear term since x_1 has degree 2. We will use the algorithm mentioned in the following theorem as a black box for detecting multilinear terms in a given polynomial.

▶ **Theorem 5** (Theorem 3.1 in [36]). *Let $P(x_1, \cdots, x_n)$ be a polynomial of degree at most k, represented by an arithmetic circuit of size $s(n)$ with additive gates (of unbounded fan-in), multiplicative gates (of fan-in two), and no scalar multiplications. There is a randomized algorithm that on input P runs in $2^k s(n)\cdot\mathsf{poly}(n)\log(1/\delta)$ time, outputs "yes" with probability $1-\delta$ if there is a multilinear term in the sum-product expansion of P, and outputs "no" if there is no multilinear term.*

The idea behind solving k-Path with the help of this theorem is to construct a polynomial p_G based on the input graph G so that p_G contains a multilinear term if and only if G contains a simple path of length k. At the same time, p_G should be computable by an arithmetic circuit of size $\mathsf{poly}(n)$. Koutis and Williams achieved these properties using the following polynomial:

$$p_G(x_1, \cdots, x_n) := \sum_{(v_{i_1}, v_{i_2}, \ldots, v_{i_k}): \text{ a walk in } G} x_{i_1}x_{i_2}\ldots x_{i_k}.$$

We recall that a walk in G is a sequence of vertices in which neighbouring vertices are adjacent in G. From the definition, it is easy to observe that there is a one-to-one correspondence between simple k-paths in G and multilinear terms in p_G. Moreover, it can be shown that there is an arithmetic circuit of size $O\left(k^2(m+n)\right)$ that computes p_G, where m is the number of edges and n is the number of vertices in G. See Chapter 10.4 of [14] for alternative constructions of this polynomial.

The polynomial construction for k-BinaryTree. Following the above-mentioned approach, we construct a polynomial P_G with the property that P_G contains a multilinear term if and only if G contains a binary tree of size k. Unfortunately, there is no immediate generalization of walks of length k that characterize binary trees on k vertices. So, instead of defining the polynomial conceptually, we will define the polynomial recursively by building the arithmetic circuit that computes P_G, and will prove the correspondence between multilinear terms in P_G and binary trees of size k in G. In the definition of our polynomial, we also need to introduce an auxiliary variable to eliminate low-degree multilinear terms in P_G (which is not an issue in the construction of the polynomial for k-Path).

Let $G = (V, E)$ be the given directed graph. For $v \in V$, let $\Delta_v^{in} \coloneqq \{u \in V : (u, v) \in E\}$. We begin by defining a polynomial $P_v^{(k)}$ for every $v \in V$ and every positive integer k, in $(n+1)$ variables $\{x_v\}_{v \in V} \cup \{y\}$:

$$P_v^{(k)} \coloneqq \begin{cases} x_v & \text{if } k = 1 \\ x_v \cdot y^{k-1} & \text{if } k > 1 \text{ and } \Delta_v^{in} = \varnothing \\ x_v \left(\sum_{u \in \Delta_v^{in}} P_u^{(k-1)} + \sum_{\ell=1}^{k-2} \left(\sum_{u_1 \in \Delta_v^{in}} P_{u_1}^{(\ell)} \right) \left(\sum_{u_2 \in \Delta_v^{in}} P_{u_2}^{(k-1-\ell)} \right) \right) & \text{if } k > 1 \text{ and } \Delta_v^{in} \neq \varnothing \end{cases}$$

Next, we define $P_G^{(k)} \coloneqq \sum_{v \in V} P_v^{(k)}$. We recall that a polynomial is homogenous if every monomial has the same degree. By induction on k, the polynomial $P_v^{(k)}$ is a degree-k homogeneous polynomial and so is $P_G^{(k)}$. Moreover, by the recursive definition, we see that $P_v^{(k)}$ can be represented as an arithmetic circuit of size $O(k^2 n)$ since there are kn polynomials in total, and computing each requires $O(1)$ addition gates (with unbounded fan-in) and $O(k)$ multiplication gates (with fan-in two). We show the following connection between multilinear terms in $P_G^{(k)}$ and binary trees in G.

▶ **Lemma 6.** *The graph G has a binary tree of size k rooted at r if and only if there is a multilinear term of the form $\prod_{v \in S} x_v$ in $P_r^{(k)}$ where $|S| = k$.*

Proof. We first show the forward direction, i.e., if G has a binary tree T of size k rooted at r, then there is a multilinear term of the form $\prod_{v \in T} x_v$ in $P_r^{(k)}$. We prove this by induction on k. The base case $k = 1$ follows since $P_r^{(1)} = x_r$. Suppose that the forward direction holds when $|T| \leq k - 1$. For $|T| = k$, we consider two cases.

1. The root r has only one child c. The subtree T_c of T rooted at c has size $k - 1$. By induction hypothesis there is a multilinear term $\prod_{v \in T_c} x_v$ in $P_c^{(k-1)}$. Since $c \in \Delta_r^{in}$, for some polynomial Q we can write

$$P_r^{(k)} = x_r \left(P_c^{(k-1)} + Q \right).$$

   Therefore $x_r \cdot \prod_{v \in T_c} x_v$ is a term in $P_r^{(k)}$. This term is multilinear and equals to $\prod_{v \in T} x_v$ since $r \notin T_c$.
2. The root r has two children c_1, c_2. Suppose that the subtree T_{c_1} rooted at c_1 has size ℓ, thus the subtree T_{c_2} rooted at c_2 has size $k - 1 - \ell$. The induction hypothesis implies that $P_{c_1}^{(\ell)}$ has a multilinear term $\prod_{v \in T_{c_1}} x_v$, and $P_{c_2}^{(k-1-\ell)}$ has a multilinear term $\prod_{v \in T_{c_2}} x_v$. Since $c_1, c_2 \in \Delta_r^{in}$, for some polynomial Q we can write

$$P_r^{(k)} = x_r \left(P_{c_1}^{(\ell)} P_{c_2}^{(k-1-\ell)} + Q \right).$$

   Therefore $x_r \left(\prod_{v \in T_{c_1}} x_v \right) \left(\prod_{v \in T_{c_2}} x_v \right)$ is a term in $P_r^{(k)}$. This term is multilinear and equals to $\prod_{v \in T} x_v$ because T is the disjoint union of r, T_{c_1} and T_{c_2}.

In both cases, the polynomial $P_r^{(k)}$ has a multilinear term $\prod_{v \in T} x_v$. This completes the inductive step.

Next, we show that if $P_r^{(k)}$ has a multilinear term of the form $\prod_{v \in S} x_v$ where $|S| = k$, then there is a binary tree T rooted at r in G with vertex set S. We prove this also by induction on k. The base case $k = 1$ is trivial since $P_r^{(1)} = x_r$ and there is a binary tree of size 1 rooted at r. Suppose that the statement holds for $k-1$ or less ($k > 1$).

Let $\prod_{v \in S} x_v$ be a multilinear term in $P_r^{(k)}$. We note that $r \in S$ since every term in $P_r^{(k)}$ contains x_r. Moreover, we may assume that $\Delta_r^{in} \neq \varnothing$ since otherwise $P_r^{(k)} = x_r \cdot y^{k-1}$ which does not contain any term of the form $\prod_{v \in S} x_v$. According to the definition of $P_r^{(k)}$, we could have two cases.

1. The term $\prod_{v \in S \setminus \{r\}} x_v$ is a multilinear term in $P_c^{(k-1)}$ for some $c \in \Delta_r^{in}$. The induction hypothesis implies that there is a binary tree T_c rooted at c with vertex set $S \setminus \{r\}$. Let T be the binary tree obtained by adding the edge (c, r) to T_c. Then T is a binary tree rooted at r with vertex set S.
2. The term $\prod_{v \in S \setminus \{r\}} x_v$ is a multilinear term in $P_{c_1}^{(\ell)} P_{c_2}^{(k-1-\ell)}$ for some $c_1, c_2 \in \Delta_r^{in}$ and some integer $1 \leq \ell \leq k-2$. In this case, since $P_{c_1}^{(\ell)}$ and $P_{c_2}^{(k-1-\ell)}$ are homogeneous polynomials of degree ℓ and $k-1-\ell$, we can partition $S \setminus \{r\}$ into two sets S_1 and S_2 with $|S_1| = \ell$ and $|S_2| = k-1-\ell$ such that $\prod_{v \in S_1} x_v$ is a multilinear term in $P_{c_1}^{(\ell)}$, and $\prod_{v \in S_2} x_v$ is a multilinear term in $P_{c_2}^{(\ell)}$. Applying the induction hypothesis, we obtain a binary tree T_{c_1} (rooted at c_1) with vertex set S_1 and a binary tree T_{c_2} (rooted at c_2) with vertex set S_2. Let T be the binary tree obtained by adding edges (c_1, r) and (c_2, r) to $T_{c_1} \cup T_{c_2}$. Then T is a binary tree rooted at r with vertex set $S_1 \cup S_2 \cup \{r\} = S$.

In both cases, we can find a binary tree T rooted at r with vertex set S. This completes the inductive step. ◀

With this choice of $P_G^{(k)}$, we call the algorithm appearing in Theorem 5 on input polynomial $\tilde{P}_G^{(k)} \coloneqq y \cdot P_G^{(k)}$, and output the result. We note that every multilinear term of the form $\prod_{v \in S} x_v$ in $P_G^{(k)}$ becomes a multilinear term of the form $y \cdot \prod_{v \in S} x_v$ in $\tilde{P}_G^{(k)}$, and every multilinear term of the form $y \cdot \prod_{v \in S} x_v$ in $P_G^{(k)}$ becomes $y^2 \cdot \prod_{v \in S} x_v$ in $\tilde{P}_G^{(k)}$, which is no longer a multilinear term. In light of Lemma 6, the graph G contains a binary tree of size k if and only if the degree-$(k+1)$ homogeneous polynomial $\tilde{P}_G^{(k)}$ has a multilinear term. The running time is $2^{k+1} \cdot O(k^2 n) \cdot \mathsf{poly}(n+1) \log(1/\delta) = 2^k \cdot \mathsf{poly}(n) \log(1/\delta)$.

We remark that this algorithm does not immediately tell us the tree T (namely the edges in T). However, we can find the edges in T with high probability via a reduction from the search variant to the decision variant. This is formalized in the next lemma.

▶ **Lemma 7.** *Suppose that there is an algorithm $\mathcal{A}$ which takes as input a directed graph $G = (V, E)$, an integer k and $\delta' \in (0,1)$ runs in time $2^k \mathsf{poly}(|V|) \log(1/\delta')$ and*

- *outputs "yes" with probability at least $1 - \delta'$ if G contains a binary tree of size k,*
- *outputs "no" with probability 1 if G does not contain a binary tree of size k.*

Then there also exists an algorithm $\mathcal{A}'$ which for every $\delta \in (0,1)$ outputs a binary tree T of size k with probability at least $1 - \delta$ when the answer is "yes", and runs in time $2^k \mathsf{poly}(|V|) \log(1/\delta)$.

Proof. The algorithm $\mathcal{A}'$ iterates through all arcs $e \in E$ and calls $\mathcal{A}$ on $(G - e, k)$ with $\delta' = \delta/m$ where $G - e = (V, E \setminus \{e\})$ and $m = |E|$. If for some $e \in E$ the call to $\mathcal{A}$ outputs "yes", we remove e from G (i.e., set $G \leftarrow G - e$) and continue the process. We will show that when the algorithm terminates, the arcs in G constitute a binary tree of size k (if there exists one) with probability at least $1 - \delta$.

Suppose the order in which $\mathcal{A}'$ processes the arcs is $e_1, e_2, \cdots, e_m$, and the graph at iteration t is denoted by $G^{(t)}$. Let B_t denote the event "$G^{(t-1)} - e_t$ contains a binary tree of size k, but the call to $\mathcal{A}\left(G^{(t-1)} - e_t, k\right)$ returns no". Due to the assumption we made for $\mathcal{A}$, event B_t happens with probability at most δ'. Since the algorithm $\mathcal{A}$ has perfect soundness, whenever $\mathcal{A}'$ removes an edge we are certain that the remaining graph still contains a binary tree of size k (otherwise the call to $\mathcal{A}$ would never return "yes"). That means if $G^{(0)} = G$ contains a binary tree of size k then $G^{(t)}$ contains a binary tree of size k for all $0 \leq t \leq m$. Therefore if none of the events B_t happens, the final graph $G^{(m)}$ is a binary tree of size k. The probability of failure is upper bounded by

$$\Pr\left[\bigcup_{t=1}^{m} B_t\right] \leq m \cdot \delta' = m \cdot \frac{\delta}{m} = \delta.$$

Since algorithm $\mathcal{A}'$ makes m calls to algorithm $\mathcal{A}$, the running time of $\mathcal{A}'$ is $m \cdot 2^k \mathsf{poly}\left(|V|\right) \log\left(1/\delta'\right) = 2^k \mathsf{poly}\left(|V|\right) \log\left(1/\delta\right)$. ◀

Theorem 5 in conjunction with Lemmas 6 and 7 complete the proof of Theorem 3.

3 Hardness results for DAGs

In this section, we show the inapproximability of finding a maximum binary tree in DAGs. The *size* of a binary tree denotes the number of vertices in the tree.

3.1 Self-improvability for directed graphs

We show that an algorithm for ROOTED-DIRMAXBINARYTREE achieving a constant factor approximation can be used to design a PTAS in Theorem 11. We emphasize that this result holds for arbitrary directed graphs and not just DAGs. The idea is to gradually boost up the approximation ratio by running the constant-factor approximation algorithm on squared graphs. Our notion of *squared graph* will be the following.

▶ **Definition 8.** *Given a directed graph $G = (V, E)$ with root r, the* squared graph G^2 *is the directed graph obtained by performing the following operations on G:*

1. *Construct $G' = (V', E')$ by introducing a source vertex s, i.e., $V' := V \cup \{s\}$. We add arcs from s to every vertex in G, i.e., $E' := E \cup \{(s, v) : v \in V\}$.*
2. *For each $u \in V$ (we note that V does not include the source vertex), we create a copy of G' that we denote as a* vertex copy G'_u. *We will denote the root vertex of G'_u by r_u, and the source vertex of G'_u by s_u.*
3. *For each $(u, v) \in E$, we create an arc (r_u, s_v).*
4. *We declare the root of G^2 to be r_r, i.e. the root vertex of the vertex copy G'_r.*

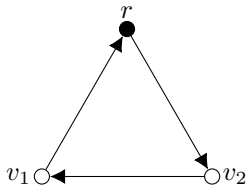
We define $G^{2^{k+1}}$ recursively as $G^{2^{k+1}} := \left(G^{2^k}\right)^2$ with the base case $G^1 := G$.

Given a directed graph G with $n-1$ vertices, the number of vertices in G^{2^k} satisfies the recurrence relation

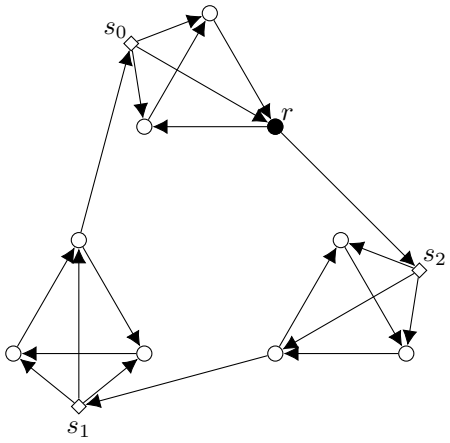
$$\left|V\left(G^{2^k}\right)\right| = \left|V\left(G^{2^{k-1}}\right)\right| \cdot \left(\left|V\left(G^{2^{k-1}}\right)\right| + 1\right) = \left|V\left(G^{2^{k-1}}\right)\right|^2 + \left|V\left(G^{2^{k-1}}\right)\right|.$$

Hence, we have

$$\left|V\left(G^{2^k}\right)\right| + 1 \leq \left(\left|V\left(G^{2^{k-1}}\right)\right| + 1\right)^2 \leq \left(\left|V\left(G^{2^{k-2}}\right)\right| + 1\right)^{2^2} \leq \cdots \leq \left(\left|V\left(G^{2^0}\right)\right| + 1\right)^{2^k} = n^{2^k}.$$



(a) G rooted at the black node.



(b) G^2 rooted at the black node. Source nodes are represented by diamonds.

Figure 1 Directed Squared Graph.

We use $OPT(G)$ to denote the size (number of vertices) of a maximum binary tree in G. The following lemma shows that $OPT(G)$ is super-multiplicative under the squaring operation.

▶ **Lemma 9.** *For any fixed root r, $OPT(G^2) \geq OPT(G)^2$.*

Proof. Suppose we have an optimal r-rooted binary tree T_1 of G, i.e. $|V(T_1)| = OPT(G)$. We construct an r_r-rooted binary tree T_2 of G^2 as follows:

1. For $v \in V(G)$, define $T'_v = T_v \cup \{s_v\}$ to be the optimal r_v-rooted binary tree in the vertex copy G'_v where T_v is identical to T_1 and the source vertex s_v is connected to an arbitrary leaf node in T_v.
2. For every vertex $v \in T_1$, add T'_v to T_2. This step generates $|V(T_1)| \cdot (|V(T_1)| + 1)$ vertices in T_2.
3. Connect the copies selected in step 2 by adding the arc (r_u, s_v) to T_2 for every arc $(u, v) \in T_1$.

Since T_1 is an r-rooted binary tree (in G), it follows that T_2 is an r_r-rooted binary tree (in G^2). Moreover, the size of T_2 is

$$|V(T_2)| = |V(T_1)| \cdot (|V(T_1)| + 1) \geq OPT(G)^2,$$

which cannot exceed $OPT\left(G^2\right)$. ◀

The following lemma shows that a large binary tree in G^2 can be used to obtain a large binary tree in G.

▶ **Lemma 10.** *For every $\alpha \in (0, 1]$, given an r_r-rooted binary tree T_2 in G^2 with size*

$$|V(T_2)| \geq \alpha OPT\left(G^2\right) - 1,$$

there is a linear-time (in the size of G^2) algorithm that finds an r-rooted binary tree T_1 of G with size

$$|V(T_1)| \geq \sqrt{\alpha} OPT(G) - 1.$$

Proof. Let $U \coloneqq \{v\colon v \in V(G) \text{ such that } r_v \in V(T_2)\}$ and $A \coloneqq \{(v,w) : v,w \in V(G), (r_v, s_w) \in E(T_2)\}$. We note that $T_1' := (U, A)$ is an r-rooted binary tree in G. This is because the path from every $v \in U$ to the root r is preserved, and the in-degree of every node $w \in U$ is bounded by the in-degree of s_w (in T_2), which is thus at most 2, and similarly the out-degree of every node is at most 1. We also remark that T_1' can be found in linear time. If $|U| \geq \sqrt{\alpha}OPT(G) > \sqrt{\alpha}OPT(G) - 1$, then the lemma is already proved. So, we may assume that $|U| < \sqrt{\alpha}OPT(G)$.

We now consider $T_v' \coloneqq (V(T_2) \cap V(G_v'), E(T_2) \cap E(G_v'))$ for $v \in U$. We can view T_v' as the restriction of T_2 to G_v', hence every node of T_v' has out-degree at most 2. Since T_2 is an r_r-rooted binary tree in G^2, every vertex in $V(T_2) \cap V(G_v')$ has a unique directed path (in T_2) to r_r, which must go through r_v, thus every vertex in $V(T_2) \cap V(G_v')$ has a unique directed path to r_v. It follows that T_v' is an r_v-rooted binary tree in the vertex copy G_v'.

We now show that there exists $v \in U$ such that $|V(T_v')| \geq \sqrt{\alpha}OPT(G)$. Suppose not, which means for every $v \in U$ we have $|V(T_v')| < \sqrt{\alpha}OPT(G)$. Then

$$\begin{aligned} |V(T_2)| &= \sum_{v \in U} |V(T_v')| < \sum_{v \in U} \left(\sqrt{\alpha}OPT(G)\right) < \sqrt{\alpha}OPT(G) \cdot \sqrt{\alpha}OPT(G) \\ &= \alpha OPT(G)^2 \leq \alpha \cdot OPT\left(G^2\right), \end{aligned}$$

a contradiction. The last inequality is due to Lemma 9.

In linear time we can find a binary tree T_v' with the desired size $|V(T_v')| \geq \sqrt{\alpha}OPT(G)$. To complete the proof of the lemma, we let $T_1 \coloneqq T_v' \setminus \{s_v\}$ which is (isomorphic to) an r-rooted binary tree in G with size at least $\sqrt{\alpha}OPT(G) - 1$. ◀

▶ **Theorem 11.** *If* rooted-DirMaxBinaryTree *has a polynomial-time algorithm that achieves a constant-factor approximation, then it has a PTAS.*

Proof. Suppose that we have a polynomial-time algorithm $\mathcal{A}$ that achieves an α-approximation for rooted-DirMaxBinaryTree. Given a directed graph G, root r and $\varepsilon > 0$, let

$$k \coloneqq 1 + \left\lceil \log_2 \frac{\log_2 \alpha}{\log_2(1-\varepsilon)} \right\rceil$$

be an integer constant that depends on α and ε. We construct G^{2^k} and run algorithm $\mathcal{A}$ on G^{2^k}. Then, we get a binary tree in G^{2^k} of size at least $\alpha OPT\left(G^{2^k}\right) - 1$. By Lemma 10, we can obtain an r-rooted binary tree in G of size at least

$$\alpha^{2^{-k}} OPT(G) - 1 \geq \alpha^{2^{-k+1}} OPT(G) \geq (1-\varepsilon) OPT(G).$$

The first inequality holds as long as

$$OPT(G) \geq \frac{1}{\sqrt{1-\varepsilon} - (1-\varepsilon)} \geq \frac{1}{\alpha^{2^{-k}} - \alpha^{2^{-k+1}}}.$$

We note that if $OPT(G)$ is smaller than $1/\left(\alpha^{2^{-k}} - \alpha^{2^{-k+1}}\right)$ which is a constant, then we can solve the problem exactly by brute force in polynomial time. Finally, we also observe that for fixed ε, the running time of this algorithm is polynomial since there are at most $n^{2^k} = n^{O(1)}$ vertices in the graph G^{2^k}. ◀

3.2 APX-hardness for DAGs

Next, we show the inapproximability results for DAGs. We begin by recalling DAGMaxBinaryTree: We begin by recalling the problem:

DAGMaxBinaryTree
Given: A directed acyclic graph $G = (V, E)$ and a root $r \in V$.
Goal: An r-rooted binary tree in G with the largest number of nodes.

We may assume that the root is the only vertex that has no outgoing arcs as we may discard all vertices that cannot reach the root. We show that DAGMaxBinaryTree is APX-hard by reducing from the following problem.

Max-3-Colorable-Subgraph
Given: An undirected graph G that is 3-colorable.
Goal: A 3-coloring of G that maximizes the fraction of properly colored edges.

It is known that finding a 3-coloring that properly colors at least 32/33-fraction of edges in a given 3-colorable graph is NP-hard [5,20]. In particular, Max-3-Colorable-Subgraph is APX-hard. We reduce Max-3-Colorable-Subgraph to DAGMaxBinaryTree. Let $G = (V, E)$ be the input 3-colorable undirected graph with $n := |V|$ and $m := |E|$. For $\varepsilon > 0$ to be fixed later, we construct a DAG, denoted $D(G, \varepsilon)$, as follows (see Figure 2 for an illustration):

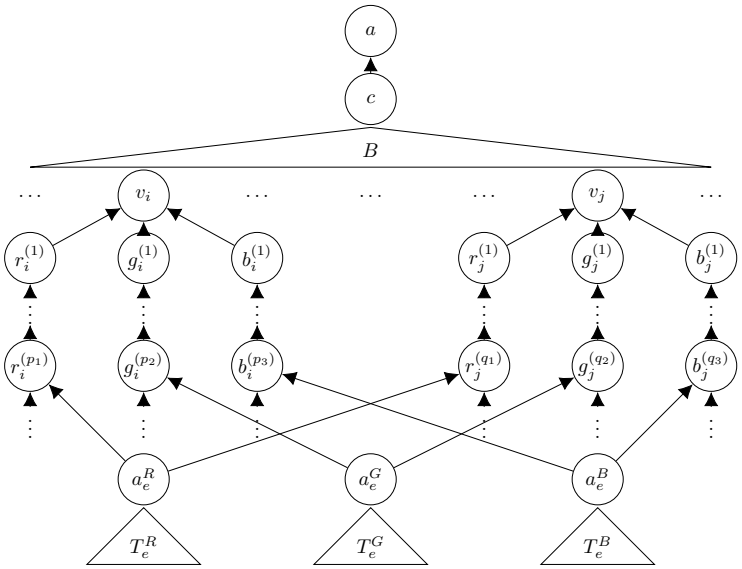


Figure 2 DAG $D(G, \varepsilon)$ constructed in the reduction from Max-3-Colorable-Subgraph to DAGMaxBinaryTree.

1. Create a directed binary tree B rooted at node c with $n := |V|$ leaf nodes. We will identify each leaf node by a unique vertex $v \in V$. Create a super root a and arc $c \to a$. This tree and the super root would have $2n$ nodes, including the super root node a, n leaf nodes, and $n - 1$ internal nodes.
2. For every $i \in V$, we introduce three directed paths of length n that will be referred to as R_i, G_i and B_i. Let R_i be structured as $r_i^{(1)} \leftarrow r_i^{(2)} \leftarrow \cdots \leftarrow r_i^{(n)}$, and similarly introduce $g_i^{(k)}$ and $b_i^{(k)}$ with the same structure. Also add arcs $r_i^{(1)} \to v_i$, $g_i^{(1)} \to v_i$ and $b_i^{(1)} \to v_i$.

3. For every edge $e = \{i, j\} \in E$, introduce three directed binary trees that will be referred to as T_e^R, T_e^G, and T_e^B, each with $t = \left\lceil \frac{2\varepsilon n(n+1)+4n^2}{\varepsilon m} \right\rceil$ nodes. Let the roots of the binary trees T_e^R, T_e^G, and T_e^B be a_e^R, a_e^G, and a_e^B respectively. Add arcs $a_e^R \to r_i^{(p_1)}$ and $a_e^R \to r_j^{(q_1)}$ where $r_i^{(p_1)}$ and $r_j^{(q_1)}$ are two nodes in R_i and R_j with in-degree strictly smaller than 2. We note that R_i is a path with n nodes so such a node always exists. Similarly connect a_e^G to $g_i^{(p_2)}$ and $g_j^{(q_2)}$, and a_e^B to $b_i^{(p_3)}$ and $b_j^{(q_3)}$ in the directed paths G_i and B_i, respectively.

The constructed graph $D(G, \varepsilon)$ is a DAG. We fix a to be the root. The number of nodes N in $D(G, \varepsilon)$ is $N = 3mt + 3n \cdot n + 2n = 3mt + 3n^2 + 2n$. We note that every node $v_i \in V$ has in-degree exactly 2 in every a-rooted maximal binary tree in $D(G, \varepsilon)$. The idea of this reduction is to encode the color of v_i as the unique path among R_i, G_i, B_i that is *not* in the subtree under v_i. The following two lemmas summarize the main properties of the DAG constructed above.

▶ **Lemma 12.** *Let T be a maximal a-rooted binary tree of $D(G, \varepsilon)$. If $|V(T)| \geq (1-\varepsilon/4)(N - n^2)$, then at most εm nodes among $\cup_{e \in E}\{a_e^R, a_e^G, a_e^B\}$ are not in T.*

Proof. Suppose more than εm such nodes are missing from T. For each node a_e^R that is not in T, the corresponding subtree T_e^R is also not in T (same for a_e^G and a_e^B). Therefore

$$|V(T)| < N - \varepsilon mt = 3mt + 3n^2 + 2n - \varepsilon mt = \left(1 - \frac{\varepsilon}{4}\right) \cdot 3mt + 3n^2 + 2n - \frac{\varepsilon}{4} mt.$$

The choice of t implies that $\varepsilon mt/4 > \varepsilon n(n+1)/2 + n^2$. Therefore

$$\begin{aligned} |V(T)| &< \left(1 - \frac{\varepsilon}{4}\right) \cdot 3mt + 3n^2 + 2n - \frac{\varepsilon n(n+1)}{2} - n^2 \\ &< \left(1 - \frac{\varepsilon}{4}\right) \cdot 3mt + \left(1 - \frac{\varepsilon}{4}\right)(2n^2 + 2n) \\ &= \left(1 - \frac{\varepsilon}{4}\right)(N - n^2), \end{aligned}$$

a contradiction. ◀

▶ **Lemma 13.** *If G is 3-colorable, then every a-rooted maximum binary tree in $D(G, \varepsilon)$ has size exactly $N - n^2$.*

Proof. We first note that every binary subtree of $D(G, \varepsilon)$ has size at most $N - n^2$. This is because there are n vertices with in-degree 3 (namely $v_1, v_2, \cdots, v_n$). For each such vertex v_i, there are 3 vertices $r_i^{(1)}, g_i^{(1)}$ and $b_i^{(1)}$ whose only outgoing arc is to v_i. Moreover, each vertex $r_i^{(1)}$ (and similarly $g_i^{(1)}$ and $b_i^{(1)}$) is the end-vertex of an induced path of length n.

Suppose G is 3-colorable. We now construct an a-rooted binary tree T of size $N - n^2$ in $D(G, \varepsilon)$. We focus on the nodes to be discarded so that we may construct a binary spanning tree with the remaining nodes. Let $\sigma \colon V \to \{Red, Green, Blue\}$ be a proper 3-coloring of G. If $\sigma(v_i) = Red$, we discard the path R_i. The cases where $\sigma(v_i) \in \{Green, Blue\}$ are similar. Since there are no monochromatic edges, there do not exist $e = \{v_i, v_j\} \in E$ and $C \in \{R, G, B\}$ such that both parents of a_e^C are not in T. Therefore every binary tree T_e^C is contained as a subtree in T. ◀

▶ **Theorem 14.** *Suppose there is a PTAS for DAGMaxBinaryTree on DAGs, then for every $\varepsilon > 0$ there is a polynomial-time algorithm which takes as input an undirected 3-colorable graph G, and outputs a 3-coloring of G that properly colors at least $(1 - \varepsilon)m$ edges.*

Proof. Let $G = (V, E)$ be the given undirected 3-colorable graph. We construct $D(G, \varepsilon)$ in polynomial time. We note that the constructed graph $D(G, \varepsilon)$ is a directed acyclic graph. We now run the PTAS for DAGMaxBinaryTree on $D(G, \varepsilon)$ and root a to obtain a $(1 - \varepsilon/4)$-approximate maximum binary tree in $D(G, \varepsilon)$. By Lemma 13 and the fact that G is 3-colorable, the PTAS will output an a-rooted binary tree T of size at least

$$\left(1 - \frac{\varepsilon}{4}\right)(N - n^2).$$

We may assume that T is a maximal binary tree in $D(G, \varepsilon)$ (if not, then add more vertices to T until we cannot add any further). Maximality ensures that the nodes v_i are in the tree T and moreover, the in-degree of v_i in T is exactly 2. For each $v_i \in V$, let c_i be the unique node among $\left\{r_i^{(1)}, g_i^{(1)}, b_i^{(1)}\right\}$ that is not in T. We define a coloring $\sigma : V \to \{Red, Green, Blue\}$ of G as

$$\forall v_i \in V, \quad \sigma(v_i) = \begin{cases} Red & \text{if } c_i = r_i^{(1)} \\ Green & \text{if } c_i = g_i^{(1)} \\ Blue & \text{if } c_i = b_i^{(1)}. \end{cases}$$

We now argue that the coloring is proper for at least $(1 - \varepsilon)$-fraction of the edges of G. Suppose we have an edge $e = \{v_i, v_j\}$ which is monochromatic under σ, and suppose w.l.o.g. $\sigma(v_i) = \sigma(v_j) = Red$. This means that neither $r_i^{(1)}$ nor $r_j^{(1)}$ is included in T. Therefore $a_e^R \notin T$ since neither of the two vertices with incoming arcs from a_e^R are in T. By Lemma 12, we know that at most εm vertices among $\cup_{e \in E}\{a_e^R, a_e^G, a_e^B\}$ can be excluded from T. Hence, the coloring σ that we obtained can violate at most εm edges in G. ◀

Finally, we prove Theorem 1 using the self-improving argument (Theorem 11) and the APX-hardness of DAGMaxBinaryTree (Theorem 14).

Proof of Theorem 1.

1. We observe that the graph G^2 constructed in Section 3 for the self-improving reduction is a DAG if G is a DAG. Therefore, by Theorem 11, a polynomial-time constant-factor approximation for DAGMaxBinaryTree would imply a PTAS for DAGMaxBinaryTree, a contradiction to APX-hardness shown in Theorem 14.
2. Next we show hardness under the Exponential Time Hypothesis. Suppose there is a polynomial-time algorithm $\mathcal{A}$ for DAGMaxBinaryTree that achieves an $\exp\left(-C \cdot \log_2 n / \log_2 \log_2 n\right)$-approximation for some constant $C > 0$. Given the input graph G with $n - 1$ vertices, let k be an integer that satisfies

   $$2^{\sqrt{n}} \leq n^{2^k} \leq 2^{2\sqrt{n}},$$

   and run $\mathcal{A}$ on G^{2^k} to obtain a binary tree with size at least

   $$\exp\left(-C \cdot \log_2 N / \log_2 \log_2 N\right) OPT\left(G^{2^k}\right),$$

   where $N = n^{2^k}$ upper bounds the size of G^{2^k}. Recursively running the algorithm suggested in Theorem 11 k times gives us a binary tree in G with size at least

   $$\begin{aligned} &\exp\left(-C \cdot \frac{\log_2 N}{\log_2 \log_2 N \cdot 2^k}\right) OPT(G) - 1 \\ \geq &\exp\left(-C \cdot \frac{2\sqrt{n}}{\log_2 \sqrt{n}} \cdot \frac{\log_2 n}{\sqrt{n}}\right) OPT(G) - 1 \\ \geq &\exp\left(-4C\right) OPT(G) - 1 \geq \frac{1}{2} \cdot \exp\left(-4C\right) OPT(G). \end{aligned}$$

The last inequality holds as long as

$$OPT(G) \geq 2 \cdot e^{4C}.$$

We note that if $OPT(G)$ is smaller than $2e^{4C}$ which is a constant, we can solve the problem exactly by brute force in polynomial time. Otherwise the above procedure can be regarded as a constant-factor approximation for DAGMaxBinaryTree. The running time is polynomial in

$$N = n^{2^k} = \exp\left(O\left(\sqrt{n}\right)\right),$$

which is sub-exponential. Moreover, from item 1 we know that it is **NP**-hard to approximate DAGMaxBinaryTree within a constant factor, thus $\mathbf{NP} \subseteq \mathbf{DTIME}\left(\exp\left(O\left(\sqrt{n}\right)\right)\right)$.

3. The proof of this item is almost identical to the previous one except that we choose a different integer k. Suppose there is an algorithm $\mathcal{A}'$ for DAGMaxBinaryTree that achieves a $\exp\left(-C \cdot \log^{1-\varepsilon} n\right)$-approximation for some constant $C > 0$, and runs in time $\exp\left(O\left(\log^d n\right)\right)$ for some constant $d > 0$. We show that there is an algorithm that achieves a constant-factor approximation for DAGMaxBinaryTree, and runs in time $\exp\left(O\left(\log^{d/\varepsilon} n\right)\right)$.
Given a DAG G on $n-1$ vertices as input for DAGMaxBinaryTree, let $k = \left\lceil \left(\frac{1}{\varepsilon} - 1\right) \log_2 \log n \right\rceil$ be an integer that satisfies

$$\left(2^k \log n\right)^{1-\varepsilon} \leq 2^k \leq 2 \left(\log n\right)^{\frac{1}{\varepsilon}-1}.$$

Running $\mathcal{A}'$ on G^{2^k} gives us a binary tree with size at least

$$\exp\left(-C \cdot \log^{1-\varepsilon} N\right) OPT\left(G^{2^k}\right),$$

where $N = n^{2^k}$ upper bounds the size of G^{2^k}. Recursively running the algorithm suggested in Theorem 11 k times gives us a binary tree in G with size at least

$$\begin{aligned}
&\exp\left(-C \cdot \frac{\log^{1-\varepsilon} N}{2^k}\right) OPT(G) - 1 \\
\geq &\exp\left(-C \cdot \frac{\left(2^k \log n\right)^{1-\varepsilon}}{2^k}\right) OPT(G) - 1 \\
\geq &\exp\left(-C\right) OPT(G) - 1 \geq \frac{1}{2} \cdot \exp\left(-C\right) OPT(G).
\end{aligned}$$

The last inequality holds as long as

$$OPT(G) \geq 2 \cdot e^{C}.$$

We note that if $OPT(G)$ is smaller than $2e^{C}$ which is a constant, we can solve the problem exactly by brute force in polynomial time. Otherwise the above procedure can be regarded as a constant-factor approximation for DAGMaxBinaryTree. The running time is quasi-polynomial in N, i.e. for some constant $C' > 0$, the running time is upper-bounded by

$$\exp\left(C'\left(\log^d N\right)\right) = \exp\left(C'\left(\left(2^k \log n\right)^d\right)\right) \leq \exp\left(C'\left(\log^{d/\varepsilon} n\right)\right). \quad \blacktriangleleft$$

4 Conclusion and Open Problems

In this work, we introduced the maximum binary tree problem (MBT) and presented hardness of approximation results for undirected, directed, and directed acyclic graphs, a fixed-parameter algorithm with the solution as the parameter, and efficient algorithms for bipartite permutation graphs. Our work raises several open questions that we state below.

Inapproximability of DirMaxBinaryTree. The view that MBT is a variant of the longest path problem leads to the natural question of whether the inapproximability results for MBT match that of longest path: Is MBT in directed graphs (or even in DAGs) hard to approximate within a factor of $1/n^{1-\varepsilon}$ (we recall that longest path is hard to approximate within a factor of $1/n^{1-\varepsilon}$ [9])? We remark that the self-improving technique is weak to handle $1/n^{1-\varepsilon}$-approximations since the squaring operation yields no improvement. The reduction in [9] showing $1/n^{1-\varepsilon}$-inapproximability of longest paths is from a restricted version of the vertex-disjoint paths problem and is very specific to paths. Furthermore, directed cycles play a crucial role in their reduction for a fundamental reason: longest path is polynomial-time solvable in DAGs. However, it is unclear if directed cycles are the source of hardness for MBT in digraphs (since MBT is already hard in DAGs).

Bicriteria Approximations. Given our inapproximability results, one natural algorithmic possibility is that of bicriteria approximations: can we find a tree with at least $\alpha \cdot OPT$ vertices while violating the degree bound by a factor of at most β? In particular, this motivates an intriguing direction concerning the longest path problem: Given an undirected/directed graph G with a path of length k, can we find a c_1-degree tree in G with at least k/c_2 vertices for some constants c_1 and c_2 efficiently?

Maximum Binary Tree in Permutation DAGs. Finally, it would be interesting to resolve the complexity of MBT in permutation DAGs (and permutation graphs). This would also resolve the open problem posed by Byers, Heeringa, Mitzenmacher, and Zervas of whether the maximum heapable subsequence problem is solvable in polynomial time [10].

References

1. Louigi Addario-Berry, Ketan Dalal, and Bruce A Reed. Degree constrained subgraphs. *Electronic Notes in Discrete Mathematics*, 19:257–263, 2005.
2. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
3. Omid Amini, David Peleg, Stéphane Pérennes, Ignasi Sau, and Saket Saurabh. Degree-constrained subgraph problems: Hardness and approximation results. In *Approximation and Online Algorithms*, pages 29–42, 2009.
4. Omid Amini, Ignasi Sau, and Saket Saurabh. Parameterized complexity of the smallest degree-constrained subgraph problem. In *Parameterized and Exact Computation*, pages 13–29, 2008.
5. Per Austrin, Ryan O'Donnell, and John Wright. A new point of NP-hardness for 2-to-1 Label-Cover. In *Proceedings of the 15th Annual International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, APPROX '12, pages 1–12, 2012.
6. János Balogh, Cosmin Bonchiş, Diana Diniş, Gabriel Istrate, and Ioan Todinca. On the heapability of finite partial orders. *Discrete Mathematics and Theoretical Computer Science*, 22(1):paper # 17, 2020.
7. Nikhil Bansal, Rohit Khandekar, and Viswanath Nagarajan. Additive guarantees for degree-bounded directed network design. *SIAM J. Comput.*, 39(4):1413–1431, October 2009.

8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, 2017.

9 Andreas Björklund, Thore Husfeldt, and Sanjeev Khanna. Approximating longest directed paths and cycles. In *Automata, Languages and Programming*, pages 222–233, 2004.

10 John Byers, Brent Heeringa, Michael Mitzenmacher, and Georgios Zervas. Heapable sequences and subseqeuences. In *Proceedings of the Meeting on Analytic Algorithmics and Combinatorics*, ANALCO '11, pages 33–44, 2011.

11 Karthekeyan Chandrasekaran, Elena Grigorescu, Gabriel Istrate, Shubhang Kulkarni, Young-San Lin, and Minshen Zhu. The maximum binary tree problem. *arXiv preprint*, 2019. arXiv:1909.07915.

12 Kamalika Chaudhuri, Satish Rao, Samantha Riesenfeld, and Kunal Talwar. A push–relabel approximation algorithm for approximating the minimum-degree mst problem and its generalization to matroids. *Theoretical Computer Science*, 410(44):4489–4503, 2009.

13 Kamalika Chaudhuri, Satish Rao, Samantha Riesenfeld, and Kunal Talwar. What Would Edmonds Do? Augmenting Paths and Witnesses for Degree-Bounded MSTs. *Algorithmica*, 55(1):157–189, September 2009.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

15 Paul Erdös, Ralph J Faudree, CC Rousseau, and RH Schelp. Subgraphs of minimal degree k. *Discrete Math*, 85(1):53–58, 1990.

16 Martin Fürer and Balaji Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409–423, 1994. doi:10.1006/jagm.1994.1042.

17 Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 448–456, 1983.

18 Michael Garey and David Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

19 Michel X. Goemans. Minimum bounded degree spanning trees. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 273–282, 2006.

20 Venkatesan Guruswami and Ali Kemal Sinop. Improved inapproximability results for maximum k-colorable subgraph. *Theory of Computing*, 9:413–435, 2013.

21 Gabriel Istrate and Cosmin Bonchiş. Heapability, interactive particle systems, partial orders: Results and open problems. In *Proceedings of DCFS'2016, 18th International Conference on Descriptional Complexity of Formal Systems*, pages 18–28. Springer, 2016.

22 David R. Karger, Rajeev Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

23 Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. On some network design problems with degree constraints. *Journal of Computer and System Sciences*, 79(5):725–736, 2013.

24 Ton Kloks, Dieter Kratsch, and Haiko Müller. Bandwidth of chain graphs. *Information Processing Letters*, 68(6):313–315, 1998.

25 Jochen Könemann and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. *SIAM J. Comput.*, 31(6):1783–1793, June 2002.

26 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *International Colloquium on Automata, Languages, and Programming*, ICALP '08, pages 575–586, 2008.

27 Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In *International Colloquium on Automata, Languages, and Programming*, ICALP '09, pages 653–664, 2009.

28 Jochen Könemann and R. Ravi. Primal-dual meets local search: Approximating msts with nonuniform degree bounds. *SIAM Journal on Computing*, 34(3):763–773, 2005.

29 Lap Chi Lau, Joseph (Seffi) Naor, Mohammad Salavatipour, and Mohit Singh. Survivable network design with degree or order constraints. *SIAM Journal on Computing*, 39(3):1062–1087, 2009.

30 Jaclyn Porfilio. A combinatorial characterization of heapability. Master's thesis, Williams College, 2015.

31 R. Ravi, Madhav Marathe, S. S. Ravi, Daniel Rosenkrantz, and Harry B. Hunt III. Approximation algorithms for degree-constrained minimum-cost network-design problems. *Algorithmica*, 31(1):58–78, September 2001.

32 Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. *J. ACM*, 62(1):1–19, March 2015.

33 Jacqueline Smith. Minimum degree spanning trees on bipartite permutation graphs. Master's thesis, University of Alberta, 2011.

34 Jeremy Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987.

35 Ryuhei Uehara and Yushi Uno. Efficient algorithms for the longest path problem. In *Proceedings of the 15th International Conference on Algorithms and Computation*, ISAAC '04, pages 871–883, 2004.

36 Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009.

Single-Source Shortest Paths and Strong Connectivity in Dynamic Planar Graphs

Panagiotis Charalampopoulos
Department of Informatics, King's College London, UK
Institute of Informatics, University of Warsaw, Poland
panagiotis.charalampopoulos@kcl.ac.uk

Adam Karczmarz
Institute of Informatics, University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl

Abstract

Efficient algorithms for computing and processing additively weighted Voronoi diagrams on planar graphs have been instrumental in obtaining several recent breakthrough results, most notably the almost-optimal exact distance oracle for planar graphs [Charalampopoulos et al., STOC'19], and subquadratic algorithms for planar diameter [Cabello, SODA'17, Gawrychowski et al., SODA'18]. In this paper, we show how Voronoi diagrams can be useful in obtaining dynamic planar graph algorithms and apply them to classical problems such as dynamic single-source shortest paths and dynamic strongly connected components.

First, we give a fully dynamic single-source shortest paths data structure for planar weighted digraphs with $\widetilde{O}(n^{4/5})$ worst-case update time and $O(\log^2 n)$ query time. Here, a single update can either change the graph by inserting or deleting an edge, or reset the source s of interest. All known non-trivial planarity-exploiting *exact* dynamic single-source shortest paths algorithms to date had polynomial query time. Further, note that a data structure with strongly sublinear update time capable of answering distance queries between all pairs of vertices in polylogarithmic time would refute the APSP conjecture [Abboud and Dahlgaard, FOCS'16].

Somewhat surprisingly, the Voronoi diagram based approach we take for single-source shortest paths can also be used in the fully dynamic strongly connected components problem. In particular, we obtain a data structure maintaining a planar digraph under edge insertions and deletions, capable of returning the identifier of the strongly connected component of any query vertex. The worst-case update and query time bounds are the same as for our single-source distance oracle. To the best of our knowledge, this is the first fully dynamic strong-connectivity algorithm achieving both sublinear update time and polylogarithmic query time for an important class of digraphs.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Shortest paths

Keywords and phrases dynamic graph algorithms, planar graphs, single-source shortest paths, strong connectivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.31

Funding *Panagiotis Charalampopoulos*: Partially supported by ERC Starting Grant TOTAL under the EU's Horizon 2020 Research and Innovation Programme (agreement no. 677651).
Adam Karczmarz: Supported by ERC Consolidator Grant 772346 TUgbOAT and the Polish National Science Centre 2018/29/N/ST6/00757 grant.

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders; Article No. 31; pp. 31:1–31:23

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The dynamic shortest paths problem seeks for a data structure maintaining a graph under updates and supporting shortest path queries.[1] Depending on the set of supported updates, we call such a graph data structure *fully dynamic* if both edge insertions and deletions are allowed, *incremental* if only edge insertions (or edge weight decreases) are supported, or *decremental* if only edge deletions (or weight increases) are allowed. In the *all-pairs* variant of dynamic shortest paths problem one has to support shortest path queries between any pair of vertices of the graph. In the *single-source* variant all shortest paths queries have to originate in a fixed distinguished vertex and the only parameter of a query is the target vertex.

For the most general setting where one requires *exact* answers, Demetrescu and Italiano [31] gave a fully dynamic algorithm (improved slightly by Thorup [77]) recomputing the all-pairs shortest paths matrix in nearly optimal $\widetilde{O}(n^2)$ amortized time even if real edge weights are allowed. Note that recomputing the distance matrix from scratch takes $\widetilde{O}(nm) = \widetilde{O}(n^3)$ time [55]. Fully dynamic all-pairs shortest paths data structures with subcubic *worst-case* update bounds are also known [5, 46, 78]. There exist faster algorithms if the input graph is unweighted and *partially dynamic* (i.e., incremental or decremental) [6, 7]. However, none of the known results improves upon a trivial, recompute-from-scratch algorithm with $\widetilde{O}(mn)$ update time and $O(1)$ query time if the graph is *sparse*, i.e., $m = \widetilde{O}(n)$. For the single-source variant, all known non-trivial exact dynamic shortest paths algorithms [37] are partially dynamic and yield no improvement over the respective recompute-from-scratch algorithm in the sparse case either.

The lack of progress on obtaining an exact fully dynamic single-source shortest paths algorithms with $O(n^{3-\epsilon})$ initialization time, $O(m^{1-\epsilon})$ amortized update time and $O(n^{1-\epsilon})$ query time at the same time can be explained by a matching lower bound conditional on the (static) APSP conjecture [73]. In fact, breaking this barrier even in the partially dynamic setting for undirected unweighted graphs would be a large breakthrough [43].

As a result, since finding good exact algorithms for general graphs seems hopeless, one needs to look for either approximate solutions or restrict their attention to more structured graph classes. Indeed, a large body of research has been devoted to designing *approximate* dynamic shortest paths algorithms, especially in partially dynamic settings [9–13, 15, 26, 29, 43–45, 48–50, 59, 60], which find many applications, e.g., in various maximum flow related problems [29, 67]. Unfortunately, many of the known fastest approximate dynamic shortest path algorithms (e.g. [9,44,48]) suffer from assuming an oblivious adversary, which significantly limits their applicability (cf. e.g., [29]).

Similarly, dynamic shortest paths problems have also been studied for important graph classes like planar graphs [3, 4, 38, 56, 63, 66], or low treewidth-graphs [3, 58]. The primary reason why faster dynamic shortest paths algorithms in these cases are possible is the existence of non-trivial *distance oracles* for these classes. A distance oracle is a compact representation of the graph's shortest paths such that the distance (or a distance estimate) between any pair of vertices can be retrieved efficiently. For general graphs, such non-trivial distance oracles exist only for undirected graphs and assuming an approximation ratio of at least 3 [25, 79, 82]. On the contrary, for planar graphs many non-trivial exact distance oracles

[1] We will identify shortest paths queries with distance queries. Almost all known dynamic shortest paths algorithms (for some exceptions see [74, 80]) can also report the actual path in nearly linear (in the number of the path's edges) time after computing a distance estimate.

have been proposed [19,27,33,38,56,63,69]. The first exact oracles with *polylogarithmic* query time and subquadratic space have been obtained only recently [23,30,41], following Cabello's breakthrough of employing Voronoi diagrams for the planar diameter problem [20]. Also near-optimal (in terms of query time, construction time, and used space) $(1+\epsilon)$-approximate distance oracles have been known for nearly two decades [62,76], and a lot of effort has been put to push the known bounds as close to optimal as possible [22,42,61,84].

Dynamic shortest paths in planar graphs. In this paper our focus is on computing shortest paths in dynamic planar graphs and applications. Klein and Subramamian were the first to give a planarity-exploiting dynamic shortest paths algorithm [66] – their data structure worked for undirected graphs, was fully dynamic, $(1+\epsilon)$-approximate and had $\widetilde{O}(n^{2/3})$ update and query time bounds. A data structure with the same bounds (up to polylogarithmic factors), but for *exact* distances in *directed* graphs was obtained in the breakthrough work of Fakcharoenphol and Rao [38] (later extended and slightly improved in [24,40,54,56,63]). Abraham et al. [4] gave a faster $(1+\epsilon)$-approximate dynamic algorithm for undirected graphs with $\widetilde{O}(n^{1/2})$ update and query times. Karczmarz [58] matched this bound for directed planar graphs, albeit only in the $(1+\epsilon)$-approximate decremental setting. Abboud and Dahlgaard [1] showed that by the APSP conjecture, one should not expect an exact dynamic all-pairs shortest paths data structure for planar graphs with strongly sublinear product of update time and query time. However, no exact data structure to date has matched this product lower bound while retaining strongly sublinear update time.

The single-source scenario is much less studied for dynamic planar graphs. Karczmarz [58] showed a decremental $(1+\epsilon)$-approximate single-source shortest paths algorithm for minor-free (and thus also planar) digraphs with $\widetilde{O}(n^{1/2})$ update time and $O(1)$ query time. Although not explicitly stated in the literature, the all-pairs data structure of [63] can be easily converted to a fully dynamic exact single-source distance oracle with $\widetilde{O}(n^{2/3})$ update time and $\widetilde{O}(n^{1/3})$ query time. However, no *fully dynamic* single-source shortest paths algorithm for planar graphs to date has been able to achieve sublinear update time and polylogarithmic query time, or at least break through the $\widetilde{O}(n)$ update-query time product barrier, even in the approximate setting.

Our results. In this paper we show the first exact dynamic single-source shortest paths algorithm for planar graphs with *strongly sublinear* update time and *polylogarithmic* query time. Our algorithm, summarized by the following theorem and described in Section 3, is deterministic and can be easily extended to report paths.

▶ **Theorem 1.** *Let G be a real-weighted planar digraph with a source $s \in V(G)$. There exists an $O(n \log n)$-space data structure maintaining G under edge insertions, edge deletions, and source changes with $O(n^{4/5} \log^2 n)$ worst-case update time that can compute $dist_G(s,v)$ for any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^2 n)$.*

To the best of our knowledge, this result constitutes the first known application of additively weighted Voronoi diagrams machinery (first introduced by Cabello [20]) in dynamic graph algorithms. More specifically, it is obtained by combining fully dynamic maintenance of r-divisions [66,75], the shortest paths algorithm for dense distance graphs [38], the recent efficient construction of dual Voronoi diagrams [23] via FR-Dijkstra [38], and the efficient point location data structure for Voronoi diagrams [41].

We now provide a brief overview of the data structure underlying Theorem 1. We maintain distances in G from the source vertex s to each boundary vertex of each piece of an r-division of G using FR-Dijkstra. For each piece of the r-division, we maintain an additively weighted

Voronoi diagram augmented with a point location data structure, with weights equal to the distances from s. Upon a query for $\text{dist}_G(s, v)$, we perform a point location query on the Voronoi diagram of a piece of the r-division that contains v.

It is worth noting that our data structure (and in fact all data structures obtained in this paper) works in the most general model of dynamic planar graphs where we only require that G remains planar after each update. Some fully dynamic planar graph algorithms assume a weaker *plane* model (e.g., [32, 35, 54]) where some plane embedding of G is fixed and we only allow inserting edges connecting vertices that lie on a common face of (that embedding of) G.

We also generalize our single-source data structure to the case when, instead of a single source s, a set of *facilities* $F \subseteq V$ is given, and our goal is to locate the closest (i.e., minimizing $\text{dist}_G(f, v)$) facility $f \in F$ for a given query vertex.[2] We show that maintaining such a data structure under edge updates issued to G, or updates to the facilities set F, is possible using $\widetilde{O}(n^{3/4} \cdot |F|^{1/4} + n^{4/5})$ worst-case update time. The query time remains $O(\log^2 n)$. Note that even though multiple-source shortest paths or maximum flow problems can be typically easily reduced to the single-source case by adding a super-source, such a reduction does not preserve planarity and indeed handling multiple sources tends to be challenging in planar graphs (cf. e.g., [16, 17]). Our generalized data structure handles up to $O(n^{1/5})$ sources as efficiently as the single-source case. Moreover, the update time remains strongly sublinear unless the number of facilities is not strongly sublinear.

Surprisingly, we show that the same framework that we use to prove Theorem 1 can be applied to obtain interesting results not directly related to the shortest paths problem. Namely, in Section 4 we show a fully dynamic strong-connectivity algorithm for planar graphs, encapsulated in the following theorem.

▶ **Theorem 2.** *Let G be a planar digraph. There exists an $O(n \log n)$-space data structure maintaining G under edge insertions and deletions with $O(n^{4/5} \log^2 n)$ worst-case update time that can compute the identifier of the strongly connected component of any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^2 n)$.*

We now sketch the main ideas behind our fully dynamic strong-connectivity algorithm. As in Subramanian's dynamic all-pairs reachability algorithm [75], the base of our data structure is a graph X, called a *reachability certificate*, that sparsifies the reachability information between boundary vertices $\partial\mathcal{R}$ of a fully dynamic r-division $\mathcal{R}$ with few holes. Naively recomputing the strongly connected components of X gives us the restriction of the strongly connected components of G to the boundary vertices $\partial\mathcal{R}$. The main challenge, of course, is to compute the identifier of a strongly connected component (SCC) of an arbitrary non-boundary vertex v of G, internal to some piece P of the r-division $\mathcal{R}$. To this end, we use the following observation: suppose $b_1, \ldots, b_k$ are some vertices of G lying in distinct strongly connected components of G. Then, v is strongly connected to some b_j if and only if b_j is in the topologically earliest SCC of G reachable from v *and* b_j is in the topologically latest SCC of G that can reach v. Roughly speaking, this observation applied to the boundary vertices of P labeled using the topological order of their respective SCCs in the certificate X, allows us to identify the SCC of v using two point-location queries on the Voronoi diagram of piece P. Each such point location query, computes, instead of the nearest site of v, the highest (or lowest) priority site that can reach v (that v can reach, resp.), and can be simulated using a standard point location query on a Voronoi diagram [41].

[2] One can also view F as a set of sites of a graphic Voronoi diagram – then the query locates the cell of the Voronoi diagram wrt. F that a given vertex v belongs to.

Whereas maintaining strongly connected components is a well-studied problem in partially dynamic settings [8, 14, 47, 53], we are not aware of any non-trivial fully dynamic strongly connected components data structures designed specifically for this problem for *any* digraph class – note that one could use a fully dynamic transitive closure data structure for this task: for example, the dynamic plane transitive closure data structure of [32] which has $\widetilde{O}(n^{1/2})$ update and query time. Such a strongly connected components data structure (i.e., with both update and query bounds $O(n^{1-\epsilon})$) for *general graphs* is in fact ruled out by a conditional (on SETH) lower bound [2]. As a result, to the best of our knowledge, we obtain the first fully dynamic strongly connected components algorithm to achieve sublinear update-query time product for *any* important class of digraphs.

The undirected counterpart of the dynamic strongly connected components problem, the *dynamic connectivity* problem, is very well-studied. Near-optimal deterministic amortized update bounds [51,52,83] and randomized worst-case update bounds [57,81] (see also [71]) are known for fully dynamic general graphs. An almost optimal deterministic worst-case update bound was very recently achieved in [28]. For fully dynamic planar graphs polylogarithmic worst-case update bounds are known to be achievable even deterministically [34].

2 Preliminaries

Throughout the paper we consider as input a simple, directed and weighted planar graph G with n vertices, and no negative weight cycles. We call a planar graph G *plane* if some embedding of G is assumed. We use $|G|$ to denote the number of vertices of G. Since simple planar graphs are sparse, $|E(G)| = O(|G|)$ as well.

We use the terms weight and length for edges and paths interchangeably throughout the paper. For any two vertices $u, v \in V(G)$, we denote by $\mathrm{dist}_G(u, v)$ the length of some shortest $u \to v$ path in the graph G.

Multiple-source shortest paths. The multiple-source shortest paths (MSSP) data structure [21, 63] represents all shortest path trees rooted at the vertices of a single face f in a weighted plane digraph using a persistent dynamic tree. It can be constructed in $O(n \log n)$ time, requires $O(n \log n)$ space, and can report any distance between a vertex of f and any other vertex in the graph in $O(\log n)$ time. MSSP can be augmented to also return the first edge of this path (and each of its subsequent edges) in $O(\log \log n)$ time (cf. [56]).

Separators and recursive decompositions. Miller [68] showed how to compute, in a triangulated plane graph with n vertices, a simple cycle of size $2\sqrt{2}\sqrt{n}$ that separates the graph into two subgraphs, each with at most $2n/3$ vertices. Simple cycle separators can be used to recursively separate a planar graph until pieces have constant size. The authors of [64] show how to obtain a complete recursive decomposition tree $\mathcal{T}(G)$ of a triangulated graph G using cycle separators in $O(n)$ time. $\mathcal{T}(G)$ is a binary tree whose nodes correspond to subgraphs of G (pieces), with the root being all of G and the leaves being pieces of constant size. We identify each piece P with the node representing it in $\mathcal{T}(G)$. We can thus abuse notation and write $P \in \mathcal{T}(G)$. The *boundary vertices* ∂P of a *non-leaf piece* P are vertices that P shares with some other piece $Q \in \mathcal{T}(G)$ that is not P's ancestor. For convenience we extend the boundary set ∂L of a leaf piece L to its entire vertex set $V(L)$. We assume P to inherit the embedding of G. The faces of P that are faces of G are called *natural*, whereas the faces of P that are not natural are the *holes* of P. The construction of [64] additionally guarantees that for each piece $H \in \mathcal{T}(G)$, (a) H is connected, (b) if H is non-leaf, then each natural

face f of H is a face of a unique child of H, (c) H has $O(1)$ holes containing precisely the vertices ∂H. Throughout, to avoid confusion, we use *nodes* when referring to $\mathcal{T}(G)$ and *vertices* when referring to G or its subgraphs. It is well-known [18, 41, 53, 64] that by suitably choosing cycle separators one can also guarantee that (1) $\sum_{H\in\mathcal{T}(G)} |H| = O(n \log n)$, (2) $\sum_{H\in\mathcal{T}(G)} |\partial H|^2 = O(n \log n)$, and (3) $|\partial H| = O(\sqrt{n}/c^d)$, where node H of $\mathcal{T}(G)$ has depth d and $c > 1$ is some constant.

The recursive decomposition algorithm of [64] works with no changes and maintains all the properties of $\mathcal{T}(G)$ even if the initial graph G has a predefined set of boundary vertices ∂G of size $O(\sqrt{|G|})$ located on $O(1)$ of G's faces. These faces are predefined as holes of G and are the only faces of G that are allowed to be non-triangular.

An *r-division* [39] $\mathcal{R}$ of a planar graph, for $r \in [1, n]$, is a decomposition of the graph into $O(n/r)$ pieces, each of size $O(r)$, such that each piece P has $O(\sqrt{r})$ boundary vertices (denoted ∂P), i.e., vertices shared with some other piece of $\mathcal{R}$. We denote by $\partial\mathcal{R}$ the set $\bigcup_{P\in\mathcal{R}} \partial P$. If additionally all pieces are connected, and the boundary vertices of each piece P of the r-division $\mathcal{R}$ are distributed among $O(1)$ faces of P (also called holes[3] of P), we call $\mathcal{R}$ an *r-division with few holes*.

In [64] it was shown that for every r larger than some constant, $\mathcal{T}(G)$ *admits* an r-division with few holes, i.e., there exists a subset of nodes of $\mathcal{T}(G)$ forming an r-division with few holes of G. Using this property, it is shown in [64] that an r-division with few holes of a triangulated graph can be computed in linear time. More generally, given a geometrically decreasing sequence of numbers $(r_m, r_{m-1}, \ldots, r_1)$, where r_1 is a sufficiently large constant, $r_{i+1}/r_i \geq b$ for all i for some $b > 1$, and $r_m = n$, we can obtain r_i-divisions with few holes for all i in time $O(n)$ in total. For convenience, we define the only piece in the r_m-division to be G itself. These r-divisions satisfy the property that a piece in the r_i-division is a – not necessarily strict – descendant (in $\mathcal{T}(G)$) of a piece in the r_j-division for each $j > i$. We also call such sequence of r_i-divisions obtained from $\mathcal{T}(G)$ a recursive $(r_m, \ldots, r_1)$-division of G.

We assume for simplicity that all holes we ever encounter are simple cycles. Unfortunately, this is not true in general. However, non-simple holes do not pose a significant obstacle, and can be avoided by suitably extending the graphs, as discussed numerous times in the past, see e.g., [23, 53, 56, 72].

Dense distance graphs and FR-Dijkstra. For a plane digraph H with weights from $\mathbb{R}_{\geq 0} \cup \{\infty\}$ and a distinguished set $\partial H \subseteq V(H)$ of boundary vertices lying on $O(1)$ faces of H, we denote by DDG_H the complete weighted graph on ∂H whose edge weights represent distances between all pairs of vertices of ∂H in H. DDG_H can be computed in $O\left((|H| + |\partial H|^2) \log n\right)$ time using MSSP [63]. In particular, dense distance graphs for all pieces $H \in \mathcal{T}(G)$ can be computed in $O(n \log^2 n)$ time.

When $\mathcal{H} = \{H_1, \ldots, H_q\}$ is a collection of plane graphs, we set $\mathrm{DDG}(\mathcal{H}) := \bigcup_{H\in\mathcal{H}} \mathrm{DDG}_H$.

▶ **Lemma 3** (FR-Dijkstra [38, 40]). *Given all DDG_{H_i}, one can compute a single-source shortest paths tree from any source s in $\mathrm{DDG}(\mathcal{H})$ in $O\left(\sum_{i=1}^{q} |\partial H_i| \log^2 n\right)$ time, where $n = |V(\mathrm{DDG}(\mathcal{H}))|$.*[4]

[3] This definition is slightly more general than the definition of a hole of a piece $P \in \mathcal{T}(G)$. Namely, the definition of an r-division does not assume a fixed embedding of the entire G; it only assumes some fixed embeddings of individual pieces.

[4] In particular H_i may be single-edge. This way, this lemma captures also the case when we compute shortest paths in a collection of DDGs with some auxiliary vertices and edges.

We now state some fairly standard definitions and lemmas about representing distances between some vertices of G of interest using unions of dense distance graphs of a recursive decomposition's (or r-division's) pieces (for instance cf. [24]), adapted to our notation. We include their proofs for completeness in Appendix A. For example, Lemma 4 captures the well-known observation of [38] that in order to compute a shortest path between any pair of vertices of G, it is enough to compute a shortest path in a union of dense distance graphs from $\mathcal{T}(G)$ with only $O(\sqrt{n})$ vertices in total.

Let L be some leaf of $\mathcal{T}(G)$. We define the *cone* of L, denoted $\mathrm{cone}_G(L)$, to be the collection of pieces of $\mathcal{T}(G)$ containing L, all ancestors of L, and all siblings of (weak) ancestors of L. For some collection $\mathcal{L}$ of leaf pieces of $\mathcal{T}(G)$, we define $\mathrm{cone}_G(\mathcal{L}) = \bigcup_{L\in\mathcal{L}} \mathrm{cone}_G(L)$.

▶ **Lemma 4** ([24, 38]). *Let $\mathcal{L}$ be some collection of leaf pieces of $\mathcal{T}(G)$. Then:*

1. *For any $u, v \in V(\mathrm{DDG}(\mathrm{cone}_G(\mathcal{L})))$, $dist_G(u,v) = dist_{\mathrm{DDG}(\mathrm{cone}_G(\mathcal{L}))}(u,v)$.*
2. $\sum_{H\in\mathrm{cone}_G(\mathcal{L})} |\partial H| = O\left(\sqrt{n|\mathcal{L}|}\right)$.

Let $\mathcal{R}$ be an r-division with few holes of G and $\mathcal{T}(P)$ be a recursive decomposition of $P \in \mathcal{R}$ with the root boundary set to ∂P. For any $v \in V(G) \setminus \partial G$, let L_v be some leaf containing v in the unique piece $P_v \in \mathcal{R}$ containing v. For any $X \subseteq V(G)$ let us define $\mathrm{cone}_{\mathcal{R}}(X) = \mathcal{R} \cup \bigcup_{v\in X\setminus\partial\mathcal{R}} \mathrm{cone}_{P_v}(L_v)$.

▶ **Lemma 5** ([24, 38]). *Let $X \subseteq V(G)$ be non-empty. Then:*

1. *For any $u, v \in V(\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(X)))$, $dist_G(u,v) = dist_{\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(X))}(u,v)$.*
2. $\sum_{H\in\mathrm{cone}_{\mathcal{R}}(X)} |\partial H| = O\left(n/\sqrt{r} + \min\left(\sqrt{n\cdot|X|}, |X|\cdot\sqrt{r}\right)\right)$.

Fully dynamic r-divisions. Many dynamic algorithms for planar graphs maintain r-divisions and useful auxiliary data structures under dynamic updates. The exact set of supported updates to G varies; e.g., [38,56,63] support only edge weight changes, [54] assumes embedding-preserving insertions, whereas [66, 75] only assume that the graph G remains planar at all times. We stick to the last, most general setting. The core of the construction behind the following theorem is due to Klein and Subramanian [66, 75]; for completeness we give a complete proof in Appendix A.

▶ **Theorem 6.** *Let $G = (V, E)$ be a weighted planar graph. Suppose that adding infinite-weight edges to G does not have effect on any properties of G that we care about. Let $r \in [1, n]$.*

There is a data structure maintaining an r-division with few holes $\mathcal{R}$ of some G^+ such that:

1. *G^+ is obtained from G by adding infinite-weight edges.*
2. *Each P has all its faces except its holes triangular and is accompanied with some auxiliary data structures that can be constructed in $T(r)$ time given P and use $S(r)$ space.*

The data structure uses $O\left(n + \frac{n}{r}\cdot S(r)\right)$ space and can be initialized in $O\left(n + \frac{n}{r}\cdot T(r)\right)$ time. After each edge deletion and edge insertion (preserving the planarity of G), it can be updated in $O(r + T(r))$ worst-case *time.*

Additively weighted Voronoi diagrams. Let G be a directed planar graph of size n with real edge-lengths, and no negative-length cycles. Assume that all faces of G are triangles except, perhaps, a single face f. Let S be the set of vertices that lie on f, called *sites*, i.e., $S = V(f)$. Let us assign to each site $s \in S$ a weight $\omega(s) \in \mathbb{R}_{\geq 0} \cup \{\infty\}$. The additively weighted distance $\mathrm{dist}^{\omega}_G(s, v)$ between a site $s \in S$ and a vertex $v \in V(G)$ is defined as $\omega(s) + \mathrm{dist}_G(s, v)$.

The additively weighted Voronoi diagram of (S, ω) within G, denoted by $\mathsf{VD}(S, \omega)$, is a partition of $V(G)$ into pairwise disjoint sets, one set $\mathsf{Vor}(s)$ for each site $s \in S$. The set $\mathsf{Vor}(s)$, which is called the Voronoi cell of s, contains all vertices in $V(G)$ that are closer (wrt. $\mathrm{dist}^{\omega}_{G}$) to s than to any other site in S.

In the following and throughout, whenever we work with Voronoi diagrams we assume that (1) G is strongly connected, (2) shortest paths in G are unique, and (3) additively weighted shortest paths in G are unique, i.e., for each $v \in V(G)$ there is a unique site s minimizing $\mathrm{dist}^{\omega}_{G}(s, v)$. Note that these assumptions make the Voronoi cells well-defined and simply connected, and guarantee that they indeed form a partition of $V(G)$. We will explicitly ensure that these requirements are met for G, S and the weight function whenever we define a Voronoi diagram on G.

There is a dual representation $\mathsf{VD}^*(S, \omega)$ of Voronoi diagram $\mathsf{VD}(S, \omega)$ as a tree of constant degree with $O(|S|)$ vertices and edges [41]. An efficient FR-Dijkstra based algorithm for computing $\mathsf{VD}^*(S, \omega)$ was presented by Charalampopoulos et al. [23].

▶ **Theorem 7** ([23]). *Suppose that we have at hand a recursive decomposition $\mathcal{T}(G)$ of G, with the only hole of G being f and $S = V(f)$. Further suppose that we have DDG_H computed for each piece $H \in \mathcal{T}(G)$. Then, we can compute $\mathsf{VD}^*(S, \omega)$ in $O(\sqrt{n \cdot |S|} \log^2 n)$ time.*

▶ Remark 8. The algorithm underlying Theorem 7 implicitly assumes that $\mathsf{Vor}(s)$ is non-empty for all $s \in S$. In Appendix B, we discuss why this assumption is not necessary, relying on [41].

In a *point location* query for some Voronoi diagram $\mathsf{VD}(S, \omega)$, we are given a vertex $v \in V(G)$ and are requested to find the site $s \in S$ such that $v \in \mathsf{Vor}(s)$ and also the value of $\mathrm{dist}^{\omega}_{G}(s, v)$. Gawrychowski et al. [41] showed the following result.

▶ **Theorem 9** ([41]). *Suppose that we have at hand an MSSP data structure for G with sources from the face f. Given some dual representation $\mathsf{VD}^*(S, \omega)$, we can preprocess it in $O(|S|)$ time, so that point location queries for $\mathsf{VD}(S, \omega)$ can be answered in $O(\log^2 n)$ time.*

3 Fully Dynamic Single Source Shortest Paths

In this section we show our single-source exact distance oracle for planar graphs with $O(n^{4/5} \log^2 n)$ update time and $O(\log^2 n)$ query time and thus prove Theorem 1. For simplicity, let us assume that G is non-negatively weighted. Negative edges can be handled as in [56] – see Appendix C.

▶ **Theorem 1.** *Let G be a real-weighted planar digraph with a source $s \in V(G)$. There exists an $O(n \log n)$-space data structure maintaining G under edge insertions, edge deletions, and source changes with $O(n^{4/5} \log^2 n)$ worst-case update time that can compute $\mathrm{dist}_G(s, v)$ for any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^2 n)$.*

The base of our data structure is a dynamic r-division $\mathcal{R}$ with few holes, as given in Theorem 6. Note that in our shortest-paths problem, indeed adding infinite-weight edges to G does no harm. Hence, in the following we work with the graph G^+ from Theorem 6 when computing distances, but identify it, without loss of generality, with our original graph G.

For technical reasons, however, we would like to avoid dealing with infinite weights in some of our data structures handling individual pieces. In the real-weighted fully dynamic setting, however, we cannot fix a sufficiently large finite number, larger than all edge weights that will ever appear in the future graph G, beforehand. Instead, we do the following. For

each $P \in \mathcal{R}$, let M_P be a sufficiently large finite number, e.g., larger than the sum of finite edge weights in P. Consider M_P to be an auxiliary data structure of P as in Theorem 6. We will use M_P to simulate infinite edge weights in P, and also for detecting paths non-existent in the original graph G (but having infinite weight in $G^+ \cap P$). As a result, below we assume each infinite weight in P (or any auxiliary data structure related to P) is replaced by M_P in all the computations performed locally on the piece P, whereas globally (when performing some computation for many pieces at once, like the shortest paths algorithm of Lemma 3) we treat all edge or path weights in P that are at least M_P as infinite.

For each piece $P \in \mathcal{R}$ we store the following additional data structures.

- We store the recursive decomposition $\mathcal{T}(P)$ with the initial boundary set to ∂P, and also DDGs for all the pieces $H \in \mathcal{T}(P)$.
- For each hole h of P, let P_h be the piece P after applying the following standard augmentations. First, P is extended into a graph P'_h using $O(r)$ vertices and edges of weight M_P embedded inside either the piece or other (than h) holes of P that would make P strongly connected and triangulated (except for the hole h) without changing the distance between any pair of reachable vertices in P. The graph P_h is in turn obtained from P'_h by changing P'_h's edge weights into $O(1)$-size vectors as described in [36] so that there is a *unique* shortest path (wrt. the lexicographical order on path weights, defined as the coordinate-wise sum of the path's individual edge weights) between any $u, v \in V(P_h)$ with cost of the form $(\mathrm{dist}_{P'_h}(u, v), \cdot)$. As proven in [36], one can compute P_h from P deterministically in linear time. The $O(1)$-size vector weights, in turn, can be easily packed into usual single-number weights.

  For each P_h, we store an MSSP data structure initialized for the hole h. Recall that an MSSP data structure can be computed in $O(r \log r)$ time. Moreover, we store a recursive decomposition $\mathcal{T}(P_h)$ of P_h with the boundary ∂P_h of the root piece set to $\partial P \cap V(h)$ of size $O(\sqrt{r})$. For each node (piece) $H \in \mathcal{T}(P_h)$, we also store DDG_H. Since the sum of sizes of all the pieces of $\mathcal{T}(P_h)$ is $O(r \log r)$, computing all these dense distance graphs takes $O(r \log^2 r)$ time (see Section 2).

Note that computing piecewise auxiliary data structures defined so far takes $O(r \log^2 r)$ time. So, by Theorem 6, they can be updated in $O(r \log^2 r)$ worst-case time after G undergoes an update.

After the initialization and each update, once $\mathcal{R}$ and all auxiliary data structures are updated, we compute for each $P \in \mathcal{R}$ a point location data structure.

▶ **Lemma 10.** *Given a weight function $\omega : \partial P \to \mathbb{R} \cup \infty$, one can compute in $O(r^{3/4} \log^2 r)$ time a data structure $L(P)$ answering the following queries in $O(\log^2 r)$ time: given any $v \in V(P)$, compute the value $\min_{b \in \partial P}\{\omega(b) + dist_P(b, v)\}$ along with the minimizer b.*

Proof. Since ∂P is a union of $O(1)$ sets ∂P_h, we can compute the desired minimum over each ∂P_h separately and then take the minimum over all h.

Let us first note that negative values of ω are not a problem. We can turn negative weights into non-negative by adding some common large value to the weights of all sites. We can thus suppose wlog. that all values of ω are non-negative.

If all the weights are infinite, the queries can be answered trivially in $O(1)$ time. So in the following assume that there is at least one site whose weight is finite.

Let $S = \partial P_h = \{s_1, \ldots, s_k\}$ be the set of sites. In order to guarantee that for each $v \in V(G)$ there is a unique site s minimizing $\mathrm{dist}^{\omega}_G(s, v)$, we will break ties by considering $(\omega(s_i), i)$ instead of $\omega(s_i)$ as the weight of site s_i, adding a second coordinate to each edge weight in P_h, set to 0, and comparing additively weighted distances lexicographically. Clearly, this extension does not break any of the properties of P_h.

Recall that P_h has finite non-negative real weights, is strongly connected, has unique shortest paths, and has a single face h that is possibly non-triangular that contains all the sites S. Moreover, the additively weighted distances in P_h are unique. Therefore, we can invoke Theorem 7 to construct the dual representation $\mathsf{VD}^*(S,\omega)$ of the Voronoi diagram $\mathsf{VD}(S,\omega)$. This requires $O\left(\sqrt{r|\partial P_h|}\log^2 n\right) = O\left(r^{3/4}\log^2 n\right)$ time.

Then, we construct the point location data structure of Theorem 9 for $\mathsf{VD}(S,\omega)$. Note that we have an MSSP data structure for P_h for hole h and hence point location queries, given $\mathsf{VD}^*(S,\omega)$, can be answered in $O(\log^2 n)$ time. ◀

We invoke Lemma 10 with weight function $\omega(b) := \mathrm{dist}_G(a,b)$ in order to construct $L(P)$ for each $P \in \mathcal{R}$. This requires $O\left(n/r^{1/4}\cdot\log^2 n\right)$ time in total. By Lemma 5, the values $\mathrm{dist}_G(s,b)$ for all $b \in \partial R$ can be computed in $O\left(n/\sqrt{r}\cdot\log^2 n\right)$ time if we run the single-source shortest paths algorithm of Lemma 3 (FR-Dijkstra) on the graph $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s))$. We also compute $\mathrm{dist}_{P_s}(s,u)$ for all $u \in P_s$, where P_s is an arbitrary piece containing s using Dijkstra's algorithm in $O(r\log r)$ time.

Now, we can compute $\mathrm{dist}_G(s,v)$ for a query vertex v as follows. If the shortest $s \to v$ path in G does not go through $\partial\mathcal{R}$, then it is fully contained in P_s and therefore $v \in P_s$ and $\mathrm{dist}_G(s,v) = \mathrm{dist}_{P_s}(s,v)$, i.e., we have $\mathrm{dist}_G(s,v)$ already computed. Otherwise, let P_v be an arbitrary piece containing v. Observe that we have $\mathrm{dist}_G(s,v) = \min_{b\in\partial P_v}\{\mathrm{dist}_G(s,b) + \mathrm{dist}_{P_v}(b,v)\}$ where the minimizer b corresponds to the a boundary vertex of some shortest $s \to v$ path in G. So this case can be reduced to a single query to the data structure $L(P_v)$. This takes $O(\log^2 n)$ time.

The worst-case update time is $O(r\log^2 r + \frac{n}{r^{1/4}}\log^2 r)$. By setting $r = n^{4/5}$ we get $O(n^{4/5}\log^2 n)$ worst-case update time. Since the space usage per piece is $O(r\log r)$, we need $O(n\log n)$ space.

▶ Remark 11. Our data structure can be extended to report, following the computation of $\mathrm{dist}_G(s,v)$, a shortest $s \to v$ path Q in time nearly linear in the number of edges of Q. This follows easily by the fact that the MSSP data structure [63] can report shortest paths efficiently (see e.g., [56] for details). Therefore, we can efficiently expand the used edges of dense distance graphs and the shortest $b \to v$ path into actual edges in G.

3.1 A Dynamic Closest Facility Data Structure

We can generalize the dynamic single-source shortest paths data structure as follows. Suppose we replace a single source vertex s with a set of *facilities* $F \subseteq V$. Given F, for a query vertex v we would like to compute $\min_{f\in F}\{\mathrm{dist}_G(f,v)\}$, and also possibly $f \in F$ minimizing this expression. A dynamic update would consist of either an edge update or changing the set F. In other words, such a problem can be seen as dynamic point location in a Voronoi diagram wrt. F, where each update either changes the graph or resets the Voronoi diagram of interest.

In this setting, we consider the following simple generalization of the single-source data structure. Let the update procedure first compute the distances $d(b) = \min_{f\in F}\{\mathrm{dist}_G(f,b)\}$ for all $b \in \partial\mathcal{R}$. Note that by Lemmas 5 this can be achieved by computing single-source shortest paths in the graph $\mathrm{DDG}(\mathcal{D}_F)$, where $\mathcal{D}_F = \mathrm{cone}_{\mathcal{R}}(F)$, extended with 0-weight edges sf, where s is an auxiliary super-source. By Lemma 3 this can be done in $O((\sqrt{n|F|}+n/\sqrt{r}+|F|)\log^2 n) = O((\sqrt{n|F|}+n^{4/5})\log^2 n)$ time. By using weights $\omega := d$ in the individual point-location data structures $L(P)$, $P \in \mathcal{R}$, a single point location query on $L(P_v)$ (recall that P_v is some piece containing v) would compute the desired closest facility f_v minimizing $\mathrm{dist}_G(f_v,v)$ unless the sought (weighted) shortest $f_v \to v$ does not go through a boundary

vertex of $\mathcal{R}$. We could in principle handle such paths by proceeding as in the single-source case and computing shortest paths naively in each piece containing a facility. However, this could take time $\Omega(r \cdot \min(n/r, |F|))$, i.e., linear in n even for moderately large facility set sizes, e.g., $|F| = \Omega(n^{1/5})$.

To improve upon this simple approach, we proceed as follows. Let $(\rho_m, \ldots, \rho_1)$ be such a sequence of integers that $\rho_m = r$, $\rho_1 = O(1)$ and $\rho_{i+1}/\rho_i = 2$ for all $i < m$. For each $P \in \mathcal{R}$ we store a recursive $(\rho_m, \ldots, \rho_1)$-division consisting of pieces of $\mathcal{T}(P)$ (cf. Section 2). Let $\mathcal{R}_{P,i}$ be the ρ_i-division of P. All $\mathcal{R}_{P,i}$ can be computed in linear time given $\mathcal{T}(P)$ [64]. Note that $\mathcal{R}_i$, defined as the union of $\mathcal{R}_{P,i}$ over all pieces $P \in \mathcal{R}$, actually forms an ρ_i-division with few holes of the entire graph G. In particular, we have $\mathcal{R}_m = \mathcal{R}$.

We store the extended pieces Q_h (recall how we obtained extended pieces P_h with unique shortest paths from P in the single-source case) plus their recursive decompositions $\mathcal{T}(Q_h)$, DDGs, and an MSSP data structure for all pieces Q of *all* $\mathcal{T}(P)$ instead of just the pieces of $\mathcal{R}_m = \mathcal{R}$ as we did in the single-source case. However, we stress that these auxiliary components for a piece $Q \subseteq P$ where $P \in \mathcal{R}$, are counted as accompanying data structures of the piece P. So, we compute $O(1)$ fresh recursive decompositions $\mathcal{T}(Q_h)$ for each piece $Q \in \mathcal{T}(P)$ – computing each takes $O(|Q| \log^2 n)$ time. As a result, by the bound $\sum_{Q \in \mathcal{T}(P)} |Q| = O(|P| \log |P|)$, the time to compute accompanying data structures of piece P increases to $O(r \log^3 n)$.

Given the set of facilities F, let j be such that $\rho_j = \Theta\left(\min\left(\frac{n}{|F|}, r\right)\right)$. Redefine $\mathcal{D}_F = \mathrm{cone}_{\mathcal{R}_j}(F)$. Again, let us compute distances $d(b) = \min_{f \in F}\{\mathrm{dist}_G(f, b)\}$ (and the closest facilities) for all $b \in \bigcup_{H \in \mathcal{D}_F} |\partial H|$ using FR-Dijkstra on DDG$(\mathcal{D}_F)$ extended with a super-source s and auxiliary edges sf, $f \in F$. This takes $O\left(\left(\sqrt{n|F|} + n/\sqrt{\rho_j}\right) \log^2 n\right) = O(\sqrt{n|F|} \log^2 n)$ time by Lemmas 3 and 5.

It only remains to show how to handle computation of closest facilities for $v \in V \setminus \bigcup_{H \in \mathcal{D}_F} \partial H$. Recall that the pieces $\mathcal{D}_F$ cover the entire G, no facility f is an internal (non-boundary) vertex of a piece $H \in \mathcal{D}_F$, and each $v \in V \setminus \bigcup_{H \in \mathcal{D}_F} \partial H$ is clearly an internal vertex of a unique piece $H_v \in \mathcal{D}_F$. Consequently, by Lemma 10, the closest facility to v can be found in $O(\log^2 n)$ time using a single query to the data structure $L(H_v)$. For this to be possible, upon update we need to build the data structures $L(H)$ for all $H \in \mathcal{D}_F$. By Lemma 10, this takes $O\left(\sum_{H \in \mathcal{D}_F} \sqrt{|H| \cdot |\partial H|} \log^2 n\right)$ time. Let us now bound this sum. First, let us consider the sum restricted to the pieces $H \in \mathcal{D}_F \cap \mathcal{R}_j$, i.e., the pieces of r-division $\mathcal{R}_j$. Since $\rho_j = \Omega(n/|F|)$ or $\rho_j = \Omega(r)$ we get:

$$O\left(\sum_{H \in \mathcal{D}_F \cap \mathcal{R}_j} \sqrt{|H| \cdot |\partial H|} \log^2 n\right) = O\left(\frac{n}{\rho_j} \cdot \rho_j^{3/4} \log^2 n\right) = O\left(\left(n^{3/4} \cdot |F|^{1/4} + \frac{n}{r^{1/4}}\right) \log^2 n\right).$$

On the other hand, if $H \notin \mathcal{D}_F \cap \mathcal{R}_j$, then $H \in \mathrm{cone}_{P_f}(L_f)$, where $f \in F \setminus \mathcal{R}_j$, P_f is the unique piece of $\mathcal{R}_j$ containing f, and L_f is some leaf of $\mathcal{T}(P_f)$ containing f. For a fixed f, by the definition of $\mathrm{cone}_{P_f}(L_f)$, there are $O(\log n)$ pieces H satisfying this, at most two per each level i of $\mathcal{T}(P_f)$. Hence, the sum of $\sqrt{|H| \cdot |\partial H|}$ over such pieces can be bounded by $\sum_{i=0}^{\infty} \sqrt{|P_f| \cdot \sqrt{|P_f|}/c^i} = O(\rho_j^{3/4})$. Summing over all f, and using $\rho_j = O(n/|F|)$, we get

$$O\left(\sum_{H \in \mathcal{D}_F \setminus \mathcal{R}_j} \sqrt{|H| \cdot |\partial H|} \log^2 n\right) = O\left(|F| \cdot \rho_j^{3/4} \log^2 n\right) = O\left(n^{3/4} \cdot |F|^{1/4} \log^2 n\right).$$

To conclude, the update time is $O\left(\left(n^{3/4} \cdot |F|^{1/4} + \frac{n}{r^{1/4}}\right) \log^2 n + r \log^3 n\right)$. By setting $r = (n/\log n)^{4/5}$ we obtain the following theorem.

▶ **Theorem 12.** *Let G be a real-weighted planar digraph with a set $F \subseteq V$ of facilities. There exists an $O(n \log^2 n)$-space data structure maintaining G under edge insertions, edge deletions, and changes of the facilities set F with $O\left(\left(n^{3/4} \cdot |F|^{1/4} + n^{4/5}\right) \log^{11/5} n\right)$ worst-case update time that can compute $\min_{f \in F}\{dist_G(f, v)\}$ along with the respective closest facility of v for any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^3 n)$.*

4 Fully Dynamic Strongly Connected Components

In this section we show that a strategy similar to that of Section 3 can be used to obtain a fully dynamic strong-connectivity algorithm. Again, we maintain an r-division $\mathcal{R}$ and some auxiliary data structures for all the individual pieces. Formally, using a dynamic r-division as in Theorem 6 may require introducing new infinite-weight edges to G which in turn may change the reachability relation in G. We circumvent this problem by setting the weights of the original edges of G to 0, and all auxiliary edges plus infinity (simulated in the implementation by sufficiently large values M_P inside individual pieces, as in Section 3). This way, u can reach v in G if and only if $\text{dist}_G(u, v) = 0$, and otherwise $\text{dist}_G(u, v) = \infty$. All known properties of reachability in plane graphs also extend to reachability using 0-weight paths (assuming non-negative weights). In the following, whenever we say that v is reachable from u, or there exists a $u \to v$ path, we mean $\text{dist}_G(u, v) = 0$.

As in Section 3, for each piece of $\mathcal{R}$ we store a recursive decomposition, dense distance graphs and MSSP data structures. All these data structures are also maintained for pieces of $\mathcal{R}$ with all edges reversed – for a piece P we call this graph the *reverse* of P and denote it by P^{rev}.

Another ingredient is a collection of *reachability certificates* X_P for all the pieces, as defined in the following lemma due to Subramanian [75], slightly adjusted to certify 0-weight paths.

▶ **Lemma 13** ([75])**.** *Let $P \in \mathcal{R}$ be a piece. There exists a directed graph X_P, where $\partial P \subseteq V(X_P)$, of size $O(\sqrt{r} \log r)$ satisfying the following property: for any $u, v \in \partial P$, $dist_P(u, v) = 0$ if and only if there exists a $u \to v$ path in X_P. The graph X_P can be computed in $O(r \log r)$ time.*

We include the reachability certificate in the set of auxiliary piecewise data structures. Since reachability certificates can be computed in $O(r \log r)$ time, maintaining them does not incur any additional asymptotic cost. The following lemma is a direct consequence of Lemma 13.

▶ **Lemma 14.** *For any $u, v \in \partial\mathcal{R}$, u can reach v in G if and only if u can reach v in $X = \bigcup_{P \in \mathcal{R}} X_P$.*

Proof. Let $u, v \in \partial\mathcal{R}$. Since each X_P certifies the reachability between ∂P in P, clearly a $u \to v$ path in X implies an existence of a $u \to v$ path in G. Now suppose there is a $u \to v$ path Q in G. Split P into maximal subpaths $Q_1, \ldots, Q_k$, such that each Q_i is fully contained in a single piece $P_i \in \mathcal{R}$. For each i, the endpoints a, b of Q_i are contained in ∂P_i and hence there exists a $a \to b$ path in $X_P \subseteq X$. Consequently, there exist a $u \to v$ path in X. ◀

To handle an edge update, after $\mathcal{R}$ and auxiliary data structures are updated, we compute the strongly connected components of X (defined as in Lemma 14) in $O(|X|) = O(n/\sqrt{r})$ time using any classical linear-time algorithm. For any $b \in \partial\mathcal{R}$, let $s_X(b)$ denote an integer

identifier of b's strongly connected component in X. By additionally sorting the SCCs of X topologically we can further assume that s_X satisfies the following property: if $a, b \in \partial\mathcal{R}$ are not strongly connected, but a can reach b in X then $s_X(a) < s_X(b)$. By Lemma 14, for $a, b \in \partial\mathcal{R}$, we have $s_X(a) = s_X(b)$ if and only if a and b are strongly connected in G; moreover, if a can reach b in G, then $s_X(a) \leq s_X(b)$.

We also define and maintain similar SCC-identifiers s_P for individual pieces P, i.e., for $u, v \in V(P)$, $s_P(u) = s_P(v)$ implies u, v are strongly connected in P, whereas $s_P(u) < s_P(v)$ implies there is no $v \to u$ path in P. Clearly, the identifiers s_P can be recomputed in $O(r)$ time given P, so we also include them into the set of auxiliary per-piece data structures.

For any $Q \in \{X\} \cup \mathcal{R}$, let S_Q be the set of used identifiers of the form $s_Q(\cdot)$. Observe that we can easily guarantee that the sets S_Q are pairwise disjoint, e.g., by using disjoint integer ranges for different sets S_Q.

The final component of our data structure, is, again a collection of per-piece point location data structures. For each $P \in \mathcal{R}$, we have two point location data structures $L(P^{\text{rev}})$ and $L(P)$ of Lemma 10. After each edge update, $L(P^{\text{rev}})$ is computed for P^{rev} with weight function $\omega = s_X$. On the other hand, $L(P)$ is initialized with weight function $\omega = -s_X$. As in Section 3, all these point location data structures are recomputed in $O(n/r^{1/4} \cdot \log^2 n)$ time (over all pieces).

We now describe how our data structure handles a query for an SCC identifier of a vertex v. The returned identifier always comes from the set $\bigcup_{Q \in \{X\} \cup \mathcal{R}} S_Q$. Let P_v be some piece containing v. Let $s_{\min}$ be the value computed by $L(P_v^{\text{rev}})$ for vertex v. Let $s_{\max}$ be *minus* the value computed by $L(P_v)$ for v. If either of $s_{\min}, s_{\max}$ equals $\pm\infty$ or $s_{\min} \neq s_{\max}$ holds, we return $s_{P_v}(v)$. Otherwise, we return $s_{\min} \in S_X$. The following lemma establishes the correctness of this query procedure.

▶ **Lemma 15.** *Let $u, v \in V(G)$ and let s_u, s_v be the respective identifiers returned by the query procedure. Then, $s_u = s_v$ if and only if u and v are strongly connected in G.*

Proof. Suppose $s_u = s_v$. If $s_u \in S_P$ for some piece P, then $s_u = s_v$ implies that u and v are strongly connected in P and thus also in G. So suppose $s_v \in S_X$. Take any $b \in \partial\mathcal{R}$ such that $s_v = s_X(b)$. We now prove that v and b are strongly connected in G. Similarly we prove that u and b are strongly connected in G. By transitivity it will follow that u and v are indeed strongly connected.

Let $P_v, s_{\min}, s_{\max}$ be defined as in the query procedure's description. Recall that $s_v \in S_X$ implies that $s_{\min}, s_{\max}$ are finite and $s_v = s_{\min} = s_{\max}$. Since all edges of P_v have weight 0, and $s_{\max}$ is finite, $s_{\max}$ in fact represents the maximum value $s_X(a)$ among those $a \in \partial P_v$ such that a path $a \to v$ exists in P_v. Similarly, observe that $s_{\min}$ represents the minimum value $s_X(c)$ among those $c \in \partial P_v$ such that a path $c \to v$ exists in P_v^{rev}, i.e., such that a path $v \to c$ exists in P_v. Let us denote by a and c the respective vertices of ∂P_v attaining the maximum and minimum values of s_X. Since $s_X(a) = s_X(c)$, there exists a path $c \to a$ in G. However, by the definition of a and c, paths $a \to v$ and $v \to c$ also exist in G, and hence a, v and c are strongly connected in G. Since a is clearly strongly connected to b by $s_X(a) = s_X(b)$, indeed v and b are strongly connected in G.

Now let us move to proving the " $\Longleftarrow$ " direction. Suppose u and v are strongly connected in G. First consider the case when there exists some vertex $b \in \partial\mathcal{R}$ located in the same strongly connected component of G as u and v. In this case we prove that $s_v = s_X(b)$. An analogous proof that $s_u = s_X(b)$ will establish $s_u = s_v$. Since v and b are strongly connected, there exist some paths $Q_1 = v \to b$ and $Q_2 = b \to v$ in G. Let v_1 be the first vertex on Q_1 such that $v_1 \in \partial P_v$ – note that v_1 necessarily exists since $b \in \partial\mathcal{R}$. Similarly set v_2 to be

the last vertex on Q_2 such that $v_2 \in \partial P_v$. Observe that the subpaths $v \to v_1$ and $v_2 \to v$ of Q_1 and Q_2 respectively lie entirely inside P_v. Hence, $s_{\min}$ and $s_{\max}$ are finite and we have $s_{\max} \geq s_X(v_2)$ and $s_{\min} \leq s_X(v_1)$. Recall that there exists a walk $v \to v_1 \to b \to v_2 \to v$, so in fact v_1, v_2, b are strongly connected, i.e., $s_X(v_1) = s_X(v_2) = s_X(b)$. Thus, we obtain $s_{\min} \leq s_X(b) \leq s_{\max}$.

On the other hand, let $a \in \partial P_v$ be such that a path $a \to v$ exists in P_v and $s_{\max} = s_X(a)$ (a exists by $s_{\max} \neq \pm\infty$). Similarly, let $c \in \partial P_v$ be such that a path $v \to c$ exists in P_v and $s_{\min} = s_X(c)$. Since a path $a \to c$ through v exists in P_v (so also in G), we have that $s_{\max} \leq s_{\min}$ by the fact that the identifiers S_X respect the topological order of the SCCs of X. Recall that we have already proved $s_{\min} \leq s_X(b) \leq s_{\max}$ so in fact we have $s_{\min} = s_{\max} = s_X(b)$, and consequently $s_v = s_X(b)$.

Finally, suppose there is no vertex of $\partial\mathcal{R}$ in the SCC of G containing u and v. First, this implies that $u, v \notin \partial\mathcal{R}$ and all $u \to v$ and $v \to u$ paths are contained in a single, unique piece P. This implies that $s_P(u) = s_P(v)$. Hence it is sufficient to prove $s_u = s_P(u)$ and $s_v = s_P(v)$. We prove the latter equality; proving the former is analogous. Recall that s_v is *not* set to $s_P(v)$ only if both $s_{\min}, s_{\max}$ are finite and $s_{\min} = s_{\max}$. This can only happen if there exists vertices $a, c \in \partial P$ such that a can reach v in P, v can reach c in P and $s_X(c) = s_{\min} = s_{\max} = s_X(a)$, i.e., a and c are strongly connected in G. But this implies that a, c and v are strongly connected in G, which contradicts the fact that the SCC of v in G does not contain vertices of $\partial\mathcal{R}$. ◀

The running time analyses of both the update and query procedures are identical to the analyses of Section 3. Hence, we have proved the following theorem.

▶ **Theorem 2.** *Let G be a planar digraph. There exists an $O(n \log n)$-space data structure maintaining G under edge insertions and deletions with $O(n^{4/5} \log^2 n)$ worst-case update time that can compute the identifier of the strongly connected component of any $v \in V(G)$ in $O(\log^2 n)$ time. The initialization time is $O(n \log^2 n)$.*

References

1. Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016*, pages 477–486, 2016. doi:10.1109/FOCS.2016.58.
2. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.
3. Ittai Abraham, Shiri Chechik, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. On dynamic approximate shortest paths for planar graphs with worst-case costs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 740–753, 2016. doi:10.1137/1.9781611974331.ch53.
4. Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 1199–1218, 2012. doi:10.1145/2213977.2214084.
5. Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 440–452, 2017. doi:10.1137/1.9781611974782.28.

6 Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1990*, pages 12–21, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320178.

7 Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *J. Algorithms*, 62(2):74–92, 2007. doi:10.1016/j.jalgor.2004.08.004.

8 Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Robert E. Tarjan. A new approach to incremental cycle detection and related problems. *ACM Trans. Algorithms*, 12(2):14:1–14:22, 2016. doi:10.1145/2756553.

9 Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM J. Comput.*, 45(2):548–574, 2016. doi:10.1137/130938670.

10 Aaron Bernstein. Deterministic partially dynamic single source shortest paths in weighted graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 44:1–44:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.44.

11 Aaron Bernstein and Shiri Chechik. Deterministic decremental single source shortest paths: beyond the o(mn) bound. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 389–397, 2016. doi:10.1145/2897518.2897521.

12 Aaron Bernstein and Shiri Chechik. Deterministic partially dynamic single source shortest paths for sparse graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 453–469, 2017. doi:10.1137/1.9781611974782.29.

13 Aaron Bernstein, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Near-optimal decremental SSSP in dense weighted digraphs. *CoRR*, abs/2004.04496, 2020. arXiv:2004.04496.

14 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 365–376. ACM, 2019. doi:10.1145/3313276.3316335.

15 Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. *CoRR*, abs/2004.08432, 2020. arXiv:2004.08432.

16 Glencora Borradaile and Philip N. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2):9:1–9:30, 2009. doi:10.1145/1502793.1502798.

17 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.

18 Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms*, 11(3):16:1–16:29, 2015. doi:10.1145/2684068.

19 Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1-2):361–381, 2012. doi:10.1007/s00453-010-9459-0.

20 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *ACM Trans. Algorithms*, 15(2):21:1–21:38, 2019. doi:10.1145/3218821.

21 Sergio Cabello, Erin W. Chambers, and Jeff Erickson. Multiple-source shortest paths in embedded graphs. *SIAM J. Comput.*, 42(4):1542–1571, 2013. doi:10.1137/120864271.

22 Timothy M. Chan and Dimitrios Skrepetos. Faster approximate diameter and distance oracles in planar graphs. *Algorithmica*, 81(8):3075–3098, 2019. doi:10.1007/s00453-019-00570-z.

23 Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Almost optimal distance oracles for planar graphs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 138–151, 2019. doi:10.1145/3313276.3316316.

24 Panagiotis Charalampopoulos, Shay Mozes, and Benjamin Tebeka. Exact distance oracles for planar graphs with failing vertices. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2110–2123, 2019. doi:10.1137/1.9781611975482.127.

25 Shiri Chechik. Approximate distance oracles with improved bounds. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 1–10, 2015. doi:10.1145/2746539.2746562.

26 Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 170–181. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00025.

27 Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, STOC 2000*, pages 469–478, 2000. doi:10.1145/335305.335359.

28 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. *CoRR*, abs/1910.08025, 2019. arXiv:1910.08025.

29 Julia Chuzhoy and Sanjeev Khanna. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 389–400. ACM, 2019. doi:10.1145/3313276.3316320.

30 Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 962–973, 2017. doi:10.1109/FOCS.2017.93.

31 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

32 Krzysztof Diks and Piotr Sankowski. Dynamic plane transitive closure. In *Algorithms - ESA 2007, 15th Annual European Symposium, Proceedings*, pages 594–604, 2007. doi:10.1007/978-3-540-75520-3_53.

33 Hristo Djidjev. On-line algorithms for shortest path problems on planar digraphs. In *Graph-Theoretic Concepts in Computer Science, 22nd International Workshop, WG '96*, pages 151–165, 1996. doi:10.1007/3-540-62559-3_14.

34 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification. I. Planary testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996. doi:10.1006/jcss.1996.0002.

35 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery R. Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992. doi:10.1016/0196-6774(92)90004-V.

36 Jeff Erickson, Kyle Fox, and Luvsandondov Lkhamsuren. Holiest minimum-cost paths and flows in surface graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1319–1332, 2018. doi:10.1145/3188745.3188904.

37 Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981. doi:10.1145/322234.322235.

38 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007.

39 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987. doi:10.1137/0216064.

40 Pawel Gawrychowski and Adam Karczmarz. Improved bounds for shortest paths in dense distance graphs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 61:1–61:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.61.

41 Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 515–529, 2018. doi:10.1137/1.9781611975031.34.

42 Qian-Ping Gu and Gengchun Xu. Constant query time $(1+\epsilon)$ -approximate distance oracle for planar graphs. In *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Proceedings*, volume 9472 of *Lecture Notes in Computer Science*, pages 625–636. Springer, 2015. doi:10.1007/978-3-662-48971-0_53.

43 Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 153–166. ACM, 2020. doi:10.1145/3357713.3384236.

44 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2542–2561. SIAM, 2020. doi:10.1137/1.9781611975994.155.

45 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Deterministic algorithms for decremental approximate shortest paths: Faster and simpler. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2522–2541. SIAM, 2020. doi:10.1137/1.9781611975994.154.

46 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2562–2574. SIAM, 2020. doi:10.1137/1.9781611975994.156.

47 Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert Endre Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Trans. Algorithms*, 8(1):3:1–3:33, 2012. doi:10.1145/2071379.2071382.

48 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 146–155, 2014. doi:10.1109/FOCS.2014.24.

49 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Symposium on Theory of Computing, STOC 2014*, pages 674–683, 2014. doi:10.1145/2591796.2591869.

50 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the $O(mn)$ barrier and derandomization. *SIAM J. Comput.*, 45(3):947–1006, 2016. doi:10.1137/140957299.

51 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

52 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, and Seth Pettie. Fully dynamic connectivity in $O(\log\, n(\log\, \log\, n)^2)$ amortized expected time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 510–520, 2017. doi:10.1137/1.9781611974782.32.

53 Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1108–1121. ACM, 2017. doi:10.1145/3055399.3055480.

54 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 313–322, 2011. doi:10.1145/1993636.1993679.

55 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

56 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in monge matrices and partial monge matrices, and their applications. *ACM Trans. Algorithms*, 13(2):26:1–26:42, 2017. doi:10.1145/3039873.

57 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 1131–1142, 2013. doi:10.1137/1.9781611973105.81.

58 Adam Karczmarz. Decremental transitive closure and shortest paths for planar digraphs and beyond. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 73–92. SIAM, 2018. doi:10.1137/1.9781611975031.5.

59 Adam Karczmarz and Jakub Lacki. Simple label-correcting algorithms for partially dynamic approximate shortest paths in directed graphs. In *3rd Symposium on Simplicity in Algorithms, SOSA@SODA 2020*, pages 106–120. SIAM, 2020. doi:10.1137/1.9781611976014.15.

60 Adam Karczmarz and Jakub Łącki. Reliable hubs for partially-dynamic all-pairs shortest paths in directed graphs. In *27th Annual European Symposium on Algorithms, ESA 2019*, pages 65:1–65:15, 2019. doi:10.4230/LIPIcs.ESA.2019.65.

61 Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup. More compact oracles for approximate distances in undirected planar graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 550–563, 2013. doi:10.1137/1.9781611973105.40.

62 Philip N. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms,SODA 2002*, pages 820–827, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545488.

63 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 146–155, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454.

64 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Symposium on Theory of Computing Conference, STOC'13*, pages 505–514, 2013. doi:10.1145/2488608.2488672.

65 Philip N. Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n\log^2 n)$-time algorithm. *ACM Trans. Algorithms*, 6(2):30:1–30:18, 2010. doi:10.1145/1721837.1721846.

66 Philip N. Klein and Sairam Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998. doi:10.1007/PL00009223.

67 Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 121–130. ACM, 2010. doi:10.1145/1806689.1806708.

68 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, STOC 1984*, pages 376–382, 1984. doi:10.1145/800057.808703.

69 Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 209–222, 2012. doi:10.1137/1.9781611973099.19.

70 Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n\log^2 n/\log\log n)$ time. In *Algorithms - ESA 2010, 18th Annual European Symposium. Proceedings, Part II*, pages 206–217, 2010. doi:10.1007/978-3-642-15781-3_18.

71 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 950–961. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.92.

72 Yahav Nussbaum. *Network flow problems in planar graphs*. PhD thesis, Tel Aviv University, 2014.

73 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. doi:10.1007/s00453-010-9401-5.

74 Piotr Sankowski. Subquadratic algorithm for dynamic shortest distances. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Proceedings*, pages 461–470, 2005. doi:10.1007/11533719_47.

75 Sairam Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *Algorithms - ESA '93, First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993, Proceedings*, pages 372–383, 1993. doi:10.1007/3-540-57273-2_72.

76 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004. doi:10.1145/1039488.1039493.

77 Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Proceedings*, pages 384–396, 2004. doi:10.1007/978-3-540-27810-8_33.

78 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, STOC 2005*, pages 112–119, 2005. doi:10.1145/1060590.1060607.

79 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

80 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 436–455. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00035.

81 Zhengyu Wang. An improved randomized data structure for dynamic graph connectivity. *CoRR*, abs/1510.04590, 2015. arXiv:1510.04590.

82 Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 539–549. SIAM, 2013. doi:10.1137/1.9781611973105.39.

83 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 1757–1769. SIAM, 2013. doi:10.1137/1.9781611973105.126.

84 Christian Wulff-Nilsen. Approximate distance oracles for planar graphs with improved query time-space tradeoff. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 351–362. SIAM, 2016. doi:10.1137/1.9781611974331.ch26.

A Omitted Proofs

▶ **Lemma 16.** *Let $\mathcal{Q}$ be some collection of pieces from $\mathcal{T}(G)$ such that:*
1. *for each leaf piece $L \in \mathcal{T}(G)$, either L or some ancestor of L is in $\mathcal{Q}$,*
2. *for each $H \in \mathcal{Q}$, if some ancestor of H is in $\mathcal{Q}$, then the parent of H is also in $\mathcal{Q}$.*

Then for any $u, v \in V(\mathrm{DDG}(\mathcal{Q}))$, $dist_{\mathrm{DDG}(\mathcal{Q})}(u, v) = dist_G(u, v)$.

Proof. Let us note that if for some $H \in \mathcal{T}(G)$ no ancestor of H belongs to $\mathcal{Q}$, then $\partial H \subseteq V(\mathrm{DDG}(\mathcal{Q}))$. We prove this claim by induction on the level ℓ of piece H. For $\ell = 0$, we get $H \in \mathcal{Q}$, so clearly $\partial H \subseteq V(H) \subseteq V(\mathrm{DDG}(\mathcal{Q}))$. Suppose $\ell \geq 1$. The statement is trivial if $H \in \mathcal{Q}$. Otherwise, consider the children $H_1, \ldots, H_k$ of H. By induction we get $\partial H_i \subseteq V(\mathrm{DDG}(\mathcal{Q}))$. So in fact we have $\partial H \subseteq \bigcup_{i=1}^{k} \partial H_i \subseteq V(\mathrm{DDG}(\mathcal{Q}))$.

Since the edges of each DDG_H encode lengths of some paths in G, we have that $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u, v) \geq \mathrm{dist}_G(u, v)$. We now prove that there is a path of length at most $\mathrm{dist}_G(u, v)$ in $\mathrm{DDG}(\mathcal{Q})$. Let P be some shortest $u \to v$ path in G. Let H be a piece of $\mathcal{T}(G)$ of minimum level that contains P. We prove our claim by induction on the level ℓ of H.

First note that by property 1 of $\mathcal{Q}$, if $H \notin \mathcal{Q}$ and no descendant of H belongs to $\mathcal{Q}$, then H has a nearest ancestor H^* such that $H^* \in \mathcal{Q}$. Observe that $V(\mathrm{DDG}(\mathcal{Q})) \cap V(H) \subseteq \partial H^*$. Hence, $u, v \in \partial H^*$ and thus $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u,v) \leq \mathrm{dist}_{\mathrm{DDG}_{H^*}}(u,v) = \mathrm{dist}_{H^*}(u,v) \leq \mathrm{dist}_H(u,v) = \mathrm{dist}_G(u,v)$.

So we can assume that either $H \in \mathcal{Q}$ or some descendant of H belongs to $\mathcal{Q}$. Then by property 2 of $\mathcal{Q}$, we have that either $H \in \mathcal{Q}$ or no ancestor of H belongs to $\mathcal{Q}$. In either of these cases we have $\partial H \subseteq V(\mathrm{DDG}(\mathcal{Q}))$. Suppose first $\ell = 0$ – then H is a leaf piece and thus $H \in \mathcal{Q}$. So clearly $P \subseteq H \subseteq \mathrm{DDG}(\mathcal{Q})$, i.e., $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u,v) \leq \mathrm{dist}_G(u,v)$. On the other hand, if we assume $\ell \geq 1$ then we can split P into maximal subpaths $P_1, \ldots, P_k$ such that each $P_i = u_i \to v_i$ is entirely contained in a single child of H_P. Then for each i we have $\{u_i, v_i\} \subseteq \{u, v\} \cup \partial H \subseteq V(\mathrm{DDG}(\mathcal{Q}))$ so by induction we get that $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u_i, v_i) \leq \mathrm{dist}_G(u_i, v_i)$ which implies $\mathrm{dist}_{\mathrm{DDG}(\mathcal{Q})}(u,v) \leq \mathrm{dist}_G(u,v)$. ◀

▶ **Lemma 4** ([24, 38]). *Let $\mathcal{L}$ be some collection of leaf pieces of $\mathcal{T}(G)$. Then:*

1. *For any $u, v \in V(\mathrm{DDG}(\mathrm{cone}_G(\mathcal{L})))$, $dist_G(u,v) = dist_{\mathrm{DDG}(\mathrm{cone}_G(\mathcal{L}))}(u,v)$.*
2. $\sum_{H \in \mathrm{cone}_G(\mathcal{L})} |\partial H| = O\left(\sqrt{n|\mathcal{L}|}\right)$.

Proof. To obtain item 1 it is enough to note that $\mathrm{cone}_G(\mathcal{L})$ satisfies the requirements posed on the collection $\mathcal{Q}$ in Lemma 16.

Let $\mathcal{A}$ be the set containing all ancestors of all the leaf pieces $L \in \mathcal{L}$. We show item 2 by bounding the sum $X = \sum_{H \in \mathcal{A}} |\partial H|$. Since the number of boundary vertices of a piece is bounded by the sum of numbers of boundary vertices of its parent and its sibling, the sum $\sum_{H \in \mathrm{cone}_G(\mathcal{L})} |\partial H|$ of our interest can be larger from X only by a constant factor.

Recall that $\mathcal{T}(G)$ admits an r-division for any $r \in [1, n]$, i.e., there exists such r-division $\mathcal{R}$ that $\mathcal{R} \subseteq \mathcal{T}(G)$ and the boundary of each piece of $\mathcal{R}$ equals the boundary of that piece in $\mathcal{T}(G)$. Let us split $\mathcal{A}$ into two parts: let $\mathcal{A}_1$ contain those $H \in \mathcal{A}$ that are descendants of some piece $P \in \mathcal{R}$, and let $\mathcal{A}_2 = \mathcal{A} \setminus \mathcal{A}_1$. Let $X_i = \sum_{H \in \mathcal{A}_i} |\partial H|$.

Since each $L \in \mathcal{L}$ is a descendant of a unique piece $P \in \mathcal{R}$, we now bound the sum of $|\partial H|$ over all ancestors of L that are descendants of P. Recall (Section 2) that if H is a piece in a recursive decomposition of an n-vertex graph, then $|\partial H| = O(\sqrt{n}/c^d)$ where d is the depth of H in that decomposition. Consider the subtree of $\mathcal{T}(G)$ rooted at P – it forms a recursive decomposition $\mathcal{T}(P)$ of P with some initial boundary vertex set $|\partial P|$ of size $O(\sqrt{r})$. So the sum of $|\partial H|$ over all ancestors of L that are descendants of P is actually equal to the sum of $|\partial H|$ over all ancestors H of L in $\mathcal{T}(P)$. Since each ancestor has distinct integral depth, this sum is $O(\sum_{i=0}^{\infty} \sqrt{|P|}/c^i) = O(\sqrt{|P|}) = O(\sqrt{r})$. Hence $X_1 = O(|\mathcal{L}| \cdot \sqrt{r})$. On the other hand, $X_2 \leq \sum_{P \in \mathcal{R}} |\partial P| = O(n/\sqrt{r})$. So we obtain $X = X_1 + X_2 = O(n/\sqrt{r} + |\mathcal{L}| \cdot \sqrt{r})$. By choosing $r = n/|\mathcal{L}|$, we obtain $X = O(\sqrt{n \cdot |\mathcal{L}|})$ as desired. ◀

▶ **Lemma 5** ([24, 38]). *Let $X \subseteq V(G)$ be non-empty. Then:*

1. *For any $u, v \in V(\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(X)))$, $dist_G(u,v) = dist_{\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(X))}(u,v)$.*
2. $\sum_{H \in \mathrm{cone}_{\mathcal{R}}(X)} |\partial H| = O\left(n/\sqrt{r} + \min\left(\sqrt{n \cdot |X|}, |X| \cdot \sqrt{r}\right)\right)$.

Proof sketch. The proof is completely analogous to that of items 2 and 3 of Lemma 4. It is enough to glue the individual decompositions $\mathcal{T}(P)$ into a single decomposition $\mathcal{T}'(G)$ such that the root has $O(n/r)$ children instead of just 2: the individual pieces of $\mathcal{R}$. Then item 1 follows by Lemma 16. Let $Y = \sum_{H \in \mathrm{cone}_{\mathcal{R}}(X)} |\partial H|$. Note that $Y = O(n/\sqrt{r} + |X| \cdot \sqrt{r})$. If we have $r = n/|X| \leq r$, then we can obtain the bound $Y = O(\sqrt{n \cdot |X|})$ in the proof of Lemma 4. Otherwise, $|X| \leq n/r$, so $Y = O(n/\sqrt{r} + |X| \cdot \sqrt{r}) = O(n/\sqrt{r})$. So $Y = O(n/\sqrt{r} + \sqrt{n \cdot |X|})$ in all cases as well. ◀

▶ **Theorem 6.** *Let $G = (V, E)$ be a weighted planar graph. Suppose that adding infinite-weight edges to G does not have effect on any properties of G that we care about. Let $r \in [1, n]$.*

There is a data structure maintaining an r-division with few holes $\mathcal{R}$ of some G^+ such that:

1. *G^+ is obtained from G by adding infinite-weight edges.*
2. *Each P has all its faces except its holes triangular and is accompanied with some auxiliary data structures that can be constructed in $T(r)$ time given P and use $S(r)$ space.*

The data structure uses $O\left(n + \frac{n}{r} \cdot S(r)\right)$ space and can be initialized in $O\left(n + \frac{n}{r} \cdot T(r)\right)$ time. After each edge deletion and edge insertion (preserving the planarity of G), it can be updated in $O(r + T(r))$ worst-case *time.*

Proof. On initialization, we first connect and triangulate G using infinite-weight edges, thus obtaining G^+. Then, we compute an r-division $\mathcal{R}$ with few holes of G^+ in linear time [64] and subsequently initialize the auxiliary data structures. Let $h \geq 2$ and $c \geq 8$ be constants such that a single piece of the computed r-division has at most $c\sqrt{r}$ boundary vertices distributed over h holes.

We will guarantee at all times that for any single piece P, $|\partial P| \leq 3c\sqrt{r}$, and there exist at most $3h$ faces of P such that any $v \in \partial P$ lies on one of these faces, called holes of P. Moreover, each edge of G^+ is contained in at most two pieces of $\mathcal{R}$: this is satisfied initially since the r-division of [64] forms a partition of faces of G^+.

Suppose that the removal of an edge $e = uv$ is issued to G. We then remove e from each of the at most two pieces P containing it. If P is disconnected afterwards, we replace it with two connected pieces P_1, P_2. Otherwise, since removing e merges two faces of P, the total number of holes of P does not increase. If, on the other hand, a new edge $e = uv$ is inserted, we add a new piece P_e consisting of a single edge e to $\mathcal{R}$. Adding P_e may cause an endpoint of e, say u, to become a boundary vertex of $\mathcal{R}$. If u was not a boundary vertex before the insertion, it had to be a vertex of a single piece P_u. At this point, since a new boundary vertex emerges in P_u, we might have $\partial P_u > 3c\sqrt{r}$ or ∂P might no longer lie on at most $3h$ faces of P_u. However, there surely exist some $3h + 1$ faces whose vertices include the whole set ∂P_u, and $|\partial P_u| \leq 3c\sqrt{r} + 1$. To fix our invariants, we first compute a cycle separator C of P_u wrt. the boundary vertices of P_u in $O(r)$ time, and replace P_u with two pieces $P_{u,1}, P_{u,2}$ – the two subgraphs of P_u induced by vertices weakly on one side of C. Clearly, the vertices of C become new boundary vertices afterwards. Subsequently, we similarly break each of $P_{u,1}, P_{u,2}$ further into two parts using a cycle separator wrt. the holes of this piece (see [64] for details). Each of the at most four resulting pieces has at most $\frac{2}{3}(3c\sqrt{r} + 1) + 2\sqrt{2}\sqrt{r} \leq (2c + 2\sqrt{2} + 2)\sqrt{r} \leq 3c\sqrt{r}$ boundary vertices, and at most $\frac{2}{3} \cdot (3h + 1) + 1 \leq 2h + 2 \leq 3h$ holes.

Observe that a single edge update can introduce $O(1)$ new pieces of size $O(r)$ in the maintained r-division, and the sizes of the existing pieces do not increase. For each of the affected pieces we recompute the auxiliary data structures in $O(T(r))$ time. As a result, after $O(n/r)$ updates, there are still $O(n/r)$ pieces, each of size $O(r)$ and with $O(\sqrt{r})$ boundary vertices distributed over $O(1)$ holes of that piece. Consequently, after every $\Omega(n/r)$ updates, we reinitialize $\mathcal{R}$ for the current graph G in $O\left(n + \frac{n}{r} \cdot T(r)\right)$ time. Hence, the amortized time to update $\mathcal{R}$ is $O(r + T(r))$.

Finally, observe that our data structure can be modified in a standard way (see e.g., [5]) to have $O(r + T(r))$ *worst-case* update time bound instead of just an amortized one. This is possible since our update procedure actually takes $O(r + T(r))$ worst-case time apart from once every $k = \Omega(n/t)$ updates when the whole data structure is rebuilt in $O\left(n + \frac{n}{r} \cdot T(r)\right)$

worst-case time. To this end we apply the time-slicing technique. We use two copies of our data structure switching their roles every $k/2$ updates. One copy is for handling at most $k/2$ updates and answering queries, and another is being gradually reinitialized in chunks of $\Omega(r + T(r))$ time (of either initialization or updates replayed) in the background. ◀

B Empty Voronoi Cells

Here, we briefly explain why the algorithm of [23] that underlies Theorem 7 works irrespective of whether there are empty Voronoi cells.

We need a few more definitions. Let f^* be the vertex of the dual graph of G corresponding to the face f, where the sites lie. For an additively weighted Voronoi diagram $\mathsf{VD}(S, \omega)$ over a triangulated graph (possibly apart from face f), we call a face whose incident vertices belong to three different Voronoi cells *trichromatic*. Let $\mathsf{VD}_0^*(S, \omega)$ be the forest obtained by considering the dual edges of G whose endpoints belong to different Voronoi cells. Then, we obtain $\mathsf{VD}_1^*(S, \omega)$ by repeatedly contracting edges of the forest that have an endpoint of degree 2. We obtain $\mathsf{VD}_2^*(S, \omega)$ by creating one copy of f^* for each of its incident edges – inheriting only that edge. The vertices of $\mathsf{VD}_2^*(S, \omega)$ that are not copies of f^* are in one-to-one correspondence with the trichromatic faces of G, other than f. If $\mathsf{VD}_2^*(S, \omega)$ is a tree then we are done. Otherwise, is some cell of $\mathsf{VD}(S, \omega)$ is empty, $\mathsf{VD}_2^*(S, \omega)$ might be a forest. However, as shown in Section 6 of [41], $\mathsf{VD}_2^*(S, \omega)$ can be turned into the sought tree $\mathsf{VD}^*(S, \omega)$ in $O(|S|)$ time.

The algorithm of [23] for computing $\mathsf{VD}^*(S, \omega)$ implicitly assumes that all Voronoi cells are non-empty, and hence that $\mathsf{VD}_2^*(S, \omega)$ is a tree. This algorithm performs FR-Dijkstra computations on G in order to compute the trichromatic faces of G, and a representation of shortest paths from the sites to the vertices incident to these trichromatic faces. Then, $\mathsf{VD}_2^*(S, \omega)$ can be straightforwardly retrieved from this representation. If we run the same algorithm without assuming that there are no non-empty Voronoi cells, the sites with empty Voronoi cells can be easily read from the shortest paths tree obtained from FR-Dijkstra (more specifically, these are the sites whose some ancestor in this tree is also in S). Since the proof of correctness of the algorithm of [23] for computing the trichromatic faces actually only requires the set S to lie on a single face of G (and not necessarily to be equal to $V(f)$), we can re-run it with S pruned from "empty" sites. From all the trichromatic faces and the corresponding shortest paths from sites to their incident vertices, we can retrieve the forest $\mathsf{VD}_2^*(S, \omega)$ in the same way as if it were a tree. Finally, we can then turn this forest into the sought tree $\mathsf{VD}^*(S, \omega)$ in $O(|S|)$ time as in [41].

C Negative Edges in the Fully Dynamic SSSP Algorithm

Recall that $p : V(H) \to \mathbb{R}$ is called a feasible price function of H if $\mathrm{dist}_H(u, v) + p(u) - p(v) \geq 0$ for all $u, v \in V(H)$. A feasible price function is guaranteed to exist if H contains no negative-cost cycle. It is well-known that, provided that a graph H is strongly connected, a vector of distances from any vertex of H constitutes a feasible price function of H.

As in the fully dynamic all-pairs algorithm of [56], we maintain functions $\phi : \partial\mathcal{R} \to \mathbb{R}$ and $\phi_P : \partial P \to \mathbb{R}$, where $P \in \mathcal{R}$, such that ϕ is some feasible price function of G restricted to $\partial\mathcal{R}$, and each ϕ_P is a feasible price function of P.

Since single-source shortest paths in planar graphs with negative weights can be computed in $O(n \log^2 n)$ time [65,70], each ϕ_P can be seen as an accompanying data structure of piece P computable in $O(r \log^2 r)$ time and maintained by the fully dynamic r-division algorithm.

The functions ϕ_P allow to treat individual graphs P and P_h as non-negatively weighted when computing all the needed DDGs and MSSP data structures, and also point location data structures $L(P)$.

It is known [40, 56] that the FR-Dijkstra algorithm (as in Lemma 3) can handle negative weights in $\mathrm{DDG}(\mathcal{H})$ with no asymptotic overhead if a feasible price function on $\mathrm{DDG}(\mathcal{H})$ is provided. Therefore, we would like to have a feasible price function on $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s))$ to compute distances from s in $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s))$ needed by the update algorithm. We can extend ϕ from ∂R to all vertices in $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s))$ as follows. Note that all pieces in $\mathrm{cone}_{\mathcal{R}}(s)$ except of those in $\mathcal{R}$ have their parents also in $\mathrm{cone}_{\mathcal{R}}(s)$. We call those pieces $H \in \mathrm{cone}_{\mathcal{R}}(s)$ for which we know the value of ϕ on all of ∂H *processed*. Initially, only the pieces $P \in \mathcal{R}$ are processed by the definition of ϕ. While there are still unprocessed pieces, we take any unprocessed piece H whose parent $A \in \mathcal{T}(P)$ has already been processed. Let H' be the sibling of H in $\mathcal{T}(P)$. Observe that ϕ_P is a feasible price function of A as well. We extend ϕ to boundary vertices of H, H' by computing shortest paths on $\mathrm{DDG}(\{H, H'\})$ from vertices ∂A, with the initial distance to each $v \in \partial A$ set to $\phi(v)$, and using FR-Dijkstra (Lemma 3) with price function ϕ_P. This way, only the initial distances of ∂A are possibly negative from the point of view of FR-Dijkstra. This does not constitute a problem for either Dijkstra's algorithm or FR-Dijkstra though (see [56]; one can treat the initial distances as weights of edges going out of a super-source; these weights can be all increased by the same large value to be made positive). One can show from the definition of ϕ that the values of ϕ on ∂A will not be altered and the computed distances form a feasible price function on $\partial H \cup \partial H'$ in G. Hence, given that $\partial A \subseteq \partial H \cup \partial H'$, we can process the children H, H' of a processed piece A in $O((|\partial H| + |\partial H'|) \log^2 n)$ time. Summing over all pieces, we obtain by Lemma 5 that extending ϕ to all $V(\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(s)))$ takes $O(n/\sqrt{r} \log^2 n)$ time. This cost is therefore negligible.

Note that an edge deletion or weight increase cannot break the feasibility of ϕ. We might need to recompute ϕ only upon insertion or weight decrease of some edge uv. As shown by Kaplan et al. [56], the new "global" price function ϕ' (or a negative cycle) can be found by computing distances from v to $\partial\mathcal{R} \cup \{u\}$ in $\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(u, v))$ (before applying the insertion) using FR-Dijsktra and the old price function ϕ. This can be done in $O(n/\sqrt{r} \log^2 n)$ time by first extending the old ϕ to $V(\mathrm{DDG}(\mathrm{cone}_{\mathcal{R}}(u, v)))$ as described above and then running the single-source shortest paths algorithm of Lemma 3.

The Number of Repetitions in 2D-Strings

Panagiotis Charalampopoulos
Department of Informatics, King's College London, UK
Institute of Informatics, University of Warsaw, Poland
panagiotis.charalampopoulos@kcl.ac.uk

Jakub Radoszewski
Institute of Informatics, University of Warsaw, Poland
Samsung R&D Poland, Warsaw, Poland
jrad@mimuw.edu.pl

Wojciech Rytter
Institute of Informatics, University of Warsaw, Poland
rytter@mimuw.edu.pl

Tomasz Waleń
Institute of Informatics, University of Warsaw, Poland
walen@mimuw.edu.pl

Wiktor Zuba
Institute of Informatics, University of Warsaw, Poland
w.zuba@mimuw.edu.pl

Abstract

The notions of periodicity and repetitions in strings, and hence these of runs and squares, naturally extend to two-dimensional strings. We consider two types of repetitions in 2D-strings: *2D-runs* and *quartics* (quartics are a 2D-version of squares in standard strings). Amir et al. introduced 2D-runs, showed that there are $\mathcal{O}(n^3)$ of them in an $n \times n$ 2D-string and presented a simple construction giving a lower bound of $\Omega(n^2)$ for their number (*Theoretical Computer Science*, 2020). We make a significant step towards closing the gap between these bounds by showing that the number of 2D-runs in an $n \times n$ 2D-string is $\mathcal{O}(n^2 \log^2 n)$. In particular, our bound implies that the $\mathcal{O}(n^2 \log n + \mathsf{output})$ run-time of the algorithm of Amir et al. for computing 2D-runs is also $\mathcal{O}(n^2 \log^2 n)$. We expect this result to allow for exploiting 2D-runs algorithmically in the area of 2D pattern matching.

A quartic is a 2D-string composed of 2×2 identical blocks (2D-strings) that was introduced by Apostolico and Brimkov (*Theoretical Computer Science*, 2000), where by quartics they meant only *primitively rooted* quartics, i.e. built of a primitive block. Here our notion of quartics is more general and analogous to that of squares in 1D-strings. Apostolico and Brimkov showed that there are $\mathcal{O}(n^2 \log^2 n)$ occurrences of primitively rooted quartics in an $n \times n$ 2D-string and that this bound is attainable. Consequently the number of distinct primitively rooted quartics is $\mathcal{O}(n^2 \log^2 n)$. The straightforward bound for the maximal number of distinct general quartics is $\mathcal{O}(n^4)$. Here, we prove that the number of distinct general quartics is also $\mathcal{O}(n^2 \log^2 n)$. This extends the rich combinatorial study of the number of distinct squares in a 1D-string, that was initiated by Fraenkel and Simpson (*Journal of Combinatorial Theory, Series A*, 1998), to two dimensions.

Finally, we show some algorithmic applications of 2D-runs. Specifically, we present algorithms for computing all occurrences of primitively rooted quartics and counting all general distinct quartics in $\mathcal{O}(n^2 \log^2 n)$ time, which is quasi-linear with respect to the size of the input. The former algorithm is optimal due to the lower bound of Apostolico and Brimkov. The latter can be seen as a continuation of works on enumeration of distinct squares in 1D-strings using runs (Crochemore et al., *Theoretical Computer Science*, 2014). However, the methods used in 2D are different because of different properties of 2D-runs and quartics.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases 2D-run, quartic, run, square

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.32

Funding *Panagiotis Charalampopoulos*: Partially supported by ERC grant TOTAL under the EU's Horizon 2020 Research and Innovation Programme (agreement no. 677651).
Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.
Tomasz Waleń: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.
Wiktor Zuba: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

1 Introduction

Periodicity is one of the main and most elegant notions in stringology. It has been studied extensively both from the combinatorial and the algorithmic perspective; see e.g. the books [18, 25, 39]. A classic combinatorial result is the periodicity lemma due to Fine and Wilf [27]. From the algorithmic side, periodicity often poses challenges in pattern matching, due to the following fact: a pattern P can have many occurrences in a text T that are "close" to each other if and only if P has a "small" period. On the other hand, the periodic structure indeed allows us to overcome such challenges; see [18, 25].

Runs, also known as maximal repetitions, are a fundamental notion in stringology. A run is a periodic fragment of the text that cannot be extended without changing the period. Runs were introduced in [35]. Kolpakov and Kucherov presented an algorithm to compute all runs in a string in time linear with respect to the length of the string over a linearly-sortable alphabet [38]. Runs fully capture the periodicity of the underlying string and, since the publication of the algorithm for their linear-time computation, they have assumed a central role in algorithm design for strings. They have been exploited for text indexing [36], answering internal pattern matching queries in texts [16, 37], or reporting repetitions in a string [2, 15, 22], to name a few applications.

Kolpakov and Kucherov also posed the so-called runs conjecture which states that there are at most n runs in a string of length n. A long line of work on the upper [19, 20, 21, 31, 42, 43, 44] and lower bounds [30, 41, 45] was concluded by Bannai et al. who positively resolved the runs conjecture in [10] (see also an alternative proof in [23] and a tighter upper bound for binary strings from [28]).

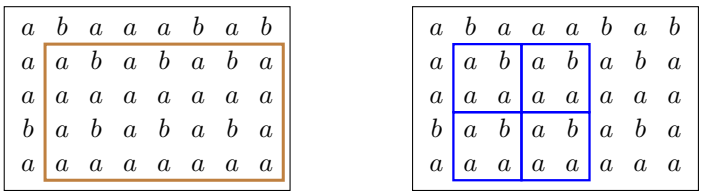
A square is a concatenation of two copies of the same string. Fraenkel and Simpson [29] showed that a string of length n contains at most $2n$ distinct square factors. This bound was improved in [26, 34]. All distinct squares in a string of length n can be computed in $\mathcal{O}(n)$ time assuming an integer alphabet [11, 22, 33] (see [46] for an earlier $\mathcal{O}(n \log n)$ algorithm).

Pattern matching and combinatorics on 2D strings have been studied for more than 40 years, see e.g. [1, 4, 9, 14, 18, 25]. In this paper we consider 2-dimensional versions of runs, introduced by Amir et al. [5, 6], and of repetitions in 2D-strings, introduced by Apostolico and Brimkov [7]. As discussed in [6, 8], one could potentially exploit such repetitions in a 2D-string, which could for instance be an image, in order to compress it.

A *2D-run* in a 2D-string A is a subarray of A that is both horizontally periodic and vertically periodic and that cannot be extended by a row or column without changing the horizontal or vertical periodicity (a formal definition follows in Section 2); see Figure 1(a). Amir et al. [5, 6] have shown that the maximum number of 2D-runs in an $n \times n$ array is $\mathcal{O}(n^3)$ and presented an example with $\Theta(n^2)$ 2D-runs. In [6] they presented an $\mathcal{O}(n^2 \log n + \mathsf{output})$-time algorithm for computing 2D-runs.

A *quartic* is a configuration that is composed of 2×2 occurrences of an array W (see Figure 1(b)) and a *tandem* is a configuration consisting of two occurrences of an array W that share one side (Apostolico and Brimkov [7] also considered another type of tandems, which

share one corner; see also [3]). An array W is called *primitive* if it cannot be partitioned into non-overlapping replicas of some array W'. Apostolico and Brimkov [7] considered only quartics and tandems with primitive W (we call them *primitively rooted*) and showed tight asymptotic bounds $\Theta(n^2 \log^2 n)$ and $\Theta(n^3 \log n)$ for the maximum number of occurrences of such quartics and tandems in an $n \times n$ array, respectively. In [8] they presented an optimal $\mathcal{O}(n^3 \log n)$-time algorithm for computing all *occurrences* of tandems with primitive W. This extends a result that a 1D-string of length n contains $\mathcal{O}(n \log n)$ occurrences of primitively rooted squares and they can all be computed in $\mathcal{O}(n \log n)$ time; see [17, 46]. In this paper we consider the numbers of *all distinct* quartics, which is a more complicated problem.



(a) a 2D-run **(b)** a quartic

Figure 1 Examples of a 2D-run and a quartic.

When computing 2D-runs we consider positioned runs: two 2D-runs with same content but starting in different points are considered distinct. However in case of quartics, similarly as in case of 1D-squares, we consider unpositioned quartics; if two quartics have the same content but start in different positions, we consider them equal.

Our Results.

- We show that the number of 2D-runs in an $n \times n$ array is $\mathcal{O}(n^2 \log^2 n)$. This improves upon the $\mathcal{O}(n^3)$ upper bound of Amir et al. [5, 6] and proves that their algorithm computes all 2D-runs in an $n \times n$ 2D-string in $\mathcal{O}(n^2 \log^2 n)$ time **(Section 3)**.
- We show that the number of distinct quartics in an $n \times n$ array is $\mathcal{O}(n^2 \log^2 n)$. This can be viewed as an extension of the bounds on the maximum number of distinct square factors in a 1D-string [26, 29] **(Section 4)**.
- We present algorithmic implications of the new upper bound for 2D-runs. We show that all occurrences of primitively rooted quartics can be computed in quasi-linear, $\mathcal{O}(n^2 \log^2 n)$ time, which is optimal by the bound of Apostolico and Brimkov [7]. Thus our algorithm complements the result of Apostolico and Brimkov [8] who gave an optimal algorithm for computing all occurrences of primitively rooted tandems. We also show that all distinct quartics can be computed in quasi-linear, $\mathcal{O}(n^2 \log^2 n)$ time, which extends efficient computation of distinct squares in 1D-strings [11, 22, 33] to 2D **(Section 5)**.
- As an easy side result, we show tight $\Theta(n^3)$ bounds for the maximum number of distinct tandems in an $n \times n$ array and how to report them in $\mathcal{O}(n^3)$ time **(Section 2)**.

2 Preliminaries

1D-Strings. We denote by $[a, b]$ the set $\{i \in \mathbb{Z} : a \leq i \leq b\}$. Let $S = S[1]S[2] \cdots S[|S|]$ be a *string* of length $|S|$ over an alphabet Σ. The elements of Σ are called *letters*. For two positions i and j on S, we denote by $S[i\,..\,j] = S[i] \cdots S[j]$ the *fragment* of S that starts at position i and ends at position j (it equals ε if $j < i$). A positive integer p is called a *period* of S if $S[i] = S[i + p]$ for all $i = 1, \ldots, |S| - p$. We refer to the smallest period as *the period* of the string, and denote it by $\mathsf{per}(S)$.

▶ **Lemma 1** (Periodicity Lemma (weak version), Fine and Wilf [27]). *If p and q are periods of a string S and satisfy $p+q \leq |S|$, then $\gcd(p,q)$ is also a period of S.*

A string S is called *periodic* if $\mathsf{per}(S) \leq |S|/2$. By ST and S^k we denote the concatenation of strings S and T and k copies of the string S, respectively. A string S is called *primitive* if it cannot be expressed as U^k for a string U and an integer $k > 1$.

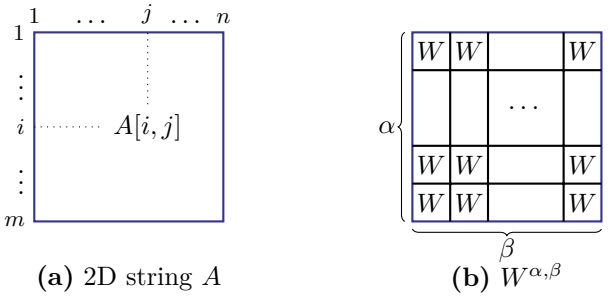
A string of the form U^2 for string U is called a *square*. A square U^2 is called *primitively rooted* if U is primitive. We will make use of the following important property of squares.

▶ **Lemma 2** (Three Squares Lemma, [24]). *Let U, V and W be three strings such that U^2 is a proper prefix of V^2, V^2 is a proper prefix of W^2 and U is primitive. Then $|U|+|V| \leq |W|$.*

A *run* (also known as *maximal repetition*) in S is a periodic fragment $R = S[i\,..\,j]$ which cannot be extended either to the left or to the right without increasing the period $p = \mathsf{per}(R)$, i.e. if $i > 1$ then $S[i-1] \neq S[i+p-1]$ and if $j < |S|$ then $S[j+1] \neq S[j-p+1]$. Let $\mathcal{R}(S)$ denote the set of all runs of string S. For periodic fragment $U = S[a\,..\,b]$, the run that extends U is the unique run $R = S[i\,..\,j]$ such that $i \leq a \leq b \leq j$ and $\mathsf{per}(R) = \mathsf{per}(U)$. An occurrence of a square U^2 is said to be *induced* by a run R if R extends U^2. Every square is induced by exactly one run [22].

2D-Strings. Let A be an $m \times n$ array (2D-string). We denote the height and width of A by $\mathsf{height}(A) = m$ and $\mathsf{width}(A) = n$, respectively. By $A[i,j]$ we denote the cell in the ith row and jth column of A; see Figure 2(a). By $A[i_1\,..\,i_2, j_1\,..\,j_2]$ we denote the subarray formed of rows $i_1,\ldots,i_2$ and columns $j_1,\ldots,j_2$.

A positive integer p is a *horizontal period* of A if the i-th column of A equals the $(i+p)$-th column of A for all $i = 1,\ldots,n-p$. We denote the smallest horizontal period of A by $\mathsf{hper}(A)$. Similarly, a positive integer q is a *vertical period* of A if the i-th row of A equals the $(i+q)$-th row of A for all $i = 1,\ldots,m-q$; the smallest vertical period of A is denoted by $\mathsf{vper}(A)$.



(a) 2D string A **(b)** $W^{\alpha,\beta}$

Figure 2 A 2D-string and the structure of $W^{\alpha,\beta}$.

An $r \times c$ subarray $B = A[i_1\,..\,i_2, j_1\,..\,j_2]$ of A is a *2D-run* if $\mathsf{hper}(B) \leq c/2$, $\mathsf{vper}(B) \leq r/2$ and extending B by a row or column, i.e. either of $A[i_1-1, j_1\,..\,j_2]$, $A[i_2+1, j_1\,..\,j_2]$, $A[i_1\,..\,i_2, j_1-1]$, or $A[i_1\,..\,i_2, j_2+1]$, would result in a change of the smallest vertical or the horizontal period.

If W is a 2D array, then by $W^{\alpha,\beta}$ we denote an array that is composed of $\alpha \times \beta$ copies of W; see Figure 2(b). A *tandem* of W is an array of the form $W^{1,2}$ and a *quartic* of W is the array $W^{2,2}$. A 2D array A is called *primitive* if $A = B^{\alpha,\beta}$ for positive integers α, β implies that $\alpha = \beta = 1$. The *primitive root* of an array A is the unique primitive array B for which $A = B^{\alpha,\beta}$ for $\alpha, \beta \geq 1$.

Apostolico and Brimkov [7] proved the following upper bound, and showed that it is tight by giving a corresponding lower bound.

▶ **Fact 3** (Lemma 5 in [7]). *A 2D array of size $n \times n$ has $\mathcal{O}(n^2 \log^2 n)$ occurrences of primitively rooted quartics.*

We say that a quartic $Q = W^{2,2}$ is *induced* by a 2D-run R if Q is a subarray of R and $\mathsf{hper}(R)$ and $\mathsf{vper}(R)$ divide the width and height of W, respectively.

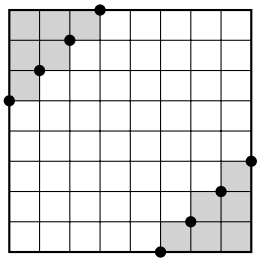


Figure 3 Shaded positions contain letters b, all the other the letters a. Each rectangle with top-left and bottom-right corners marked is a 2D-run; altogether there are 18 distinct 2D-runs, including two of the form $b^{2,2}$. There are also 10 distinct quartics $a^{\alpha,\beta}$, where $0 < \alpha, \beta \leq 8$ are even and $\alpha + \beta \leq 10$. There is also the quartic $b^{2,2}$ (altogether 11 distinct quartics). The centrally placed quartic $a^{2,2}$ is contained in 16 2D-runs. There are only two distinct primitively rooted quartics.

▶ **Observation 4.** *Every quartic is induced by a 2D-run. However; the same quartic can be induced even by $\Theta(n^2)$ 2D-runs; say the middle quartic $a^{2,2}$ in Figure 3.*

▶ Remark 5. The fact that a string of length n has $\mathcal{O}(n \log n)$ occurrences of primitively rooted squares immediately shows (by the fact that a square is induced by exactly one run) that it has $\mathcal{O}(n \log n)$ runs. However, an analogous argument applied for quartics and 2D-runs does not give a non-trivial upper bound for the number of the latter because of Observation 4.

In our algorithms, we use a variant of the Dictionary of Basic Factors in 2D (2D-DBF in short) that is similar to the one presented in [25]. Namely, to each subarray of A whose width and height is an integer power of 2 we assign an integer identifier from $[0, n^2]$ so that two arrays with the same dimensions are equal if and only if their identifiers are equal. The total number of such subarrays is $\mathcal{O}(n^2 \log^2 n)$ and the identifiers can be assigned in $\mathcal{O}(n^2 \log^2 n)$ time; see [25]. Using 2D-DBF, we can assign an identifier to a subarray of A of arbitrary dimensions $r \times c$ being a quadruple of 2D-DBF identifiers of its four $2^i \times 2^j$ subarrays that share one of its corners, where $2^i \leq r < 2^{i+1}$ and $2^j \leq c < 2^{j+1}$. Such quadruples preserve the property that two subarrays of the same dimensions are equal if and only if the 2D-DBF quadruples are the same.

As an illustration, we show a tight bound for the number of distinct tandems and an optimal algorithm for computing them.

▶ **Theorem 6.** *The maximum number of distinct tandems in an $n \times n$ array A is $\Theta(n^3)$. All distinct tandems in an $n \times n$ array can be reported in the optimal $\Theta(n^3)$ time.*

Proof. Let us fix two row numbers $i < i'$ in A. Then, the number of distinct tandems with top row i and bottom row i' is $\mathcal{O}(n)$ by the fact that a string of length n contains $\mathcal{O}(n)$ squares [26, 29]. Thus, in total there are $\mathcal{O}(n^3)$ distinct tandems. For the lower bound, let

the ith row of A be filled with occurrences of the letter i. Every subarray of A of even width is a tandem. For each distinct triplet of top and bottom rows and even width, we obtain a distinct tandem.

Let us proceed to the algorithm. For a height $h \in [1, n]$, we assign integer identifiers from $[1, n^2]$ that preserve lexicographical comparison to all height-h substrings of columns of A. They can be assigned using the generalized suffix tree [18, 47] of the columns of A in $\mathcal{O}(n^2 \log n)$ time. Let B_h be an array such that $B_h[i, j]$ stores the identifier of $A[i\,..\,i+h-1, j]$. To a subarray $W = A[i\,..\,i+h-1, j\,..\,j+w-1]$ we assign an *identifier* $\mathsf{id}(W) = B_h[i, j\,..\,j+w-1]$. Then for any two subarrays W and W' of height h, $W = W'$ if and only if $\mathsf{id}(W) = \mathsf{id}(W')$. For every height $h = 1, \ldots, n$ and row i, we find all distinct squares in $B_h[i, 1], \ldots, B_h[i, n]$ in $\mathcal{O}(n)$ time [11, 22, 33]. This corresponds to the set of distinct tandems with top row i and bottom row $i + h - 1$. Finally, we assign identifiers from 2D-DBF of A to each of the tandems and use radix sort to sort them and enumerate distinct tandems. ◀

3 Improved Upper Bound for 2D-Runs

We introduce the framework that Amir et al. used for efficiently computing 2D-runs [5, 6].

We say that a subarray $B = A[i_1\,..\,i_2, j_1\,..\,j_2]$ of A is a *horizontal run* if it is horizontally periodic (that is, $\mathsf{hper}(B) \le \mathsf{width}(B)/2$) and extending B by either of the columns $A[i_1\,..\,i_2, j_1 - 1]$ or $A[i_1\,..\,i_2, j_2 + 1]$ would result in a change of the smallest horizontal period. (Note that B does not have to be vertically periodic.)

For $k \in [1, \lfloor \log n \rfloor]$ and $i \in [1, n - 2^k + 1]$, let H_i^k be the string obtained by replacing the columns of array $A[i\,..\,i + 2^k - 1, 1\,..\,n]$ with metasymbols such that $H_i^k[j] = H_i^k[j']$ if and only if $A[i\,..\,i + 2^k - 1, j] = A[i\,..\,i + 2^k - 1, j']$. Notice that each such horizontal run of height 2^k corresponds to a run in some H_i^k.

The following lemma will enable us to "anchor" each 2D-run R in the top-left or bottom-left corner of a horizontal run of "similar" height as R. It was proved in [6], but we provide a proof for completeness.

▶ **Lemma 7** (Lemma 7 in [6]). *Let R be a 2D-run whose height is in the range $[2^k, 2^{k+1})$. Then there is a horizontal run R' of height 2^k with $\mathsf{hper}(R') = \mathsf{hper}(R)$ and $\mathsf{width}(R') \ge \mathsf{width}(R)$ such that top-left or bottom-left corners of R and R' coincide (see Figure 4).*

Proof. Let $R = A[i_1\,..\,i_2, j_1\,..\,j_2]$ be the 2D-run in scope and let $k = \lfloor \log(i_2 - i_1 + 1) \rfloor$. We have to show that at least one of the two following statements holds.

- There is a run $R_1 = S[j_1\,..\,b]$ in $S = H_{i_1}^k$ with smallest period p and $b \ge j_2$.
- There is a run $R_2 = T[j_1\,..\,d]$ in $T = H_{i_2 - 2^k + 1}^k$ with smallest period p and $d \ge j_2$.

Since $\mathsf{vper}(R) \le \mathsf{height}(R)/2$, all distinct rows of R are represented in each of $U = S[j_1\,..\,j_2]$ and $V = T[j_1\,..\,j_2]$ and hence $p = \mathsf{per}(U) = \mathsf{per}(V)$. Let $R_1 = S[a\,..\,b]$ be the run that extends U and $R_2 = T[c\,..\,d]$ be the run that extends V. Let us suppose towards a contradiction that $\max(a, c) < j_1$. Then, $A[i_1\,..\,i_2, j_1 - 1] = A[i_1\,..\,i_2, j_1 - 1 + p]$, which contradicts R being a run, since R and $B = A[i_1\,..\,i_2, j_1 - 1\,..\,j_2]$ have the same horizontal and vertical periods. ◀

The sum of the lengths of the runs in a string of length n can be $\Omega(n^2)$ as shown in [32]. However, we prove the following lemma, which is crucial for our approach. We will use it to obtain an overall bound on the possible widths of 2D-runs for our anchors.

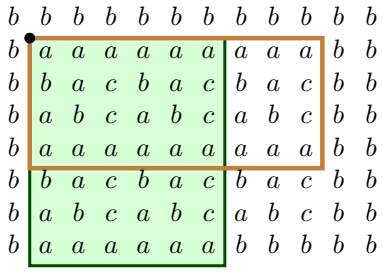


Figure 4 The shaded 7×6 subarray is a 2D-run R, with vertical period 3 and horizontal period $p = 3$. The other marked 4×9 rectangle encloses a horizontal run R' with the same top-left corner and the same horizontal period as R. We have $2 \cdot p \leq width(R) \leq width(R')$.

▶ **Lemma 8.** *For any string S of length n we have that*

$$\rho(S) := \sum_{R \in \mathcal{R}(S)} (|R| - 2 \cdot per(R) + 1) = \mathcal{O}(n \log n).$$

Proof. We consider for each run $R = S[i..j]$ of S the interval $I_R = [i, j - 2 \cdot \mathsf{per}(R) + 1]$. Note that $\rho(S) = \sum_{R \in \mathcal{R}(S)} |I_R|$.

Observe that for every $a \in I_R$ the string $S[a..a + \mathsf{per}(R) - 1]$ is primitive, since if it was of the form U^k for a string U and an integer $k > 1$, then $|U| < \mathsf{per}(R)$ would be a period of R, a contradiction. Hence, at each position $a \in I_R$ there is an occurrence of a primitively rooted square of length $2 \cdot \mathsf{per}(R)$.

A direct application of the Three Squares Lemma (Lemma 2) implies that at most $\mathcal{O}(\log n)$ primitively rooted squares can start at each position a. Each such square extends to a unique run. Thus, each position i belongs to $\mathcal{O}(\log n)$ intervals I_R for $R \in \mathcal{R}(S)$. This completes the proof. ◀

We are now ready to prove the main result of this section.

▶ **Theorem 9.** *There are $\mathcal{O}(n^2 \log^2 n)$ 2D-runs in an $n \times n$ array A.*

Proof. We will iterate over all horizontal runs $R' = A[i..i', j..j']$ whose height is a power of 2, i.e. $i' = i + 2^k - 1$ for some k. For each such horizontal run R', we consider the 2D-runs R with:

(a) top-left corner $A[i, j]$ or bottom-left corner $A[i', j]$,
(b) $\mathsf{hper}(R) = \mathsf{hper}(R')$, and
(c) $\mathsf{height}(R) \in [2^k, 2^{k+1})$.

For each such 2D-run R, we have $\mathsf{width}(R) \in [2 \cdot \mathsf{hper}(R'), \mathsf{width}(R')]$, else the horizontal period would break, i.e. property (b) would be violated. Let us notice that R' corresponds to a run $U = H_i^k[j..j'] \in \mathcal{R}(H_i^k)$. In particular, $\mathsf{width}(R) \in [2 \cdot \mathsf{per}(U), |U|]$.

Lemma 7 implies that each 2D-run is accounted for at least once in this manner. It is thus enough to bound the number of considered runs. We have n choices for i and $\log n$ choices for k. Further, due to Lemma 8, for each corresponding meta-string H_i^k we have $\mathcal{O}(n \log n)$ choices for a pair (j, c) such that $U = H_i^k[j..j'] \in \mathcal{R}(H_i^k)$ and $c \in [2 \cdot \mathsf{per}(U), |U|]$. In total, we thus have $\mathcal{O}(n^2 \log^2 n)$ choices for (i, k, j, c). We will complete the proof by showing that there is only a constant number of 2D-runs with top-left corner $A[i, j]$, width w and whose height is in the range $[2^k, 2^{k+1})$. (2D-runs with bottom-left corner $A[i', j]$ can be bounded symmetrically.)

▷ Claim 10 (cf. Lemma 10 in [6]). Let B be an $r \times c$ array with $r \in [2^k, 2^{k+1})$. Then, there are at most two integers $p > 2^{k-1}$ such that $p = \mathsf{vper}(B') \leq \mathsf{height}(B')/2$ for B' consisting of the top $\mathsf{height}(B') \geq 2^k$ rows of B.

Proof. Consider S to be the meta-string obtained by replacing the rows of B by single letters. Then, a direct application of the Three Squares Lemma (Lemma 2) to S yields the claimed bound. ◁

We apply Claim 10 to $B = A[i\,..\,\min(i + 2^{k+1} - 2, n), j\,..\,j + c - 1]$. If $\mathsf{vper}(R) \leq 2^{k-1}$, then $\mathsf{vper}(R) = \mathsf{vper}(R')$ by the Periodicity Lemma (Lemma 1) applied to the meta-string obtained by replacing the rows of the intersection of R' and B by single letters. Now Claim 10 implies that there are at most three choices to make for the vertical period: $\mathsf{vper}(R')$ and the two integers from the claim. Finally, for fixed top-left corner, width and vertical period we can have a single 2D-run. This concludes the proof. ◀

Amir et al. [6] presented the following algorithmic result.

▶ **Theorem 11** ([6]). *All 2D-runs in an $n \times n$ array can be computed in $\mathcal{O}(n^2 \log n + \mathsf{output})$ time, where output is the number of 2D-runs reported.*

By combining Theorems 9 and 11 we get the following corollary.

▶ **Corollary 12.** *All 2D-runs in an $n \times n$ array can be computed in $\mathcal{O}(n^2 \log^2 n)$ time.*

4 Upper Bound on the Number of Distinct Quartics

Fact 3 that originates from [7] shows that an $n \times n$ array A has $\mathcal{O}(n^2 \log^2 n)$ occurrences of primitively rooted quartics. This obviously implies that the number of distinct primitively rooted quartics is upper bounded by $\mathcal{O}(n^2 \log^2 n)$. Unfortunately, an array can contain $\Theta(n^4)$ occurrences of general quartics; this takes place e.g. for a unary array. In this section we show that $\mathcal{O}(n^2 \log^2 n)$ is also an upper bound for the number of *distinct* general quartics, i.e. subarrays of A of the form $W^{\alpha,\beta}$ for even $\alpha, \beta \geq 2$ and primitive W.

The following lemma and its corollary are the combinatorial foundation of our proofs. An array W with $\mathsf{height}(W) \in [2^a, 2^{a+1})$ and $\mathsf{width}(W) \in [2^b, 2^{b+1})$ will be called an (a, b)-*array*.

▶ **Lemma 13.** *Let a, b be non-negative integers and W, W' be different primitive (a, b)-arrays. If occurrences of $W^{2,3}$ and $(W')^{2,3}$ (of $W^{3,2}$ and $(W')^{3,2}$, respectively) in A share the same corner (i.e., top-left, top-right, bottom-left or bottom-right), then $\mathsf{width}(W) = \mathsf{width}(W')$ ($\mathsf{height}(W) = \mathsf{height}(W')$, respectively).*

Proof. Clearly it is sufficient to prove the lemma for $W^{2,3}$ and $(W')^{2,3}$. Assume w.l.o.g. that occurrences of $W^{2,3}$ and $(W')^{2,3}$ in A share the top-left corner and consider their overlap X.

Each of the rows of X has periods $\mathsf{width}(W)$ and $\mathsf{width}(W')$. Assume w.l.o.g. that $\mathsf{width}(W) \leq \mathsf{width}(W')$. Then

$$\mathsf{width}(X) = 3 \cdot \mathsf{width}(W) \geq \mathsf{width}(W) + 2^{a+1} \geq \mathsf{width}(W) + \mathsf{width}(W').$$

By the Periodicity Lemma (Lemma 1), $p = \gcd(\mathsf{width}(W), \mathsf{width}(W'))$ is a horizontal period of X.

The array X contains at least one occurrence of W and W' in its top-left corner. Hence, W and W' have a horizontal period p. If $\mathsf{width}(W) < \mathsf{width}(W')$, then $\mathsf{width}(W')$ cannot be a multiple of $\mathsf{width}(W)$, because then we would have $\mathsf{width}(W') > 2^{a+1}$. Hence, if $\mathsf{width}(W) < \mathsf{width}(W')$, we would have $p < \mathsf{width}(W)$ which by $p \mid \mathsf{width}(W)$ would mean that W is not primitive. This indeed shows that $\mathsf{width}(W) = \mathsf{width}(W')$. ◀

▶ **Corollary 14.** *Let a, b be non-negative integers and W, W' be different (a, b)-arrays. If occurrences of $W^{3,3}$ and $(W')^{3,3}$ in A share the same corner (i.e., top-left, top-right, bottom-left or bottom-right), then at least one of W, W' is not primitive.*

If $V^{2,2}$ is a non-primitively rooted quartic, then there exists a primitive array W such that $V = W^{\alpha,\beta}$ and at least one of α, β is greater than one. We will call the quartic $W^{2\alpha,2\beta}$ *thin* if $\alpha = 1$ or $\beta = 1$ for this decomposition, and *thick* otherwise. We refer to *points* in A as the $(n+1)^2$ positions where row and column delimiters intersect. Let us first bound the number of distinct thin quartics. For $\beta > 1$, we consider any rightmost occurrence of every such quartic, that is, any occurrence $A[i_1 \,..\, i_2, j_1 \,..\, j_2]$ that maximizes j_1.

▶ **Lemma 15.** *The total number of distinct thin quartics in A is $\mathcal{O}(n^2 \log^2 n)$.*

Proof. We give a proof for quartics of the form $W^{2,2\beta}$ for primitive W and $\beta > 1$; the proof for quartics of the form $W^{2\alpha,2}$ for $\alpha > 1$ is symmetric. We consider each pair of positive integers a, b and show that each point holds the top-left corner of at most two rightmost occurrences of $W^{2,2\beta}$ for primitive (a, b)-arrays W and $\beta > 1$.

Assume to the contrary that the rightmost occurrences of $W^{2,2\beta}$, $(W')^{2,2\beta'}$ and $(W'')^{2,2\beta''}$ share their top-left corner for primitive (a, b)-arrays W, W', W''. The arrays W, W', W'' are pairwise different, since otherwise one of the occurrences would not be the rightmost. By Lemma 13, we have $\mathsf{width}(W) = \mathsf{width}(W') = \mathsf{width}(W'')$. Assume w.l.o.g. that $\mathsf{height}(W) < \mathsf{height}(W') < \mathsf{height}(W'')$.

Let (i, j) denote the top-left corner of the three quartics. Let us consider three length-2ℓ strings formed of metacharacters that correspond to row fragments:

$$(A[i, j \,..\, j + w - 1]), \ldots, (A[i + 2\ell - 1, j \,..\, j + w - 1])$$

for $w = \mathsf{width}(W)$ and $\ell \in \{\mathsf{height}(W), \mathsf{height}(W'), \mathsf{height}(W'')\}$. All the three strings need to be primitively rooted squares. We apply the Three Squares Lemma (Lemma 2) to conclude that $\mathsf{height}(W'') > \mathsf{height}(W) + \mathsf{height}(W') > 2^{a+1}$, a contradiction. ◀

Now let us proceed to thick quartics. Unfortunately, in this case a single point can be the top-left corner of a linear number of rightmost occurrences of thick quartics; see the example in Figure 3. Let us consider an occurrence of $W^{\alpha,\beta}$ for even $\alpha, \beta > 2$ and primitive W, called a *positioned quartic*. It implies $\alpha \cdot \beta$ occurrences of W. Let us call all corners of all these occurrences of W *special points* of this positioned quartic. Each special point stores a direction in $\{\text{top-left}, \text{top-right}, \text{bottom-left}, \text{bottom-right}\}$. A special point has one of the directions if it is the respective corner of an occurrence of $W^{3,3}$ in this positioned quartic. Clearly, since $\alpha, \beta \geq 4$, for every special point in $W^{\alpha,\beta}$ except for the middle row if $\alpha = 4$ or middle column if $\beta = 4$, one can assign such a direction (if many directions are possible, we choose an arbitrary one); see Figure 5.

The quartics with primitive root W are called *W-quartics*. The set of all special points (with directions) of *all* positioned thick W-quartics for a given W is denoted by *SpecialPoints*(W). Among W-quartics of the same height we distinguish the ones with maximal width, which we call *h-maximal* (horizontally maximal). Let us observe that each W-quartic is contained in an occurrence of some h-maximal W-quartic.

▶ **Theorem 16.** *The number of distinct quartics in an $n \times n$ array is $\mathcal{O}(n^2 \log^2 n)$.*

Proof. By Fact 3 and Lemma 15 it suffices to show that the total number of distinct thick quartics in A is $\mathcal{O}(n^2 \log^2 n)$. Let us fix non-negative integers a, b. It is enough to show that the number of distinct subarrays of A of the form $W^{\alpha,\beta}$ for even $\alpha, \beta > 2$ and any primitive (a, b)-array W is $\mathcal{O}(n^2)$.

The sets of special points have the following properties. Claim 17 follows from Corollary 14.

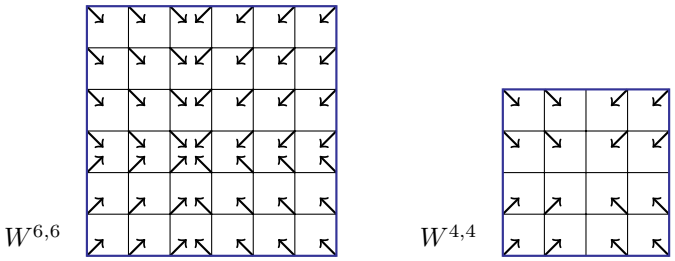


Figure 5 Special points of a positioned quartic with primitive root W with associated directions of four types. The arrow indicates the corner (four possibilities) of $W^{3,3}$ which is contained in the quartic. If several assignments of directions are possible, only one of them is chosen (it does not matter which one). In case of $W^{4,4}$ the middle row and column are not special.

▷ Claim 17. For primitive (a, b)-arrays $W \neq W'$, $\mathit{SpecialPoints}(W) \cap \mathit{SpecialPoints}(W') = \emptyset$.

For an array W, let us denote by $\mathit{ThickQuartics}(W)$ the total number of thick quartics in A with primitive root W.

▷ Claim 18. For a primitive (a, b)-array W, $\mathit{ThickQuartics}(W) < |\mathit{SpecialPoints}(W)|$.

Proof. For each $\alpha = 4, 6, \ldots$ in this order, we select one positioned h-maximal W-quartic U_α of height $\alpha \cdot \mathsf{height}(W)$. The number of distinct W-quartics in A of height $\alpha \cdot \mathsf{height}(W)$ is at most the number of special points in U_α in any of its rows. Note that this statement also holds if $U_\alpha = W^{\alpha,4}$; then there are still four special points in each (non-middle if $\alpha = 4$) row.

We describe a process of assigning distinct W-quartics to distinct special points in $\mathit{SpecialPoints}(W)$. Assume all points in this set are initially not marked. We choose any single row from U_α with all special points in this row still not marked. Then we mark all these special points. We can always choose a suitable row because the heights are increasing.

This way each W-quartic is assigned to only one special point from $\mathit{SpecialPoints}(W)$. ◁

By the claims, the total number of thick W-quartics for primitive (a, b)-arrays W is bounded by:

$$\sum_W \mathit{ThickQuartics}(W) < \sum_W |\mathit{SpecialPoints}(W)| \leq 4(n+1)^2,$$

where the sum is over all primitive (a, b)-arrays W. The conclusion follows. ◀

5 Algorithms for Computing Quartics

In this section we show algorithmic applications of 2D-runs related to quartics.

▶ **Theorem 19.** *All occurrences of primitively rooted quartics in an $n \times n$ array A can be computed in the optimal $\mathcal{O}(n^2 \log^2 n)$ time.*

Proof. Let us consider a 2D-run $R = A[i_1 \,..\, i_2, j_1 \,..\, j_2]$ with periods $\mathsf{hper}(R) = p$ and $\mathsf{vper}(R) = q$. It induces primitively rooted quartics of width $2p$ and height $2q$. The set of top-left corners of these quartics forms a rectangle $\hat{R} = [i_1, i_2 - 2p + 1] \times [j_1, j_2 - 2q + 1]$. We denote by $\mathcal{F}_{p,q}$ the family of such rectangles $\hat{R}$ over 2D-runs R with the same periods p, q.

Such rectangles for different 2D-runs may overlap, even when the dimensions of the quartic are fixed (see Observation 4). In order not to report the same occurrence multiple times, we need to compute, for every dimensions of a quartic, all points in the union of

the corresponding rectangles. This could be done with an additional $\log n$-factor in the complexity using a standard line sweep algorithm [12]. However, we can achieve $\mathcal{O}(n^2 \log^2 n)$ total time using the fact that the total number of occurrences reported is $\mathcal{O}(n^2 \log^2 n)$.

▷ Claim 20. Let $\mathcal{F}_1, \ldots, \mathcal{F}_k$ be families of 2D rectangles in $[1, n]^2$ and let $r = \sum_{i=1}^{k} |\mathcal{F}_i|$. We can compute k (not necessarily disjoint) sets of grid points $Out_i = \bigcup \mathcal{F}_i$ in $\mathcal{O}(n + r + \mathsf{output})$ total time, where $\mathsf{output} = \sum_i |Out_i|$ is the total number of reported points.

Proof. We design an efficient line sweep algorithm. We will perform a separate line sweep, left to right, for each family $\mathcal{F}_i$.

The sweep goes over horizontal (x) coordinates in a left-to-right manner. The broom stores vertical (y) coordinates of horizontal sides of rectangles that it currently intersects. They are stored in a sorted list L of pairs (y, c), where y is the coordinate, and c is the count of rectangles with bottom side at coordinate y minus the count of the rectangles with top side at coordinate y. Only pairs with non-zero second component are stored. Clearly, the second components of the list elements always sum up to 0.

A coordinate x is processed if L is non-empty before accessing it or there exist any vertical sides of rectangles at x. All vertical sides with the same y-coordinate are processed in a batch. For every such batch we want to guarantee that endpoints of all sides are stored in a list B in a top-down order.

A top (bottom) endpoint at vertical coordinate y is stored as $(y, +1)$ ($(y, -1)$, respectively).

Let us now describe how to process a horizontal coordinate x. Let us merge the list L that is currently in the broom with the list B of the batch by the first components. If there is more than one pair with the same first component, we merge all of them together, summing up the second components.

Let us denote by L' the resulting list. We iterate over all elements of L', keeping track of the partial sum of second components, denoted as s. For every element (y, c) of L', the point (x, y) is reported for $\bigcup \mathcal{F}_i$. Moreover, if the partial sum s before considering c was positive and the previous element of L' is (y', c'), all points $(x, y' + 1), \ldots, (x, y - 1)$ are reported to Out_i.

Finally, all pairs with second component equal to zero are removed from L' which becomes the new list L.

Let us now analyze the complexity of the algorithm. The line sweep makes n steps. The total size of lists B across all families $\mathcal{F}_i$ is $\mathcal{O}(r)$ and they can be constructed simultaneously in $\mathcal{O}(n + r)$ time via bucket sort.

Processing a batch with list B takes $\mathcal{O}(|L| + |B|)$ time plus the time to report points in Out_i. As we have already noticed, the sum of $\mathcal{O}(|B|)$ components is $\mathcal{O}(r)$. For every element (y, c) of the initial list L, a point with the vertical coordinate y is reported upon merging; hence, the sum of $\mathcal{O}(|L|)$ components is dominated by $\mathcal{O}(\mathsf{output})$. Overall we achieve time complexity $\mathcal{O}(n + r + \mathsf{output})$. ◁

We apply the claim to the families $\mathcal{F}_{p,q}$. Then r and output are upper bounded by $\mathcal{O}(n^2 \log^2 n)$ by Theorem 9 and Fact 3, respectively. The optimality of our algorithm's complexity is due to the $\Omega(n^2 \log^2 n)$ lower bound on the maximum number of occurrences of primitively rooted quartics from [7]. ◀

We proceed to an efficient algorithm for enumerating distinct, not necessarily primitively rooted, quartics using 2D-runs. The solution for an analogous problem for 1-dimensional strings (computing distinct squares from runs) uses Lyndon roots of runs [22]. However, in 2 dimensions it is not clear if a similar approach could be applied efficiently, say, with the aid

of 2D Lyndon words [40] as Lyndon roots of 2D-runs. We develop a different approach in which the workhorse is the following auxiliary problem related to the folklore nearest smaller value problem.

Let us consider a grid of height m in which every cell can be black or white. We say that the grid forms a *staircase* if the set of white cells in each row is nonempty and is a prefix of this row (see Figure 6). A staircase can be uniquely determined by an array $Whites[1\,..\,m]$ such that $Whites[i]$ is the number of white cells in the ith row. We consider *shapes* of white rectangles. Each shape is a pair (p, q) that represents the dimensions of the rectangle. These shapes (and corresponding rectangles) are partially ordered by: $(p, q) < (p', q') \;\Leftrightarrow\; (p, q) \neq (p', q') \;\wedge\; p \leq p' \;\wedge\; q \leq q'$.

MAX WHITE RECTANGLES
Input: An array $Whites[1\,..\,m]$ that represents a staircase.
Output: Shapes of all maximal white rectangles in this staircase.

▶ **Lemma 21.** *MAX WHITE RECTANGLES problem can be solved in $\mathcal{O}(m)$ time.*

Proof. Assume that $Whites[0] = Whites[m + 1] = -1$. Let us define two tables of size m:

$$NSVUp[i] = \max\{j \,:\, j < i,\ Whites[j] < Whites[i]\},$$
$$NSVDown[i] = \min\{j \,:\, j > i,\ Whites[j] < Whites[i]\}.$$

They can be computed in $\mathcal{O}(m)$ time by a folklore algorithm for the nearest smaller value table; see e.g. [13]. Then the problem can be solved as in Algorithm 1 presented below. After the first for-loop, for each maximal white rectangle R we have $MaxWidth[\mathsf{height}(R)] = \mathsf{width}(R)$, but we could have redundant values for non-maximal rectangles. In order to filter out non-maximal rectangles, we process the candidates by decreasing height and remove the ones that are dominated by the previous maximal rectangle in the partial order of shapes. ◀

Algorithm 1 The first phase computes a set of shapes of type $(h, MaxWidth[h])$, at most one for each height h; see also Figure 6. In the second phase only inclusion-maximal shapes from this set are reported.

ComputeCandidates:
$MaxWidth[1\,..\,m] := (0, \ldots, 0)$
for $i := 1$ **to** m **do**
  $h := NSVDown[i] - NSVUp[i] - 1$
  $MaxWidth[h] := \max(MaxWidth[h], Whites[i])$

ReportMaximal:
$mw := 0$
for $h := m$ **down to** 1 **do**
  if $MaxWidth[h] > mw$ **then**
    Report the shape $(h, MaxWidth[h])$
    $mw := MaxWidth[h]$

▶ Remark 22. Note that the total area (and width) of a staircase can be large but the complexity of our algorithm is linear with respect to the number of rows, thanks to the small representation (array $Whites$).

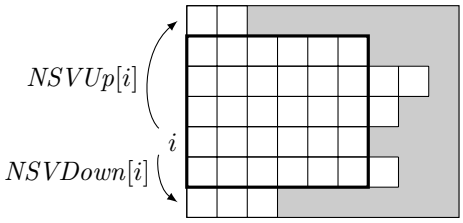


Figure 6 A maximal white rectangle containing row i is computed using the NSV tables for i.

Now our approach is graph-theoretic. The graph nodes correspond to occurrences of primitively rooted quartics. For a fixed primitively rooted quartic $W^{2,2}$ we consider the graph $G_W = (V, E)$, where V is the set of top-left corners of occurrences of $W^{2,2}$. Let $r = \mathsf{height}(W)$ and $c = \mathsf{width}(W)$. The edges in G connect vertex (i, j) with vertices $(i \pm r, j)$ and $(i, j \pm c)$, if they exist. See also Figure 7. This graph can be efficiently computed since we know its nodes due to Theorem 19.

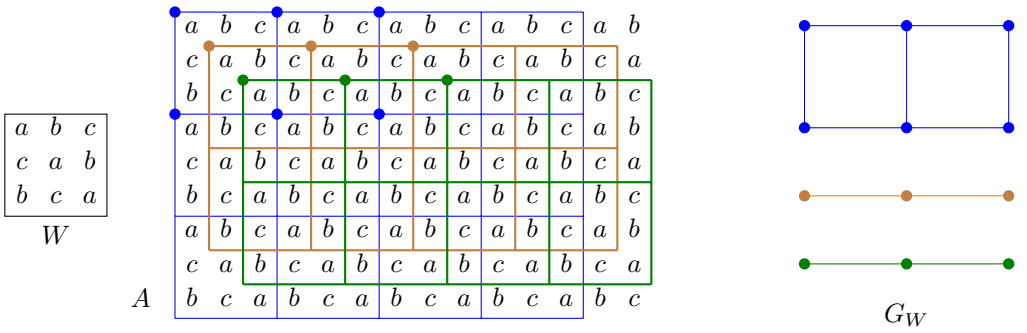


Figure 7 Graph G_W has 12 vertices that form two components with 3 vertices each (green and brown) and one component with 6 vertices (blue). Note the non-trivial occurrences of W in $W^{3,4}$.

▶ **Lemma 23.** *All graphs G_W, and their connected components, for all W which are primitive roots of quartics in A can be constructed in $\mathcal{O}(n^2 \log^2 n)$ time.*

Proof. We first compute all occurrences of primitively rooted quartics in A using Theorem 19. By Fact 3, there are $\mathcal{O}(n^2 \log^2 n)$ of them in total.

We can assign 2D-DBF identifiers (quadruples) to each of the occurrences and group the occurrences by distinct primitively rooted quartics via radix sort in $\mathcal{O}(n^2 \log^2 n)$ time. This gives us the vertices of G_W.

To compute the edges, we use an auxiliary $n \times n$ Boolean array D that will store top-left corners of occurrences of each subsequent primitively rooted quartic $W^{2,2}$.

Initially D is set to zeroes and after each W, all cells with ones are zeroed in $\mathcal{O}(|G_W|)$ time. Using this array and the positions of occurrences of $W^{2,2}$, the edges of G_W can be computed in $\mathcal{O}(|G_W|)$ time. It also allows to divide G_W into connected components via graph search in $\mathcal{O}(|G_W|)$ time. ◀

▶ **Theorem 24.** *All distinct quartics in an $n \times n$ array A can be computed in $\mathcal{O}(n^2 \log^2 n)$ time.*

Proof. We first apply Lemma 23. Now consider a fixed primitive W of height c and width r. Let us note that if $(i,j),(i',j')$ belong to the same connected component H of G_W, then $i \equiv i' \pmod r$ and $j \equiv j' \pmod c$. We say that a connected component H of G_W *generates* an occurrence of a power $W^{\alpha,\beta}$ if the $\alpha\beta$ occurrences of W that are implied by it belong to H. If $W^{\alpha,\beta}$ has an occurrence in A, then it is generated by some connected component H of G_W, unless $\min(\alpha,\beta)=1$.

We say that $W^{\alpha,\beta}$ is a *maximal* power if there is no other power $W^{\alpha',\beta'}$ in A such that $\alpha' \geq \alpha$, $\beta' \geq \beta$, and $(\alpha',\beta') \neq (\alpha,\beta)$. Similarly, we consider powers that are maximal among ones that are generated by a connected component H. Let $\mathit{MaxPowers}_W(H)$ be the set of maximal powers generated by a connected component H. It can be computed in linear time using Lemma 21 as shown in Algorithm 2, which we now explain.

For each vertex (i,j) in H, we insert four points to a set S, which correspond to the four occurrences of W underlying the occurrence of quartic $W^{2,2}$ at position (i,j). If S is treated as a set of white cells in a grid, then $W^{\alpha,\beta}$ for $\alpha > 1$ is a power generated by H if and only if the grid contains a white rectangle of shape (α,β). For a cell $(i,j) \in S$, we denote $R[i,j] = \min\{p \geq 0 : (i,j+p) \notin S\}$. Assuming that the cells of S are sorted by non-increasing second component, each value $R[i,j]$ can be computed from $R[i,j+1]$ in constant time, for a total of $\mathcal{O}(|S|)$ time. The sorting for all S can be done globally, using radix sort. Also, the array R can be stored globally and used for all S, cleared after each use. Finally, we process each maximal set of consecutive cells $(i,j),\ldots,(i+m-1,j) \in S$ that are located in the same column and apply Lemma 21 to solve the resulting instance of the Max White Rectangles problem. The total time required by this step is $\mathcal{O}(|S|)$.

Algorithm 2 Computing $\mathit{MaxPowers}_W(H)$ for a component H of G_W.

$S := \emptyset$
foreach (i,j) **in** $V(H)$ **do**
  $a := \lfloor i/r \rfloor$; $b = \lfloor j/c \rfloor$
  $S := S \cup \{(a,b),(a+1,b),(a,b+1),(a+1,b+1)\}$
$R[0\,..\,n, 0\,..\,n] := (0,\ldots,0)$
foreach (i,j) **in** S *in non-increasing order of* j **do**
  $R[i,j] := R[i,j+1]+1$
Result $:= \emptyset$
foreach *maximal set* $\{(i,j),(i+1,j)\ldots,(i+m-1,j)\} \subseteq S$ **do**
  Whites$[1\,..\,m] := R[i\,..\,i+m-1,j]$
  Result $:=$ *Result* $\cup$ MaxWhiteRectangles(*Whites*)
remove redundant rectangles from *Result*
return *Result*

In the end we filter out the powers $W^{\alpha,\beta}$ that are not maximal in A similarly as in the proof of Lemma 21, using a global array *MaxWidth*. Let $W^{\alpha_1,\beta_1},\ldots,W^{\alpha_k,\beta_k}$ be the resulting sequence of maximal powers, sorted by increasing first component, and let $\alpha_0 = \beta_0 = 0$. Then the set of all quartics in A with primitive root W contains all $W^{2\alpha,2\beta}$ over $\alpha_{p-1} < 2\alpha \leq \alpha_p$, $1 \leq 2\beta \leq \beta_p$, for $p \in [2,k]$. They can be reported in $\mathcal{O}(n^2 \log^2 n)$ total time over all W due to the upper bound of Theorem 16. ◀

6 Final Remarks

We showed that the numbers of distinct runs and quartics in an $n \times n$ array are $\mathcal{O}(n^2 \log^2 n)$. This improves upon previously known estimations. We also proposed $\mathcal{O}(n^2 \log^2 n)$-time algorithms for computing all occurrences of primitively rooted quartics and all distinct quartics. A straightforward adaptation shows that for an $m \times n$ array these bounds and complexities all become $\mathcal{O}(mn \log m \log n)$.

We pose two conjectures for $n \times n$ 2D-strings:

- The number of 2D-runs is $\mathcal{O}(n^2)$.
- The number of distinct quartics is $\mathcal{O}(n^2)$.

References

1 Amihood Amir, Gary Benson, and Martin Farach. An alphabet independent approach to two-dimensional pattern matching. *SIAM Journal on Computing*, 23(2):313–323, 1994. doi:10.1137/S0097539792226321.

2 Amihood Amir, Itai Boneh, Panagiotis Charalampopoulos, and Eitan Kondratovsky. Repetition detection in a dynamic string. In *27th Annual European Symposium on Algorithms, ESA 2019*, volume 144 of *LIPIcs*, pages 5:1–5:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.5.

3 Amihood Amir, Ayelet Butman, Gad M. Landau, Shoshana Marcus, and Dina Sokol. Double string tandem repeats. In *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, volume 161 of *LIPIcs*, pages 3:1–3:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CPM.2020.3.

4 Amihood Amir and Martin Farach. Efficient 2-dimensional approximate matching of non-rectangular figures. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, pages 212–223. ACM/SIAM, 1991. URL: http://dl.acm.org/citation.cfm?id=127787.127829.

5 Amihood Amir, Gad M. Landau, Shoshana Marcus, and Dina Sokol. Two-dimensional maximal repetitions. In *26th Annual European Symposium on Algorithms, ESA 2018*, volume 112 of *LIPIcs*, pages 2:1–2:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.2.

6 Amihood Amir, Gad M. Landau, Shoshana Marcus, and Dina Sokol. Two-dimensional maximal repetitions. *Theoretical Computer Science*, 812:49–61, 2020. doi:10.1016/j.tcs.2019.07.006.

7 Alberto Apostolico and Valentin E. Brimkov. Fibonacci arrays and their two-dimensional repetitions. *Theoretical Computer Science*, 237(1-2):263–273, 2000. doi:10.1016/S0304-3975(98)00182-0.

8 Alberto Apostolico and Valentin E. Brimkov. Optimal discovery of repetitions in 2D. *Discrete Applied Mathematics*, 151(1-3):5–20, 2005. doi:10.1016/j.dam.2005.02.019.

9 Theodore P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM Journal on Computing*, 7(4):533–541, 1978. doi:10.1137/0207043.

10 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

11 Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.22.

12 Jon Louis Bentley. Algorithms for Klee's rectangle problems. Unpublished notes, Computer Science Department, Carnegie Mellon University, 1977.

13 Omer Berkman, Baruch Schieber, and Uzi Vishkin. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms*, 14(3):344–370, 1993. doi:10.1006/jagm.1993.1018.

14 Richard S. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170, 1977. doi:10.1016/0020-0190(77)90017-5.

15 Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, and Wiktor Zuba. Counting distinct patterns in internal dictionary matching. In *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, volume 161 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CPM.2020.8.

16 Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal dictionary matching. In *30th International Symposium on Algorithms and Computation, ISAAC 2019*, volume 149 of *LIPIcs*, pages 22:1–22:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ISAAC.2019.22.

17 Maxime Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981. doi:10.1016/0020-0190(81)90024-7.

18 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

19 Maxime Crochemore and Lucian Ilie. Analysis of maximal repetitions in strings. In *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007*, volume 4708 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2007. doi:10.1007/978-3-540-74456-6_42.

20 Maxime Crochemore and Lucian Ilie. Maximal repetitions in strings. *Journal of Computer and System Sciences*, 74(5):796–807, 2008. doi:10.1016/j.jcss.2007.09.003.

21 Maxime Crochemore, Lucian Ilie, and Liviu Tinta. The "runs" conjecture. *Theoretical Computer Science*, 412(27):2931–2941, 2011. doi:10.1016/j.tcs.2010.06.019.

22 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.

23 Maxime Crochemore and Robert Mercaş. On the density of Lyndon roots in factors. *Theoretical Computer Science*, 656:234–240, 2016. doi:10.1016/j.tcs.2016.02.015.

24 Maxime Crochemore and Wojciech Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995. doi:10.1007/BF01190846.

25 Maxime Crochemore and Wojciech Rytter. *Jewels of stringology*. World Scientific, 2002. doi:10.1142/4838.

26 Antoine Deza, Frantisek Franek, and Adrien Thierry. How many double squares can a string contain? *Discrete Applied Mathematics*, 180:52–69, 2015. doi:10.1016/j.dam.2014.08.016.

27 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

28 Johannes Fischer, Stepan Holub, Tomohiro I, and Moshe Lewenstein. Beyond the runs theorem. In *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015*, volume 9309 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2015. doi:10.1007/978-3-319-23826-5_27.

29 Aviezri S. Fraenkel and Jamie Simpson. How many squares can a string contain? *Journal of Combinatorial Theory, Series A*, 82(1):112–120, 1998. doi:10.1006/jcta.1997.2843.

30 Frantisek Franek and Qian Yang. An asymptotic lower bound for the maximal number of runs in a string. *International Journal of Foundations of Computer Science*, 19(1):195–203, 2008. doi:10.1142/S0129054108005620.

31 Mathieu Giraud. Not so many runs in strings. In *Language and Automata Theory and Applications, Second International Conference, LATA 2008*, volume 5196 of *Lecture Notes in Computer Science*, pages 232–239. Springer, 2008. doi:10.1007/978-3-540-88282-4_22.

32 Amy Glen and Jamie Simpson. The total run length of a word. *Theoretical Computer Science*, 501:41–48, 2013. doi:10.1016/j.tcs.2013.06.004.

33 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004. doi:10.1016/j.jcss.2004.03.004.

34 Lucian Ilie. A note on the number of squares in a word. *Theoretical Computer Science*, 380(3):373–376, 2007. doi:10.1016/j.tcs.2007.03.025.

35 Costas S. Iliopoulos, Dennis W. G. Moore, and William F. Smyth. A characterization of the squares in a Fibonacci string. *Theoretical Computer Science*, 172(1-2):281–291, 1997. doi:10.1016/S0304-3975(96)00141-7.

36 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.

37 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.

38 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

39 M. Lothaire. *Combinatorics on words, Second Edition.* Cambridge mathematical library. Cambridge University Press, 1997.

40 Shoshana Marcus and Dina Sokol. 2D Lyndon words and applications. *Algorithmica*, 77(1):116–133, 2017. doi:10.1007/s00453-015-0065-z.

41 Wataru Matsubara, Kazuhiko Kusano, Akira Ishino, Hideo Bannai, and Ayumi Shinohara. New lower bounds for the maximum number of runs in a string. In *Proceedings of the Prague Stringology Conference 2008*, pages 140–145, 2008. URL: http://www.stringology.org/event/2008/p13.html.

42 Simon J. Puglisi, Jamie Simpson, and William F. Smyth. How many runs can a string contain? *Theoretical Computer Science*, 401(1-3):165–171, 2008. doi:10.1016/j.tcs.2008.04.020.

43 Wojciech Rytter. The number of runs in a string: Improved analysis of the linear upper bound. In *23rd Annual Symposium on Theoretical Aspects of Computer Science, STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 184–195. Springer, 2006. doi:10.1007/11672142_14.

44 Wojciech Rytter. The number of runs in a string. *Information and Computation*, 205(9):1459–1469, 2007. doi:10.1016/j.ic.2007.01.007.

45 Jamie Simpson. Modified Padovan words and the maximum number of runs in a word. *The Australasian Journal of Combinatorics*, 46:129–146, 2010. URL: http://ajc.maths.uq.edu.au/pdf/46/ajc_v46_p129.pdf.

46 Jens Stoye and Dan Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. *Theoretical Computer Science*, 270(1-2):843–856, 2002. doi:10.1016/S0304-3975(01)00121-9.

47 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. doi:10.1007/BF01206331.

A Alternative Algorithm for the Proof of Lemma 21

An alternative, space efficient and more direct algorithm that does not use additional tables *NSVDown* and *NSVUp*, is shown below. The algorithm computes only the table *MaxWidth*. Then, we can use the second phase from Algorithm 1. We assume that the table *MaxWidth* is initially filled with zeros.

Algorithm 3 Alternative implementation of the first phase in Algorithm 1.

```
Whites[0] := Whites[m + 1] := 0
S := empty stack; push(S, 0)
for i := m down to 0 do
    while Whites[i] < Whites[top(S)] do
        k := top(S); h := top(S) − i − 1
        MaxWidth[h] := max(MaxWidth[h], Whites[k])
        pop(S)
    if Whites[top(S)] = Whites[i] then pop(S)
    push(S, i)
```

The algorithm is a version of a folklore algorithm for the Nearest Smaller Values problem and correctness can be shown using the same arguments. If $Whites[i] < Whites[i+1]$, then the algorithm produces shapes of all Max White Rectangles anchored at $i+1$, otherwise $i+1$ is "nonproductive". Observe that $i+1 = top(S)$ when we start processing $i \geq 1$.

Let us analyze the time complexity of the algorithm. In total $m+2$ elements are pushed to the stack. Each iteration of the while-loop pops an element, so the total number of iterations of this loop is $\mathcal{O}(m)$. Consequently, the algorithm works in $\mathcal{O}(m)$ time. In the end one needs to filter out non-maximal rectangles as in the previous proof of Lemma 21.

New Bounds on Augmenting Steps of Block-Structured Integer Programs

Lin Chen
Department of Computer Science, Texas Tech University, Lubbock, TX, US
chenlin198662@gmail.com

Martin Koutecký
Computer Science Institute, Charles University, Prague, Czech Republic
koutecky@iuuk.mff.cuni.cz

Lei Xu
Department of Computer Science, University of Texas Rio Grande Valley, TX, US
xuleimath@gmail.com

Weidong Shi
Department of Computer Science, University of Houston, TX, US
larryshi@ymail.com

Abstract

Iterative augmentation has recently emerged as an overarching method for solving Integer Programs (IP) in variable dimension, in stark contrast with the volume and flatness techniques of IP in fixed dimension. Here we consider *4-block n-fold integer programs*, which are the most general class considered so far. A 4-block n-fold IP has a constraint matrix which consists of n copies of small matrices A, B, and D, and one copy of C, in a specific block structure. Iterative augmentation methods rely on the so-called *Graver basis* of the constraint matrix, which constitutes a set of fundamental augmenting steps. All existing algorithms rely on bounding the ℓ_1- or ℓ_∞-norm of elements of the Graver basis. Hemmecke et al. [Math. Prog. 2014] showed that 4-block n-fold IP has Graver elements of ℓ_∞-norm at most $\mathcal{O}_{FPT}(n^{2^{s_D}})$, leading to an algorithm with a similar runtime; here, s_D is the number of rows of matrix D and $\mathcal{O}_{FPT}$ hides a multiplicative factor that is only dependent on the small matrices A, B, C, D, However, it remained open whether their bounds are tight, in particular, whether they could be improved to $\mathcal{O}_{FPT}(1)$, perhaps at least in some restricted cases.

We prove that the ℓ_∞-norm of the Graver elements of 4-block n-fold IP is upper bounded by $\mathcal{O}_{FPT}(n^{s_D})$, improving significantly over the previous bound $\mathcal{O}_{FPT}(n^{2^{s_D}})$. We also provide a matching lower bound of $\Omega(n^{s_D})$ which even holds for arbitrary non-zero lattice elements, ruling out augmenting algorithm relying on even more restricted notions of augmentation than the Graver basis. We then consider a special case of 4-block n-fold in which C is a zero matrix, called 3-block n-fold IP. We show that while the ℓ_∞-norm of its Graver elements is $\Omega(n^{s_D})$, there exists a different decomposition into lattice elements whose ℓ_∞-norm is bounded by $\mathcal{O}_{FPT}(1)$, which allows us to provide improved upper bounds on the ℓ_∞-norm of Graver elements for 3-block n-fold IP. The key difference between the respective decompositions is that a Graver basis guarantees a *sign-compatible* decomposition; this property is critical in applications because it guarantees each step of the decomposition to be feasible. Consequently, our improved upper bounds let us establish faster algorithms for 3-block n-fold IP and 4-block IP, and our lower bounds strongly hint at parameterized hardness of 4-block and even 3-block n-fold IP. Furthermore, we show that 3-block n-fold IP is without loss of generality in the sense that 4-block n-fold IP can be solved in FPT oracle time by taking an algorithm for 3-block n-fold IP as an oracle.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Integer Programming, Graver basis, Fixed parameter tractable

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.33

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders; Article No. 33; pp. 33:1–33:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of this paper is available at `https://arxiv.org/abs/1805.03741`.

Funding *Lin Chen*: Research was supported in part by NSF 1756014.
Martin Koutecký: Partially supported by Israel Science Foundation grant 308/18, Charles University project UNCE/SCI/004 and by the project 19-27871X of GA ČR.

1 Introduction

A powerful mathematical tool for modeling of various optimization problems is Integer Programming:

$$\min\{\mathbf{w}\cdot\mathbf{x} \,:\, \mathcal{A}\mathbf{x}=\mathbf{b},\, \mathbf{l}\leq\mathbf{x}\leq\mathbf{u},\, \mathbf{x}\in\mathbb{Z}^N\}, \tag{IP}$$

where $\mathbf{w},\mathbf{b},\mathbf{l},\mathbf{u}$ are integer vectors of the *objective function*, *right hand side*, and *lower and upper bounds*, respectively, $\mathcal{A}$ is an integer *constraint matrix*, and $\mathbf{x}$ is a vector of *variables*. It plays a key role in theory as a component in the design of approximation and parameterized algorithms, as well as in practice, with current solvers being routinely utilized in industry and capable of handling models with thousands of variables.

In general, Integer Programming is NP-hard, as was shown already by Karp [23], which motivates the search for tractable special cases. Famous polynomially solvable cases are IPs with few rows and small coefficients as shown by Papadimitriou in 1981 [30], and IPs with few variables as shown by Lenstra in 1983. Arguably the most significant development in the last 20 years has been the introduction of *iterative augmentation* methods which led to the development of fast algorithms for wide classes of IPs whose constraint matrix has a special block structure, and to subsequent breakthrough applications in parameterized and approximation algorithms [5, 21, 27]. In fact, essentially *all* known tractable classes of IP in variable dimension are of this kind, except total unimodular IPs from the '60s.

An iterative augmentation algorithm starts with an initial feasible solution $\mathbf{x}$ and iteratively finds *augmenting steps* $\mathbf{g}\in\mathbb{Z}^N$, i.e., $\mathbf{x}+\mathbf{g}$ is feasible and $\mathbf{w}(\mathbf{x}+\mathbf{g})<\mathbf{w}\mathbf{x}$. A major question is *where* to obtain "good" augmenting steps. The *Graver basis of* $\mathcal{A}$, $\mathcal{G}(\mathcal{A})$, has emerged as an excellent choice, with good guarantees on convergence to optimal solutions while still being algorithmically "tame". Specifically, at the heart of iterative augmentation techniques are bounds on the ℓ_1- and ℓ_∞-norm of elements of the Graver basis, which enable dynamic programming to be used to find Graver elements.

We stress the role of bounds on the elements of $\mathcal{G}(\mathcal{A})$. Historically, all tractable classes of IP were discovered by proving new norm bounds and subsequently designing a dynamic program around them, with the former typically being much harder than the latter. Moreover, recent runtime improvements have followed from improving existing bounds [8, 28], and the most challenging questions in the field are tightly connected to norm bounds. Our focus here is the currently least understood class of IPs, *4-block n-fold IP*:

$$(\mathrm{IP})_{n,\mathbf{b},\mathbf{l},\mathbf{u},\mathbf{w}}: \quad \min\{\mathbf{w}\cdot\mathbf{x} : H\mathbf{x}=\mathbf{b},\, \mathbf{l}\leq\mathbf{x}\leq\mathbf{u},\, \mathbf{x}\in\mathbb{Z}^{t_B+nt_A}\}, \tag{1}$$

where H (called a 4-*block n-fold matrix*) is build from smaller blocks A, B, C and D:

$$H=\begin{pmatrix} C & D \\ B & A \end{pmatrix}^{(n)} := \begin{pmatrix} C & D & D & \cdots & D \\ B & A & 0 & & 0 \\ B & 0 & A & & 0 \\ \vdots & & & \ddots & \\ B & 0 & 0 & & A \end{pmatrix}.$$

Here, A, B, C, D are $s_i \times t_i$ matrices, $i = A, B, C, D$, respectively, and H consists of n copies of A, B, D and one copy of C. Notice that by plugging A, B, C, D into the above block structure we require that $s_C = s_D$, $s_A = s_B$, $t_B = t_C$ and $t_A = t_D$. Let Δ be the largest absolute value among all the entries of A, B, C, D. Let H_0 be a matrix obtained from H by setting $C = \mathbf{0}$. We also study *3-block n-fold IP*, obtained by replacing H with H_0.

For ease of presentation, we introduce the submatrices E and F such that

$$E := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & & 0 \\ 0 & A & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & A \end{pmatrix} \qquad F := \begin{pmatrix} B & A & 0 & & 0 \\ B & 0 & A & & 0 \\ \vdots & & & \ddots & \\ B & 0 & 0 & & A \end{pmatrix}, \tag{2}$$

4-block n-fold IP remains the simplest case of block-structured IPs for which an algorithm of runtime $f(s_A, t_A, \ldots, s_D, t_D, \Delta)n^{O(1)}$ (i.e., an FPT algorithm; see below) remains unknown. From another perspective, Koutecký et al. [28] has recently resolved the complexity of IP with respect to the structural parameters *primal* and *dual treedepth* td_P and td_D, respectively, by showing that IPs with small td_P and td_D are efficiently solvable. IPs with small *incidence treedepth* td_I subsume both of the aforementioned classes as well as 4-block n-fold IP, and 4-block n-fold IP remains the simplest open case with respect to td_I.

Our Contribution. Because we are interested in efficient algorithms, we wish to confine the exponential dependence on the input into the small numbers s_i, t_i, $i = A, B, C, D$, and Δ. Thus we take the perspective of *parameterized complexity*: for a problem instance I with a *parameter* k, we call an algorithm with runtime $f(k)|I|^{O(1)}$ a *fixed-parameter tractable (FPT)* algorithm, and an algorithm with runtime $|I|^{f(k)}$ an *XP algorithm* (for *slice-wise polynomial*). If such algorithms exist, we say that the problem is *FPT* or *XP parameterized by* k, respectively.

In this paper, we provide new and improved upper bounds and resulting algorithms for 4-block and 3-block n-fold IP, as well as the very first lower bounds for these classes which we believe to hint at the parameterized hardness of these problems. We denote by $\ker_{\mathbb{Z}}(H) = \{\mathbf{x} \in \mathbb{Z}^{t_B + nt_A} \mid H\mathbf{x} = \mathbf{0}\}$ the *integer kernel of* H, also called the *lattice of* H, and by $g_\infty(H) = \max_{\mathbf{g} \in \mathcal{G}(H)} \|\mathbf{g}\|_\infty$ the largest ℓ_∞-norm of an element of the Graver basis $\mathcal{G}(H)$ (a precise definition is given below in Section 2); analogously for H_0. First, we show an upper bound on $g_\infty(H)$.

▶ **Theorem 1.** *For any 4-block n-fold matrix* H, $g_\infty(H) \leq \mathcal{O}_{FPT}(n^{s_D})$.

This improves on the previous bound of $\mathcal{O}_{FPT}(n^{2^{s_D}})$ [16]. We also establish the first explicit lower bound matching our upper bound, making it tight up to an FPT factor. Importantly, our lower bound even applies to the first t_B coordinates (denoted $\mathbf{x}^0$ for a vector $\mathbf{x} \in \mathbb{Z}^{t_B + nt_A}$) which play a special role in algorithms for 4-block n-fold IP. What is more, our lower bound even applies to *any* non-zero element of $\ker_{\mathbb{Z}}(H)$:

▶ **Theorem 2.** *For arbitrary integer* $t \in \mathbb{N}$, *there exists a 4-block n-fold matrix* H *such that* $s_i, t_i \in O(t)$ *for* $i = A, B, C, D$, *and for* any $\mathbf{g} \in \ker_{\mathbb{Z}}(H)$ *we have* $\|\mathbf{g}^0\|_\infty = \Omega(n^t)$.

Therefore, even augmenting via a different set of steps may have to deal with steps that are unbounded by $\mathcal{O}_{FPT}(1)$. Combining Theorem 1 with the original idea of Hemmecke et al. [16] and a strongly polynomial framework of Koutecký et al. [28], we obtain the currently fastest algorithm for 4-block n-fold IP:

▶ **Theorem 3.** *4-block n-fold IP can be solved in time* $\mathcal{O}_{FPT}(n^{O(s_D t_B)})$.

Second, we restrict our attention to 3-block n-fold IP. The motivation is that 3-block n-fold IP is essentially no less general than 4-block n-fold IP. Indeed, for any 4-block n-fold IP, there exists an equivalent 3-block n-fold IP where the largest coefficient, number of rows and columns of the submatrices only increase by $\mathcal{O}(1)$ times (see Theorem 19 and Definition 17 in Appendix 5 for a formal statement).

Interestingly, the lattice elements (i.e., augmenting step candidates) of 3-block n-fold IP admit a decomposition with ℓ_∞-norm bounded by $\mathcal{O}_{FPT}(1)$:

▶ **Theorem 4.** *Any* $\mathbf{g} \in \ker_{\mathbb{Z}}(H_0)$ *decomposes to* $\sum_{i=1}^{N} \mathbf{e}_i$ *for some* $N \in \mathbb{Z}_{\geq 0}$ *with* $\mathbf{e}_i \in \ker_{\mathbb{Z}}(H_0)$ *and* $\|\mathbf{e}_i\|_\infty \leq \mathcal{O}_{FPT}(1)$ *for each* i.

However, this decomposition is not "sign-compatible", meaning possibly none of its elements is a feasible step on its own, which makes its immediate algorithmic use complicated. Nevertheless, we are able to use it to establish an upper bound of $\min\{\mathcal{O}_{FPT}(n^{s_D}), \mathcal{O}_{FPT}(n^{t_A^2+1})\}$ (below, and Theorem 1):

▶ **Theorem 5.** *For any 3-block n-fold matrix* H_0, $g_\infty(H_0) \leq \mathcal{O}_{FPT}(n^{t_A^2+1})$.

This upper bound of $\mathcal{O}_{FPT}(n^{t_A^2+1})$, which is singly exponential in t_A, is much more involved compared with the upper bound of Theorem 1. This coincides with the existing results for 4-block n-fold IP [16], where an upper bound depending on A, B (instead of C, D) is much more complicated. Our proof relies on a completely new approach, which first establishes the decomposition of Theorem 4 and then modifies it into a sign-compatible decomposition through merging summands. This may be of separate interest for deriving upper bounds on $g_\infty(\mathcal{A})$ for other classes of matrices $\mathcal{A}$, particularly for deriving an upper bound on $g_\infty(H)$ which has an explicit dependency on s_A, s_B, t_A, t_B in the exponent of n. Moreover, we show that any 4-block n-fold IP can be embedded in a 3-block n-fold IP (Theorem 19) in a particular way, which allows us to transfer the 4-block n-fold lower bound (now restricted to *feasible* lattice elements):

▶ **Theorem 6.** *For arbitrary integer* $t \in \mathbb{N}$, *there exists a 3-block n-fold IP with a matrix* H *such that* $s_i, t_i \in O(t)$ *for* $i = A, B, C, D$, *and for any* feasible *nonzero* $\mathbf{g} \in \ker_{\mathbb{Z}}(H_0)$ *we have* $\|\mathbf{g}^0\|_\infty = \Omega(n^t)$.

Finally, using our new upper bound of Theorem 5, we get that:

▶ **Theorem 7.** *3-block n-fold IP can be solved in time* $\min\{\mathcal{O}_{FPT}(n^{O(s_D t_B)}, \mathcal{O}_{FPT}(n^{O(t_A^2 t_B)})\}$.

Related Work

4-block n-fold IP originated as a generalization of two previously studied classes of IP, the *n-fold* and *2-stage stochastic* IP, which are obtained by substituting the constraint matrix H with E and F we defined before. We also call E the n-fold matrix and F the 2-stage stochastic matrix, respectively. The origins of iterative augmentation methods for 2-stage stochastic IP reach the work of Hemmecke and Schultz in 2001 [19]. De Loera et al. [7] first studied n-fold IP in 2008. Later, Hemmecke et al. [17] showed an FPT algorithm for n-fold IP based on dynamic programming, which led to a breakthrough in computational social choice [26] and was also applied in the context of scheduling by Knop and Koutecký [25]. Later, this FPT algorithm inspired a better algorithm for a special case of *combinatorial*

n-fold IP developed by Knop et al. [27], who also apply it to problems in stringology and graph algorithms. Finally, this algorithm was lifted to the general n-fold IP by Koutecký et al. [28] and Eisenbrand et al. [8].

An extension of n-fold IP to tree-structured matrices called *tree-fold IP* was developed by Chen and Marx [5] and applied to scheduling problems. Jansen et al. [21] have used n-fold IP to obtain efficient PTASes for scheduling problems. An extension of 2-stage stochastic IP analogous to tree-folds is called multi-stage stochastic and was studied by Aschenbrenner and Hemmecke [4]. Ganian and Ordyniak [12] studied the structural parameters primal treedepth and treewidth, and later Ganian et al. [13] studied dual and incidence treedepth and treewidth. Koutecký et al. [28] discovered that tree-fold and multi-stage stochastic IPs are essentially equivalent to IPs with small dual and primal treedepth, settling the parameterized complexity with respect to these parameters. The work of Koutecký et al. [28] subsumes essentially all current knowledge about the solvability of IP in variable dimension with the exception of totally unimodular constraint matrices and two related classes [2, 3], with the main remaining open problem being the complexity of 4-block n-fold IP and, more generally, IP with respect to incidence treedepth.

Bounds on $g_\infty(\mathcal{A})$ and $g_1(\mathcal{A}) = \max_{\mathbf{g}\in\mathcal{G}(\mathcal{A})} \|\mathbf{g}\|_1$ play a central role in the recent developments. For example, Chen and Marx [5] showed that tree-fold IP is FPT, but a naïve analysis yields a tower-of-exponentials dependence on the parameters. Eisenbrand et al. [8] lower this to double-exponential by improving the bounds on $g_1(\mathcal{A})$, and, at least with the current approach, the only way to obtain a single-exponential algorithm is by obtaining single-exponential bounds on $g_1(\mathcal{A})$. It has been known for a long time that 2-stage stochastic IP is FPT [19], however, there are no known bounds at all for this algorithm except for the computability of the parameter dependence f due to no bounds being available for $g_\infty(F)$. Very recently, Klein [24] is able to obtain such a bound for $g_\infty(F)$, which yields an FPT algorithm with a concrete running time. Lower bounds on $g_\infty(\mathcal{A})$ have been rare so far. Finhold and Hemmecke [11] study them in the context of n-fold IP. Koutecký et al. [28] show lower bounds (only using elementary techniques) for IPs in terms of their primal and dual treewidth.

We use the Steinitz Lemma, which has recently gained renewed attention [10, 8, 22].

2 Preliminaries

Notations

We write vectors in boldface, e.g. $\mathbf{x}, \mathbf{y}$, and their entries in normal font, e.g. x_i, y_i. Any (t_B+nt_A)-dimensional vector $\mathbf{x}$ can be divided into $n+1$ *bricks*, such that $\mathbf{x} = (\mathbf{x}^0, \mathbf{x}^1, \cdots, \mathbf{x}^n)$ where $\mathbf{x}^0 \in \mathbb{Z}^{t_B}$ and each $\mathbf{x}^i \in \mathbb{Z}^{t_A}$, $1 \leq i \leq n$. We call $\mathbf{x}^i$ the *i-th brick* for $0 \leq i \leq n$. We write $0_{s\times t}$ for an $s \times t$ matrix consisting of 0, and I_t for an $t \times t$ identity matrix. For a vector or a matrix, we write $\|\cdot\|_\infty$ to denote the maximal absolute value of its elements. For two vectors $\mathbf{x}, \mathbf{y}$ of the same dimension, $\mathbf{x} \cdot \mathbf{y}$ denotes their inner product.

Throughout this paper, we write $\mathcal{O}_{FPT}(1)$ to represent a parameter that is only dependent on $\Delta, s_A, s_B, s_C, s_D, t_A, t_B, t_C, t_D$ where Δ is the maximal absolute value among all the entries of A, B, C, D, that is, $\mathcal{O}_{FPT}(1)$ is only dependent on the small matrices A, B, C, D and is independent of n. For any computable function $f(x)$, we write $\mathcal{O}_{FPT}(f)$ to represent a computable function $f'(x)$ such that $|f'(x)| \leq \mathcal{O}_{FPT}(1) \cdot |f(x)|$, and $\Omega_{FPT}(f)$ to represent a function f'' such that $|f''(x)| \geq \Omega(1) \cdot |f(x)|$. If no FPT-term is hidden, we will use $\mathcal{O}$ in its standard meaning (e.g., in Theorem 6).

Two vectors $\mathbf{x}$ and $\mathbf{y}$ are called *sign-compatible* if $x_i \cdot y_i \geq 0$ holds for every pair of coordinates (x_i, y_i). Furthermore, we call a summation $\sum_i \mathbf{x}_i$ *sign-compatible* if the summands are pair-wise sign-compatible.

Graver basis

Consider the general integer linear programming in the standard form (IP). Let $\sqsubseteq$ be the *conformal order* in $\mathbb{R}^m$ defined such that $\mathbf{x} \sqsubseteq \mathbf{y}$ if $\mathbf{x}$ and $\mathbf{y}$ lie in the same orthant, i.e., $x_i \cdot y_i \geq 0$ for each $i = 1, \ldots, m$, and $|x_i| \leq |y_i|$ for each $i = 1, \ldots, m$. Given any subset $X \subseteq \mathbb{R}^n$, we say $\mathbf{x}$ is an $\sqsubseteq$-*minimal* element of X if $\mathbf{x} \in X$ and there does not exist $\mathbf{y} \in X$, $\mathbf{y} \neq \mathbf{x}$ such that $\mathbf{y} \sqsubseteq \mathbf{x}$. It is known that every subset of $\mathbb{Z}^m$ has finitely many $\sqsubseteq$-minimal elements. We study the Graver basis:

▶ **Definition 8** (Graver basis [14])**.** *The* Graver basis *of an integer matrix E is the finite set $\mathcal{G}(E) \subseteq \ker_{\mathbb{Z}}(E)$ of all $\sqsubseteq$-minimal elements of $\ker_{\mathbb{Z}}(E) \setminus \{\mathbf{0}\}$.*

For clarity, we sometimes emphasize that $\mathbf{g}$ comes from $\mathcal{G}(H)$ by writing it as $\mathbf{g}(H)$, and similarly for other vectors. We use the fact that any $\mathbf{x} \in \ker_{\mathbb{Z}}(H)$, $\mathbf{x} \neq 0$ can be written as $\mathbf{x} = \sum_i \alpha_i \mathbf{g}_i(H)$, where $\alpha_i \in \mathbb{Z}_+$, $\mathbf{g}_i(H) \in \mathcal{G}(H)$ and $\mathbf{g}_i(H) \sqsubseteq \mathbf{x}$ [29, Lemma 3.4].

The Graver basis $\mathcal{G}(H)$ is only dependent on H. Let $\|B\|_\infty$ be the largest absolute value over all entries. For any $\mathbf{g} \in \mathcal{G}(A)$, we have the following rough estimation for some constant c_1, c_2 [29]:

$$|\mathcal{G}(H)| \leq (c_1 \|H\|_\infty)^{mn} \quad \text{and } \|g\|_\infty \leq (c_2 \|A\|_\infty)^{mn}.$$

Augmentation algorithms for IP and Graver-best oracle

There is a general framework for solving (IP) by utilizing $\mathcal{G}(\mathcal{A})$, which was developed in a series of papers [5, 17, 21, 27]. A recent paper by Koutecký et al. [28] formalizes this framework and extends it to also obtaining *strongly polynomial* algorithms (algorithms whose number of arithmetic operations does not depend on the length of the numbers on input).

We say that $\mathbf{x}$ is *feasible* for (IP) if $\mathcal{A}\mathbf{x} = \mathbf{b}$ and $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$. Let $\mathbf{x}$ be a feasible solution for (IP). We call $\mathbf{g}$ a *feasible step* if $\mathbf{x} + \mathbf{g}$ is feasible for (IP). Further, call a feasible step $\mathbf{g}$ *augmenting* if $\mathbf{w}(\mathbf{x} + \mathbf{g}) < \mathbf{w}(\mathbf{x})$. An augmenting step $\mathbf{g}$ and a *step length* $\rho \in \mathbb{Z}$ form an $\mathbf{x}$-*feasible step pair* with respect to a feasible solution $\mathbf{x}$ if $\mathbf{l} \leq \mathbf{x} + \rho\mathbf{g} \leq \mathbf{u}$. An augmenting step $\mathbf{h}$ is a *Graver-best step* for $\mathbf{x}$ if $\mathbf{w}(\mathbf{x} + \mathbf{h}) \leq \mathbf{w}(\mathbf{x} + \rho\mathbf{g})$ for all $\mathbf{x}$-feasible step pairs $(\mathbf{g}, \rho) \in \mathcal{G}(\mathcal{A}) \times \mathbb{Z}$. The next definition and theorem show that it is sufficient to focus all our attention on finding Graver-best steps. This takes care of matters such as finding an initial feasible solution, using a proximity theorem to shrink $\mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}$ and so on.

▶ **Definition 9** (Graver-best oracle)**.** *A* Graver-best oracle *for an integer matrix $\mathcal{A}$ is one that, queried on $\mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}$ and $\mathbf{x}$ feasible to* (IP)*, returns a Graver-best step $\mathbf{h}$ for $\mathbf{x}$.*

▶ **Theorem 10** ([28])**.** *Given a Graver-best oracle for E, (IP) can be solved in strongly polynomial oracle time.*

We remark that the polynomial dependence on the dimension N and in particular the number of bricks n when it comes to 4-block n-fold IP, can be reduced using an *approximate Graver-best oracle* introduced by Altmanová et al. [1] and implicitly by Eisenbrand et al. [8].

Finiteness theorems for n-fold and 2-stage stochastic matrices

Consider an n-fold matrix E that consists of A and D (i.e., $B = C = 0$ in a 4-block n-fold matrix). It is shown that $g_\infty(E)$ is $\mathcal{O}_{FPT}(1)$. More precisely, we have the following lemma.

▶ **Lemma 11** ([9, Lemma 28]). *Let E be an n-fold matrix. Then $g_1(E) \leq (\|E\|_\infty s_D s_A)^{O(s_D s_A)}$.*

▶ **Lemma 12** ([9, Lemma 26]). *Let F be a two-stage stochastic matrix. Then $g_\infty(F) \leq f(t_B, t_A, \|A, B\|_\infty)$ for a double-exponential function f.*

Both lemmas hold for more general classes of tree-fold and multi-stage stochastic matrices.

The Steinitz lemma

The Steinitz lemma has been utilized in several recent papers [8, 10, 22] to establish better algorithms for IP. We use it as well.

▶ **Lemma 13** ([15]). *Let an arbitrary norm be given in $\mathbb{R}^\kappa$ and assume that $\|\mathbf{x}_i\| \leq \zeta$ for $1 \leq i \leq m$ and $\sum_{i=1}^m \mathbf{x}_i = \mathbf{x}$. Then there exists a permutation π such that for all positive integers $\ell \leq m$, $\|\sum_{i=1}^{\ell} \mathbf{x}_{\pi(i)} - \frac{\ell-\kappa}{m}\mathbf{x}\| \leq \kappa\zeta$.*

3 4-block n-fold IP

In this section we consider IP (1) for arbitrary H and derive matching upper and lower bounds on the ℓ_∞-norm of its Graver basis depending on the parameter $s_{\mathrm{C}} = s_{\mathrm{D}}$.

We first establish the following upper bound that improves significantly the current result.

▶ **Theorem 1.** *For any 4-block n-fold matrix H, $g_\infty(H) \leq \mathcal{O}_{FPT}(n^{s_D})$.*

Proof. Let $\mathbf{g} \in \mathcal{G}(H)$. Recall the definition of F in Eq (2). As $F \cdot \mathbf{g} = 0$, there exist $\alpha_j \in \mathbb{Z}_+$, $\mathbf{g}_j(F) \in \mathcal{G}(F)$ and $\mathbf{g}_j(F) \sqsubseteq \mathbf{g}$ such that $\mathbf{g} = \sum_{j=1}^m \alpha_j \mathbf{g}_j(F)$. Furthermore, $\|\mathbf{g}_j(F)\|_\infty = \mathcal{O}_{FPT}(1)$ according to Lemma 12. Let $\mathbf{h}_j = C \cdot \mathbf{g}_j^0(F) + \sum_{i=1}^n D\mathbf{g}_j^i(F)$, which is an s_{D}-dimensional vector such that $\|\mathbf{h}_j\|_\infty = \mathcal{O}_{FPT}(n)$. As $H\mathbf{g} = 0$, it follows that

$$\sum_{j=1}^m \alpha_j \mathbf{h}_j = \underbrace{\mathbf{h}_1 + \mathbf{h}_1 + \cdots + \mathbf{h}_1}_{\alpha_1} + \underbrace{\mathbf{h}_2 + \mathbf{h}_2 + \cdots + \mathbf{h}_2}_{\alpha_2} + \cdots + \underbrace{\mathbf{h}_m + \mathbf{h}_m + \cdots + \mathbf{h}_m}_{\alpha_m} = 0,$$

i.e., the sequence of $\mathbf{h}_i$'s sum up to 0. According to Lemma 13, there exists a permutation of the sequence such that $\|\sum_{i=1}^{\ell} \mathbf{z}_i\|_\infty \leq s_{\mathrm{D}} \cdot \mathcal{O}_{FPT}(n) = \mathcal{O}_{FPT}(n)$ for all $\ell \leq m'$, where $m' = \sum_{i=1}^m \alpha_i$ and $\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_{m'}$ is a permutation of the sequence $\underbrace{\mathbf{h}_1, \mathbf{h}_1, \cdots, \mathbf{h}_1}_{\alpha_1}, \underbrace{\mathbf{h}_2, \mathbf{h}_2, \cdots, \mathbf{h}_2}_{\alpha_2}$, $\cdots, \underbrace{\mathbf{h}_m, \mathbf{h}_m, \cdots, \mathbf{h}_m}_{\alpha_m}$. Let $\tau = \mathcal{O}_{FPT}(n)$ be the upper bound on $\|\sum_{i=1}^{\ell} \mathbf{z}_i\|_\infty$, then we know that $\sum_{i=1}^{\ell} \mathbf{z}_i \in \{-\tau, -\tau+1, \cdots, \tau\}^{s_{\mathrm{D}}}$. Consequently, if $m' > (2\tau+1)^{s_{\mathrm{D}}} + 1$, there exists $\ell_1 < \ell_2$ such that $\sum_{i=1}^{\ell_1} \mathbf{z}_i = \sum_{i=1}^{\ell_2} \mathbf{z}_i$, i.e., $\sum_{i=1}^{\ell_2-\ell_1} \mathbf{z}_i = 0$. Recall that every $\mathbf{z}_i$ corresponds to some $\mathbf{h}_{i'}$. Suppose $\sum_{i=1}^{\ell_2-\ell_1} \mathbf{z}_i = \sum_{j=1}^m \alpha'_j \mathbf{h}_j$ for $\alpha'_j \leq \alpha_j$, then by the definition of $\mathbf{h}_j$ it follows that

$$C\left(\sum_{j=1}^m \alpha'_j \mathbf{g}_j^0(F)\right) + \sum_{i=1}^n D\left(\sum_{j=1}^m \alpha'_j \mathbf{g}_j^i(F)\right) = 0.$$

Hence, $H\sum_{j=1}^m \alpha'_j \mathbf{g}_j(F) = 0$. That is, if $m' = \sum_{j=1}^m \alpha_j > (2\tau+1)^{s_{\mathrm{D}}} + 1$, then there exists some $\mathbf{g}' = \alpha'_j \mathbf{g}_j(F)$ such that $H\mathbf{g}' = 0$, $\mathbf{g}' \sqsubset \mathbf{g}$ and $\mathbf{g}' \neq \mathbf{g}$, contradicting the fact that $\mathbf{g} \in \mathcal{G}(H)$. Thus, $\sum_{j=1}^m \alpha_j \leq (2\tau+1)^{s_{\mathrm{D}}} + 1$, implying that $\|\mathbf{g}\|_\infty = \mathcal{O}_{FPT}(n^{s_{\mathrm{D}}})$. ◀

We complement our upper bound by establishing a matching lower bound. We remark that lower bound from Theorem 2 not only holds for the ℓ_∞-norm of Graver basis elements, but even holds for any non-zero lattice element. This gives a sharp contrast to 3-block n-fold IP. As we will show later in Theorem 6 and Theorem 4, a similar lower bound also exists for the ℓ_∞-norm of Graver basis elements of $\ker_{\mathbb{Z}}(H_0)$, however, $\ker_{\mathbb{Z}}(H_0)$ does admit a decomposition into lattice elements whose ℓ_∞-norm is bounded by $\mathcal{O}_{FPT}(1)$.

▶ **Theorem 2.** *For arbitrary integer $t \in \mathbb{N}$, there exists a 4-block n-fold matrix H such that $s_i, t_i \in O(t)$ for $i = A, B, C, D$, and for* any *$\mathbf{g} \in \ker_{\mathbb{Z}}(H)$ we have $\|\mathbf{g}^0\|_\infty = \Omega(n^t)$.*

Proof. We let $A = I_{t\times t}$, $B = -I_{t\times t}$. We define $(t-1)\times t$ matrices D and C such that

$$D = \begin{pmatrix} 1 & -1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & \\ 0 & 0 & 0 & \cdots & 1 & -1 \end{pmatrix} \qquad C = \begin{pmatrix} -1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & \\ 0 & 0 & 0 & \cdots & -1 & 0 \end{pmatrix}$$

Consider any nonzero $\mathbf{y} \in \ker_{\mathbb{Z}}(H)$. According to $A\mathbf{y}^0 - B\mathbf{y}^i = 0$, we know that $\mathbf{y}^0 = \mathbf{y}^i$ for every $1 \le i \le n$. According to $C\mathbf{y}^0 + \sum_{i=1}^{n} D\mathbf{y}^i = 0$, we have $(C + nD)\mathbf{y}^0 = 0$, i.e.,

$$\begin{pmatrix} n-1 & -n & 0 & \cdots & 0 & 0 \\ 0 & n-1 & -n & \cdots & 0 & 0 \\ \vdots & & & \ddots & & \\ 0 & 0 & 0 & \cdots & n-1 & -n \end{pmatrix} \cdot \mathbf{y} = 0$$

Let $\mathbf{y}^0 = (y_1, y_2, \cdots, y_t)$, the following is true:

$$(n-1)y_i = ny_{i+1}, \quad 1 \le i \le t-1 \tag{3}$$

It is easy to see that as long as $\mathbf{y} \neq 0$, we have $\mathbf{y}^0 \neq 0$ and consequently $y_i \neq 0$ for every $1 \le i \le t$. According to $(n-1)y_{t-1} = ny_t$, y_{t-1} is dividable by n. Let $y_{t-1} = nz_{t-1}$ for some $z_{t-1} \in \mathbb{Z}_{\neq 0}$. According to $(n-1)y_{t-2} = ny_{t-1} = n^2 z_{t-1}$, we know that y_{t-2} is dividable by n^2. Let $y_{t-2} = n^2 z_{t-2}$ and we plug it into $(n-1)y_{t-3} = ny_{t-2}$. In general, suppose we have shown that $y_{t-k} = n^k z_{t-k}$ for all $k \le k_0$. Now for $k = k_0 + 1$, we have $(n-1)y_{t-k_0-1} = ny_{t-k_0} = n^{k_0+1} z_{n-k_0}$, then y_{t-k_0-1} is dividable by n^{k_0+1}. Hence, we conclude that y_1 is dividable by n^{t-1}, i.e., $\|\mathbf{y}\|_\infty = \Omega(n^{t-1})$ and Theorem 2 is proved. ◀

4 3-block n-fold IP

In this section we focus on 3-block n-fold IP where $H = H_0$, i.e., $C = \mathbf{0}$. As we will show in this section, 3-block n-fold IP admits several properties that make it a particularly interesting and important special case. First, 3-block n-fold IP is without loss of generality – any 4-block n-fold IP reduces to 3-block n-fold IP with a constant increase in the parameters. Second, any element of $\ker_{\mathbb{Z}}(H_0)$ admits a decomposition into lattice elements with bounded ℓ_∞-norm, which is in certain contrast to Theorem 2. Unfortunately, a strong lower bound of $\Omega(n^t)$ for *feasible* lattice elements still exists for $s_i = t_i = \mathcal{O}(t)$. Nevertheless, we establish an alternative upper bound of $\mathcal{O}_{FPT}(n^{t_A^2+1})$ on the ℓ_∞-norm of the Graver basis elements for 3-block n-fold IP which only depends on parameters of A.

4.1 Decomposition into lattice elements with bounded ℓ_∞-norm

The goal of this subsection is to prove the following theorem.

▶ **Theorem 4.** *Any* $\mathbf{g} \in \ker_{\mathbb{Z}}(H_0)$ *decomposes to* $\sum_{i=1}^{N} \mathbf{e}_i$ *for some* $N \in \mathbb{Z}_{\geq 0}$ *with* $\mathbf{e}_i \in \ker_{\mathbb{Z}}(H_0)$ *and* $\|\mathbf{e}_i\|_\infty \leq \mathcal{O}_{FPT}(1)$ *for each* i.

Proof. Since $H_0\mathbf{g} = 0$, we know that $F \cdot \mathbf{g} = 0$. Therefore, there exist $\alpha_j \in \mathbb{Z}_+$ and $\mathbf{g}_j(F) \sqsubseteq \mathbf{g}$ such that $\mathbf{g} = \sum_j \alpha_j \mathbf{g}_j(F)$, where $\mathbf{g}_j(F) \in \mathcal{G}(F)$. Consider each $\mathbf{g}_j(F)$. As F is a two-stage stochastic matrix (recall its definition in Eq (2)), by Lemma 12 it holds for every j that $\|\mathbf{g}_j(F)\|_\infty = \mathcal{O}_{FPT}(1)$. Note that each $\mathbf{g}_j(F)$ can be written into $n+1$ bricks such that $\mathbf{g}_j(F) = \left(\mathbf{g}_j^0(F), \mathbf{g}_j^1(F), \cdots, \mathbf{g}_j^n(F)\right)$ where $\mathbf{g}_j^0(F)$ is a t_B-dimensional vector, and $\mathbf{g}_j^i(F)$ is a t_A-dimensional vector for every $1 \leq i \leq n$. It is obvious that $\|\mathbf{g}_j^i(F)\|_\infty = \mathcal{O}_{FPT}(1)$ for every $0 \leq i \leq n$, and it holds that

$$B\mathbf{g}_j^0(F) + A\mathbf{g}_j^i(F) = 0, \quad \forall 1 \leq i \leq n.$$

The claim below follows from picking a suitable $\mathbf{v}_j^*$ such that $\mathbf{g}_j^i(F) - \mathbf{v}_j^*$ has "balanced" coefficients.

▷ Claim 14. For every $\mathbf{g}_j(F)$ and $1 \leq \ell \leq |\mathcal{G}(A)|$, there exist some $\mathbf{v}_j^*$ and $\alpha_{j,\ell}^i \in \mathbb{Z}_{\geq 0}$ with

- $\mathbf{g}_j^i(F) - \mathbf{v}_j^* = \sum_{\ell=1}^{|\mathcal{G}(A)|} \alpha_{j,\ell}^i \mathbf{g}_\ell(A), \quad \forall 1 \leq i \leq n.$
- For every $1 \leq \ell \leq |\mathcal{G}(A)|$, either $|\{i : \alpha_{j,\ell}^i > 0\}| = 0$, or $|\{i : \alpha_{j,\ell}^i > 0\}| \geq n/2$.
- $\max_{i,j,\ell} |\alpha_{j,\ell}^i| \leq \alpha_{\max}$, where $\alpha_{\max} = 2g_\infty(F) = \mathcal{O}_{FPT}(1)$
- $\|\mathbf{v}_j^*\|_\infty = \mathcal{O}_{FPT}(1)$.

Proof. Consider an arbitrary $\mathbf{v}_j$ such that $\begin{pmatrix} \mathbf{g}_j^0(F) \\ \mathbf{v}_j \end{pmatrix} \in \ker_{\mathbb{Z}}([B, A])$ and $\| \begin{pmatrix} \mathbf{g}_j^0(F) \\ \mathbf{v}_j \end{pmatrix} \|_\infty \leq g_\infty(F) = \alpha_{\max}/2$. Such $\mathbf{v}_j$ exists since $\mathbf{g}_j(F)$ satisfies that $B\mathbf{g}_j^0(F) + A\mathbf{g}_j^i(F) = 0$ for any $1 \leq i \leq n$. We have $A(\mathbf{g}_j^i(F) - \mathbf{v}_j) = 0$ for every $1 \leq i \leq n$, hence there exist $\bar{\alpha}_{j,\ell}^i \in \mathbb{Z}_+$, $\mathbf{g}_\ell(A) \in \mathcal{G}(A)$ and $\mathbf{g}_\ell(A) \sqsubseteq \mathbf{g}_j^i(F) - \mathbf{v}_j$ such that for some integer m,

$$\mathbf{g}_j^i(F) - \mathbf{v}_j = \sum_{\ell=1}^{m} \bar{\alpha}_{j,\ell}^i \mathbf{g}_\ell(A), \quad \forall 1 \leq i \leq n.$$

Since $\| \begin{pmatrix} \mathbf{g}_j^0(F) \\ \mathbf{v}_j \end{pmatrix} \|_\infty \leq \alpha_{\max}/2$, consequently $\|\mathbf{g}_j^i(F) - \mathbf{v}_j\|_\infty \leq \alpha_{\max}$, and $\bar{\alpha}_{j,\ell}^i \leq \alpha_{\max}$. Consider the cardinality of the set $\{i : \bar{\alpha}_{j,\ell}^i > 0\}$. If $1 \leq |\{i : \bar{\alpha}_{j,\ell}^i > 0\}| \leq \lfloor n/2 \rfloor$, we say ℓ is *unbalanced* for $\mathbf{g}_j(F)$. Let $\bar{\alpha}_{j,\max}^i = \max_{1 \leq \ell \leq m} \bar{\alpha}_{j,\ell}^i$ and UB_j be the set of all unbalanced indices ℓ, we define

$$\mathbf{v}_j^* := \mathbf{v}_j + \sum_{\ell \in UB_j} \bar{\alpha}_{j,\max}^i \mathbf{g}_\ell(A),$$

$$\text{then, } \mathbf{g}_j^i(F) - \mathbf{v}_j^* = \sum_{\ell \in \{1,2,\cdots,m\} \setminus UB_j} \bar{\alpha}_{j,\ell}^i \mathbf{g}_\ell(A) + \sum_{\ell \in UB_j} (\bar{\alpha}_{j,\max}^i - \bar{\alpha}_{j,\ell}^i) \cdot (-\mathbf{g}_\ell(A)), \quad \forall 1 \leq i \leq n.$$

Note that $-\mathbf{g}_\ell(A) \in \mathcal{G}(A)$. For all the $\mathbf{g}_\ell(A)$'s in $\mathcal{G}(A)$ that do not appear in the above equation, we take their coefficients as 0. Furthermore, we have $|\bar{\alpha}_{j,\ell}^i| \leq \alpha_{\max}$ and $|\bar{\alpha}_{j,\max}^i - \bar{\alpha}_{j,\ell}^i| \leq \alpha_{\max}$ for all i, ℓ. As $\|\mathbf{v}_j\|_\infty = \mathcal{O}_{FPT}(1)$, $\|\mathbf{g}_\ell(A)\|_\infty = \mathcal{O}_{FPT}(1)$, we know that $\|\mathbf{v}_j^*\|_\infty = \mathcal{O}_{FPT}(1)$. Thus, the claim is proved. ◁

We call $(\mathbf{g}_j^0(F), \mathbf{v}_j^*, \mathbf{v}_j^*, \cdots, \mathbf{v}_j^*)$ as a canonical vector (of $\mathbf{g}_j(F)$). It is easy to see that $F(\mathbf{g}_j^0(F), \mathbf{v}_j^*, \mathbf{v}_j^*, \cdots, \mathbf{v}_j^*) = 0$. Since $\|\mathbf{v}_j^*\|_\infty = \mathcal{O}_{FPT}(1)$ and $\|\mathbf{g}_j^0(F)\|_\infty = \mathcal{O}_{FPT}(1)$, there are at most $\tau = \mathcal{O}_{FPT}(1)$ different kinds of canonical vectors. This means, there may be different $\mathbf{g}_k(F)$'s with the same canonical vector. We list all the τ possible canonical vectors and let $\mathbf{r}_j := (\mathbf{p}_j^*, \mathbf{v}_j^*, \mathbf{v}_j^*, \cdots, \mathbf{v}_j^*)$ be the j-th one. Let CA_j be the set of indices of all $\mathbf{g}_k(F)$'s whose canonical vector is $\mathbf{r}_j$, then we have

$$\mathbf{g} = \sum_{j=1}^{\tau}\Big(\sum_{k\in CA_j}\alpha_k\Big)\mathbf{r}_j + \sum_{j=1}^{\tau}\sum_{k\in CA_j}\alpha_k\left(\mathbf{g}_k(F) - \mathbf{r}_j\right). \tag{4}$$

We say an n-dimensional vector $\boldsymbol{\alpha} = (\alpha^1, \alpha^2, \cdots, \alpha^n) \in \mathbb{Z}_{\geq 0}^n$ is *balanced*, if $\boldsymbol{\alpha} = 0$, or $\|\boldsymbol{\alpha}\|_\infty \leq \alpha_{\max} = \mathcal{O}_{FPT}(1)$ and $|\{i : \alpha^i > 0\}| \geq n/2$. Then the following observation is true.

▶ **Observation 15.** *For any nonzero balanced vector $\boldsymbol{\alpha}$ it holds that $\|\boldsymbol{\alpha}\|_1 \geq n/2 \cdot \alpha^i/\alpha_{\max}$ for every $1 \leq i \leq n$.*

Using the concept of a balanced vector, Claim 14 indicates that if $\mathbf{r}_j$ is a canonical vector of $\mathbf{g}_k(F)$, then $\mathbf{g}_k^i(F) - \mathbf{v}_j^* = \sum_{\ell=1}^{|\mathcal{G}(A)|}\alpha_{k,\ell}^i\mathbf{g}_\ell(A)$ such that the vector $(\alpha_{k,\ell}^1, \alpha_{k,\ell}^2, \cdots, \alpha_{k,\ell}^n)$ is a balanced vector. The nice thing about balanced vectors is that we can have the following claim, which will be used several times later.

▷ Claim 16. Let $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_k$ be a sequence of balanced vectors in $\mathbb{Z}_{\geq 0}^n$ such that $\|\sum_{h=1}^k \mathbf{y}_h\|_1 \leq n\Lambda$ where $\Lambda = \mathcal{O}_{FPT}(1)$, then $\|\sum_{h=1}^k \mathbf{y}_h\|_\infty \leq 2\alpha_{\max}\Lambda = \mathcal{O}_{FPT}(1)$.

Proof of Claim 16. We prove by contradiction. Suppose on the contrary that $\|\sum_{h=1}^k \mathbf{y}_h\|_\infty > 2\alpha_{\max}\Lambda$, then there exists some i^* such that $\sum_{h=1}^k \mathbf{y}_h^{i*} > 2\alpha_{\max}\Lambda$. Since $\mathbf{y}_h$'s are balanced vectors, according to Observation 15, we have

$$\|\sum_{h=1}^k \mathbf{y}_h\|_1 = \sum_{h=1}^k \|\mathbf{y}_h\|_1 \geq n \cdot \frac{\sum_{h=1}^k \mathbf{y}_h^{i*}}{2\alpha_{\max}} > n\Lambda,$$

which contradicts the fact that $\|\sum_{h=1}^k \mathbf{y}_h\|_1 \leq n\Lambda$. Hence, the claim is true. ◁

Since $\mathbf{r}_j$ is a canonical vector of $\mathbf{g}_k(F)$, by Claim 14, there exist balanced vectors $\boldsymbol{\beta}_{k,\ell}$ such that Eq (4) can be rewritten as (ignoring $\mathbf{g}^0$):

$$\mathbf{g}^i = \sum_{j=1}^{\tau}\Big(\sum_{k\in CA_j}\alpha_k\Big)\mathbf{v}_j^* + \sum_{j=1}^{\tau}\sum_{k\in CA_j}\alpha_k\left(\sum_{\ell=1}^{|\mathcal{G}(A)|}\beta_{k,\ell}^i\mathbf{g}_\ell(A)\right), \quad \forall 1 \leq i \leq n,$$

$$\text{or equivalently,} \quad \mathbf{g}^i = \sum_{j=1}^{\tau}\alpha_j'\mathbf{v}_j^* + \sum_{\ell=1}^{|\mathcal{G}(A)|}\beta_\ell^i\mathbf{g}_\ell(A), \quad \forall 1 \leq i \leq n, \tag{5}$$

where $\alpha_j' = \sum_{k\in CA_j}\alpha_k$ and each $\beta_\ell = (\beta_\ell^1, \cdots, \beta_\ell^n)$ is the summation of balanced vectors.

$$\text{As } [0, D, D, \cdots, D]\mathbf{g} = 0, \text{ we have } \sum_{j=1}^{\tau} n\alpha_j' D\mathbf{v}_j^* + \sum_{\ell=1}^{|\mathcal{G}(A)|}\Big(\sum_{i=1}^{n}\beta_\ell^i\Big)D\mathbf{g}_\ell(A) = 0. \tag{6}$$

Note that $|\mathcal{G}(A)| = \mathcal{O}_{FPT}(1)$, the equation above can be rewritten as

$$[D\mathbf{v}_1^*, \cdots, D\mathbf{v}_\tau^*, D\mathbf{g}_1(A), \cdots, D\mathbf{g}_{|\mathcal{G}(A)|}(A)] \cdot (n\alpha_1', \cdots, n\alpha_\tau', \sum_{i=1}^{n}\beta_1^i, \cdots, \sum_{i=1}^{n}\beta_{|\mathcal{G}(A)|}^i) = 0. \tag{7}$$

Let $V = [D\mathbf{v}_1^*, D\mathbf{v}_2^*, \cdots, D\mathbf{v}_\tau^*, D\mathbf{g}_1(A), D\mathbf{g}_2(A), \cdots, D\mathbf{g}_{|\mathcal{G}(A)|}(A)]$, which is an $\mathcal{O}_{FPT}(1)\times \mathcal{O}_{FPT}(1)$-matrix with $\|V\|_\infty = \mathcal{O}_{FPT}(1)$, then there exist $\lambda_k \in \mathbb{Z}_+$ and $\mathbf{g}_k(V) \in \mathcal{G}(V)$, such that $(n\alpha'_1, n\alpha'_2, \cdots, n\alpha'_\tau, \sum_{i=1}^n \beta^i_1, \cdots, \sum_{i=1}^n \beta^i_{|\mathcal{G}(A)|}) = \sum_k \lambda_k \mathbf{g}_k(V)$. Note that since $\alpha'_j, \beta^i_\ell \geq 0$, we can restrict that every $\mathbf{g}_k(V) \in \mathbb{Z}^{\tau+|\mathcal{G}(A)|}_{\geq 0}$.

For ease of description, from now on we take the viewpoint of a packing problem. We view each canonical vector $\mathbf{r}^*_j$ and $\mathbf{g}_\ell(A)$ as an item, whereas there are $\tau + |\mathcal{G}(A)|$ different kinds of items. There are $n+1$ different bins. Bin 0 can only be used to pack items $\mathbf{r}^*_j$, $1 \leq j \leq \tau$, and bin i ($1 \leq i \leq n$) can only be used to pack items $\mathbf{g}_\ell(A)$, $1 \leq \ell \leq |\mathcal{G}(A)|$. Currently there are α'_j copies of item $\mathbf{r}^*_j$ in bin 0, and β^i_ℓ copies of item $\mathbf{g}_\ell(A)$ in bin i. This is called a packing profile. Now we want to split this packing profile into several "sub-profiles", i.e., we want to determine integers $\mu^h_j, \sigma^{i,h}_\ell \in \mathbb{Z}_{\geq 0}$ such that the followings are true:

- **(i)** $\mu^h_j, \sigma^{i,h}_\ell = \mathcal{O}_{FPT}(1)$ and $\mu^h_j + \sigma^{i,h}_\ell > 0$.
- **(ii)** $\sum_h \mu^h_j = \alpha'_j$, $\sum_h \sigma^{i,h}_\ell = \beta^i_\ell$;
- **(iii)** $[D\mathbf{v}_1^*, \cdots, D\mathbf{v}_\tau^*, D\mathbf{g}_1(A), \cdots, D\mathbf{g}_{|\mathcal{G}(A)|}(A)]$ $\cdot (n\mu^h_1, \cdots, n\mu^h_\tau, \sum_{i=1}^n \sigma^{i,h}_\ell, \cdots, \sum_{i=1}^n \sigma^{i,h}_{|\mathcal{G}(A)|}) = 0$ for every h.

A packing with μ^h_j copies of $\mathbf{r}^*_j$ in bin 0 and $\sigma^{i,h}_\ell$ copies of $\mathbf{g}_\ell(A)$ in bin i is called a sub-profile. Any sub-profile corresponds to a $(t_A + nt_B)$-dimensional vector $\mathbf{e}_h = (\mathbf{e}^0_h, \mathbf{e}^1_h, \cdots, \mathbf{e}^n_h)$ where

$$\mathbf{e}^0_h = \sum_{j=1}^{\tau} \mu^h_j \mathbf{p}^*_j$$

$$\mathbf{e}^i_h = \sum_{j=1}^{\tau} \mu^h_j \mathbf{v}^*_j + \sum_{\ell=1}^{|\mathcal{G}(A)|} \sigma^{i,h}_\ell \mathbf{g}_\ell(A), \quad \forall 1 \leq i \leq n$$

If all the three conditions on sub-profiles hold, then we know that $\|\mathbf{e}_h\|_\infty = \mathcal{O}_{FPT}(1)$, $\mathbf{g} = \sum_h \mathbf{e}_h$ and $H_0\mathbf{e}_h = 0$ (to see why $H_0\mathbf{e}_h = 0$ holds, simply recall that $F\mathbf{r}^*_j = 0$ and condition (iii) implies that $[0, D, D, \cdots, D]\mathbf{e}_h = 0$), and furthermore, there are at most $\sum_j \alpha'_j + \sum_{i,\ell} \beta^i_\ell$ sub-profiles, which is finite. Hence, $\mathbf{g} = \sum_h \mathbf{e}_h$ and the theorem is proved.

We will construct $\mathbf{e}_h$'s iteratively. Once $\mathbf{e}_h$ is constructed, we continue our decomposition procedure on $\mathbf{g} - \sum_{k=1}^h \mathbf{e}_k$.

Suppose we have constructed $\mathbf{e}_1$ to $\mathbf{e}_{h_0-1}$ where conditions (i) and (iii) are satisfied for each $\mathbf{e}_h$, $\alpha'_j - \sum_{h=1}^{h_0-1} \mu^h_j \geq 0$, $\bar{\beta}^i_\ell := \beta^i_\ell - \sum_{h=1}^{h_0-1} \sigma^{i,h}_\ell \geq 0$ and furthermore, each vector $\bar{\boldsymbol{\beta}}_\ell = (\bar{\beta}^1_\ell, \cdots, \bar{\beta}^n_\ell)$ can be expressed as a summation of all but one balanced vectors, more precisely, there exist balanced vectors $\phi_{\ell,k} \in \mathbb{Z}^n_{\geq 0}$, $1 \leq k \leq k_{\max}$ such that

$$\bar{\beta}_\ell = \sum_{k=1}^{k_{\max}-1} \phi_{\ell,k} + \bar{\phi}_{\ell,k_{\max}}, \quad \text{where } \bar{\phi}_{\ell,k_{\max}} \sqsubseteq \phi_{\ell,k_{\max}}.$$

We show how to construct $\mathbf{e}_{h_0}$. Let $\bar{\alpha}'_j = \alpha'_j - \sum_{h=1}^{h_0-1} \mu^h_j$. According to condition (iii) of each $\mathbf{e}_h$, we know that

$$[D\mathbf{v}_1^*, \cdots, D\mathbf{v}_\tau^*, D\mathbf{g}_1(A), \cdots, D\mathbf{g}_{|\mathcal{G}(A)|}(A)] \cdot (n\bar{\alpha}'_1, \cdots, n\bar{\alpha}'_\tau, \sum_{i=1}^n \bar{\beta}^i_1, \cdots, \sum_{i=1}^n \bar{\beta}^i_{|\mathcal{G}(A)|}) = 0$$

Consequently, there exist $\lambda'_k \in \mathbb{Z}_{\geq 0}$ and $\mathbf{g}_k \in \mathbb{Z}^{\tau+|\mathcal{G}(A)|}_{\geq 0} \cap \mathcal{G}(V)$ such that

$$(n\bar{\alpha}'_1, n\bar{\alpha}'_2, \cdots, n\bar{\alpha}'_\tau, \sum_{i=1}^n \bar{\beta}^i_1, \cdots, \sum_{i=1}^n \bar{\beta}^i_{|\mathcal{G}(A)|}) = \sum_k \lambda'_k \mathbf{g}_k(V).$$

There are two possibilities.

Case 1. If there exists some $\lambda'_k \geq n$, we consider the vector-summand $n\mathbf{g}_k(V)$ out of $\lambda'_k\mathbf{g}_k(V)$. Let $n\mathbf{g}_k(V) = (n\zeta_1, n\zeta_2, \cdots, n\zeta_{\tau+|\mathcal{G}(A)|})$. We set $\mu_j^{h_0} = \zeta_j = \mathcal{O}_{FPT}(1)$ for $1 \leq j \leq \tau$. We set the values of σ_ℓ^{i,h_0} such that $\sum_{i=1}^{n} \sigma_\ell^{i,h_0} = n\zeta_{\tau+\ell}$. Consequently, condition (iii) is satisfied for $\mathbf{e}_{h_0}$. Now it suffices to set the values of each σ_ℓ^{i,h_0} such that they are bounded by $\mathcal{O}_{FPT}(1)$. Equivalently, this means out of the $\bar{\beta}_\ell^i$ copies of $\mathbf{g}_\ell(A)$, our goal is to take σ_ℓ^{i,h_0} copies such that in total we take $n\zeta_{\tau+\ell}$ copies and $\sigma_\ell^{i,h_0} = \mathcal{O}_{FPT}(1)$. We achieve this in a simple greedy way. Let k^* be the index such that

$$\sum_{k=k^*+1}^{k_{\max}-1} \|\phi_{\ell,k}\|_1 + \|\bar{\phi}_{\ell,k_{\max}}\|_1 < n\zeta_{\tau+\ell} \leq \sum_{k=k^*}^{k_{\max}-1} \|\phi_{\ell,k}\|_1 + \|\bar{\phi}_{\ell,k_{\max}}\|_1$$

Let $\bar{\phi}_{\ell,k^*} \sqsubseteq \phi_{\ell,k^*}$ be an arbitrary vector such that

$$\|\bar{\phi}_{\ell,k^*}\|_1 + \sum_{k=k^*+1}^{k_{\max}-1} \|\phi_{\ell,k}\|_1 + \|\bar{\phi}_{\ell,k_{\max}}\|_1 = n\zeta_{\tau+\ell}.$$

We set $\sigma_\ell^{i,h_0} = \bar{\phi}_{\ell,k^*}^i + \sum_{k=k^*+1}^{k_{\max}-1} \phi_{\ell,k}^i + \bar{\phi}_{\ell,k_{\max}}^i$. It is obvious that in total we have taken $n\zeta_{\tau+\ell}$ copies of $\mathbf{g}_\ell(A)$. Now it remains to show that $\|\sigma_\ell^{h_0}\|_\infty = \|\bar{\phi}_{\ell,k^*} + \sum_{k=k^*+1}^{k_{\max}-1} \phi_{\ell,k} + \bar{\phi}_{\ell,k_{\max}}\|_\infty = \mathcal{O}_{FPT}(1)$. To see this, notice that each $\phi_{\ell,k}$ is a balanced vector, hence

$$\|\phi_{\ell,k^*}\|_1 + \sum_{k=k^*+1}^{k_{\max}-1} \|\phi_{\ell,k}\|_1 + \|\phi_{\ell,k_{\max}}\|_1 \leq n\zeta_{\tau+\ell} + 2n\alpha_{\max} = \mathcal{O}_{FPT}(n).$$

According to Claim 16, $\|\phi_{\ell,k^*} + \sum_{k=k^*+1}^{k_{\max}-1} \phi_{\ell,k} + \phi_{\ell,k_{\max}}\|_\infty = \mathcal{O}_{FPT}(1)$. Consequently, $\|\sigma_\ell^{h_0}\|_\infty = \mathcal{O}_{FPT}(1)$.

Also notice that after we take σ_ℓ^{i,h_0} copies of $\mathbf{g}_\ell(A)$, $\bar{\boldsymbol{\beta}}_\ell - \sigma_\ell^{h_0} = \sum_{k=1}^{k^*-1} \phi_{\ell,k} + (\phi_{\ell,k^*} - \bar{\phi}_{\ell,k^*})$, which is still the summation of all but one balanced vector. Hence we can continue to decompose $\mathbf{g} - \sum_{h=1}^{h_0} \mathbf{e}_h$.

Case 2. $\lambda'_k < n$ for every k. We claim that $\|\mathbf{g} - \sum_{h=1}^{h_0-1} \mathbf{e}_h\|_\infty = \mathcal{O}_{FPT}(1)$. If this claim is true, then $\mathbf{g} = \sum_{h=1}^{h_0-1} \mathbf{e}_h + (\mathbf{g} - \sum_{h=1}^{h_0-1} \mathbf{e}_h)$, and Theorem 4 is proved. To show the claim, we use a similar argument as that of case 1. First, $n\bar{\alpha}'_j \leq (\sum_k \lambda_k) \cdot \max_k \|\mathbf{g}_k(V)\|_\infty = \mathcal{O}_{FPT}(n)$, hence $\bar{\alpha}'_j = \mathcal{O}_{FPT}(1)$. Second, we consider the n-dimensional vector $\boldsymbol{\beta} = \sum_{\ell=1}^{|\mathcal{G}(A)|} \boldsymbol{\beta}_\ell$. Recall that $\bar{\beta}_\ell^i := \beta_\ell^i - \sum_{h=1}^{h_0-1} \sigma_\ell^{i,h} \geq 0$ and each vector $\bar{\boldsymbol{\beta}}_\ell$ satisfy that

$$\bar{\beta}_\ell = \sum_{k=1}^{k_{\max}-1} \phi_{\ell,k} + \bar{\phi}_{\ell,k_{\max}}$$

where $\bar{\phi}_{\ell,k_{\max}} \sqsubseteq \phi_{\ell,k_{\max}}$. Let $\bar{\beta}'_\ell = \sum_{k=1}^{k_{\max}} \phi_{\ell,k}$ and $\boldsymbol{\beta}' = \sum_{\ell=1}^{|\mathcal{G}(A)|} \boldsymbol{\beta}'_\ell$. Given that $\bar{\phi}_{\ell,k_{\max}} \sqsubseteq \phi_{\ell,k_{\max}}$ and $\phi_{\ell,k_{\max}}$ is a balanced vector, $\|\bar{\beta}'_\ell\|_1 \leq \|\bar{\beta}_\ell\|_1 + n\alpha_{\max}$. Consequently

$$\begin{aligned} \|\beta'\|_1 \leq \sum_{\ell=1}^{|\mathcal{G}(A)|} \|\bar{\beta}'_\ell\|_1 &\leq \sum_{\ell=1}^{|\mathcal{G}(A)|} \|\bar{\beta}_\ell\|_1 + n\alpha_{\max} \cdot |\mathcal{G}(A)| \\ &\leq \sum_k \lambda'_k \cdot \max_k \|\mathbf{g}_k(V)\|_1 + n\alpha_{\max} \cdot |\mathcal{G}(A)| = \mathcal{O}_{FPT}(n). \end{aligned}$$

Note that $\boldsymbol{\beta}'$ is the summation of balanced vectors. According to Claim 16, $\|\boldsymbol{\beta}'\|_\infty = \mathcal{O}_{FPT}(1)$, consequently $\|\boldsymbol{\beta}\|_\infty \leq \|\boldsymbol{\beta}'\|_\infty = \mathcal{O}_{FPT}(1)$. Combining the fact that $\|\mathbf{p}_j^*\|_\infty = \mathcal{O}_{FPT}(1)$, $\|\mathbf{v}_j^*\|_\infty = \mathcal{O}_{FPT}(1)$ and $\|\mathbf{g}_\ell(A)\|_\infty = \mathcal{O}_{FPT}(1)$, we have $\|\mathbf{g} - \sum_{i=1}^{h_0-1} \mathbf{e}_h\|_\infty = \mathcal{O}_{FPT}(1)$. ◀

Theorem 4 indicates that, there exists some "basis" for 3-block n-fold IP with FPT-bounded ℓ_∞-norms. Unfortunately, this basis need not be Graver basis; indeed, we will show later that the Graver basis of 3-block n-fold IP does not have an FPT-bounded ℓ_∞-norm. However, Theorem 4 provides a new perspective on the structure of the kernel space, which can be utilized to bound the ℓ_∞-norm of the Graver basis through a "merging" technique for the proof of Theorem 5 as we illustrate in the following subsection.

4.2 A sign-compatible decomposition

We have shown in the previous subsection that any element of $\ker_{\mathbb{Z}}(H_0)$ admits a decomposition into lattice elements whose ℓ_∞-norm is bounded by $\mathcal{O}_{FPT}(1)$. However, this decomposition is not necessarily "sign-compatible", meaning that possibly none of its elements is a feasible step on its own, which makes its immediate algorithmic use complicated. Towards the algorithm for 3-block n-fold IP, we resort to Graver basis. The goal of this subsection is to prove the following theorem.

▶ **Theorem 5.** *For any 3-block n-fold matrix H_0, $g_\infty(H_0) \leq \mathcal{O}_{FPT}(n^{t_A^2+1})$.*

Following the line of arguments in previous papers [4, 16, 18, 20], it seems very difficult to derive an upper bound singly exponential in t_A. To prove Theorem 5, we use a completely different approach. We give a brief overview of the proof idea. The reader is referred to the full version of this paper [6] for details.

Proof idea. The basic idea is to show that if $\|\mathbf{g}(H_0)\|_\infty$ is too large for some $\mathbf{g}(H_0) \in \mathcal{G}(H_0)$, then we are able to find some $\mathbf{z} \sqsubset \mathbf{g}(H_0)$ and $H_0\mathbf{z} = 0$, contradicting the fact that $\mathbf{g}(H_0)$ is a Graver basis element. Suppose $\mathbf{y} = \mathbf{g}(H_0)$ and $\|\mathbf{y}\|_\infty$ is very large. The crucial idea is that we do not search directly for $\mathbf{z} \sqsubset \mathbf{y}$, but rather search for $\mathbf{z} \sqsubset \tilde{\mathbf{y}}$ where $\tilde{\mathbf{y}}$ is an "equalization", of $\mathbf{y}$, and then prove that such a $\mathbf{z}$ also satisfies that $\mathbf{z} \sqsubset \mathbf{y}$.

Roughly speaking, we will divide the n bricks of $\mathbf{y}$, i.e., $\mathbf{y}^i$ for $i = 1, 2, \cdots, n$, into $\sigma = \mathcal{O}_{FPT}(1)$ groups $N_1, N_2, \cdots, N_\sigma$ such that for any $k \in N_j$, $\tilde{\mathbf{y}}^k \approx \frac{1}{|N_j|}\sum_{i\in N_j}\mathbf{y}^i$. Why do we need to take such a detour in the proof? The problem is that by directly arguing on $\mathbf{y}$ we run into a bound that is similar as [16]. Therefore, we use a completely different approach – we adopt the decomposition of Theorem 4, and then modify such a decomposition into a sign-compatible one by "merging" summands. Towards this, we first prove a merging lemma (see Lemma 5 of the full version) which states that given a summation of a sequence of vectors, we can always divide them into disjoint subsets such the summation of vectors in each subset becomes sign-compatible. The merging lemma can turn an arbitrary decomposition into a sign-compatible one, despite the fact that the cardinality of each subset is exponential in the dimension of the vectors (which means the ℓ_∞-norm of the summands will explode by a factor that is exponential in the dimension). Consequently, if we directly merge the $\mathcal{O}_{FPT}(n)$-dimensional vectors in the decomposition of Theorem 4, we get a very weak bound. To handle this, we consider an alternative sum $\tilde{\mathbf{y}}$, which is derived by averaging multiple bricks of $\mathbf{y}$ as we mentioned above.

By altering the decomposition of $\mathbf{y}$, we get a decomposition of $\tilde{\mathbf{y}}$ such that the following is true: all the $n+1$ bricks of each vector-summand can be divided into $\mathcal{O}_{FPT}(1)$ subsets where in each subset the bricks are identical. This indicates that, although we are summing up $\mathcal{O}_{FPT}(n)$-dimensional vectors to $\tilde{\mathbf{y}}$, it is essentially the same as summing up $\mathcal{O}_{FPT}(1)$-dimensional vectors. Such a transformation comes at a cost – summands summing up to $\tilde{\mathbf{y}}$ do not have $\mathcal{O}_{FPT}(1)$-bounded ℓ_∞-norms, indeed, each vector-summand consists of n bricks whose ℓ_∞-norm is $\mathcal{O}_{FPT}(1)$, and at most 1 brick (which is a t_A-dimensional vector) whose ℓ_∞-norm is $\mathcal{O}_{FPT}(n)$. Applying our merging lemma, we derive a sign-compatible decomposition of $\tilde{\mathbf{y}}$ where the summands have an ℓ_∞-norm bounded by $\mathcal{O}_{FPT}(n^{t_A^2+1})$.

It remains to show that at least one summand $\mathbf{z}$ in the sign-compatible decomposition of $\tilde{\mathbf{y}}$ also satisfies that $\mathbf{z} \sqsubset \mathbf{y}$. To show this we need to go back to the definition of $\tilde{\mathbf{y}}$. We are averaging bricks of $\mathbf{y}$, but which bricks shall we average? Each brick is a t_A-dimensional vector and we consider each coordinate. We set up a threshold Γ. If the absolute value of a coordinate is larger than Γ, we say it is (positive or negative) large. Otherwise it is small. Therefore, each brick can be characterized by identifying its coordinates being positive large, negative large or small (which is defined as the *quantity type* of a brick). We only average the bricks of the same quantity type. By doing so, we can ensure that $\tilde{\mathbf{y}}^i$ is roughly sign-compatible with $\mathbf{y}^i$ – if the j-th coordinate of $\mathbf{y}^i$ is positive or negative large, then this coordinate of $\tilde{\mathbf{y}}^i$ is also positive or negative. Hence, any $\mathbf{z} \sqsubset \tilde{\mathbf{y}}^i$ is almost sign-compatible with $\mathbf{y}$ – indeed, if we can ensure additionally that the j-th coordinate of $\mathbf{z}^i$ is 0 as long as the j-th coordinate of $\mathbf{y}^i$ is small, then we can conclude that $\mathbf{z} \sqsubset \mathbf{y}$. This "if" can be proved using a counting argument, and we get Theorem 5. ◀

5 4-block n-fold IP reduces to 3-block n-fold IP

In this section, we will show that for any 4-block n-fold IP, there exists an equivalent 3-block n-fold IP which is *kernel preserving*, as we define in the following.

▶ **Definition 17** (Extended formulation). *Let $n' \geq n$, $m' \in \mathbb{N}$, $\mathcal{A} \in \mathbb{Z}^{m \times n}$, $\mathbf{b} \in \mathbb{Z}^m$, $\mathbf{l}, \mathbf{u} \in (\mathbb{Z} \cup \{\pm\infty\})^n$ and $\mathcal{A}' \in \mathbb{Z}^{m' \times n'}$, $\mathbf{b}' \in \mathbb{Z}^{m'}$, $\mathbf{l}', \mathbf{u}' \in (\mathbb{Z} \cup \{\pm\infty\})^{n'}$. We say that*

$$\mathcal{A}'(\mathbf{x}, \mathbf{y}) = \mathbf{b}',\ \mathbf{l}' \leq (\mathbf{x}, \mathbf{y}) \leq \mathbf{u}' \tag{EF}$$

is an extended formulation *of*

$$\mathcal{A}\mathbf{x} = \mathbf{b},\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{OrigF}$$

if $\{\mathbf{x} \mid \mathcal{A}\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\} = \{\mathbf{x} \mid \exists \mathbf{y} : \mathcal{A}'(\mathbf{x}, \mathbf{y}) = \mathbf{b}', \mathbf{l}' \leq (\mathbf{x}, \mathbf{y}) \leq \mathbf{u}'\}$.

▶ **Definition 18** (Feasibly kernel-preserving extended formulation). *We say that* (EF) *is a* feasibly kernel preserving *extended formulation of* (OrigF) *if for each* $(\mathbf{x}, \mathbf{y})$ *feasible in* (EF),

$$\mathcal{A}'(\mathbf{g}, \mathbf{h}) = \mathbf{0},\ \mathbf{l}' \leq (\mathbf{x}, \mathbf{y}) + (\mathbf{g}, \mathbf{h}) \leq \mathbf{u}' \qquad \Longrightarrow \qquad \mathcal{A}\mathbf{g} = \mathbf{0},\ \mathbf{l} \leq \mathbf{x} + \mathbf{g} \leq \mathbf{u},$$

that is, each element $(\mathbf{g}, \mathbf{h})$ *of* $\ker(\mathcal{A}')$ *which is feasible with respect to* $(\mathbf{x}, \mathbf{y})$ *corresponds to an element* $\mathbf{g} \in \ker(\mathcal{A})$ *which is feasible with respect to* $\mathbf{x}$.

Extended formulations are commonly used to show how a set of solutions can be embedded in an *extended space*, perhaps using less inequalities or obeying some extra structural requirements. The basic observation is that if we take an objective function f over the original formulation (OrigF) and optimize $f'(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})$ over (EF), the optimal solution $(\mathbf{x}, \mathbf{y})$ over (EF) is such that $\mathbf{x}$ is an optimum over (OrigF). In the subsequent theorem we will use it to show that any 4-block n-fold IP can be embedded in a 3-block n-fold IP without blowing up the block sizes too much. The specific notion of a feasibly kernel-preserving extended formulation is useful to show that also our lower bounds on lattice elements are transferred, as we will show subsequently in Theorem 6.

Now we come to the main result of this subsection.

▶ **Theorem 19.** *Any 4-block n-fold IP with parameters $\Delta, s_A, s_B, s_C, s_D, t_A, t_B, t_C, t_D$ has a feasibly kernel-preserving extended formulation whose constraint matrix is a 3-block n-fold matrix with parameters $\hat{\Delta}, \hat{s}_A, \hat{s}_B, \hat{s}_D, \hat{t}_A, \hat{t}_B, \hat{t}_D$ satisfying*

$$\hat{\Delta} = \Delta \quad \hat{t}_A = \hat{t}_D = 2t_C + t_D + s_A \quad \hat{t}_B = t_B \quad \hat{s}_A = \hat{s}_B = s_B + t_C \quad \hat{s}_D = s_D = s_C\ .$$

Proof. Let us construct a 3-block n-fold IP instance which models the given 4-block IP instance. It's matrix $\hat{H}_0$ is a 3-block n-fold matrix composed of blocks $\hat{A}, \hat{B}$ and $\hat{D}$, and the remaining data is $\hat{\mathbf{b}}, \hat{\mathbf{l}}, \hat{\mathbf{u}}$ and $\hat{\mathbf{w}}$. Let the blocks be defined as follows.

$$\hat{D} = (C\ D\ \mathbf{0}\ \mathbf{0}) \qquad \hat{A} = \begin{pmatrix} -I & \mathbf{0} & I & \mathbf{0} \\ \mathbf{0} & A & \mathbf{0} & I \end{pmatrix} \qquad \hat{B} = \begin{pmatrix} I \\ B \end{pmatrix}$$

We call the four block columns of $\hat{A}$ and $\hat{D}$ *subbricks* and index them by greek letters α, β, γ and δ, i.e., $\mathbf{x}^{1\alpha}$ is the α-subbrick of the first brick.

Now, we add an extra brick which we call an *aggregation* brick, denoted $\mathbf{x}^d$ where $d = n+1$. The idea is that the α subbrick is non-zero only at the aggregation brick and corresponds to the first-stage variables of the original 4-block n-fold IP. We shall ensure that this is true using lower and upper bounds. However, to subsequently "assign" the aggregated values to the first stage variables, we also need to modify the B block, which, in turn, forces us to introduce new slack variables. This is the meaning of the γ subbrick (slack variables for bricks $i \neq d$) and δ subbrick (slack variables for the dth brick).

The right hand side $\hat{\mathbf{b}}$ is simply $\hat{\mathbf{b}}^0 = \mathbf{b}^0$ and $\hat{\mathbf{b}}^i = (\mathbf{0}\ \mathbf{b}^i)$ for $i \neq d$ and $\hat{\mathbf{b}}^d = (\mathbf{0}\ \mathbf{0})$. We set the new lower and upper bounds $\hat{\mathbf{l}}, \hat{\mathbf{u}}$ as follows:

- α **subbrick** $\hat{\mathbf{l}}^{i\alpha} = \hat{\mathbf{u}}^{i\alpha} = \mathbf{0}$ for all $i \neq d$, and $\hat{\mathbf{l}}^{d\alpha} = -\infty$, $\hat{\mathbf{u}}^{d\alpha} = +\infty$. This ensures the α subbrick to be only possibly non-zero in brick d.
- β **subbrick** $\hat{\mathbf{l}}^{i\beta} = \mathbf{l}^i$ and $\hat{\mathbf{u}}^{i\beta} = \mathbf{u}^i$ for all $i \neq d$ and $\hat{\mathbf{l}}^{d\beta} = \hat{\mathbf{u}}^{d\beta} = \mathbf{0}$. This ensures that the β subbrick has the meaning of the original variables $\mathbf{x}^i$ for all bricks except brick d, where we enforce $\hat{\mathbf{x}}^{d\beta} = \mathbf{0}$.
- γ **subbrick** $\hat{\mathbf{l}}^{d\gamma} = \hat{\mathbf{u}}^{d\gamma} = \mathbf{0}$ and $\hat{\mathbf{l}}^{i\gamma} = -\infty$, $\hat{\mathbf{u}}^{i\gamma} = +\infty$ for $i \neq d$. Without these variables and due to the structure of $\hat{A}$ and $\hat{B}$, we would be enforcing for *each* brick $i \neq d$ that $\hat{\mathbf{x}}^0 = \hat{\mathbf{x}}^{i\alpha}$, and since $\hat{\mathbf{x}}^{i\alpha} = \mathbf{0}$ this would mean $\hat{\mathbf{x}}^0 = \mathbf{0}$. The γ subbrick relaxes this to $\hat{\mathbf{x}}^0 = \hat{\mathbf{x}}^{i\alpha} + \hat{\mathbf{x}}^{i\gamma} = \mathbf{0} + \hat{\mathbf{x}}^{i\gamma}$ which is trivially satisfiable considering our setting of the bounds $\hat{\mathbf{l}}^{d\gamma}$ and $\hat{\mathbf{u}}^{d\gamma}$.
- δ **subbrick** $\hat{\mathbf{l}}^{i\delta} = \hat{\mathbf{u}}^{i\delta} = \mathbf{0}$ for all $i \neq d$, and $\hat{\mathbf{l}}^{d\delta} = -\infty$, $\hat{\mathbf{u}}^{d\delta} = +\infty$, i.e., the same as for the α subbrick. Similarly to the γ subbrick, without the δ subbrick we would be enforcing $B\hat{\mathbf{x}}^0 + A\hat{\mathbf{x}}^{d\beta} = \mathbf{0}$, however $\hat{\mathbf{x}}^{d\beta} = \mathbf{0}$ so we would be forcing $B\hat{\mathbf{x}}^0 = \mathbf{0}$, which is undesired. Thus we relax it to $B\hat{\mathbf{x}}^0 + A\hat{\mathbf{x}}^{d\beta} + \hat{\mathbf{x}}^{d\delta} = B\hat{\mathbf{x}}^0 + \hat{\mathbf{x}}^{d\delta} = \mathbf{0}$ which is trivially satisfiable.

To show that the constructed system

$$H^0\hat{\mathbf{x}} = \hat{\mathbf{b}},\ \hat{\mathbf{l}} \leq \hat{\mathbf{x}} \leq \hat{\mathbf{u}},\ \hat{\mathbf{x}} \in \mathbb{Z}^{\hat{t}_B + (n+1)\hat{t}_A} \tag{8}$$

is truly an extended formulation of $H\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^{t_C + nt_A}$, let us define a projection $\pi : \mathbb{Z}^{\hat{t}_B + (n+1)\hat{t}_A} \to \mathbb{Z}^{t_C + nt_A}$ which defines the mapping from the extended formulation to the original instance. Specifically, we let

$$\pi((\hat{x}^0, \hat{x}^{1\alpha}, \hat{x}^{1\beta}, \hat{x}^{1\gamma}, \hat{x}^{1\delta}, \hat{x}^{2\alpha}, \ldots, \hat{x}^{n\delta}, \hat{x}^{d\alpha}, \ldots, \hat{x}^{d\delta}) = (\hat{x}^0, \hat{x}^{1\beta}, \hat{x}^{2\beta}, \ldots, \hat{x}^{n\beta})\ .$$

By the arguments above we see that $\hat{\mathbf{x}}^0$ has precisely the meaning of $\mathbf{x}^0$ and $\hat{\mathbf{x}}^{i\beta}$ for $i \neq d$ has the meaning of $\mathbf{x}^i$.

Finally, let us argue that this extended formulation is also feasibly kernel-preserving. Consider now a feasible solution $\hat{\mathbf{x}}$ of (8), and consider any $\hat{\mathbf{g}}$ in $\ker(H_0)$ such that $\hat{\mathbf{x}} + \hat{\mathbf{g}}$ is again feasible. We have to show that $H\pi(\hat{\mathbf{x}}) = \mathbf{0}$ and $\mathbf{l} \leq \mathbf{x} + \pi(\hat{\mathbf{x}}) \leq \mathbf{u}$. The latter follows easily from the fact that $\hat{\mathbf{l}} \leq \hat{\mathbf{x}} + \hat{\mathbf{g}} \leq \hat{\mathbf{u}}$ and that $\pi(\hat{\mathbf{l}}) = \mathbf{l}$ and $\pi(\hat{\mathbf{u}}) = \mathbf{u}$. To see the former, consider separately first the upper row $(C\ D\ \cdots\ D)$ of H and after that the remaining rows.

We have that

$$C\hat{\mathbf{x}}^{d\alpha} + D\hat{\mathbf{x}}^{d\beta} + \mathbf{0}\hat{\mathbf{x}}^{d\gamma} + \mathbf{0}\hat{\mathbf{x}}^{d\delta} + \sum_{i=1}^{n} C\hat{\mathbf{x}}^{i\alpha} + D\hat{\mathbf{x}}^{i\beta} + \mathbf{0}\hat{\mathbf{x}}^{i\gamma} + \mathbf{0}\hat{\mathbf{x}}^{i\delta} = \mathbf{0} \ .$$

Omitting the zero blocks, we obtain

$$C\hat{\mathbf{x}}^{a\alpha} + D\hat{\mathbf{x}}^{a\beta} + \sum_{i=1}^{n} C\hat{\mathbf{x}}^{i\alpha} + D\hat{\mathbf{x}}^{i\beta} = \mathbf{0} \ .$$

Recall that our bounds enforce $\hat{\mathbf{x}}^{d\beta} = \mathbf{0}$ and $\hat{\mathbf{x}}^{i\alpha} = \mathbf{0}$ for $i \neq d$, and finally $\hat{\mathbf{x}}^{0} = \hat{\mathbf{x}}^{d\alpha}$, so plugging these in we obtain

$$C\hat{\mathbf{x}}^{0} + \sum_{i=1}^{n} D\hat{\mathbf{x}}^{i\beta} = \mathbf{0},$$

which by the definition of π implies that $C\pi(\mathbf{x})^0 + \sum_{i=1}^{n} D\pi(\mathbf{x})^i = \mathbf{0}$ as desired. Now it is left to show that, for each $i \neq d$, $B\pi(\mathbf{x})^0 + A\pi(\mathbf{x})^i = \mathbf{0}$. We have that

$$B\hat{\mathbf{x}}^{0} + \mathbf{0}\hat{\mathbf{x}}^{i\alpha} + A\hat{\mathbf{x}}^{i\beta} + \mathbf{0}\hat{\mathbf{x}}^{i\gamma} + I\hat{\mathbf{x}}^{i\delta} = \mathbf{0} \ .$$

Omitting the zero blocks and recalling that our bounds enforce $\hat{\mathbf{x}}^{i\delta} = \mathbf{0}$ for each $i \neq d$, we have

$$A\hat{\mathbf{x}}^{0} + A\hat{\mathbf{x}}^{i\beta} = \mathbf{0},$$

which, by definition of π, is what we wanted to show. ◀

▶ Remark 20. Theorem 19 has several consequences. One is that 4-block n-fold IP is in FPT if and only if 3-block n-fold IP is in FPT. Furthermore, as the reduction is kernel preserving, we can also utilize Theorem 19 to transfer the Graver basis elements between 4-block n-fold IP and 3-block n-fold IP, as is implied by Theorem 6.

▶ **Theorem 6.** *For arbitrary integer $t \in \mathbb{N}$, there exists a 3-block n-fold IP with a matrix H such that $s_i, t_i \in O(t)$ for $i = A, B, C, D$, and for any* feasible *nonzero $\mathbf{g} \in \ker_{\mathbb{Z}}(H_0)$ we have $\|\mathbf{g}^0\|_\infty = \Omega(n^t)$.*

Proof. Consider the instance constructed in Theorem 2 with H being the 4-block n-fold matrix from the proof. Apply Theorem 19 to this instance to obtain its feasible kernel-preserving extended formulation, which is a 3-block n-fold IP, and consider any $\hat{\mathbf{x}}$ which is a feasible solution for it. Denote by π the projection from the proof of Theorem 19.

Now let $\hat{\mathbf{g}} \in \ker_{\mathbb{Z}}(H_0) \subseteq \ker(H_0)$ be feasible with respect to $\hat{\mathbf{x}}$. By Definition 18, we have $\mathbf{g} = \pi(\hat{\mathbf{g}}) \in \ker_{\mathbb{Z}}(H)$, and by Theorem 2 we have $\|\mathbf{g}\|_\infty = \Omega(n^{t-1})$ and in particular $\|\mathbf{g}^0\|_\infty = \Omega(n^{t-1})$. By the definition of π these lower bounds transfer to $\hat{\mathbf{g}}$ and $\hat{\mathbf{g}}^0$. ◀

▶ Remark 21. The reader may wonder what if we take a "fat" kernel element of a 4-block n-fold IP, which cannot be decomposed into "thin" kernel element whose infinity norm bounded by $\mathcal{O}_{FPT}(1)$, then use Theorem 19 to construct an equivalent 3-block n-fold IP, and apply Theorem 4 to decompose the kernel element of the 3-block n-fold IP into thin elements and transform them back to the original 4-block n-fold IP. This seems to suggest that Theorem 19 is contradicting Theorem 2 and Theorem 4. We emphasize that such a contradiction does not exist. Indeed, our definition of a feasibly-preserving extended formulation is such that it takes kernel elements from 3-block n-fold IP to 4-block n-fold IP

only if they are feasible in the 3-block n-fold IP. Note that the construction of Theorem 19 requires specific lower and upper bounds on the extended variables $\mathbf{y}$. This is where the non-conformality of the decomposition of Thm 4 comes into play: what happens is that we decompose a kernel element of 3-block into "thin" kernel elements, however, we cannot take them back to the 4-block because they do not satisfy the bounds on the extended variables, and thus the definition of feasibly kernel-preserving extended formulation does not guarantee anything for them.

6 Algorithms

Using the upper bound on the Graver basis elements, we can derive algorithms for 3-block and 4-block n-fold IP by combining the idea from [16] and the recent progress in [28, 8], as indicated by Theorem 7 and Theorem 3.

In the following we prove Theorem 7. Theorem 3 can be proved by plugging in the upper bound of 4-block n-fold IP and proceed with the same argument.

▶ **Theorem 7.** *3-block n-fold IP can be solved in time* $\min\{\mathcal{O}_{FPT}(n^{O(s_D t_B)}, \mathcal{O}_{FPT}(n^{O(t_A^2 t_B)})\}$.

Proof. Using the idea of approximate Graver-best oracle introduced by Altmanová et al. [1] and implicitly by Eisenbrand et al. [8], it suffices for us to solve the following IP for each fixed value $\rho_0 = 2^0, 2^1, 2^2, \cdots$:

$$\min\{\mathbf{wx} : H_0\mathbf{x} = 0, \mathbf{l} \leq \mathbf{x}_0 + \rho_0\mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^m, \|\mathbf{x}\|_\infty \leq \min\{\mathcal{O}_{FPT}(n^{s_c}), \mathcal{O}_{FPT}(n^{t_A^2+1})\}\}$$

Let $\mathbf{x}_*$ be the optimal solution. Given that $\|\mathbf{x}_*\|_\infty \leq \mathcal{O}_{FPT}(n^{t_A^2+1})$, we can guess $\mathbf{x}_*^0$ and there are $\mathcal{O}_{FPT}(n^{(t_A^2+1)t_B})$ different possibilities. For each guess, say, $\mathbf{x}_*^0 = \mathbf{v}$, we solve the following problem:

$$\min\{\mathbf{w} \cdot \mathbf{x} : H_0\mathbf{x} = 0, \mathbf{l} \leq \mathbf{x}_0 + \rho_0\mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^m, \mathbf{x}^0 = \mathbf{v}\}$$

By fixing $\mathbf{x}^0$, the above problem becomes exactly an n-fold IP, which can be solved efficiently in $\mathcal{O}_{FPT}(n^2 \log n^2)$ time [8]. Notice that ρ_0 may take $\mathcal{O}_{FPT}(n \log n)$ distinct values, the overall running time is $\min\{\mathcal{O}_{FPT}(n^{s_D t_B+3}) \log^3 n, \mathcal{O}_{FPT}(n^{(t_A^2+1)t_B+3} \log^3 n)\}$. ◀

7 Conclusion

We consider 4-block n-fold IP and its important special case 3-block n-fold IP, both generalizing the two-stage stochastic IP and n-fold IP. We show that lattice elements of 3-block n-fold IP admit a decomposition whose ℓ_∞-norm is bounded in $\mathcal{O}_{FPT}(1)$, while any non-zero integral element in the kernel of 4-block n-fold IP may have an ℓ_∞-norm at least $\Omega(n^{s_c})$. We provide a matching upper bound on the ℓ_∞-norm of the Graver basis for 4-block n-fold IP, which gives an exponential improvement upon the best known result. We also establish an upper bound of $\min\{\mathcal{O}_{FPT}(n^{s_c}), \mathcal{O}_{FPT}(n^{t_A^2}+1)\}$ on the ℓ_∞-norm of the Graver basis for 3-block n-fold IP. A remaining important open problem is whether 4-block n-fold IP, or equivalently, 3-block n-fold IP, is in FPT. Our lower bounds give some indication that this is unlikely.

References

1 Katerina Altmanová, Dusan Knop, and Martin Koutecký. Evaluating and tuning n-fold integer programming. *ACM Journal of Experimental Algorithmics*, 24(1):2.2:1–2.2:22, 2019. doi:10.1145/3330137.

2 Gautam Appa, Balázs Kotnyek, Konstantinos Papalamprou, and Leonidas Pitsoulis. Optimization with binet matrices. *Operations research letters*, 35(3):345–352, 2007.

3 Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1206–1219. ACM, 2017.

4 Matthias Aschenbrenner and Raymond Hemmecke. Finiteness theorems in stochastic integer programming. *Foundations of Computational Mathematics*, 7(2):183–227, 2007.

5 Lin Chen and Daniel Marx. Covering a tree with rooted subtrees–parameterized and approximation algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2801–2820. SIAM, 2018.

6 Lin Chen, Lei Xu, Weidong Shi, and Martin Koutecký. New bounds on augmenting steps of block-structured integer programs. *arXiv preprint*, 2018. arXiv:1805.03741.

7 Jesús A De Loera, Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. N-fold integer programming. *Discrete Optimization*, 5(2):231–241, 2008.

8 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.49.

9 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *arXiv preprint*, 2019. arXiv:1904.01361.

10 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 808–816. SIAM, 2018.

11 Elisabeth Finhold and Raymond Hemmecke. Lower bounds on the graver complexity of m-fold matrices. *Annals of Combinatorics*, 20(1):73–85, 2016.

12 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artificial Intelligence*, 2018.

13 Robert Ganian, Sebastian Ordyniak, and MS Ramanujan. Going beyond primal treewidth for (m) ilp. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

14 Jack E Graver. On the foundations of linear and integer linear programming i. *Mathematical Programming*, 9(1):207–226, 1975.

15 Victor S Grinberg and Sergey V Sevast'yanov. Value of the steinitz constant. *Functional Analysis and Its Applications*, 14(2):125–126, 1980.

16 Raymond Hemmecke, Matthias Köppe, and Robert Weismantel. Graver basis and proximity techniques for block-structured separable convex integer minimization problems. *Mathematical Programming*, 145(1-2):1–18, 2014.

17 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N-fold integer programming in cubic time. *Mathematical Programming*, 137(1-2):325–341, 2013.

18 Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. A polynomial oracle-time algorithm for convex integer minimization. *Mathematical Programming*, 126(1):97–117, 2011.

19 Raymond Hemmecke and Rüdiger Schultz. Decomposition methods for two-stage stochastic integer programs. In *Online optimization of large scale systems*, pages 601–622. Springer, 2001.

20 Serkan Hoşten and Seth Sullivant. A finiteness theorem for markov bases of hierarchical models. *Journal of Combinatorial Theory, Series A*, 114(2):311–321, 2007.

21 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-ip - new PTAS results for scheduling with setups times. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 44:1–44:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.44.

22 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.43.

23 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

24 Kim-Manuel Klein. About the complexity of two-stage stochastic ips. In Daniel Bienstock and Giacomo Zambelli, editors, *Integer Programming and Combinatorial Optimization - 21st International Conference, IPCO 2020, London, UK, June 8-10, 2020, Proceedings*, volume 12125 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2020. doi:10.1007/978-3-030-45771-6_20.

25 Dusan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *J. Sched.*, 21(5):493–503, 2018. doi:10.1007/s10951-017-0550-0.

26 Dusan Knop, Martin Kouteckỳ, and Matthias Mnich. Voting and bribing in single-exponential time. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 66. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

27 Dušan Knop, Martin Kouteckỳ, and Matthias Mnich. Combinatorial n-fold integer programming and applications. *Mathematical Programming*, pages 1–34, 2019.

28 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 85:1–85:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.85.

29 Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010.

30 Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

Distance Bounds for High Dimensional Consistent Digital Rays and 2-D Partially-Consistent Digital Rays

Man-Kwun Chiu
Institut für Informatik, Freie Universität Berlin, Germany
chiumk@zedat.fu-berlin.de

Matias Korman
Department of Computer Science, Tufts University, Medford, MA, USA
matias.korman@tufts.edu

Martin Suderland
Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland
martin.suderland@usi.ch

Takeshi Tokuyama
Kwansei Gakuin University, Sanda, Japan
tokuyama@kwansei.ac.jp

Abstract

We consider the problem of digitalizing Euclidean segments. Specifically, we look for a constructive method to connect any two points in $\mathbb{Z}^d$. The construction must be *consistent* (that is, satisfy the natural extension of the Euclidean axioms) while resembling them as much as possible. Previous work has shown asymptotically tight results in two dimensions with $\Theta(\log N)$ error, where resemblance between segments is measured with the Hausdorff distance, and N is the L_1 distance between the two points. This construction was considered tight because of a $\Omega(\log N)$ lower bound that applies to any consistent construction in $\mathbb{Z}^2$.

In this paper we observe that the lower bound does not directly extend to higher dimensions. We give an alternative argument showing that any consistent construction in d dimensions must have $\Omega(\log^{1/(d-1)} N)$ error. We tie the error of a consistent construction in high dimensions to the error of similar *weak* constructions in two dimensions (constructions for which some points need not satisfy all the axioms). This not only opens the possibility for having constructions with $o(\log N)$ error in high dimensions, but also opens up an interesting line of research in the tradeoff between the number of axiom violations and the error of the construction. In order to show our lower bound, we also consider a colored variation of the concept of discrepancy of a set of points that we find of independent interest.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Consistent Digital Line Segments, Digital Geometry, Discrepancy

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.34

Related Version A full version of the paper is available at `https://arxiv.org/abs/2006.14059`.

Funding *Man-Kwun Chiu*: Partially supported by ERC StG 757609.
Matias Korman: Supported by MEXT Kakenhi No. 17K12635 and the NSF award CCF-1422311.
Takeshi Tokuyama: Supported by MEXT Kakenhi 17K19954 and 18H05291.

Acknowledgements The authors would like to thank Matthew Gibson, Evanthia Papadopoulou, André van Renssen and Marcel Roeloffzen for their helpful discussions during the creation of this paper. The authors would also like to thank the anonymous reviewers for the many comments that helped improve the paper. We would especially like to thank the SODA reviewer that showed us how to improve the lower bound from $\Omega(\log^{1/d} N)$ to $\Omega(\log^{1/(d-1)} N)$.

1 Introduction

Euclidean line segments are one of the most fundamental objects of geometry. Although often loosely referred to as *the shortest path connecting the endpoints*, segments have a clear and unique axiomatic definition out of which many interesting properties follow. For example, it is well-known that the intersection of two segments is always a segment (that could possibly degenerate to a point or even become empty). The definition of other mathematical concepts heavily depends on the definition of segments (such as convex regions).

The definition of segment works very well in a Euclidean or similar spaces with infinite precision. Digital representation (such as pixels in a screen) introduces imprecision. The most common approach used in practice is to somehow *round* the Euclidean segment into the digital space. The digital segments will look very similar to the Euclidean counterparts (that is, the *error* is very small). However, we cannot guarantee the useful properties and concepts that follow from the axiomatic definition of Euclidean segment (see Figure 1).

In the aspect of the consistency of digital segments, we look for a deterministic method to construct digital segments in a way that (i) the analogous of Euclidean axioms are satisfied and (ii) the digital segments resemble the Euclidean ones as much as possible.

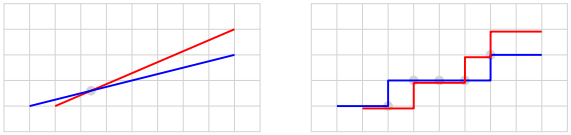


Figure 1 Left: Two Euclidean line segments that intersect in a point. Right: Rounding produces polylines that intersect in three disconnected components.

Preliminaries

Our aim is to construct a digital path $dig(p, q)$ for any two points $p, q \in \mathbb{Z}^d$. Ideally, we want dig to be defined for any pairs of points in $\mathbb{Z}^d$ (full list of requirements is described below), but sometimes we consider the case in which dig is only defined for a subset of $\mathbb{Z}^d \times \mathbb{Z}^d$.

▶ **Definition 1.** *For any $S \subseteq \mathbb{Z}^d \times \mathbb{Z}^d$, let $DS(S)$ be a set of digital segments such that $dig(p, q) \in DS(S)$ for all $(p, q) \in S$. We say that $DS(S)$ forms a* partial set of consistent digital segments *on S (partial CDS for short) if for every pair $(p, q) \in S$ it satisfies the following five axioms:*

(S1) *Grid path property: $dig(p, q)$ is a path between p and q under the $2d$-neighbor topology*[1]*.*
(S2) *Symmetry property: if $(q, p) \in S$, $dig(p, q) = dig(q, p)$.*
(S3) *Subsegment property: for any $r \in dig(p, q)$, $dig(p, r) \in DS(S)$ and $dig(p, r) \subseteq dig(p, q)$.*
(S4) *Prolongation property: $\exists\ r \in \mathbb{Z}^d$ such that $dig(p, r) \in DS(S)$ and $dig(p, q) \subset dig(p, r)$.*
(S5) *Monotonicity property: for all $i \leq d$ such that $p_i = q_i$, it holds that every point $r \in dig(p, q)$ satisfies $r_i = p_i = q_i$.*

These axioms give nice properties of digital segments analogous to Euclidean line segments. For example, (S1) and (S3) imply that the intersection of two digital segments is another segment (that could degenerate to a single point or an empty set). (S5) implies that the intersection of a segment with an axis-aligned halfspace is a segment.

[1] The $2d$-neighbor topology is the natural one that connects to your predecessor and successor in each dimension. Formally speaking, two points are connected if and only if their $||\cdot||_1$ distance is exactly one.

A partial CDS for $S = \mathbb{Z}^d \times \mathbb{Z}^d$ is called a set of *consistent digital segments* (CDS for short). Although our final goal is to have such a construction that works for the case in which $S = \mathbb{Z}^d \times \mathbb{Z}^d$, in this paper we consider subsets of the form $S = \{o\} \times \mathbb{Z}^d$ (where o is the origin or any fixed point in $\mathbb{Z}^d$). We say that a partial CDS on such a set is a *consistent digital ray* system (CDR for short), as it contains all segments (or rays) from o to $\mathbb{Z}^d$.

Another property that we want from partial CDS is that they visually resemble the Euclidean segments. The resemblance between the digital segment $dig(p, q)$ and the Euclidean counterpart $\overline{pq}$ is measured using the *Hausdorff* distance. The Hausdorff distance $H(A, B)$ of two objects A and B is defined by $H(A, B) = \max\{h(A, B), h(B, A)\}$, where $h(A, B) = \max_{a\in A} \min_{b\in B} \delta(a, b)$, and $\delta(a, b)$ is the standard $||\cdot||_\infty$ L-infinity norm.

Thus, the resemblance of a partial CDS on S is defined as $\max_{(p,q)\in S} H(dig(p, q), \overline{pq})$ (that is, the biggest error created between a digital segment and its Euclidean counterpart). This value is simply referred to as the *error* of the partial CDS construction. We are interested to see how the error grows as we enlarge our focus of interest. Thus, we limit the domain to the case in which both points are in the L_1 ball of radius N centered at the origin (i.e. $\mathcal{G}_N = \mathbb{Z}^d \cap B_1(o, N)$). Rather than looking for the exact function, we are interested in the asymptotic behavior of the error as a function of N. For simplicity, we will actually restrict ourselves to the positive orthant $\mathcal{G}_N^+ = \mathcal{G}_N \cap_i H_i$, where $H_i = \{p \in \mathbb{Z}^d : p_i \geq 0\}$ and p_i is the i-th coordinate of p (the results extend to other orthants by symmetry).

Previous Work

Research on the digital representation of line segments has been an active area of research for over half a century [10]. Many different approaches have been considered. Most common techniques look for methods that implicitly encode the properties we desire. For example, a popular approach is to consider a *dynamic* method to digitize line segments. In this setting, the way we transform a Euclidean segment into a digital one will depend on which other segments are present (and their specific coordinates). It is known that a grid of exponential size is needed if we want to preserve the combinatorial types [9]. Another workaround is known as *snap rounding* that represents line segments by polygonal chains: Each segment is carefully rounded to avoid inconsistencies. Note that both of these ideas implicitly keep the error small while making sure that the intersection of two digital segments is a connected component. Although they work well in practice, they have the drawback that they cannot be used to define objects that are based on digital segments (such as digital starshapes or convex region).

The first paper to explicitly look for an axiomatic approach was in 1987 by Luby [11]: in his work he introduced the concept of CDS (under the name of *smooth geometries*) and gave a method to construct CDSs in $\mathbb{Z}^2$ based on a characterization of CDRs in $\mathbb{Z}^2$: any CDR can be uniquely identified by four total orders of the integers (and vice versa). By choosing a proper total order and using it for all points of $\mathbb{Z}^2$ we obtain a CDR with $O(\log N)$ error. Håstad[2] gave a matching lower bound for any such construction. The lower bound is based on discrepancy theory [12]: any CDR is mapped to a sequence of real numbers in $[0, 1)$ in a way that the error of the CDR is proportional to the discrepancy of the sequence (intuitively speaking, a measure on how well shuffled the numbers are).

These results were rediscovered by Chun et al. [8] and Christ et al. [6]. They renewed interest in the topic and sparked other related research: Chowdhury and Gibson [4] gave necessary and sufficient conditions for a collection of CDRs to form a CDS. In a companion

[2] The lower bound was published by Luby, but credit given to Håstad (see Theorem 19 of [11]).

paper, the same authors [5] afterwards provided an alternative characterization together with a constructive algorithm; specifically, they gave an algorithm that, given a collection of segments in an $N \times N$ grid that satisfies the five axioms, computes a CDS that contains those segments. The algorithm runs in polynomial time of N.

Unfortunately, most of these results only work on the digital plane. Out of the previously mentioned results, only the CDR construction of Chun et al. [8] extends to three and higher dimensions. The construction has $O(\log N)$ error regardless of the dimension. Chun et al. [8] also considered the case in which the monotonicity property (S5) is not preserved. They showed that if we remove (S5), we can obtain a CDR with $O(1)$ error in any dimension. Although the error is small, the resulting segments are far from what we would consider similar to the Euclidean segments (because they loop around many times). Recently, Chiu and Korman [2] showed that the problem in higher dimensions behaves very differently from the two dimensional case. Specifically, they studied how to extend the CDS construction of Christ et al. [6] and showed that it is very limiting in three (and higher) dimensions. We can use their method to get arbitrarily many CDRs (with $\Omega(\log N)$ error) and sometimes we can get a CDS. However, whenever the construction yields a CDS, it will have $\Omega(N)$ error.

Our interest in higher dimensions comes motivated by an application in *image segmentation*. Image segmentation is the act of separating an object from its background in an image (that is, determining which pixels are part of the background and which ones not). Chun et al. [8] showed how to combine their CDR construction with the framework of Asano et al. [1] to segment two dimensional images. This idea has been extended to consider other shapes (see [7] for a detailed list), but always two dimensional. The hope is that a high dimensional CDR with low error will produce more accurate segmentation algorithms. Although traditional images taken with a camera are two dimensional, images from a medical equipment such as those taken with an MRI machine can have three or even higher dimensions (say, when we want to track changes of a particular object along time).

Results and paper organization

When approximating some geometric object, it often happens that higher dimensions create a larger error than in a lower dimension setting. Since the high dimensional setting contains a two dimensional subspace, it is common for lower bounds to extend to higher dimensions. However, this is not true for the case of CDRs: although a three dimensional CDR contains two dimensional subspaces, those subspaces need not exactly be CDRs (and thus the $\Omega(\log N)$ lower bound does not directly hold). In this paper, we further explain the reason and investigate the lower bound for the higher dimensional case.

The main reason why a subspace is not a CDR is because of the prolongation property (S4): we require that every segment is extendable, but has no constraints on the dimension in which it does so. In particular, a subspace of a high dimensional CDR need not be a CDR (see an example in Figure 2). Subspaces of CDRs are what we call *weak CDR*: it is a construction that almost always behaves like a CDR but some vertices may not satisfy the prolongation property (S4). Each vertex that does not extend is called an *inner leaf*. In this paper we study weak CDRs in two dimensions and the implications that they have for (proper) CDRs in higher dimensions.

The new found properties of weak CDRs allow us to extend the two-dimensional lower bound to higher dimensions. Håstad's bound was based on a mapping from a (two-dimensional) CDR into a pointset in $[0, 1) \subset \mathbb{R}$ and tied the error of the CDR to the discrepancy of the transformed pointset. Our lower bound uses an additional intermediate step: from any CDR we consider the weak CDR it generates in the x_1x_2-plane. We then

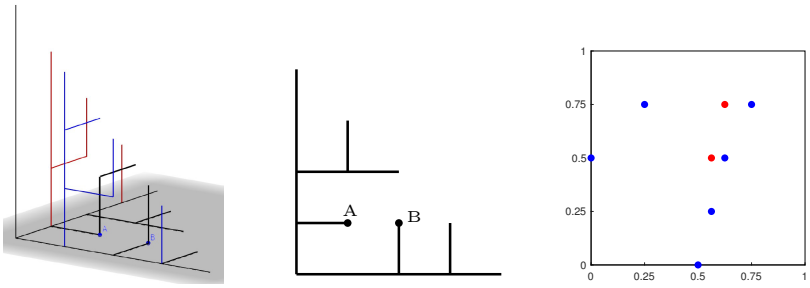


Figure 2 (left) A drawing of a CDR in $\mathcal{G}_N^+ \subset \mathbb{Z}^3$ for $N = 4$. Notice that the CDR is a tree whose leaves are at the plane $x + y + z = N$. (middle) A cross section on the xy-plane of the same CDR. Observe that vertices A and B do not extend within the xy-plane. Thus, the subspace is a weak CDR (rather than a proper CDR). (right) A map of the weak CDR into a two-colored pointset. Regions with many blue points and few red correspond to portions of the CDR with high error.

map this weak CDR into a set of points in the unit square and *then* use discrepancy theory to obtain a lower bound for the weak CDR and eventually to the high dimensional CDR. Overall, we show a very strong link between the three spaces (CDR in high dimensions, weak CDR in the x_1x_2-plane and set of points created by our mapping). Along the paper we will analyze properties of each of the spaces, and see what implications it has for the other two. Specifically, we show the following:

(i) Because we now need to account for more general constructions (weak CDRs instead of proper CDRs), the mapping needs to be changed. Instead of creating points in the $[0, 1)$ interval, in Section 2 we map into a two-colored pointset in $[0, 1) \times [0, 1)$.

(ii) Similar to the two dimensional case, we can tie the error of the weak CDR to the discrepancy of the mapped pointset. First, we extend the discrepancy results [12] to our exact setting. Let R and B be a set of red and blue points in the unit square, respectively. Let $m = |B| - |R|$ and assume $m > 0$. For any set P of points in the unit square and $x, y \in [0, 1]$ let $P[x, y]$ be the number of points in $P \cap [0, x] \times [0, y]$. For any two real numbers $0 \leq x, y \leq 1$ we define the discrepancy of R and B at (x, y) as $D_{R,B}(x, y) = mxy - (B[x, y] - R[x, y])$. The *discrepancy* of R and B is simply defined as $D^*_{R,B} = \max_{(x,y)\in[0,1]^2} |D_{R,B}(x, y)|$ (i.e., the highest discrepancy we can achieve among all possible rectangles). The discrepancy $D^*_{R,B}$ of a two-colored pointset is high if and only if there is an axis-aligned rectangle with the origin as corner in which the difference of the cardinalities is far from the expected difference.

▶ **Theorem 2** (Two colors discrepancy). *For any set R and B of points such that $|B| > |R|$ it holds that $D^*_{R,B} = \Omega\left(\frac{(|B|-|R|)\cdot\log(|B|+|R|)}{|B|+|R|}\right)$.*

The proof is given in Section 3.

(iii) With this new discrepancy result we obtain a trade-off between the error of any weak CDR and the number of inner leaves (i.e., vertices that do not satisfy (S4)). When the weak CDR has zero inner leaves (and thus is a proper CDR) our bound matches the lower bound of Håstad. As the number of inner leaves increases, the lower bound decreases. In Section 4 we prove the following relationship.

▶ **Theorem 3.** *For any $N \in \mathbb{N}$, any weak CDR defined on $\mathcal{G}_N^+ \subset \mathbb{Z}^2$ with κ_2 inner leaves between lines $x + y = \lceil N/2 \rceil$ and $x + y = N$ has $\Omega(\frac{N \log N}{N+\kappa_2})$ error.*

(iv) We then apply Theorem 3 to obtain a lower bound for CDRs in d dimensions: intuitively speaking, if the 2-D subspace has few inner leaves (say, $o(N \log N)$), then it will have $\omega(1)$ error. On the other hand, a weak CDR with many inner leaves in the 2-D

subspace will cause too many points to extend to one of the remaining dimensions, and create large error as well. This gives a lower bound of $\Omega(\log^{1/(d-1)} N)$ for any CDR construction in d dimensions (see Section 5):

▶ **Theorem 4.** *Any CDR in $\mathbb{Z}^d$ has $\Omega(\log^{1/(d-1)} N)$ error.*

Although we believe our analysis to be loose (especially in Theorem 4), we are not certain that the existing CDR constructions with $O(\log N)$ error are tight either. In the full version of the paper in [3], we explore the possibility of having a CDR in high dimensions with $o(\log N)$ error (rather than directly looking at CDRs in high dimensions, we see what properties it would imply in the other two subspaces). Although we cannot explicitly find a construction with $o(\log N)$ error, we provide interesting insight on how further research can solve this question. In particular, we give a weak CDR construction with 5/2 error and $\Theta(N^2)$ inner leaves. In order to further reduce the number of inner leaves in weak CDRs with constant error we instead look at how to create a two-colored pointset with constant discrepancy. We show that it is not possible to have $o(N^2)$ red points in some pattern of the pointset with constant discrepancy, which gives us a condition on any weak CDR with $o(N^2)$ inner leaves.

Further discussion on the implication of these results is given in Section 6.

2 Mapping a weak CDR into a pointset

We start by showing how to transform a weak CDR in two dimensions into a two-colored pointset in $[0,1)^2$. Given any weak CDR, its restriction to $\mathcal{G}_N^+$ forms a spanning tree T of $\mathcal{G}_N^+$ because of axioms (S1) and (S3). Although the tree is undirected, we see it as a directed graph (rooted tree) whose edges are oriented away from the origin (root). Then, (S5) implies that the parent of each vertex (x,y) (except the root) is either $(x-1,y)$ or $(x,y-1)$. For any edge $e=uv$ of T, where u is the parent of v, we define $T(e)$ as the subtree of T that is rooted at the child node v of e. We slightly abuse the notation and use $T(v)$ to denote the subtree that is emanating from v towards the leaves (that is, $T(v)=T(e)$).

For any $n \le N$ let L_n be the points of $\mathcal{G}_N^+$ whose sum of coordinates is n (i.e., $L_n = \{(x,y) \in \mathcal{G}_N^+ \colon x+y=n\}$). We follow the usual terminology that we call a vertex of degree one a *leaf*. We further consider two subcategories: we say that a leaf v of T is an *inner leaf* if it is not in L_N. All the vertices in L_N are called *boundary leaves*. Note that, by properties of CDR, all vertices of L_N are proper leaves (since any children should be in L_{N+1}, which is outside $\mathcal{G}_N^+$). Further note that in a proper CDR there will be no inner leaves. A vertex v of T is a *split vertex* if it has degree three or it is the origin. Let $\mathcal{S}$ be the set of split vertices and $\mathcal{D}$ the set of inner leaves.

2.1 Auxiliary function

Before giving the transformation from a tree to a point set we first define an auxiliary function $M : \mathcal{G}_N^+ \to [0,1]$. For any $p \in L_N$ we set $M(p) = \frac{p_x}{p_x+p_y}$. For any subtree $T'(v)$ of T we define two more functions inductively for $v \in L_n$ from $n=N$ to 0 as follows:

$$\max(T'(v)) = \max_{p \in T'(v) \cap (\mathcal{D} \cup L_N)} M(p) \quad \text{and} \quad \min(T'(v)) = \min_{p \in T'(v) \cap (\mathcal{D} \cup L_N)} M(p),$$

where $M(p)$ for $p \in \mathcal{D}$ is defined in the next paragraph.

For any inner leaf $\ell \in \mathcal{D}$, we know that the edges $e_1 = (\ell_x - 1, \ell_y + 1)(\ell_x, \ell_y + 1)$ and $e_2 = (\ell_x + 1, \ell_y - 1)(\ell_x + 1, \ell_y)$ must be present in T. Thus, we define $M(\ell)$ as $M(\ell) = \frac{\max(T(e_1)) + \min(T(e_2))}{2}$. Intuitively speaking, we look at the leaves above and to the right of ℓ, and assign a value that is in between the two of them (see Figure 3, left). The following statement shows that these values are sorted along L_n.

▶ **Lemma 5.** *Let $T(u), T(v) \subset T$ be two subtrees of T rooted at the vertices $u, v \in L_n$ (respectively) for some $n \leq N$ such that $u_x < v_x$. Then, it holds that* $\max(T(u)) < \min(T(v))$.

Proof. We prove this statement by induction on n from N to 1. If both $u, v \in L_N$ then both $T(u)$ and $T(v)$ consist of a single vertex and the proof trivially follows. Now, assume that the statement is true for any two vertices $u', v' \in L_i$ for $i > n$. We need to show that the statement holds for any two vertices $u, v \in L_n$ such that $u_x < v_x$.

First observe that if we have two descendants u' and v' from u and v respectively such that $u', v' \in L_{n'}$ for some $n' > n$, then it holds that $u'_x < v'_x$. Indeed, this follows from the fact that when we embed T in the natural way with edges drawn as straight segments, the result is a tree with no crossings. Thus, if $v'_x < u'_x$ happened for some descendants, then the two paths in T from u to u' and from v to v' would either cross or form a cycle. Any of those two situations would contradict with the fact that T is a weak CDR.

Back to our original proof, consider the case in which neither u nor v are inner leaves. By the above argument we have that the x-coordinate of any child $u' \in L_{n+1}$ of u must be smaller than any child $v' \in L_{n+1}$ of v. By induction, this implies that $\max(T(u')) < \min(T(v'))$ and thus $\max(T(u)) < \min(T(v))$.

The cases in which u or v are inner leaves are similar: if u is an inner leaf, we have $\max(T(u)) = M(u) = \frac{\max(T(u_1))+\min(T(u_2))}{2}$, where $u_1 = (u_x, u_y + 1) \in L_{n+1}$ and $u_2 = (u_x + 1, u_y) \in L_{n+1}$. By induction on u_1 and u_2 we have $\max(T(u_1)) < \min(T(u_2))$ and $\max(T(u)) < \min(T(u_2))$, thus we need to compare $\min(T(u_2))$ with any children of v. If v is also an inner leaf, we can do a similar argument and have that $\max(T(v_1)) < \min(T(v))$ where $v_1 = (v_x, v_y + 1)$.

In general, given u, let $u' \in L_{n+1}$ be the child of u with the largest x-coordinate (or $u' = u_2$ if u is an inner leaf). Similarly, we define v' as the child of v with the smallest x-coordinate (or $v' = v_1$ if v is an inner leaf). Again, by planarity of the natural embedding, we have that $u'_x \leq v'_x$ if at least one of u, v is an inner leaf. In either case, we can use induction and get that $\max(T(u')) \leq \min(T(v'))$ which implies $\max(T(u)) < \max(T(u')) \leq \min(T(v')) \leq \min(T(v))$ (if u is an inner leaf) or $\max(T(u)) \leq \max(T(u')) \leq \min(T(v')) < \min(T(v))$ (if v is an inner leaf) completing the proof. ◀

For any subtree T' of T, its *depth* is the longest possible length of a path from its root to any of its leaves. Any split vertex $s \in \mathcal{S}$ has two branching edges e_1 and e_2, each defining a subtree. The subtree of higher depth is the *preferred* subtree of s (in case of tie, we choose the tree emanating from $(s_x + 1, s_y)$). For any point $p \in \mathcal{G}_N^+$ we define a walk from p to some leaf of T. If $p \in L_n$ has degree two, we follow the single edge to L_{n+1}. If $p \in \mathcal{S}$, we follow the edge to the preferred subtree. This process is continued until we reach a leaf $\gamma(p)$.

With this virtual walk we can define the function M to all points $p \in \mathcal{G}_N^+$ (not only leaves) of the domain as follows. If p is neither a split nor a leaf, we define $M(p)$ as $M(p) = M(\gamma(p))$. For a split vertex s, let s' be the child of s that is not on the preferred subtree of s. Then, we define $M(s)$ as $M(s) = M(\gamma(s'))$.

Intuitively speaking, from any vertex we always follow its only edge away from the root (if it has degree 2) or the preferred edge (if it has degree 3) until we reach a leaf. The only exception is if we start on a split vertex, in which case we do not follow the preferred edge at the first step. This exception is needed to make sure that the end points of the walk starting from split vertices are distinct.

▶ **Lemma 6.** *For any split vertex $s \in \mathcal{S}$, there exists a unique leaf $\ell \in \mathcal{D} \cup L_N$ such that $M(s) = M(\ell)$. And for any leaf $\ell \in \mathcal{D} \cup L_N \setminus \{(N, 0)\}$, there exists a unique split vertex $s \in \mathcal{S}$ such that $M(s) = M(\ell)$.*

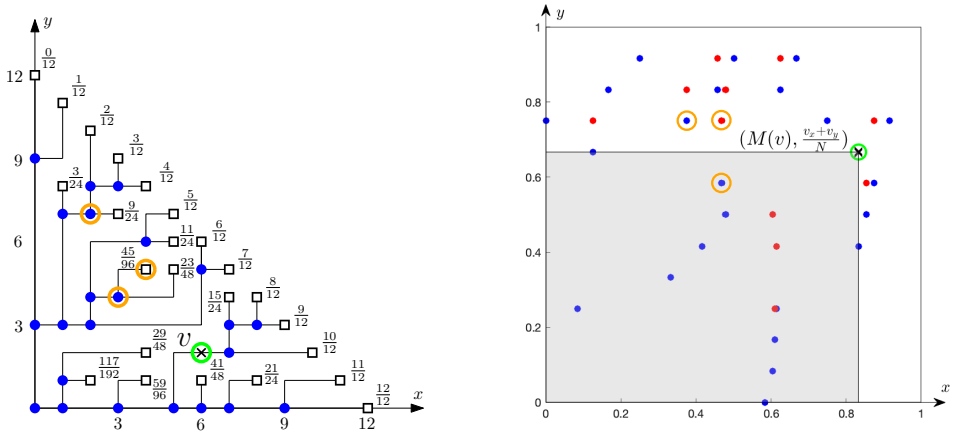


Figure 3 (left) A tree of a weak CDR and the value of the auxiliary function M applied to all leaves of the tree. (right) The tree transformed into blue and red point sets. Two vertices of the same layer are mapped to points with the same y-coordinate and an inner leaf and its corresponding split vertex are mapped to points with the same x-coordinate (see the highlighted orange circles). (For Theorem 8) The x-coordinate of $v = (6, 2)$ (green circle) can be bounded in terms of the difference between blue and red point in the axis-aligned rectangle with corners $(0, 0)$ and $\pi(v) = (M(v), \frac{v_x+v_y}{N}) = (\frac{10}{12}, \frac{8}{12})$. The rectangle contains 11 blue points and 3 red ones.

Proof. By definition of the auxiliary function, two leaves do not have the same mapping. Thus, it remains to show that the walk of two different split vertices cannot end at the same leaf. Imagine doing the walk backwards: start at any leaf, walk towards the origin and stop as soon as you reach a split vertex by traversing its non-preferred edge. Since each split vertex has exactly two children, it follows that exactly one leaf will stop at each split vertex. The exceptional case is the leaf $(N, 0)$, from which walking backwards to the origin is a horizontal path and the path does not contain any non-preferred edge. That is, in the inverse walk we follow preferred edges until we reach a non-preferred edge. This is equivalent to starting at a split vertex and follow the non-preferred edge once and continue with the preferred edges, which is the exact definition of our auxiliary function. ◀

2.2 Transforming the tree into a pointset

With the auxiliary function M we can define the mapping between a weak CDR into a bicolored pointset in the unit square. For any vertex $v = (v_x, v_y) \in \mathcal{G}_N^+$ we define its *transformation* as $\pi(v) = (M(v), \frac{v_x+v_y}{N})$. Given any weak CDR, we look at the tree T it defines in $\mathcal{G}_N^+$. Each vertex $v \in \mathcal{D}$ creates a red point $\pi(v)$ and each split vertex $w \in \mathcal{S}$ creates a blue point $\pi(w)$ (note that we do not transform the boundary leaves in L_N into points). We define the *mapping* of T as the union of the sets $R = \{\pi(v) \colon v \in \mathcal{D}\}$ and $B = \{\pi(v) \colon v \in \mathcal{S}\}$ (see Figure 3, right). Note that the two sets depend on the tree T (and thus $R = R(T)$ and $B = B(T)$). From now on we assume that T is fixed, and thus we simplify the notation for ease of reading. For any set P of points in the unit square and $x, y \in [0, 1]$ let $P[x, y]$ be the number of points in $P \cap [0, x] \times [0, y]$.

▶ **Lemma 7.** *For any weak CDR T in $\mathcal{G}_N^+ \subset \mathbb{Z}^2$ and $n < N$, the red and blue points on the horizontal line $y = n/N$ alternate in color starting and ending with a blue point. In particular, we have $B[1, n/N] - R[1, n/N] = n + 1$.*

Proof. For the first statement we observe that only points that lie in L_n will have y-coordinates equal to n/N. Moreover, since L_{n+1} has one more vertex than L_n, each diagonal must have exactly one more split vertex than inner leaves. Indeed, Chun et al. showed that in proper CDRs each diagonal has exactly one split vertex (and of course, zero inner leaves).

Now we need to show that split vertices and inner leaves appear alternatingly on the diagonal line. Consider two consecutive split vertices $u, v \in L_n$ such that $u_x < v_x$. By definition of split, the edges $e_u = (u_x, u_y)(u_x + 1, u_y)$ and $e_v = (v_x, v_y)(v_x, v_y + 1)$ are all in T. Observe that there are $v_x - u_x - 1$ vertices in L_n and $v_x - u_x - 2$ vertices in L_{n+1} between e_u and e_v. Since two different vertices of L_n cannot connect to the same vertex of L_{n+1}, one of them will not reach L_{n+1}. That vertex will be an inner leaf and will be between u and v as claimed.

That is, the blue pointset has one more point than the red pointset in each horizontal line $y = i/N$. Summing up the differences from $i = 0$ to n, we get that in total there are $n + 1$ additional blue points $p = (x, y)$ with $y \leq n/N$. ◀

With the above observations we can now state the main relationship between the weak CDR and its mapped pointset. For any vertex $v \in L_n$, its path to the origin splits the tree into two portions. Consider the portion of the tree up to L_n that is above the path from v to the origin. In L_0, the subtree contains a single vertex (the root) whereas at the diagonal L_n contains $v_x + 1$ vertices. Since the number of leaves grows with split vertices and shrinks with inner leaves, this means that in the portion of the tree that we are looking at, the difference between split vertices and inner leaves must be v_x, see Figure 3. Note that if the two children of a split vertex (e.g., $(5, 0)$ in Figure 3) are not in the same portion, the number of leaves does not grow with that split vertex. However, these split vertices may be still contained in the rectangle that we consider in the mapped pointset. This is the reason why we do not have an equality in Theorem 8.

▶ **Theorem 8.** *For any vertex $v \in \mathcal{G}_N^+$ it holds that $B[M(v), \frac{v_x+v_y}{N}] - R[M(v), \frac{v_x+v_y}{N}] - 2 \leq v_x \leq B[M(v), \frac{v_x+v_y}{N}] - R[M(v), \frac{v_x+v_y}{N}]$.*

Proof. We split the proof into two auxiliary lemmas.

▶ **Lemma 9.** *Let $v \in L_n$ be a split vertex such that $v_x < n$. If $M(v) < M(\gamma(v))$ the rectangle $[M(v), M(\gamma(v))] \times \left[0, \frac{n-1}{N}\right]$ contains exactly one point, which is blue and has $M(\gamma(v))$ as x-coordinate. If $M(\gamma(v)) < M(v)$ the rectangle $[M(\gamma(v)), M(v)] \times \left[0, \frac{n-1}{N}\right]$ contains exactly one point, which is blue and has $M(\gamma(v))$ as x-coordinate. When $v = (n, 0) \in L_n$ the rectangle $[M(v), M(\gamma(v))] \times \left[0, \frac{n-1}{N}\right]$ is empty.*

Proof. We first consider the case of $M(v) < M(\gamma(v))$. When we keep following from v to the preferred subtree, we end up in a leaf, called ℓ. By definition of M we have $M(\ell) = M(\gamma(v))$. Since $v_x < n$ we have $M(\gamma(v)) \neq 1$. By Lemma 6 there is a unique split vertex $s \in \mathcal{S}$ such that $M(s) = M(\ell)$. This split vertex is below layer L_n (indeed, we reach L_n from ℓ by following only preferred edges and the inverse walk has to stop when we traverse a non-preferred edge of s) and therefore s is transformed to a blue point in the rectangle. Now let s' be a split vertex which is mapped to a blue point in the rectangle. We will show that $s' = s$. Let ℓ' be the unique leaf such that $M(\ell') = M(s')$. Consider first the case in which ℓ' is below layer L_n (that is, $\ell'_x + \ell'_y < n$). Then let v' be the vertex on $dig(o, v)$ and $L_{\ell'_x+\ell'_y}$. If $\ell'_x < v'_x$ (resp. $v'_x < \ell'_x$) then Lemma 5 implies that $M(\ell') < min(T_{v'}) \leq M(v)$ (resp. $M(\gamma(v)) \leq max(T_{v'}) < M(\ell')$). This would be a contradiction to s' being mapped to a blue point in the rectangle.

It remains to consider the case in which ℓ' is above layer L_n. Define ℓ'' to be the vertex on $dig(o, \ell')$ and L_n. Lemma 5 implies that $\ell'' = v$ (otherwise we have either $M(\ell') < M(v)$ or $M(\gamma(v)) < M(\ell')$ which would again be a contradiction). Recall that there is only one split vertex whose walk to its corresponding leaf through preferred subtrees passes through v. Hence $s' = s$ and there is exactly one blue point in the rectangle.

We now show that there cannot be any red point either. Indeed, recall that for every red point there is a blue point with the same x-coordinate and smaller y-coordinate because for each inner leaf ℓ there is a unique split vertex s defined by the walk from s to ℓ such that $M(\ell) = M(s)$. From the previous argument, we know that s with $M(s) = M(\gamma(v))$ is mapped to the only one blue point in the rectangle and its corresponding leaf ℓ defined by the walk is above L_n. Hence, even if ℓ is an inner leaf, the mapped red point is not in the rectangle. Moreover, there cannot be any other red point in the rectangle (since it would imply that the corresponding blue point would also be in and we already ruled out this case).

In the same way we can also prove that if $M(\gamma(v)) < M(v)$ the rectangle $[M(\gamma(v)), M(v)] \times \left[0, \frac{n-1}{N}\right]$ contains exactly one point, which is blue and has $M(\gamma(v))$ as x-coordinate. If $v_x = n$ then ℓ as defined above is the leaf $(N, 0)$ and $M(\ell) = 1$. Lemma 6 implies that there is no split vertex s with $M(s) = 1$. ◀

▶ **Lemma 10.** *For any vertex $v \in \mathcal{G}_N^+$ it holds that*

$$v_x - B\left[M(v), \frac{v_x + v_y - 1}{N}\right] + R\left[M(v), \frac{v_x + v_y - 1}{N}\right] + 1 \in \{0, 1\}. \tag{1}$$

Proof. We first prove by induction over n that $\forall n \in \{0, 1, ..., N\}$ the following statement holds.

$$\begin{aligned} & \{M(\gamma(p)) | p \in L_n\} \\ = \; & \left\{x \in [0,1] : \left|B \cap \{x\} \times \left[0, \frac{n-1}{N}\right]\right| - \left|R \cap \{x\} \times \left[0, \frac{n-1}{N}\right]\right| = 1\right\} \cup \{1\}. \end{aligned} \tag{2}$$

The quantity $|B \cap \{x\} \times [0, \frac{n-1}{N}]| - |R \cap \{x\} \times [0, \frac{n-1}{N}]|$ counts the difference between the number of blue points and red points on the vertical segment with x-coordinate x and length $\frac{n-1}{N}$. Because of Lemma 6 we know that each split vertex shares the same value with a leaf in the auxiliary function M. If the leaf is an inner leaf, both blue (split) and red (inner) points lie on the same unit segment $\{x\} \times [0, 1]$. Otherwise, there is only one blue point on $\{x\} \times [0, 1]$ because $M(p)$ for $p \in L_N$ are all different. Hence the quantity $|B \cap \{x\} \times [0, \frac{n-1}{N}]| - |R \cap \{x\} \times [0, \frac{n-1}{N}]|$ can either be 0 or 1.

The base case $n = 0$ trivially holds. We have $\{M(\gamma(p)) | p \in L_0\} = \{1\}$ and

$$B \cap \{x\} \times \left[0, \frac{n-1}{N}\right] = R \cap \{x\} \times \left[0, \frac{n-1}{N}\right] = \emptyset$$

We assume that Section 2.2 holds for layer L_n and we prove that it also holds for L_{n+1}. We distinguish 3 cases for any vertex q in layer L_n.

- If q has degree 2 then q and its child $r \in L_{n+1}$ are mapped by $M \circ \gamma$ to the same value. Moreover q does not create any vertex in the set B nor R.
- If q is an inner leaf, then the value $M(\gamma(q))$ will not appear in $\{M(\gamma(p)) | p \in L_{n+1}\}$ any more. The value $M(\gamma(q))$ also disappears in

  $$\left\{x \in [0,1] : \left|B \cap \{x\} \times \left[0, \frac{n}{N}\right]\right| - \left|R \cap \{x\} \times \left[0, \frac{n}{N}\right]\right| = 1\right\} \cup \{1\}.$$

  because q created a red point in R with the coordinates $(M(\gamma(q)), \frac{n}{N}) = (M(q), \frac{n}{N})$.

- If q is a split vertex, then the value $M(\gamma(q))$ will stay in $\{M(\gamma(p))|p \in L_{n+1}\}$. Moreover $\{M(\gamma(p))|p \in L_{n+1}\}$ contains the additional value $M(q)$. The value $M(q)$ also appears in

$$\left\{x \in [0,1] : \left|B \cap \{x\} \times \left[0, \frac{n}{N}\right]\right| - \left|R \cap \{x\} \times \left[0, \frac{n}{N}\right]\right| = 1\right\} \cup \{1\}$$

because q creates a blue point in B with the coordinates $(M(q), \frac{n}{N})$.

Hence Section 2.2 holds.

Let v be a vertex in layer L_n, i.e. $n = v_x + v_y$. By Lemma 5 we know that a vertex $u \in L_n$ with $u_x < v_x$ satisfies $M(\gamma(u)) < M(\gamma(v))$. By Lemma 5 we also know that a vertex $w \in L_n$ with $v_x < w_x$ satisfies $M(\gamma(v)) < M(\gamma(w))$. Hence the number of vertices in layer L_n with smaller x-coordinate than that of v is exactly the number of vertices which are mapped by $M \circ \gamma$ to a smaller value than that of v. If $v_x < n$:

$$\begin{aligned} v_x &= |\{u \in L_n | u_x < v_x\}| \overset{\text{Lemma 5}}{=} |\{u \in L_n | M(\gamma(u)) < M(\gamma(v))\}| \\ &= |\{u \in L_n | M(\gamma(u)) \le M(\gamma(v))\}| - 1 \\ &\overset{(2)}{=} B\left[M(\gamma(v)), \frac{n-1}{N}\right] - R\left[M(\gamma(v)), \frac{n-1}{N}\right] - 1 \\ &\overset{\text{Lemma 9}}{=} \begin{cases} B[M(v), \frac{n-1}{N}] - R\left[M(v), \frac{n-1}{N}\right] - 1 & \text{if } M(\gamma(v)) \le M(v) \\ B[M(v), \frac{n-1}{N}] - R\left[M(v), \frac{n-1}{N}\right] & \text{if } M(v) < M(\gamma(v)) \end{cases} \end{aligned}$$

If $v_x = n$ then:

$$\begin{aligned} v_x &= |\{u \in L_n | M(\gamma(u)) \le M(\gamma(v))\}| - 1 \overset{(2)}{=} B\left[M(\gamma(v)), \frac{n-1}{N}\right] - R\left[M(\gamma(v)), \frac{n-1}{N}\right] \\ &\overset{\text{Lemma 9}}{=} B\left[M(v), \frac{n-1}{N}\right] - R\left[M(v), \frac{n-1}{N}\right] \end{aligned}$$

◀

By Lemma 7, the red and blue points on the line $y = v_x + v_y$ alternate in color starting and ending with a blue point. Hence, any interval $[0, x]$ on the line $y = v_x + v_y$ contains at most one more blue points. Therefore, $B[M(v), \frac{v_x+v_y}{N}] - R[M(v), \frac{v_x+v_y}{N}] - (B[M(v), \frac{v_x+v_y-1}{N}] - R[M(v), \frac{v_x+v_y-1}{N}])$ is at most one. Lemmas 10 and 7 directly imply Theorem 8. ◀

3 Bichromatic discrepancy

Let R and B be a set of red and blue points in the unit square, respectively. Let $r = |R|$ and $b = |B|$, and further assume that $b > r$. Let $m = b - r$ (which is positive since $b > r$). For any set P of points in the unit square and $x, y \in [0, 1]$ let $P[x, y]$ be the number of points in $P \cap [0, x] \times [0, y]$.

For any two sets R and B and real numbers $x, y \le 1$ we define the discrepancy of R and B at (x, y) as

$$D_{R,B}(x, y) = (b - r)xy - (B[x, y] - R[x, y]). \tag{3}$$

The *discrepancy* of R and B is simply defined as $D^*_{R,B} = \max_{(x,y)\in[0,1]^2} |D_{R,B}(x, y)|$ (i.e., the highest discrepancy we can achieve among all possible rectangles).

▶ **Theorem 2** (Two colors discrepancy). *For any set R and B of points such that $|B| > |R|$ it holds that $D^*_{R,B} = \Omega\left(\frac{(|B|-|R|)\cdot\log(|B|+|R|)}{|B|+|R|}\right)$.*

Note that if we set $R = \emptyset$ we get the classic two dimensional discrepancy result for which there are several proofs (see [12] for a detailed survey). In order to extend the bound for the case of $R \neq \emptyset$, we make minor changes to Schmidt's proof [13]. We start by using an auxiliary function G (defined below) and combining it with the trivial inequality

$$\int_{(x,y)\in[0,1]^2} D_{R,B}(x,y)G(x,y)dxdy \leq \max_{(x,y)\in[0,1]^2} |D_{R,B}(x,y)| \int_{(x,y)\in[0,1]^2} |G(x,y)|dxdy$$

to obtain $D^*_{R,B} = \max_{(x,y)\in[0,1]^2} |D_{R,B}(x,y)| \geq \frac{\int D_{R,B}G}{\int |G|}$.

Note that for simplicity in the notation we removed the integration limits. Our definition of G is identical to the one used by Schimdt: Let $m = \lceil \log_2(b+r) \rceil + 1$ and observe that, by definition of m we have $2(b+r) \leq 2^m \leq 4(b+r)$. For any $j \in \{0,\ldots,m\}$ we define function $f_j : [0,1]^2 \to \{-1,0,1\}$ as follows: subdivide the unit square with 2^j equally spaced vertical lines and 2^{m-j} horizontal lines.

For any value of j we subdivide the unit square into rectangles of area 2^{-m} (larger values of j will result in thinner but wider rectangles). Let A be a rectangle of subdivision associated to f_j. We define f_j within the rectangle to be 0 if A contains any point of $R \cup B$. If A does not have neither red nor blue points, we further subdivide it into four congruent quadrants. The function value of f_j is equal to 1 in the upper right and lower left quadrants, and -1 in upper left and lower right quadrants (see a visual representation of f_j in [12], page 173).

Then, we define G as $G = (1+cf_0)(1+cf_1)\ldots(1+cf_m) - 1$, where $c > 0$ is a small constant (whose value will be chosen afterwards). Note that G can also be expressed as $G = G_1 + \ldots G_m$, where $G_k = c^k \sum_{0\leq j_1 \leq \ldots \leq j_m \leq m} f_{j_1} f_{j_2} \cdots f_{j_k}$.

Schmidt showed that $\int |G| \leq 2$ (regardless of the value of m). Thus, we now focus in giving an upper bound for $\int D_{R,B}G$.

▶ **Lemma 11.** *There exists a constant c_1 such that $\int D_{R,B}G_1 \geq cc_1 \frac{b-r}{b+r} \log(b+r)$.*

Proof. By definition of G_1 we have $\int D_{R,B}G_1 = c\sum_{j=0}^{m} \int D_{R,B}f_j$. Thus, it suffices to show that for any value of j it holds that $\int D_{R,B}f_j \geq c' \frac{b-r}{b+r}$ (for some other constant $c' > 0$).

Recall that, when defining f_j, we subdivided the unit square into at least $2(b+r)$ rectangles. For the rectangles that contain at least one point of $R \cup B$, f_j is set to zero, and thus they do not contribute to the integral. Since we have $b+r$ many points, we know that there must exist at least $b+r$ rectangles that do not contain any point of R or B. Let A be any such rectangle, and let $A_{SW}, A_{NW}, A_{SE}, A_{NE}$ be the four subquadrants of A (where the subindex refers to the cardinal position of the quadrant). Recall that f_j is equal to 1 for any point of $A_{SW} \cup A_{NE}$ and -1 for points of $A_{SE} \cup A_{NW}$.

Let w and h be vectors defined by the horizontal and vertical sides of A_{SW}, respectively. Observe that their lengths are 2^{-j-1} and 2^{j-m-1}, respectively. Then, we have

$$\begin{aligned}
&\int_A f_j D_{R,B} \\
=\ &\int_{A_{SW}} D_{R,B} - \int_{A_{NW}} D_{R,B} + \int_{A_{NE}} D_{R,B} - \int_{A_{SE}} D_{R,B} \\
=\ &\int_{A_{SW}} [D_{R,B}(x,y) + D_{R,B}(x+\mathsf{w}, y+\mathsf{h}) - D_{R,B}(x, y+\mathsf{h}) - D_{R,B}(x+\mathsf{w}, y)]dxdy.
\end{aligned}$$

If we apply the definition of $D_{R,B}$ (Eq. (3)) to the four terms inside the integral we get

$$\begin{aligned}\int_A f_j D_{R,B} &= \int_{A_{SW}} ((b-r)[xy+(x+\mathsf{w})(y+\mathsf{h})-x(y+\mathsf{h})-(x+\mathsf{w})y])dxdy \\ &- \int_{A_{SW}} (B[x,y]+B[x+\mathsf{w},y+\mathsf{h}]-B[x,y+\mathsf{h}]-B[x+\mathsf{w},y])dxdy \\ &+ \int_{A_{SW}} (R[x,y]+R[x+\mathsf{w},y+\mathsf{h}]-R[x,y+\mathsf{h}]-R[x+\mathsf{w},y])dxdy.\end{aligned}$$

Observe that we are integrating twice positively and twice negatively over almost identical functions. In fact, the terms of the first integral all cancel out except along the rectangle $[x,x+\mathsf{w})\times[y,y+\mathsf{h})$. Similarly, when we look at the second and third terms, the contribution of any point in $R\cup B$ is cancelled out unless it is in the rectangle $[x,x+\mathsf{w})\times[y,y+\mathsf{h})$. However, by definition of A there are no such points. Thus, we obtain

$$\int_A f_j D_{R,B} = \int_{A_{SW}} (b-r)\mathsf{w}\cdot\mathsf{h}\,dxdy = \int_{A_{SW}} (b-r)2^{-m-2}dxdy = (b-r)2^{-2m-4}$$

That is, when we integrate $f_j D_{R,B}$ over a rectangle A containing no point of $R\cup B$, the result is $(b-r)2^{-2m-4}$. We know that there are at least $b+r$ rectangles not containing points of $R\cup B$, thus their contribution is at least $\frac{(b+r)(b-r)}{2^{2m+4}} = \frac{(b+r)}{2^m}\frac{(b-r)}{16\cdot 2^m} \geq \frac{1}{4}\frac{(b-r)}{16\cdot 4(b+r)} = \Omega(\frac{b-r}{b+r})$. ◀

▶ **Lemma 12.** *There exists a constant c_2 such that $\sum_{k=2}^{m}\int D_{R,B}G_k \leq c^2 c_2 \frac{b-r}{b+r}\log(b+r)$.*

Proof. Recall that $G_k = c^k \sum_{0\leq j_1<j_2<\ldots<j_k\leq m} f_{j_1}\ldots f_{j_k}$. Fix any valid set of indices and consider the value of $\int f_{j_1}\ldots f_{j_k}D_{R,B}$.

As shown in [12], function $f_{j_1}\ldots f_{j_k}$ is largely defined by f_{j_1} and f_{j_k}. Indeed, if we overlay the rectangular partition defined by functions $f_{j_1},\ldots,f_{j_k}$ we obtain a grid of rectangles whose width is 2^{-j_k} and height $2^{-(m-j_1)}$. In each of these rectangles, the function is zero (if any of the rectangles associated to the f_{j_i} functions contains a point of $R\cup B$), or is further subdivided into four equal sized quadrants and in each one it is $+1$ or -1 alternatively.

Let A be one of the rectangles of the refined grid. As shown in Lemma 11, we have that

$$\int_A f_{j_1}\ldots f_{j_k}D_{R,B} = \tau(b-r)2^{-2(m+j_k-j_1)-4},$$

where $\tau\in\{-1,1\}$. This extra term appears because the product of the different functions involved can change the sign of each of the four quadrants. In any case, we have $\int_A f_{j_1}\ldots f_{j_k}D_{R,B} \leq (b-r)2^{-2(m+g)-4}$ where $g=j_k-j_1$.

By the way the grid is constructed, there are $2^{m-j_1}\times 2^{j_k} = 2^{m+g}$ many rectangles, and thus we conclude that $\int f_{j_1}\ldots f_{j_k}D_{R,B} \leq (b-r)2^{-m-g-4}$. In order to obtain a bound $\int D_{R,B}G_k$ we sum over all possible indices.

$$\int D_{R,B}G_k = c^k \sum_{0\leq j_1<\ldots<j_k\leq m}\int f_{j_1}\ldots f_{j_k}D_{R,B} \leq \frac{c^k(b-r)}{2^{m+4}}\sum_{0\leq j_1<\ldots<j_k\leq m} 2^{-(j_k-j_1)}.$$

Note that in the sum, the indices $j_2,\ldots j_{k-1}$ do not matter. Thus, we group the terms by the gap between the indices j_1 and j_k (say, if $j_1=3$ and $j_k=7$ the gap is 4). Note that the minimum gap is at least $k-1$ (since otherwise we do not have enough space to choose the $k-2$ indices in between) and at most m. Once we have a gap of g there are $m-g$ options for index j_1.

$$\begin{aligned}\int D_{R,B}G_k &\leq \frac{c^k(b-r)}{2^{m+4}} \sum_{g=k-1}^{m} \sum_{j_1=0}^{m-g} \sum_{j_1<j_2<\ldots<j_{k-1}<j_1+g} 2^{-g} \\ &= \frac{c^k(b-r)}{2^{m+4}} \sum_{g=k-1}^{m} \sum_{j_1=0}^{m-g} \binom{g-1}{k-2} 2^{-g} \leq \frac{c^k(b-r)m}{2^{m+4}} \sum_{g=k-1}^{m} \binom{g-1}{k-2} 2^{-g}.\end{aligned}$$

In order to upper bound the sum over all G_k, we first reorder the summation order.

$$\begin{aligned}\sum_{k=2}^{m} \int D_{R,B}G_k &\leq \sum_{k=2}^{m} \frac{c^k(b-r)m}{2^{m+4}} \sum_{g=k-1}^{m} \binom{g-1}{k-2} 2^{-g} \\ &= \frac{(b-r)m}{2^{m+4}} \sum_{g=1}^{m} 2^{-g} c^2 \sum_{k=2}^{g+1} \binom{g-1}{k-2} c^{k-2} \\ &= \frac{(b-r)m}{2^{m+4}} \sum_{g=1}^{m} 2^{-g} c^2 (1+c)^{g-1} \\ &= \frac{(b-r)mc^2}{2^{m+5}} \sum_{g=1}^{m} \left(\frac{1+c}{2}\right)^{g-1}.\end{aligned}$$

The sum contains the first terms of the geometric sum $\sum_{g=1}^{\infty} \left(\frac{1+c}{2}\right)^{g-1} \leq \frac{2}{1-c}$ (for any $c < 1$). In particular, if we set $c \leq 1/2$ we can upper bound the partial sum by 4. Recall that $m = \Theta(\log(b+r))$ and $2^m = \Theta(b+r)$. Thus, the lemma is proven. ◀

▶ **Corollary 13.** *There exists a constant $\kappa > 0$ such that $\int D_{R,B}G \geq \kappa \left(\frac{(b-r)\cdot\log(b+r)}{b+r}\right)$.*

Proof. Apply the inequality $\int(A+B) \geq \int A - \int |B|$ and Lemmas 11 and 12 to obtain:

$$\int D_{R,B}G = \int D_{R,B}G_1 + \sum_{k=2}^{m} \int D_{R,B}G_k \geq c(c_1 - cc_2) \left(\frac{(b-r)\cdot\log(b+r)}{b+r}\right)$$

Note that Lemmas 11 and 12 holds for any value of c such that $c \in (0, 1/2]$. By choosing a sufficiently small value of c (say, $c = \min\{\frac{1}{2}, \frac{c_1}{2c_2}\}$) we obtain $\int D_{R,B}G \geq \frac{cc_1}{2} \left(\frac{(b-r)\cdot\log(b+r)}{b+r}\right)$. ◀

This completes the proof of Theorem 2.

When $R = \emptyset$, it would be expected that we need to distribute the blue points uniformly in the unit square to have a low discrepancy. Indeed, it is also held for the red points. The following theorem implies that even if there are many red points, but the red points are concentrated in the lower half of the unit square, the discrepancy cannot be reduced. For simplicity, we only show a special case of how the discrepancy is depended on the points in $[0,1] \times [1/2,1]$, which is good enough for our purpose in Section 5. Notice that the same argument can be applied in a more general case.

▶ **Theorem 14.** *For any set R and B of points in the unit square such that $|R| = r$, $|B| = b$ and $b > r$. Let r_2 and b_2 be the number of red and blue points in $[0,1] \times [1/2,1]$ respectively. It holds that*

$$D^*_{R,B} = \Omega\left(\frac{(b_2 - r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right).$$

Proof. Let R_2 and B_2 be the set of red and blue points in $[0,1] \times [1/2,1]$ respectively. Consider the upper half of the unit square $[0,1] \times [1/2,1]$ and rescale the vertical length to be 1. By Theorem 2, there exists a point $(x,2y)$ such that $|D_{R_2,B_2}(x,2y)| = |2xy(b_2 - r_2) - (B_2[x,2y] - R_2[x,2y])| \geq 2c\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right)$ for some constant c.

Then, we map the point $(x,2y)$ back to a point $(x,1/2+y)$ in the original unit square. We will show that either $D_{R,B}(x,1/2+y)$ or $D_{R,B}(x,1/2-\epsilon)$ would give us the desired lower bound, where ϵ is an arbitrarily small constant such that rectangle $[0,1] \times [0,1/2-\epsilon]$ only contains $B \setminus B_2$ and $R \setminus R_2$.

If $|D_{R,B}(x,1/2+y)| \geq c\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right)$, we are done.

If $|(b-r)/2 - (b_2-r_2)| \geq c/4\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right)$, the proof is also done. Because

$$\begin{aligned}
& |D_{R,B}(1,1/2-\epsilon)| \\
\overset{(3)}{=} \; & |(b-r)(1/2-\epsilon) - (B[1,1/2-\epsilon] - R[1,1/2-\epsilon])| \\
= \; & |(b-r)(1/2-\epsilon) - (b-r-(b_2-r_2))| \\
= \; & |(b-r)/2 - (b_2-r_2) - (b-r)\epsilon| \\
> \; & c/8\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right).
\end{aligned}$$

Suppose that the two cases do not hold, we have $|D_{R,B}(x,1/2+y)| < c\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right)$ and $|(b-r)/2 - (b_2-r_2)| < c/4\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right)$. Let $R_1 = R \setminus R_2$ and $B_1 = B \setminus B_2$, which are inside the rectangle $[0,1] \times [0,1/2-\epsilon]$. Consider

$$\begin{aligned}
& D_{R,B}(x,1/2+y) \\
= \; & (b-r)x(1/2+y) - (B[x,1/2+y] - R[x,1/2+y]) \\
= \; & (b-r)x(1/2+y) - (B_2[x,1/2+y] - R_2[x,1/2+y] + B_1[x,1/2-\epsilon] - R_1[x,1/2-\epsilon]) \\
= \; & (b-r)x(1/2-\epsilon) - (B_1[x,1/2-\epsilon] - R_1[x,1/2-\epsilon]) + (b-r)x\epsilon \\
& +(b-r)xy - (B_2[x,1/2+y] - R_2[x,1/2+y]) \\
= \; & D_{R,B}(x,1/2-\epsilon) + (b-r)xy - (B_2[x,1/2+y] - R_2[x,1/2+y]) + (b-r)x\epsilon \\
> \; & D_{R,B}(x,1/2-\epsilon) + 2(b_2-r_2)xy - (B_2[x,1/2+y] - R_2[x,1/2+y]) \\
& -c/2\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right) + (b-r)x\epsilon \\
= \; & D_{R,B}(x,1/2-\epsilon) + D_{R_2,B_2}(x,2y) - c/2\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right) + (b-r)x\epsilon
\end{aligned}$$

The first inequality is given by $b-r > 2(b_2-r_2) - c/2\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right)$. Since $|D_{R,B}(x,1/2+y)| < c\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right)$ and $|D_{R_2,B_2}(x,2y)| \geq 2c\left(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2}\right)$, we can conclude that $|D_{R,B}(x,1/2-\epsilon)| = \Omega(\frac{(b_2-r_2)\cdot\log(b_2+r_2)}{b_2+r_2})$. ◀

4 Lower bound for two dimensional weak CDRs

Before giving the proof of Theorem 3, we recall that a proof for a proper CDR (i.e., one without inner leaves) was given in [8]. Our proof follows the same spirit, so we first give an overview of their proof and describe what changes when we introduce inner leaves.

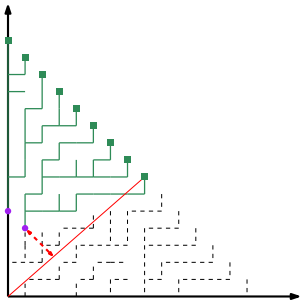


Figure 4 Illustration of why the two sets I (purple disks) and $\mathcal{L}(I)$ (green squares) should have proportional sizes. If the size of $\mathcal{L}(I)$ grows drastically (as shown in the figure), the point of the highest x-coordinate in $\mathcal{L}(I)$ must make a significant detour to pass through I, causing a large error. A similar effect happens if the size of $\mathcal{L}(I)$ is comparatively small.

▶ **Lemma 15.** *Given a CDR, a point $p = (x, y) \in L_N$, and an integer $n < N$, let $p' = (x', y') \in L_n$ be the unique point of L_n that is in $dig(o, p)$. The Hausdorff error of the CDR is at least $|x' - x \cdot \frac{n}{N}|$.*

Proof. This result was shown by Chun et al. [8] (Lemma 3.5, in Cases 1 and 2). We give the proof for completeness. Consider the L-infinity ball of radius $|x' - x \cdot \frac{n}{N}|$ centered at $p \cdot \frac{n}{N}$. By construction, this ball contains p' in its boundary. Because of the monotonicity axiom, no vertex of $dig(o, p)$ can be in the interior of the ball. In particular, when measuring the Hausdorff distance of point $p \cdot \frac{n}{N} \in \overline{op}$ we get an error of at least $|x' - x \cdot \frac{n}{N}|$. ◀

Consider any point $p \in L_N$ and virtually sweep a line of slope -1 from the origin all the way to L_N. During the sweep, the intersection between the diagonal line and either the Euclidean segment $\overline{op}$ or the digital one $dig(o, p)$ will be a point. Lemma 15 says that if we can find an instant of time for which two intersection points are at distance ∂ from each other, then the Hausdorff error of the whole CDR must be $\Omega(\partial)$ (see Figure 4).

In order to find this instant of time we see how much the subtrees grow. Consider a consecutive set of I vertices in some intermediate layer L_n. Let $\mathcal{L}(I)$ be the vertices of L_N whose digital path to the origin passes through some vertex of I. If the CDR has small error, we need $\mathcal{L}(I)$ to have roughly $\frac{N}{n}|I|$ many points. The difference between the expected number of vertices and $|\mathcal{L}(I)|$ combined with Lemma 15 will give a lower bound on the Hausdorff error.

Our proof follows the same spirit (transform the tree into a pointset, use discrepancy to find a subset with too many/too few children and use Lemma 15 to find a large error). Although all three steps follow the same spirit, they need major changes to account for the possibility of inner leaves.

The biggest change is how we map the tree. In proper CDRs each line has a unique split vertex and always extends to L_N. Thus, a region with a large number of split vertices directly implies a large error. In our setting, we could potentially have a region with many split vertices followed by a large number of inner leaves to cancel out the growth. This is why we need two major changes: first we now color the points red and blue depending on whether they are split vertices or inner leaves. We also introduce a second dimension to track when the children of a split vertex stop extending. Intuitively speaking, the x-coordinate of our mapping will be similar to the mapping done by Chun et al. [8] whereas the y-coordinate represents time. Thus, the difference in y-coordinates between red and blue points can be used to determine for how long are the two children of a split vertex alive (the longer the difference in y-coordinates, the further away that the two children extend).

We now use the mapping of Section 2 together with the two colors discrepancy (Theorems 2 and 14) to show a lower bound on the error of weak CDRs. The discrepancy result in Theorem 2 considers the points in the whole unit square. Due to some technical reasons, in Section 5 we will need a discrepancy result for the points in the *upper half* of the unit square instead (Theorem 14). The difference between the two theorems is just a constant factor and thus would have little implication. Here we use Theorem 14 and prove the result in terms of the number of inner leaves in the upper half. Specifically, we show the following result.

▶ **Theorem 3.** *For any $N \in \mathbb{N}$, any weak CDR defined on $\mathcal{G}_N^+ \subset \mathbb{Z}^2$ with κ_2 inner leaves between lines $x+y=\lceil N/2 \rceil$ and $x+y=N$ has $\Omega(\frac{N \log N}{N+\kappa_2})$ error.*

Proof. Given a weak CDR and its associated tree T, consider its transformation into the sets R and B of red and blue points defined by π. Let b_2 and r_2 be the numbers of blue and red points in the rectangle $[0,1] \times [1/2,1]$ respectively. By Lemma 7, we have $b_2 - r_2 = \lfloor N/2 \rfloor$. We apply the discrepancy result (Theorem 14) with $b_2 - r_2 = \lfloor N/2 \rfloor$ and $r_2 = \kappa_2$, and obtain that there exists $\alpha, \beta \in [0,1]$ such that $|B[\alpha,\beta] - R[\alpha,\beta] - N \cdot \alpha \cdot \beta| > c' \cdot \frac{N \cdot \log N}{N+\kappa_2}$.

We want to use Theorem 8 on the vertex of T whose image is (α, β). Naturally, such a vertex need not exist, but we will find one nearby whose associated discrepancy is also high. Let $n = \lfloor N \cdot \beta \rfloor$ and observe that $B[\alpha,\beta] = B[\alpha, \frac{n}{N}]$; indeed, by the way we transform points, their y-coordinates are of the form i/N. However, by definition of n we know that β is between n/N and $(n+1)/N$ and thus no point can lie in the horizontal strip $y \in (n/N, \beta]$ (by the same argument we also have $R[\alpha,\beta] = R[\alpha, \frac{n}{N}]$).

If we substitute β in the previous equation we get

$$\left| B\left[\alpha, \frac{n}{N}\right] - R\left[\alpha, \frac{n}{N}\right] - \alpha n \right| > c' \cdot \frac{N \log N}{N+\kappa_2} - 1 \geq c'' \cdot \frac{N \cdot \log N}{N+\kappa_2}$$

for a large enough N, $\kappa_2 \in O(N \log N)$ and for some $c'' > 0$. We get the additional 1 term because of the rounding in the definition of n.

Now we need to do a similar operation for α. Let $q_i = (i, n-i)$ be a vertex of L_n. By Lemma 5 the image of the auxiliary function $M(q_i)$ monotonically increases as i grows. Let $Q = \{q_i \colon M(q_i) \leq \alpha\}$ and $\alpha' = max_{q_i \in Q} M(q_i)$. Note that, by definition of the set Q, it trivially holds that $\alpha' \leq \alpha$.

▶ **Lemma 16.** $B[\alpha, \frac{n}{N}] - R[\alpha, \frac{n}{N}] = B[\alpha', \frac{n}{N}] - R[\alpha', \frac{n}{N}]$

Proof. The difference between the two rectangles is the rectangle Δ whose opposite corners are $(\alpha', 0)$ and $(\alpha, n/N)$, and one of the boundary $\overline{(\alpha',0)(\alpha', \frac{n}{N})}$ is open. We claim that red and blue points are paired (sharing the same x-coordinate) in Δ (and thus, for each red point that we remove we are also removing a blue one). By Lemma 6, we know that all the blue points have different x-coordinates, so do red points. Hence, if there are red and blue points on the same vertical line, they must be the only pair in that vertical line. First notice that if there is a red point in Δ, there also exists a blue point in Δ with the same x-coordinate and below the red point. By the virtual walk that we define the auxiliary function, every split vertex is closer to the origin than the corresponding leaf. Hence, after the transformation π, if there is a red point, then there must exist a blue point with the same x-coordinate (by Lemma 6) and smaller y-coordinate. Then, we will show that if there is a blue point in Δ, there also exists a red point in Δ with the same x-coordinate.

Assume, for the sake of contradiction that there exists a blue point p in Δ such that there does not exist a red point q with the same x-coordinate as p in Δ. Let s be the split vertex whose image is p. By definition of the transformation π, the x-coordinate of p is $M(s)$, which

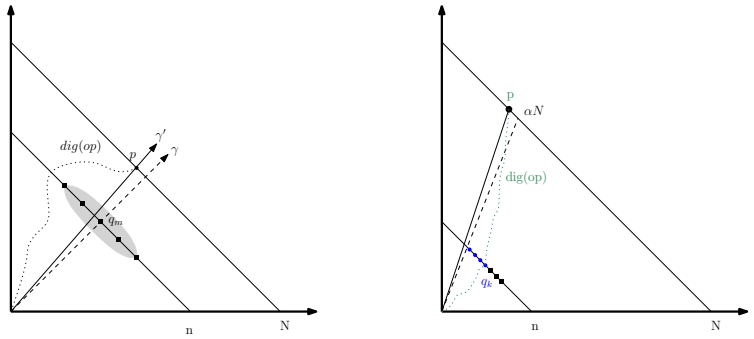


Figure 5 (left) When k is small we have $\Omega(\frac{N \log N}{N+\kappa_2})$ consecutive vertices in L_n that are not productive (shown as squares). In particular, the ray γ through the middle point must make a large detour. (right) When k is large, there is a digital path through q_k with a big detour.

is between α' and α. We apply Lemma 6 to find the unique leaf ℓ such that $M(s) = M(\ell)$. Since $\pi(\ell) \notin \Delta$, we have that $\ell_x + \ell_y > n$. Let m be the unique vertex of L_n that is in the path from s to ℓ. It follows that $\pi(m) = (M(\ell), \frac{n}{N}) \in \Delta$. This gives a contradiction with the definition of α', and thus implies that if there exists a blue point in Δ, then there also exists a red point in Δ with the same x-coordinate. ◀

Thus, given a pair (α, β) whose associated rectangle has high discrepancy, we have *snapped* it to the pair $(\alpha', \frac{n}{N})$ that defines another rectangle with high discrepancy. More importantly, by definition of Q, we know that $\pi(q_{|Q|-1}) = (\alpha', \frac{n}{N})$. Note that $q_{|Q|-1}$ need not be a split vertex or an inner leaf (and thus, $(\alpha', \frac{n}{N})$ may not be a point of $R \cup B$).

Let $b' = B[\alpha', \frac{n}{N}]$ and $r' = R[\alpha', \frac{n}{N}]$. If we apply Theorem 8 to point $q_{|Q|-1}$ we get that $b' - r' - 2 \leq |Q| - 1 \leq b' - r'$. This set Q is the one that makes the role of I in the proof overview: we know that vertices of Q are the ones that extend to cover all the vertices of L_N whose image is α' or less. As such, we would expect $|Q|$ to contain roughly $n\alpha'$ elements. However, the discrepancy result tells us that the size of Q is $c'' \frac{N \log N}{N+\kappa_2}$ units away from that value. We say that p is *productive* if some point of $T(p)$ is in L_N (this is equivalent to the fact that p can be extended to reach the boundary). Let $k \leq b' - r' - 2$ be the biggest integer such that q_k is productive. Note that k is well defined because q_0 is always productive ($(0, n)$ always extends to $(0, N)$). The proof now considers a few cases depending on whether k is small or large (specifically, we say that k is *small* if $|Q| - 1 - k \geq (b' - r' - 2) - k > \frac{c''}{2} \cdot \frac{N \log N}{N+\kappa_2}$, *large* otherwise) and if Q contains too few or too many points.

k is small. Recall that we looked for the largest possible k (such that q_k is productive). Thus, if k is small, we have many points in layer L_n that are consecutive and not productive. In particular, none of the vertices in $q_{b'-r'-\lfloor \frac{c''}{2} \cdot \frac{N \log N}{N+\kappa_2} \rfloor}, \ldots, q_{b'-r'-2}$ are productive. Let $q_m = q_{b'-r'-\lfloor \frac{c''}{4} \cdot \frac{N \log N}{N+\kappa_2} \rfloor}$ (note that this point is surrounded by non-productive points in both sides along L_n).

Shoot a ray γ from o towards q_m. Let p be the vertex on L_N that is closest to γ. Observe that the $||\cdot||_\infty$ distance between γ and p is at most $1/2$. Let γ' be the ray shooting from o towards p. Similarly, the $||\cdot||_\infty$ distance between γ' and q_m is at most $1/2$ (see Figure 5, left).

We now apply Lemma 15 to $dig(o, p)$. We know that the Euclidean segment $\overline{op}$ is close to q_m. The digital segment must cross L_n and is far from q_m (the closest it can pass is either $q_{b'-r'-\lfloor \frac{c''}{2} \cdot \frac{N \log N}{N+\kappa_2} \rfloor - 1}$ or $q_{b'-r'-1}$). That is, we know that the intersection of $\overline{op}$

with the line $x + y = n$ is at most half a unit away from q_m. Similarly, the intersection with $dig(o, p)$ is at least $\lfloor \frac{c''}{4} \cdot \frac{N \log N}{N+\kappa_2} \rfloor$ from q_m. Thus, by triangle inequality the $||\cdot||_\infty$ distance between $dig(o, p)$ and $\overline{op}$ is at least $\lfloor \frac{c''}{4} \cdot \frac{N \log N}{N+\kappa_2} \rfloor - 3/2 \in \Omega(\frac{N \log N}{N+\kappa_2})$.

k is large and $b' - r' \geq n\alpha + c'' \cdot \frac{N \log N}{N+\kappa_2}$. Look at the x-coordinate of q_k. We know that Q has at least $b' - r' - 1 \geq n\alpha + c'' \cdot \frac{N \log N}{N+\kappa_2} - 1$ many elements, and k is among the productive vertices with the largest x-coordinate. In particular, the x-coordinate of q_k is at least $b' - r' - 2 \geq n\alpha + \frac{c''}{2} \cdot \frac{N \log N}{N+\kappa_2} - 2$.

Let p be the unique leaf of L_N such that $M(p) = M(q_k)$. We now apply Lemma 15 to $dig(o, p)$ at the line $x + y = n$. By definition of p, we have that $dig(o, p)$ passes through q_k. Now, by definition of Q, we know that $M(q_k) \leq \alpha$ and in particular the x-coordinate of p is at most αN (see Figure 5, right). Thus, the Euclidean segment $\overline{op}$ must intersect at a point whose x-coordinate is at most αn.

That is, when we look at the Euclidean and the digital segments along line $x + y = n$, the Euclidean crossing happens at x-coordinate at most αn. However, the x-coordinate of the digital crossing is at least $\alpha n + \frac{c''}{2} \cdot \frac{N \log N}{N+\kappa_2} - 1$. By Lemma 15 we conclude that the error must be $\Omega(\frac{N \log N}{N+\kappa_2})$ as claimed.

$b' - r' < n\alpha - c'' \cdot \frac{N \log N}{N+\kappa_2}$. This proof is very similar to the previous case. Consider the vertex $p = (\lfloor \alpha N \rfloor, N - \lfloor \alpha N \rfloor) \in L_N$ and apply Lemma 15 to $dig(o, p)$ and $\overline{op}$.

At line $x + y = n$ the Euclidean segment $\overline{op}$ passes through a point whose x-coordinate is $\lfloor \alpha N \rfloor \cdot \frac{n}{N} \geq \lfloor \alpha n \rfloor - 1$. By definition, $M(p) \leq \alpha$ and thus $dig(o, p)$ must pass through some vertex q of Q. In particular, the x-coordinate of q is at most $b' - r' < n\alpha - c'' \cdot \frac{N \log N}{N+\kappa_2}$, giving the $\Omega(\frac{N \log N}{N+\kappa_2})$ error and completing the proof of Theorem 3. ◀

Note that if we use Theorem 2 instead, the same argument follows and we would get the following result.

▶ **Theorem 17.** *For any $N \in \mathbb{N}$, any weak CDR defined on $\mathcal{G}_N^+ \subset \mathbb{Z}^2$ with κ_1 inner leaves has $\Omega(\frac{N \log N}{N+\kappa_1})$ error.*

5 Lower bound for CDRs in high dimensions

We now use the lower bound of weak CDRs to obtain a lower bound for CDRs in three or higher dimensions. Consider the restriction of any d-dimensional CDR T to the x_1x_2-plane (we call this restriction the *x_1x_2-restriction of T* and denote it by $T_{x_1x_2}$). Recall that the key observation is that $T_{x_1x_2}$ is a (possibly weak) CDR and that any inner leaf in $T_{x_1x_2}$ must extend in some x_i-direction in T for some $i \in [3..d]$. We have seen that $T_{x_1x_2}$ needs to have a large number of inner leaves to have $o(\log N)$ error. In the following, we will show that a large number of inner leaves will cause constraints for $\mathbb{Z}^d$ and have an impact in the overall error of T.

We do a slight abuse of notation and use the same terms as in two dimensions. For simplicity of the notation, we assume that N is a positive even number. For any $n \leq N$, let $L_n = \{(x_1, x_2, \ldots, x_d) \in \mathcal{G}_N^+ \colon \sum_{i=1}^d x_i = n\}$. Given any CDR in $\mathcal{G}_N^+$, we consider the CDR as a tree rooted at the origin. Let $T(v)$ be the subtree rooted at v.

From Theorem 3, we already know that in order for $T_{x_1x_2}$ to have sublogarithmic error we must have $\kappa_2 \in \omega(N)$ inner leaves. However, each inner leaf ties to a boundary leaf in L_N in d dimensions. In other words, the subtrees rooted at the vertices in $L_{N/2-1} \cap T_{x_1x_2}$ must cover all these boundary vertices. We now observe that a weak CDR with inner leaves in the x_1x_2-plane induces subtrees which are too big for the high dimensional proper CDR (See Figure 6).

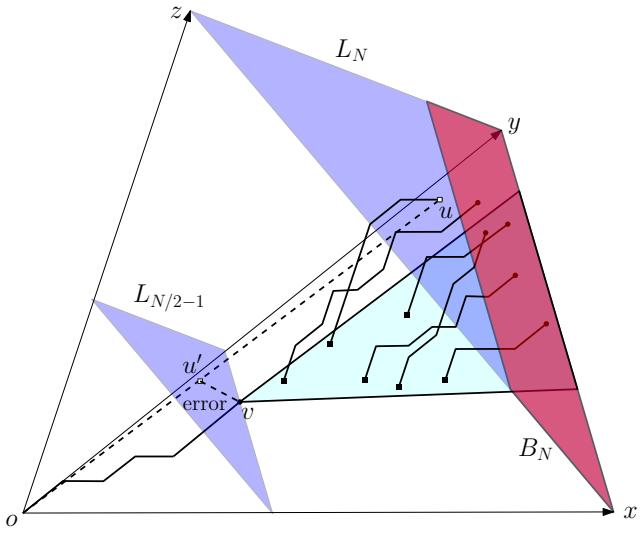


Figure 6 Illustration of Lemmas 18 and 19: the red region represents the region of B_N. If we have lots of inner leaves in T_{xy}, it will have many descendants in the three dimensional CDR at layer L_N so that the height of the red region attempting to contain them is large. In particular, we can find a vertex v on the xy-plane such that v is on the $dig(o, u)$ and u is far away from the xy-plane. For simplicity, we show the Euclidean error between v and u', but we note that the proof argues under the $||\cdot||_\infty$ metric.

▶ **Lemma 18.** *Given any CDR in $\mathcal{G}_N^+$, let κ_2 be the number of inner leaves in $T_{x_1x_2}$ between $L_{N/2}$ and L_N. There exists a vertex $v \in L_{N/2-1}$ such that $v_i = 0$ for $i = 3, \ldots, d$ and some boundary leaf $u \in T(v) \cap L_N$ has $u_j \geq (\kappa_2/N)^{\frac{1}{d-2}} - 1$ for some $j \in [3..d]$.*

Proof. The proof follows from a packing argument. Consider the set $V = \{(0, N/2 - 1, 0, \ldots, 0), (1, N/2 - 2, 0, \ldots, 0), \ldots, (N/2 - 1, 0, 0, \ldots, 0)\}$. Note that these vertices lie in the x_1x_2-plane and thus are in $T_{x_1x_2}$. Because they are the two dimensional equivalent of $L_{N/2-1}$, the union of their subtrees covers $T_{x_1x_2}$ between $N/2$ and N. In this region we know that we have κ_2 many inner leaves, which will extend to L_N with the first step in the x_i-direction for some $i \in [3..d]$. Let Y_N be the extended vertices on L_N from these κ_2 inner leaves, i.e., $|Y_N| \geq \kappa_2$.

Let $B_N = \{(x_1, x_2, \ldots, x_d) \in \mathcal{G}_N^+ \colon \sum_{i=1}^d x_i = N, x_1 + x_2 < N \text{ and } \forall i \in [3..d], x_i < (\kappa_2/N)^{\frac{1}{d-2}} - 1\}$, see Figure 6. Since we have less than $(\kappa_2/N)^{\frac{1}{d-2}}$ choices for $x_3, \ldots, x_d$, at most N choices for x_1 and the value of x_2 is adjusted to satisfy the constraint $\sum_{i=1}^d x_i = N$, the size of B_N is less than κ_2. Hence, B_N cannot contain all vertices of Y_N. Moreover, no vertices of Y_N lie on x_1x_2-plane, so there exists some vertex $u \in Y_N$ such that $u_j \geq (\kappa_2/N)^{\frac{1}{d-2}} - 1$ for some $j \in [3..d]$, which is in $T(v) \cap L_N$ for some $v \in V$. ◀

The existence of this vertex v is the root of the problem. We conclude with the following statement.

▶ **Lemma 19.** *Any CDR defined on $\mathcal{G}_N^+ \subset \mathbb{Z}^d$ with κ_2 inner leaves in $T_{x_1x_2}$ between $L_{N/2}$ and L_N has $\Omega((\kappa_2/N)^{\frac{1}{d-2}})$ error.*

Proof. Apply Lemma 18 to obtain a vertex $v \in L_{N/2-1} \cap T_{x_1x_2}$ that satisfies some $u \in T(v) \cap L_N$ with $u_j \geq (\kappa_2/N)^{\frac{1}{d-2}} - 1$ for some $j \in [3..d]$. Let u' be the intersection of $\overline{ou}$ and the affine plane containing $L_{N/2-1}$, see Figure 6. As L_N and $L_{N/2-1}$ are parallel, $\frac{u'_j - o_j}{u_j - o_j} \geq \frac{1}{3}$ for $N \geq 6$, this implies that $u'_j = \Omega((\kappa_2/N)^{\frac{1}{d-2}})$. By construction, we have that v is on the $dig(o, u)$ and $v_j = 0$, hence $||\cdot||_\infty$ distance between $dig(o, u)$ and $\overline{ou}$ is $\Omega((\kappa_2/N)^{\frac{1}{d-2}})$. ◀

Combining with Theorem 3 gives us a lower bound for CDRs in d dimensions.

▶ **Theorem 4.** *Any CDR in $\mathbb{Z}^d$ has $\Omega(\log^{1/(d-1)} N)$ error.*

Proof. By Theorem 3 and Lemma 19, the error is $\Omega(\frac{N \log N}{N+\kappa_2})$ and $\Omega((\kappa_2/N)^{\frac{1}{d-2}})$, where κ_2 is the number of inner leaves in $T_{x_1x_2}$ between $L_{N/2}$ and L_N. The balance between the two is obtained by choosing $\kappa_2 = \Theta(N \log^{\frac{d-2}{d-1}} N)$, giving the $\Omega(\log^{1/(d-1)} N)$ lower bound. ◀

6 Final remarks

Common intuition would say that the $\Omega(\log N)$ lower bound for the error of two-dimensional CDR and CDS automatically extends to higher dimensions. The observation that this is not true opens up new ways in which research can continue. We believe that further analysis of the mapping between the three spaces (from CDR in high dimensions to the 2-D weak CDR to the two-colored pointset) and the high interdependence between the three spaces can help in designing better lower and upper bounds.

Our lower bound $\Omega(\log^{1/(d-1)} N)$ extends the previous lower bound. The next step would be to close the gap between $\Omega(\log^{1/2} N)$ and $O(\log N)$ bounds in three dimensions. Even if the final answer ends up being $\Theta(\log N)$ we believe that the relationship between high dimensional CDRs, weak CDRs induced in subspaces and the mapping to pointset gives a better understanding of CDRs.

We also find that weak CDRs are an interesting research topic on their own. In particular, we would like to find the relationship between the number of inner leaves and the error of the construction. That is, say that we want a CDR with $O(e)$ error (for some $e \leq \log n$). What is the minimum number of leaves $\ell = \ell(e)$ that such a CDR must have? Can we find such a construction?

Theorem 3 seems to indicate a linear relationship between the two, and it is not hard to obtain one (an example is given in [3]). However, this construction is most likely not the best possible one. Indeed, even if we are interested in $O(\log N)$ error, this construction creates a large number of inner leaves, but we know of CDRs with the same error and no inner leaves. Thus, the question becomes, can we significantly improve upon the greedy construction, which can be found in the full version of the paper in [3]? Or is there some exponential dependency between the number of inner leaves and the error of the weak CDR?

References

1 Tetsuo Asano, Danny Z. Chen, Naoki Katoh, and Takeshi Tokuyama. Efficient algorithms for optimization-based image segmentation. *International Journal of Computational Geometry and Applications*, 11(2):145–166, 2001.

2 Man-Kwun Chiu and Matias Korman. High dimensional consistent digital segments. *SIAM Journal on Discrete Mathematics*, 32(4):2566–2590, 2018.

3 Man-Kwun Chiu, Matias Korman, Martin Suderland, and Takeshi Tokuyama. Distance bounds for high dimensional consistent digital rays and 2-d partially-consistent digital rays, 2020. `arXiv:2006.14059`.

4 Iffat Chowdhury and Matt Gibson. A characterization of consistent digital line segments in $\mathbb{Z}^2$. In Nikhil Bansal and Irene Finocchi, editors, *Proceedings of the 23rd Annual European Symposium on Algorithms*, volume 9294, pages 337–348, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

5 Iffat Chowdhury and Matt Gibson. Constructing consistent digital line segments. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *Proceedings of the 12th Latin American Theoretical Informatics Symposium*, volume 9644, pages 263–274, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

6 Tobias Christ, Dömötör Pálvölgyi, and Miloš Stojaković. Consistent digital line segments. *Discrete & Computational Geometry*, 47(4):691–710, 2012.

7 Jinhee Chun, Natsuda Kaothanthong, Ryosei Kasai, Matias Korman, Martin Nöllenburg, and Takeshi Tokuyama. Algorithms for computing the maximum weight region decomposable into elementary shapes. *Computer Vision and Image Understanding*, 116(7):803–814, 2012.

8 Jinhee Chun, Matias Korman, Martin Nöllenburg, and Takeshi Tokuyama. Consistent digital rays. *Discrete and Computational Geometry*, 42(3):359–378, 2009.

9 Jacob E. Goodman, Richard Pollack, and Bernd Sturmfels. Coordinate representation of order types requires exponential storage. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 405–410. ACM, 1989.

10 Reinhard Klette and Azriel Rosenfeld. Digital straightness—a review. *Discrete Appl. Math.*, 139(1–3):197–230, 2004.

11 Michael G. Luby. Grid geometries which preserve properties of Euclidean geometry: A study of graphics line drawing algorithms. In *NATO Conference on Graphics/CAD*, pages 397–432, 1987.

12 Jiří Matoušek. *Geometric Discrepancy: An Illustrated Guide*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 1999. URL: `https://books.google.co.jp/books?id=BKvXj1GisP0C`.

13 Wolfgang Schmidt. Irregularities of distribution, vii. *Acta Arithmetica*, 21(1):45–50, 1972. URL: `http://eudml.org/doc/205130`.

Finding Large H-Colorable Subgraphs in Hereditary Graph Classes

Maria Chudnovsky
Princeton University, NJ, USA
mchudnov@math.princeton.edu

Jason King
Princeton University, NJ, USA
jtking@princeton.edu

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Paweł Rzążewski
Warsaw University of Technology, Faculty of Mathematics and Information Science, Poland
University of Warsaw, Institute of Informatics, Poland
p.rzazewski@mini.pw.edu.pl

Sophie Spirkl
Princeton University, NJ, USA
sspirkl@math.princeton.edu

Abstract

We study the Max Partial H-Coloring problem: given a graph G, find the largest induced subgraph of G that admits a homomorphism into H, where H is a fixed pattern graph without loops. Note that when H is a complete graph on k vertices, the problem reduces to finding the largest induced k-colorable subgraph, which for $k = 2$ is equivalent (by complementation) to Odd Cycle Transversal.

We prove that for every fixed pattern graph H without loops, Max Partial H-Coloring can be solved:

- in $\{P_5, F\}$-free graphs in polynomial time, whenever F is a threshold graph;
- in $\{P_5, \text{bull}\}$-free graphs in polynomial time;
- in P_5-free graphs in time $n^{\mathcal{O}(\omega(G))}$;
- in $\{P_6, \text{1-subdivided claw}\}$-free graphs in time $n^{\mathcal{O}(\omega(G)^3)}$.

Here, n is the number of vertices of the input graph G and $\omega(G)$ is the maximum size of a clique in G. Furthermore, by combining the mentioned algorithms for P_5-free and for $\{P_6, \text{1-subdivided claw}\}$-free graphs with a simple branching procedure, we obtain subexponential-time algorithms for Max Partial H-Coloring in these classes of graphs.

Finally, we show that even a restricted variant of Max Partial H-Coloring is NP-hard in the considered subclasses of P_5-free graphs, if we allow loops on H.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Problems, reductions and completeness; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases homomorphisms, hereditary graph classes, odd cycle transversal

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.35

Related Version A full version of the paper is available at [8], https://arxiv.org/abs/2004.09425.

Funding *Maria Chudnovsky*: This material is based upon work supported in part by the U. S. Army Research Office under grant number W911NF-16-1-0404, and by NSF grant DMS-1763817.
Michał Pilipczuk: This work is a part of project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 677651).

Paweł Rzążewski: Supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062.
Sophie Spirkl: This material is based upon work supported by the National Science Foundation under Award No. DMS1802201.

Acknowledgements We acknowledge the welcoming and productive atmosphere at Dagstuhl Seminar 19271 "Graph Colouring: from Structure to Algorithms", where this work has been initiated.

1 Introduction

Many computational graph problems that are (NP-)hard in general become tractable in restricted classes of input graphs. In this work we are interested in *hereditary* graph classes, or equivalently classes defined by forbidding induced subgraphs. For a set of graphs $\mathcal{F}$, we say that a graph G is *$\mathcal{F}$-free* if G does not contain any induced subgraph isomorphic to a graph from $\mathcal{F}$. By forbidding different sets $\mathcal{F}$ we obtain graph classes with various structural properties, which can be used in the algorithmic context. This highlights an interesting interplay between structural graph theory and algorithm design.

Perhaps the best known example of this paradigm is the case of the Maximum Independent Set problem: given a graph G, find the largest set of pairwise non-adjacent vertices in G. It is known that the problem is NP-hard on F-free graphs unless F is a forest whose every component is a path or a subdivided claw [2]; here, a *claw* is a star with 3 leaves. However, the remaining cases, when F is a *subdivided claw forest*, remain largely unexplored despite significant effort. Polynomial-time algorithms have been given for P_5-free graphs [24], P_6-free graphs [20], claw-free graphs [26, 29], and fork-free graphs [3, 25]. While the complexity status in all the other cases remains open, it has been observed that relaxing the goal of polynomial-time solvability leads to positive results in a larger generality. For instance, for every $t \in \mathbb{N}$, Maximum Independent Set can be solved in time $2^{\mathcal{O}(\sqrt{tn \log n})}$ in P_t-free graphs [4]. The existence of such a *subexponential-time algorithm* for F-free graphs is excluded under the Exponential Time Hypothesis whenever F is not a subdivided claw forest (see e.g. the discussion in [27]), which shows a qualitative difference between the negative and the potentially positive cases. Also, Chudnovsky et al. [10] recently gave a quasi-polynomial-time approximation scheme (QPTAS) for Maximum Independent Set in F-free graphs, for every fixed subdivided claw forest F.

The abovementioned positive results use a variety of structural techniques related to the considered graph classes, for instance: the concept of *Gyárfás path* that gives useful separators in P_t-free graphs [4, 6, 10], the dynamic programming approach based on potential maximal cliques [24, 20], or structural properties of claw-free and fork-free graphs that relate them to line graphs [25, 26, 29]. Some of these techniques can be used to give algorithms for related problems, which can be expressed as looking for the largest (in terms of the number of vertices) induced subgraph satisfying a fixed property. For Maximum Independent Set this property is being edgeless, but for instance the property of being acyclic corresponds to the Maximum Induced Forest problem, which by complementation is equivalent to Feedback Vertex Set. Work in this direction so far focused on properties that imply bounded treewidth [1, 17] or, more generally, that imply sparsity [27].

A different class of problems that admits an interesting complexity landscape on hereditary graphs classes are coloring problems. For fixed $k \in \mathbb{N}$, the k-Coloring problem asks whether the input graph admits a proper coloring with k colors. For every $k \geqslant 3$, the problem is NP-hard on F-free graphs unless F is a forest of paths (a *linear forest*) [18]. The classification of the remaining cases is more advanced than in the case of Maximum Independent Set, but not yet complete. On one hand, Hoàng et al. [22] showed that for every fixed k, k-Coloring

is polynomial-time solvable on P_5-free graphs. On the other hand, the problem becomes NP-hard already on P_6-free graphs for all $k \geqslant 5$ [23]. The cases $k = 3$ and $k = 4$ turn out to be very interesting. 4-Coloring is polynomial-time solvable on P_6-free graphs [14] and NP-hard in P_7-free graphs [23]. While there is a polynomial-time algorithm for 3-Coloring in P_7-free graphs [5], the complexity status in P_t-free graphs for $t \geqslant 8$ remains open. However, relaxing the goal again leads to positive results in a wider generality: for every $t \in \mathbb{N}$, there is a subexponential-time algorithm with running time $2^{\mathcal{O}(\sqrt{tn \log n})}$ for 3-Coloring in P_t-free graphs [19], and there is also a polynomial-time algorithm that given a 3-colorable P_t-free graph outputs its proper coloring with $\mathcal{O}(t)$ colors [12].

We are interested in using the toolbox developed for coloring problems in P_t-free graphs to the setting of finding maximum induced subgraphs with certain properties. Specifically, consider the following Maximum Induced k-Colorable Subgraph problem: given a graph G, find the largest induced subgraph of G that admits a proper coloring with k colors. While this problem clearly generalizes k-Coloring, for $k = 1$ it boils down to Maximum Independent Set. For $k = 2$ it can be expressed as Maximum Induced Bipartite Subgraph, which by complementation is equivalent to the well-studied Odd Cycle Transversal problem: find the smallest subset of vertices that intersects all odd cycles in a given graph. While polynomial-time solvability of Odd Cycle Transversal on P_4-free graphs (also known as *cographs*) follows from the fact that these graphs have bounded cliquewidth (see [15]), it is known that the problem is NP-hard in P_6-free graphs [16]. The complexity status of Odd Cycle Transversal in P_5-free graphs remains open [9, Problem 4.4]: resolving this question was the original motivation of our work.

Our contribution. Following the work of Groenland et al. [19], we work with a very general form of coloring problems, defined through homomorphisms. For graphs G and H, a *homomorphism* from G to H, or an *H-coloring* of G, is a function $\phi\colon V(G) \to V(H)$ such that for every edge uv in G, we have $\phi(u)\phi(v) \in E(H)$. We study the Max Partial H-Coloring problem defined as follows: given a graph G, find the largest induced subgraph of G that admits an H-coloring. Note that if H is the complete graph on k vertices, then an H-coloring is simply a proper coloring with k colors, hence this formulation generalizes the Maximum Induced k-Colorable Subgraph problem. Unless stated explicitly, we will always assume that the pattern graph H does not have loops, hence an H-coloring is a proper coloring with $|V(H)|$ colors.

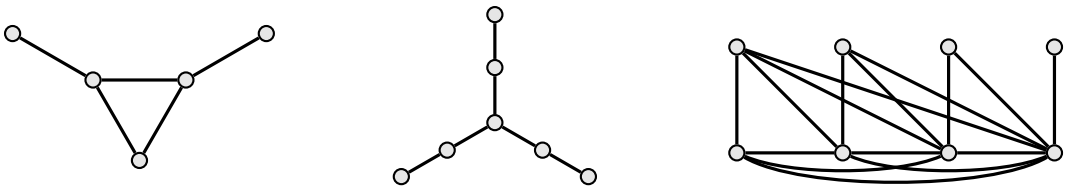


Figure 1 A bull, a 1-subdivided claw, and an example threshold graph.

Fix a pattern graph H without loops. We prove that Max Partial H-Coloring can be solved:

(R1) in $\{P_5, F\}$-free graphs in polynomial time, whenever F is a threshold graph;
(R2) in $\{P_5, \text{bull}\}$-free graphs in polynomial time;
(R3) in P_5-free graphs in time $n^{\mathcal{O}(\omega(G))}$; and
(R4) in $\{P_6, \text{1-subdivided claw}\}$-free graphs in time $n^{\mathcal{O}(\omega(G)^3)}$.

Here, n is the number of vertices of the input graph G and $\omega(G)$ is the size of the maximum clique in G. Also, recall that a graph G is a *threshold graph* if $V(G)$ can be partitioned into an independent set A and a clique B such that for each $a, a' \in A$, we have either $N(a) \supseteq N(a')$ or $N(a) \subseteq N(a')$. There is also a characterization via forbidden induced subgraphs: threshold graphs are exactly $\{2P_2, C_4, P_4\}$-free graphs, where $2P_2$ is an induced matching of size 2. Figure 1 depicts a bull, a 1-subdivided claw, and an example threshold graph.

Further, we observe that by employing a simple branching strategy, an $n^{\mathcal{O}(\omega(G)^\alpha)}$-time algorithm for Max Partial H-Coloring in $\mathcal{F}$-free graphs can be used to give also a subexponential-time algorithm in this setting, with running time $n^{\mathcal{O}(n^{\alpha/(\alpha+1)})}$. Thus, results (R3) and (R4) imply that for every fixed irreflexive H, the Max Partial H-Coloring problem can be solved in time $n^{\mathcal{O}(\sqrt{n})}$ in P_5-free graphs and in time $n^{\mathcal{O}(n^{3/4})}$ in $\{P_6, \text{1-subdivided claw}\}$-free graphs. This in particular applies to the Odd Cycle Transversal problem. We note here that Dabrowski et al. [16] proved that Odd Cycle Transversal in $\{P_6, K_4\}$-free graphs is NP-hard and does not admit a subexponential-time algorithm under the Exponential Time Hypothesis. Thus, it is unlikely that any of our algorithmic results – the $n^{\mathcal{O}(\omega(G))}$-time algorithm and the $n^{\mathcal{O}(\sqrt{n})}$-time algorithm – can be extended from P_5-free graphs to P_6-free graphs.

All our algorithms work in a weighted setting, where instead of just maximizing the size of the domain of an H-coloring, we maximize its total *revenue*, where for each pair $(u, v) \in V(G) \times V(H)$ we have a prescribed revenue yielded by sending u to v. This setting allows encoding a broader range of coloring problems. For instance, list variants can be expressed by giving negative revenues for forbidden assignments (see e.g. [21, 28]). Also, our algorithms work in a slightly larger generality than stated above, see Section 5 for precise statements.

Finally, we investigate the possibility of extending our algorithmic results to pattern graphs with possible loops. We show an example of a graph H with loops, for which Max Partial H-Coloring is NP-hard and admits no subexponential-time algorithm under the ETH even in very restricted subclasses of P_5-free graphs, including $\{P_5, \text{bull}\}$-free graphs. This shows that whether the pattern graph is allowed to have loops has a major impact on the complexity of the problem.

Full version. In this extended abstract we focus on proving results (R3) and (R4). Results (R1) and (R2), as well as of the abovementioned lower bound, are proved in the full version of the paper, which is available on arXiv [8]. Also, the main branching step is given here in a simplified form that is sufficient for results (R3) and (R4), but not for results (R1) and (R2).

Our techniques. The key element of our approach is a branching procedure that, given an instance (G, rev) of Max Partial H-Coloring, where rev is the revenue function, produces a relatively small set of instances Π such that solving (G, rev) reduces to solving all the instances in Π. Moreover, every instance $(G', \mathsf{rev}') \in \Pi$ is simpler in the following sense: either it is an instance of Max Partial H'-Coloring for H' being a proper induced subgraph of H (hence it can be solved by induction on $|V(H)|$), or for any connected graph F on at least two vertices, G' is F-free provided we assume that G is $F^{\bullet\!-\!\circ}$-free. Here, $F^{\bullet\!-\!\circ}$ is the graph obtained from F by adding a universal vertex y and a degree-1 vertex x adjacent only to y. In particular we have $\omega(G') < \omega(G)$, so applying the branching procedure exhaustively in a recursion scheme yields a recursion tree of depth bounded by $\omega(G)$. Now, for results (R3) and (R4) we respectively have $|\Pi| \leqslant n^{\mathcal{O}(1)}$ and $|\Pi| \leqslant n^{\mathcal{O}(\omega(G)^2)}$, giving bounds of $n^{\mathcal{O}(\omega(G))}$ and $n^{\mathcal{O}(\omega(G)^3)}$ on the total size of the recursion tree and on the overall time complexity.

For result (R1) we apply the branching procedure not exhaustively, but a constant number of times: if the original graph G is $\{P_5, F\}$-free for some threshold graph F, it suffices to apply the branching procedure $\mathcal{O}(|V(F)|)$ times to reduce the original instances to a set of edgeless instances, which can be solved trivially. As $\mathcal{O}(|V(F)|) = \mathcal{O}(1)$, this gives recursion tree of polynomial size, and hence a polynomial-time complexity due to always having $|\Pi| \leqslant n^{\mathcal{O}(1)}$ in this setting. For result (R2), we show that two applications of the branching procedure reduce the input instance to a polynomial number of instances that are P_4-free, which can be solved in polynomial time due to P_4-free graphs (also known as *cographs*) having cliquewidth at most 2. However, these applications are interleaved with a reduction to the case of *prime graphs* – graphs with no non-trivial modules – which we achieve using dynamic programming on the modular decomposition of the input graph. This is in order to apply some results on the structure of prime bull-free graphs [11, 13], so that P_4-freeness is achieved at the end.

Let us briefly discuss the key branching procedure. The first step is finding a useful dominating structure that we call a *monitor*: a subset of vertices M of a connected graph G is a monitor if for every connected component C of $G - M$, there is a vertex in M that is complete to C. We prove that in a connected P_6-free graph there is always a monitor that is the closed neighborhood of a set of at most three vertices. After finding such a monitor $N[X]$ for $|X| \leqslant 3$, we perform a structural analysis of the graph centered around the set X. This analysis shows that there exists a subset of $\mathcal{O}(|V(H)|)$ vertices such that after guessing this subset and the H-coloring on it, the instance can be partitioned into several separate subinstances, each of which has a strictly smaller clique number. This structural analysis, and in particular the way the separation of subinstances is achieved, is inspired by the polynomial-time algorithm of Hoàng et al. [22] for k-Coloring in P_5-free graphs.

Other related work. We remark that very recently and independently of us, Brettell et al. [7] proved that for every fixed $s, t \in \mathbb{N}$, the class of $\{K_t, sK_1 + P_5\}$-free graphs has bounded mim-width. Here, *mim-width* is a graph parameter that is less restrictive than cliquewidth, but the important aspect is that a wide range of vertex-partitioning problems, including the Max Partial H-Coloring problem considered in this work, can be solved in polynomial time on every class of graphs where the mim-width is universally bounded and a corresponding decomposition can be computed efficiently. The result of Brettell et al. thus shows that in P_5-free graphs, the mim-width is bounded by a function of the clique number. This gives an $n^{f(\omega(G))}$-time algorithm for Max Partial H-Coloring in P_5-free graphs (for fixed H), for some function f. However, the proof presented in [7] gives only an exponential upper bound on the function f, which in particular does not imply the existence of a subexponential-time algorithm. To compare, our reasoning leads to an $n^{\mathcal{O}(\omega(G))}$-time algorithm and a subexponential-time algorithm with complexity $n^{\mathcal{O}(\sqrt{n})}$.

We remark that the techniques used by Brettell et al. [7] also rely on revisiting the approach of Hoàng et al. [22], and they similarly observe that this approach can be used to apply induction based on the clique number of the graph.

2 Preliminaries

Graphs. For a graph G, the vertex and edge sets of G are denoted by $V(G)$ and $E(G)$, respectively. The *open neighborhood* of a vertex u is the set $N_G(u) := \{v \colon uv \in E(G)\}$, while the *closed neighborhood* is $N_G[u] := N_G(u) \cup \{u\}$. This notation is extended to sets of vertices: for $X \subseteq V(G)$, we set $N_G[X] := \bigcup_{u \in X} N_G[u]$ and $N_G(X) := N_G[X] - X$. We may omit the subscript if the graph G is clear from the context. By C_t, P_t, and K_t we respectively denote the cycle, the path, and the complete graph on t vertices.

The *clique number* $\omega(G)$ is the size of the largest clique in a graph G. A clique K in G is *maximal* if no proper superset of K is a clique.

For $s, t \in \mathbb{N}$, the *Ramsey number* of s and t is the smallest integer k such that every graph on k vertices contains either a clique of size s or an independent set of size t. It is well-known that the Ramsey number of s and t is bounded from above by $\binom{s+t-2}{s-1}$, hence we will denote $\mathsf{Ramsey}(s,t) \coloneqq \binom{s+t-2}{s-1}$.

For a graph G and $A \subseteq V(G)$, by $G[A]$ we denote the subgraph of G induced by A. We write $G - A \coloneqq G[V(G) - A]$. We say that F is an *induced subgraph* of G if there is $A \subseteq V(G)$ such that $G[A]$ is isomorphic to F; this containment is *proper* if in addition $A \neq V(G)$. For a family of graphs $\mathcal{F}$, a graph G is *$\mathcal{F}$-free* if G does not contain any induced subgraph from $\mathcal{F}$. If $\mathcal{F} = \{H\}$, then we may speak about *H-free* graphs as well.

If G is a graph and $A \subseteq V(G)$ is a subset of vertices, then a vertex $u \notin A$ is *complete* to A if u is adjacent to all the vertices of A, and u is *anti-complete* to A if u has no neighbors in A. We will use the following simple claim several times.

▶ **Lemma 1.** *Suppose G is a graph, A is a subset of its vertices such that $G[A]$ is connected, and $u \notin A$ is a vertex that is neither complete nor anti-complete to A in G. Then there are vertices $a, b \in A$ such that $u - a - b$ is an induced P_3 in G.*

Proof. Since u is neither complete nor anticomplete to A, both the sets $A \cap N(u)$ and $A - N(u)$ are non-empty. As A is connected, there exist $a \in A \cap N(u)$ and $b \in A - N(u)$ such that a and b are adjacent. Now $u - a - b$ is the desired induced P_3. ◀

For a graph F, by $F^\bullet$ we denote the graph obtained from F by adding a *universal vertex*: a vertex adjacent to all the other vertices. Similarly, by $F^{\bullet-\circ}$ we denote the graph obtained from F by adding first an isolated vertex, say x, and then a universal vertex, say y. Note that thus y is adjacent to all the other vertices of $F^{\bullet-\circ}$, while x is adjacent only to y.

***H*-colorings.** For graphs H and G, a function $\phi\colon V(G) \to V(H)$ is a *homomorphism* from G to H if for every $uv \in E(G)$, we also have $\phi(u)\phi(v) \in E(H)$. Note that a homomorphism from G to the complete graph K_t is nothing else than a proper coloring of G with t colors. Therefore, a homomorphism from G to H will be also called an *H-coloring* of G, and we will refer to vertices of H as colors. Note that we will always assume that H is a simple graph without loops, so no two adjacent vertices of G can be mapped by a homomorphism to the same vertex of H. To stress this, we will call such H an *irreflexive pattern graph*.

A *partial homomorphism* from G to H, or a *partial H-coloring* of G, is a partial function $\phi\colon V(G) \rightharpoonup V(H)$ that is a homomorphism from $G[\mathsf{dom}\,\phi]$ to H, where $\mathsf{dom}\,\phi$ denotes the domain of ϕ.

Suppose that with graphs G and H we associate a *revenue function* $\mathsf{rev}\colon V(G) \times V(H) \to \mathbb{R}$. Then the *revenue* of a partial H-coloring ϕ is defined as

$$\mathsf{rev}(\phi) \coloneqq \sum_{u \in \mathsf{dom}\,\phi} \mathsf{rev}(u, \phi(u)).$$

In other words, for $u \in V(G)$ and $v \in V(H)$, $\mathsf{rev}(u,v)$ denotes the revenue yielded by assigning $\phi(u) \coloneqq v$.

We now define the main problem studied in this work. In the following, we consider the graph H fixed.

Max Partial H-Coloring
Input: Graph G and a revenue function $\mathsf{rev}\colon V(G) \times V(H) \to \mathbb{R}$
Output: A partial H-coloring ϕ of G that maximizes $\mathsf{rev}(\phi)$

An *instance* of the Max Partial H-Coloring problem is a pair (G, rev) as above. A *solution* to an instance (G, rev) is a partial H-coloring of G, and it is *optimum* if it maximizes $\mathsf{rev}(\phi)$ among solutions. By $\mathrm{OPT}(G, \mathsf{rev})$ we denote the maximum possible revenue of a solution to the instance (G, rev).

Let us note one aspect that will be used later on. Observe that in revenue functions we allow negative revenues for some assignments. However, if we are interested in maximizing the total revenue, there is no point in using such assignments: if $u \in \mathsf{dom}\,\phi$ and $\mathsf{rev}(u, \phi(u)) < 0$, then just removing u from the domain of ϕ increases the revenue. Thus, optimal solutions never use assignments with negative revenues. Note that this feature can be used to model list versions of partial coloring problems, where each vertex $v \in V(G)$ is assigned a list of colors $L(v) \subseteq V(H)$ and can only be mapped to a vertex from $L(v)$.

3 Monitors in P_6-free graphs

In this section we prove an auxiliary result about finding useful separators in P_6-free graphs. The desired property is expressed in the following definition.

▶ **Definition 2.** *Let G be a connected graph. A subset of vertices $M \subseteq V(G)$ is a* monitor *in G if for every connected component C of $G - M$, there exists a vertex $w \in M$ that is complete to C.*

Let us note the following property of monitors.

▶ **Lemma 3.** *If M is a monitor in a connected graph G, then every maximal clique in G intersects M. In particular, $\omega(G - M) < \omega(G)$.*

Proof. If K is a clique in $G - M$, then K has to be entirely contained in some connected component C of $G - M$. Since M is a monitor, there exists $w \in M$ that is complete to C. Then $K \cup \{w\}$ is also a clique in G, hence K cannot be a maximal clique in G. ◀

We now prove that in P_6-free graphs we can always find easily describable monitors.

▶ **Lemma 4.** *Let G be a connected P_6-free graph. Then for every $u \in V(G)$ there exists a subset of vertices X such that $u \in X$, $|X| \leqslant 3$, $G[X]$ is a path whose one endpoint is u, and $N_G[X]$ is a monitor in G.*

Lemma 4 follows immediately from the following statement applied for $t = 6$.

▶ **Lemma 5.** *Let $t \in \{4, 5, 6\}$, G be a connected P_6-free graph, and $u \in V(G)$ be a vertex such that in G there is no induced P_t with u being one of the endpoints. Then there exists a subset X of vertices such that $u \in X$, $|X| \leqslant t - 3$, $G[X]$ is a path whose one endpoint is u, and $N_G[X]$ is a monitor in G.*

Proof. We proceed by induction on t. The base case for $t = 4$ will be proved directly within the analysis.

In the following, by *slabs* we mean connected components of the graph $G - N_G[u]$. We shall say that a vertex $w \in N_G(u)$ is *mixed* on a slab C if w is neither complete nor anti-complete to C. A slab C is *simple* if there exists a vertex $w \in N_G(u)$ that is complete to C, and *difficult* otherwise.

Note that since G is connected, for every difficult slab D there exists some vertex $w \in N_G(u)$ that is mixed on D. Then, by Lemma 1, we can find vertices $a, b \in D$ such that $u - w - a - b$ is an induced P_4 in G. If $t = 4$ then no such induced P_4 can exists, so we infer that in this case there are no difficult slabs. Then $N_G[u]$ is a monitor, so we may set $X := \{u\}$. This proves the claim for $t = 4$; from now on we assume that $t \geqslant 5$.

Let us choose a vertex $v \in N_G(u)$ that maximizes the number of difficult slabs on which v is mixed. Suppose there is a difficult slab D' such that v is anti-complete to D'. As we argued, there exists a vertex $v' \in N_G(u)$ such that v' is mixed on D'; clearly $v' \neq v$. By the choice of v, there exists a difficult slab D such that v is mixed on D and v' is anti-complete to D. By applying Lemma 1 twice, we find vertices $a, b \in D$ and $a', b' \in D'$ such that $v - a - b$ and $v' - a' - b'$ are induced P_3s in G. Now, if v and v' were adjacent, then $b - a - v - v' - a' - b'$ would be an induced P_6 in G, a contradiction. Otherwise $b - a - v - u - v' - a' - b'$ is an induced P_7 in G, again a contradiction (see Figure 2).

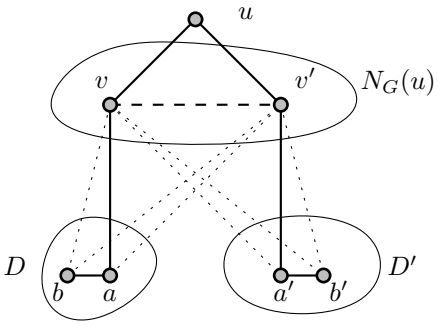


Figure 2 The graph G in the proof of Lemma 5 when v anti-complete to some difficult slab D'. Dotted lines show non-edges. The edge vv' might be present.

We conclude that v is mixed on every difficult slab. Let

$$A := \{v\} \cup \bigcup_{D\colon \text{difficult slab}} V(D).$$

Then $G[A]$ is connected and P_6-free. Moreover, in $G[A]$ there is no P_{t-1} with one endpoint being v, because otherwise we would be able to extend such an induced P_{t-1} using u, and thus obtain an induced P_t in G with one endpoint being u. Consequently, by induction we find a subset $Y \subseteq A$ such that $|Y| \leqslant (t-1) - 3 = t - 4$, $G[Y]$ is a path with one of the endpoints being v, and $N_{G[A]}[Y]$ is a monitor in $G[A]$. Let $X := Y \cup \{u\}$. Then $|X| \leqslant t - 3$ and $G[X]$ is a path with u being one of the endpoints.

We verify that $N_G[X]$ is a monitor in G. Consider any connected component C of $G - N_G[X]$. As $N_G[X] \supseteq N_G[u]$, C is contained in some slab D. If D is simple, then by definition there exists a vertex $w \in N_G[u] \subseteq N_G[X]$ that is complete to D, hence also complete to C. Otherwise D is difficult, hence C is a connected component of $G[A] - N_{G[A]}[Y]$. Since $N_{G[A]}[Y]$ is a monitor in $G[A]$, there exists a vertex $w \in N_{G[A]}[Y] \subseteq N_G[X]$ that is complete to C. This completes the proof. ◀

We remark that no statement analogous to Lemma 4 can hold for P_7-free graphs, even if from X we only require that $N_G[X]$ intersects all the maximum-size cliques in G (which is implied by the property of being a monitor, see Lemma 3). Consider the following example. Let G be a graph obtained from the union of $n+1$ complete graphs $K^{(0)}, \ldots, K^{(n)}$, each on n vertices, by making one vertex from each of the graphs $K^{(1)}, \ldots, K^{(n)}$ adjacent to a different vertex of $K^{(0)}$. Then G is P_7-free, but the minimum size of a set $X \subseteq V(G)$ such that $N_G[X]$ intersects all maximum-size cliques in G is n.

4 Branching

We now present the core branching step that is used by all our algorithms. This part is inspired by the approach of Hoàng et al. [22]. We will rely on the following two graph families; see Figure 3. For $t \in \mathbb{N}$, the graph S_t is obtained from the star $K_{1,t}$ by subdividing every edge once. Then $L_1 := P_3$ and for $t \geqslant 2$ the graph L_t is obtained from S_t by making all the leaves of S_t pairwise adjacent.

Figure 3 Graphs S_4 and L_4.

▶ **Lemma 6.** *Let H be a fixed irreflexive pattern graph. Suppose we are given integers s, t and an instance (G, rev) of Max Partial H-Coloring such that G is connected and $\{P_6, L_s, S_t\}$-free. Denoting $n := |V(G)|$, one can in time $n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$ construct a subgraph G' of G with $V(G') = V(G)$ and a set Π consisting of at most $n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$ revenue functions with domain $V(G) \times V(H)$ such that the following conditions hold:*

(C1) *The graph G' is $\{P_6, L_s, S_t\}$-free. Moreover, if G is $F^\bullet$-free for some connected graph F on at least two vertices, then G' is F-free.*

(C2) *We have $\mathrm{OPT}(G, \mathsf{rev}) = \max_{\mathsf{rev}' \in \Pi} \mathrm{OPT}(G', \mathsf{rev}')$. Moreover, for any $\mathsf{rev}' \in \Pi$ for which the maximum is reached, every optimum solution ϕ to (G', rev') is also an optimum solution to (G, rev) with $\mathsf{rev}(\phi) = \mathsf{rev}'(\phi)$.*

We remark that the statement above is a simplified variant of the lemma, and it is sufficient for proving results (R3) and (R4), but not for results (R1) and (R2). In the full variant, presented in the full version of the paper, solving the instance (G, rev) is reduced to solving a list Π of *pairs* of instances. Each pair $((G_1, \mathsf{rev}_1), (G_2, \mathsf{rev}_2)) \in \Pi$ satisfies the following: (G_1, rev_1) is an instance of Max Partial H'-Coloring for some proper induced subgraph H' of G; and if G_2 contains some induced connected graph F on at least two vertices, then G contains not only an induced $F^\bullet$, but even an induced $F^{\bullet\text{-}\circ}$. This gives a stronger reduction of structure upon application of Lemma 6, which is vitally used in the proofs of results (R1) and (R2).

The remainder of this section is devoted to the proof of Lemma 6. We fix the irreflexive pattern graph H and consider an input instance (G, rev). We find it more didactic to first perform an analysis of (G, rev), and only provide the algorithm at the end. Thus, the correctness will be clear from the previous observations.

Since G is connected, by Lemma 4 there exists $X \subseteq V(G)$ such that $|X| \leqslant 3$ and $N[X]$ is a monitor in G. Note that such a set X can be found in polynomial time by checking all subsets of $V(G)$ of size at most 3. In case $|X| < 3$, we may add arbitrary vertices to X so that $|X| = 3$, note that the property of being a monitor still holds. Let us arbitrarily enumerate the vertices of X as $\{x_1, x_2, x_3\}$.

We partition $V(G) - X$ into A_1, A_2, A_3, A_4 as follows (see Figure 4):

$$A_1 := N(x_1) - X,\ A_2 := N(x_2) - (X \cup A_1),\ A_3 := N(x_3) - (X \cup A_1 \cup A_2),\ A_4 := V(G) - N[X].$$

Note that $\{A_1, A_2, A_3\}$ is a partition of $N(X)$. For $i \in \{1,2,3\}$, denote $A_{>i} \coloneqq \bigcup_{j=i+1}^{4} A_j$ and observe that x_i is complete to A_i and anti-complete to $A_{>i}$. Moreover, we have the following.

▷ Claim 7. For every connected graph F and $i \in \{1,2,3,4\}$, if $G[A_i]$ contains an induced F, then G contains an induced $F^\bullet$.

Proof. Suppose $B \subseteq A_i$ induces F in G. If $i \in \{1,2,3\}$ then $B \cup \{x_i\}$ induces $F^\bullet$ in G, hence assume that $i = 4$. Since F is connected, B is entirely contained in one connected component C of $G[A_4]$. As $N[X]$ is a monitor in G, there exists a vertex $w \in N[X]$ that is complete to C. Now $B \cup \{w\}$ induces $F^\bullet$ in G. ◁

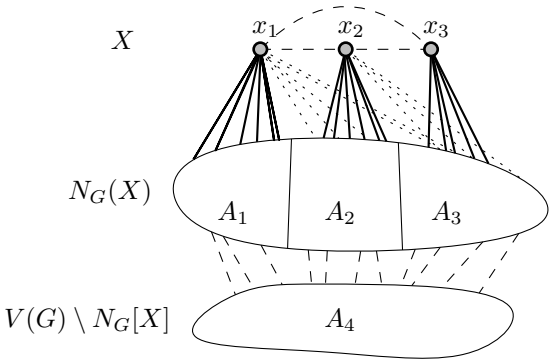


Figure 4 The partition on $V(G)$ in the proof of Lemma 6. Solid and dotted lines respectively indicate that a vertex is complete or anti-complete to a set. Dashed edges might, but do not have to exist.

The next claim contains the core combinatorial observation of the proof.

▷ Claim 8. Let ϕ be a solution to the instance (G, rev). Then for every $i \in \{1,2,3\}$ and $v \in V(H)$, there exists a set $S \subseteq A_i$ such that:

- $|S| < \mathsf{Ramsey}(s,t)$ and $S \subseteq A_i \cap \phi^{-1}(v)$; and
- every vertex $u \in A_{>i}$ that has a neighbor in $A_i \cap \phi^{-1}(v)$, also has a neighbor in S.

Proof. Let S be the smallest set contained in $A_i \cap \phi^{-1}(v)$ and satisfying the second condition, it exists, as this condition is satisfied by $A_i \cap \phi^{-1}(v)$. Note that since H is irreflexive, it follows that $\phi^{-1}(v)$ is an independent set in G, hence S is independent as well.

Suppose for contradiction that $|S| \geqslant \mathsf{Ramsey}(s,t)$. By minimality, for every $u \in S$ there exists $u' \in A_{>i}$ such that u is the only neighbor of u' in S. Let $S' \coloneqq \{u' \colon u \in S\}$. Since $|S'| = |S| \geqslant \mathsf{Ramsey}(s,t)$, in $G[S']$ we can either find a clique K' of size s or an independent set I' of size t; denote $K \coloneqq \{u \colon u' \in K'\}$ and $I \coloneqq \{u \colon u' \in I'\}$. In the former case, we find that $\{x_i\} \cup K \cup K'$ induces the graph L_s in G, a contradiction. Similarly, in the latter case we have that $\{x_i\} \cup I \cup I'$ induces S_t in G, again a contradiction. This completes the proof of the claim. ◁

Claim 8 suggests the following notion. A *guess* is a function $R \colon V(H) \to 2^{N[X]}$ satisfying that:

- for each $v \in V(H)$, $R(v)$ is a subset of $N[X]$ such that $|R(v) \cap A_i| < \mathsf{Ramsey}(s,t)$ for all $i \in \{1,2,3\}$; and
- sets $R(v)$ are pairwise disjoint for different $v \in V(H)$.

Let $\mathcal{R}$ be the family of all possible guesses; then we easily have the following.

▷ Claim 9. We have that $|\mathcal{R}| \leqslant n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$ and $\mathcal{R}$ can be enumerated in time $n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$.

Proof. For each $v \in V(H)$, the number of choices for $R(v)$ in a guess R is bounded by $2^3 \cdot n^{3\cdot\mathsf{Ramsey}(s,t)}$: the first factor corresponds to the choice of $R(v) \cap X$, while the second factor bounds the number of choices of $R(v) \cap A_i$ for $i \in \{1,2,3\}$. Since the guess R is determined by choosing $R(v)$ for each $v \in V(H)$ and $|V(H)|$ is considered a constant, the number of different guesses is bounded by $\left(2^3 \cdot n^{3\cdot\mathsf{Ramsey}(s,t)}\right)^{|V(H)|} = n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$. Clearly, they can be also enumerated in time $n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$. ◁

Now, we say that a guess R is *compatible* with a solution ϕ to (G, rev) if the following conditions hold for every $v \in V(H)$:

(C1) $R(v) \subseteq \phi^{-1}(v)$;
(C2) $R(v) \cap X = \phi^{-1}(v) \cap X$; and
(C3) for all $i \in \{1,2,3\}$ and $u \in A_{>i}$, if u has a neighbor in $\phi^{-1}(v) \cap A_i$, then u also has a neighbor in $R(v) \cap A_i$.

The following statement follows immediately from Claim 8.

▷ Claim 10. For every solution ϕ to the instance (G, rev), there exists a guess $R \in \mathcal{R}$ that is compatible with ϕ.

Consider a guess $R \in \mathcal{R}$. We define a set $B^R \subseteq V(G) \times V(H)$ of *disallowed pairs* for R as follows. We include a pair $(u,v) \in V(G) \times V(H)$ in B^R if any of the following conditions holds:

(D1) $u \in X$ and $u \notin R(v)$;
(D2) $u \in R(v')$ for some $v' \in V(H)$ that is different from v;
(D3) u has a neighbor in G that belongs to $R(v')$ for some $v' \in V(H)$ such that $vv' \notin E(H)$; or
(D4) $u \in A_i - R(v)$ for some $i \in \{1,2,3\}$ and there exists $u' \in A_{>i}$ such that $uu' \in E(G)$ and $N_G(u') \cap A_i \cap R(v) = \emptyset$.

Intuitively, B^R contains assignments that contradict the supposition that R is compatible with a considered solution.

Based on B^R, we define a new revenue function $\mathsf{rev}^R \colon V(G) \times V(H) \to \mathbb{R}$ as follows:

$$\mathsf{rev}^R(u,v) = \begin{cases} -1 & \text{if } (u,v) \in B^R; \\ \mathsf{rev}(u,v) & \text{otherwise.} \end{cases}$$

The intuition is that disallowing a pair (u,v) is modelled by assigning a negative revenue to the corresponding assignment. This forbids optimum solutions from using this assignment.

We define a subgraph G' of G as follows: $V(G') \coloneqq V(G)$ and $E(G')$ comprises all edges of G whose both endpoints belong to the same set A_i, for some $i \in \{1,2,3,4\}$. Thus, in G' the vertices of X are isolated, and there are no edges between any A_i and A_j for $i \neq j$, nor between any A_i and X. For every guess $R \in \mathcal{R}$, we may consider a new instance (G', rev^R) of Max Partial H-Coloring. In the following two claims we establish the relationship between solutions to the instance (G, rev) and solutions to instances (G', rev^R) for $R \in \mathcal{R}$. The proofs essentially boil down to a verification that all the previous definitions work as expected. In particular, the key point is that the modification of revenues applied when constructing rev^R implies automatic satisfaction of all the constraints associated with edges that were present in G, but got removed in G'.

▷ Claim 11. For every guess $R \in \mathcal{R}$, every optimum solution ϕ to the instance (G', rev^R) is also a solution to the instance (G, rev), and moreover $\mathsf{rev}^R(\phi) = \mathsf{rev}(\phi)$.

Proof. Recall that ϕ is a solution to (G, rev) if and only if ϕ is a partial H-coloring of G. Hence, we need to prove that for every $uu' \in E(G)$ with $u, u' \in \mathsf{dom}\,\phi$, we have $\phi(u)\phi(u') \in E(H)$. Denote $v \coloneqq \phi(u)$ and $v' \coloneqq \phi(u')$ and suppose for contradiction that $vv' \notin E(H)$. Since ϕ is an optimum solution to (G', rev^R), we have $\mathsf{rev}^R(u, v) \geqslant 0$, which implies that $(u, v) \notin B^R$. Similarly $(u', v') \notin B^R$. We now consider cases depending on the alignment of u and u' in G.

If $u, u' \in A_i$ for some $i \in \{1, 2, 3, 4\}$ then $uu' \in E(G')$, so the supposition $vv' \notin E(H)$ would contradict the assumption that ϕ is a solution to (G', rev^R).

Suppose $u \in A_i$ and $u' \in A_j$ for $i, j \in \{1, 2, 3, 4\}$, $i \neq j$; by symmetry, assume $i < j$. As $vv' \notin E(H)$, we infer that u' does not have any neighbors in $R(v)$ in G, for otherwise we would have $(u', v') \in B^R$ by (D3). As $uu' \in E(G)$, $u \in A_i$, and $u' \in A_{>i}$, this implies that $(u, v) \in B^R$ by (D4), a contradiction.

Finally, suppose that $\{u, u'\} \cap X \neq \emptyset$, say $u \in X$. Since $(u, v) \notin B^R$, by (D1) we infer that $u \in R(v)$. Then, by (D3), $vv' \notin E(H)$ and $uu' \in E(G)$ together imply that $(u', v') \in B^R$, a contradiction.

This completes the proof that ϕ is a solution to (G, rev). To see that $\mathsf{rev}^R(\phi) = \mathsf{rev}(\phi)$ note that ϕ, being an optimum solution to (G', rev^R), does not use any assignments with negative revenues in rev^R, while $\mathsf{rev}(u, v) = \mathsf{rev}^R(u, v)$ for all (u, v) satisfying $\mathsf{rev}^R(u, v) \geqslant 0$. ◁

▷ Claim 12. If ϕ is a solution to (G, rev) that is compatible with a guess $R \in \mathcal{R}$, then ϕ is also a solution to (G', rev^R) and $\mathsf{rev}^R(\phi) = \mathsf{rev}(\phi)$.

Proof. As ϕ is a solution to (G, rev), it is a partial H-coloring of G. Since G' is a subgraph of G with $V(G') = V(G)$, ϕ is also a partial H-coloring of G'. Hence ϕ is a solution to (G', rev^R).

To prove that $\mathsf{rev}^R(\phi) = \mathsf{rev}(\phi)$ it suffices to show that $(u, \phi(u)) \notin B^R$ for every $u \in \mathsf{dom}\,\phi$, since functions rev^R and rev differ only on the pairs from B^R. Suppose otherwise, and consider cases depending on the reason for including $(u, \phi(u))$ in B^R. Denote $v \coloneqq \phi(u)$.

First, suppose $u \in X$ and $u \notin R(v)$. By (C2) we have $u \notin R(v) \cap X = \phi^{-1}(v) \cap X \ni u$, a contradiction.

Second, suppose $u \in R(v')$ for some $v' \neq v$. By (C1) we have $v = \phi(u) = v'$, again a contradiction.

Third, suppose that u has a neighbor u' in G such that $u' \in R(v')$ for some $v' \in V(H)$ satisfying $vv' \notin E(H)$. By (C1), we have $u' \in \mathsf{dom}\,\phi$ and $\phi(u') = v'$. But then $\phi(u)\phi(u') = vv' \notin E(H)$ even though $uu' \in E(G)$, a contradiction with the assumption that ϕ is a partial H-coloring of G.

Fourth, suppose that $u \in A_i - R(v)$ for some $i \in \{1, 2, 3\}$ and there exists $u' \in A_{>i}$ such that $uu' \in E(G)$ and $N_G(u') \cap R(v) \cap A_i = \emptyset$. Observe that since $u \in A_i \cap \phi^{-1}(v)$ and $uu' \in E(G)$, by (C3) u' has a neighbor in $R(v) \cap A_i$ in the graph G. This contradicts the supposition that $N_G(u') \cap R(v) \cap A_i = \emptyset$.

As in all the cases we have obtained a contradiction, this concludes the proof of the claim. ◁

Let now $\Pi \coloneqq \{\mathsf{rev}^R \colon R \in \mathcal{R}\}$. Then, condition (C2) can be easily derived from Claim 11 and Claim 12, while condition (C1) is implied by Claim 7. Note here that G' is $\{P_6, L_s, S_t\}$-free, because it is a disjoint union of induced subgraphs of G. Finally, from Claim 9 we infer that $|\Pi| = |\mathcal{R}| \leqslant n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$ and Π can be constructed in time $n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$, because given $R \in \mathcal{R}$ it is straightforward to construct rev^R in polynomial time. Hence Π satisfies all the requested properties, and this completes the proof of Lemma 6.

5 Corollaries for subclasses of P_6-free graphs

In this section we prove results (R3) and (R4) promised in Section 1. The idea is to apply Lemma 6 exhaustively, until the considered instance becomes trivial. The main point is that with each application the clique number of the graph drops, hence we naturally obtain an upper bound of the form $n^{f(\omega(G))}$ for the total size of the recursion tree, hence also on the running time.

The following statement captures the idea of exhaustive applying Lemma 6 in a recursive scheme. For convenience, we formulate the statement so that s and t are given on input.

▶ **Theorem 13.** *Let H be a fixed irreflexive pattern graph. There exists an algorithm that given $s,t \in \mathbb{N}$ and an instance (G, rev) of Max Partial H-Coloring where G is $\{P_6, L_s, S_t\}$-free, solves this instance in time $n^{\mathcal{O}(\mathsf{Ramsey}(s,t)\cdot\omega(G))}$.*

Proof. If G is not connected, then for every connected component C of G we apply the algorithm recursively to $(C, \mathsf{rev}|_{V(C)})$. If ϕ_C is the obtained optimum solution to this instance, we may output $\phi := \bigcup_C \phi_C$. It is clear that ϕ constructed in this way is an optimum solution to (G, rev).

Assume then that G is connected. If G consists of only one vertex, say u, then we may simply output $\phi := \{(u,v)\}$ where v maximizes $\mathsf{rev}(u,v)$, or $\phi := \emptyset$ if $\mathsf{rev}(\cdot)$ has no positive value in its range. Hence, assume that G has at least two vertices, in particular $\omega(G) \geqslant 2$. We now apply Lemma 6 to G. Thus, in time $n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$ we obtain a subgraph G' of G with $V(G) = V(G')$ and a suitable set of revenue functions Π satisfying $|\Pi| \leqslant n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$. Recall here that G' is $\{P_6, L_s, S_t\}$-free. Moreover, if we set $F = K_{\omega(G)}$ then G is $F^\bullet$-free, so Lemma 6 implies that G' is F-free. This means that $\omega(G') < \omega(G)$.

Next, for every $\mathsf{rev}' \in \Pi$ we recursively solve the instance (G', rev'). Lemma 6 implies that if among the obtained optimum solutions to instances (G', rev') we pick the one with the largest revenue, then this solution is also an optimum solution to (G, rev).

We are left with analyzing the running time. Recall that every time we recurse into subproblems constructed using Lemma 6, the clique number of the currently considered graph drops by at least one. Since recursing on a disconnected graph yields connected graphs in subproblems, we conclude that the total depth of the recursion tree is bounded by $2\cdot\omega(G)$. In every recursion step we branch into $n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$ subproblems, hence the total number of nodes in the recursion tree is bounded by $\left(n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}\right)^{2\cdot\omega(G)} = n^{\mathcal{O}(\mathsf{Ramsey}(s,t)\cdot\omega(G))}$. The internal computation in each subproblem take time $n^{\mathcal{O}(\mathsf{Ramsey}(s,t))}$, hence the total running time is indeed $n^{\mathcal{O}(\mathsf{Ramsey}(s,t)\cdot\omega(G))}$. ◀

Note that since both L_3 and S_2 contain P_5 as an induced subgraph, every P_5-free graph is $\{P_6, L_3, S_2\}$-free. Hence, from Theorem 13 we may immediately conclude the following statement, where the setting of P_5-free graphs is covered by the case $s = 3$ and $t = 2$.

▶ **Corollary 14.** *For any fixed $s,t \in \mathbb{N}$ and irreflexive pattern graph H, Max Partial H-Coloring can be solved in $\{P_6, L_s, S_t\}$-free graphs in time $n^{\mathcal{O}(\omega(G))}$. This in particular applies to P_5-free graphs.*

Next, we observe that the statement of Theorem 13 can be also used for non-constant s to obtain an algorithm for the case when the graph L_s is not excluded.

▶ **Corollary 15.** *For any fixed $t \in \mathbb{N}$ and irreflexive pattern graph H, Max Partial H-Coloring can be solved in $\{P_6, S_t\}$-free graphs in time $n^{\mathcal{O}(\omega(G)^t)}$.*

Proof. Observe that since the graph L_s contains a clique of size s, every graph G is actually $L_{\omega(G)+1}$-free. Therefore, we may apply the algorithm of Theorem 13 for $s := \omega(G)+1$. Note here that $\omega(G)$ can be computed in time $n^{\omega(G)+\mathcal{O}(1)}$ by verifying whether G has cliques of size $1, 2, 3, \ldots$ up to the point when the check yields a negative answer. Since for $s = \omega(G)+1$ and fixed t we have

$$\mathsf{Ramsey}(s,t) = \binom{s+t-2}{t-1} \leqslant \mathcal{O}(\omega(G)^{t-1}),$$

the obtained running time is $n^{\mathcal{O}(\mathsf{Ramsey}(s,t)\cdot\omega(G))} \leqslant n^{\mathcal{O}(\omega(G)^t)}$. ◀

Let us note that an algorithm with running time $n^{\mathcal{O}(\omega(G)^\alpha)}$, for some constant α, can be used within a simple branching strategy to obtain a subexponential-time algorithm.

▶ **Lemma 16.** *Let H be a fixed irreflexive graph and suppose Max Partial H-Coloring can be solved in time $n^{\mathcal{O}(\omega(G)^\alpha)}$ on $\mathcal{F}$-free graphs, for some family of graphs $\mathcal{F}$ and some constant $\alpha \geqslant 1$. Then Max Partial H-Coloring can be solved in time $n^{\mathcal{O}(n^{\alpha/(\alpha+1)})}$ on $\mathcal{F}$-free graphs.*

Proof. Let (G, rev) be the input instance, where G has n vertices. We define threshold $\tau := \left\lfloor n^{\frac{1}{\alpha+1}} \right\rfloor$.

The algorithm first checks whether G contains a clique on τ vertices. This can be done in time $n^{\tau+\mathcal{O}(1)} \leqslant n^{\mathcal{O}(n^{1/(\alpha+1)})}$ by verifying all subsets of τ vertices in G. If there is no such clique then $\omega(G) < \tau$, so we can solve the problem using the assumed algorithm in time $n^{\mathcal{O}(\omega(G)^\alpha)} \leqslant n^{\mathcal{O}(\tau^\alpha)} \leqslant n^{\mathcal{O}(n^{\alpha/(\alpha+1)})}$. Hence, suppose that we have found a clique K on τ vertices.

Observe that since H is irreflexive, in any partial H-coloring ϕ of G only at most $|V(H)|$ vertices of K can be colored, that is, belong to $\mathsf{dom}\,\phi$. We recurse into $\binom{\tau}{\leqslant |V(H)|} \leqslant n^{|V(H)|}$ subproblems: in each subproblem we fix a different subset $A \subseteq K$ with $|A| \leqslant |V(H)|$ and recurse on the graph $G_A := G - (K - A)$ with revenue function $\mathsf{rev}_A := \mathsf{rev}|_{V(G_A)}$. Note here that G_A is $\mathcal{F}$-free. From the above discussion it is clear that $\mathrm{OPT}(G, \mathsf{rev}) = \max_{A\subseteq K, |A|\leqslant|V(H)|} \mathrm{OPT}(G_A, \mathsf{rev}_A)$. Therefore, the algorithm may return the solution with the highest revenue among those obtained in recursive calls.

As for the running time, observe that in every recursive call, the algorithm either solves the problem in time $n^{\mathcal{O}(n^{\alpha/(\alpha+1)})}$, or recurses into $n^{|V(H)|} = n^{\mathcal{O}(1)}$ subcalls, where in each subcall the vertex count is decremented by at least $\left\lfloor n^{\frac{1}{\alpha+1}} \right\rfloor$. It follows that the depth of the recursion is bounded by $\mathcal{O}(n^{\alpha/(\alpha+1)})$, hence the total number of nodes in the recursion tree is at most $n^{\mathcal{O}(n^{\alpha/(\alpha+1)})}$. Since the time used for each node is bounded by $n^{\mathcal{O}(n^{\alpha/(\alpha+1)})}$, the total running time of $n^{\mathcal{O}(n^{\alpha/(\alpha+1)})}$ follows. ◀

By combining Corollary 14 and Corollary 15 with Lemma 16 we conclude the following.

▶ **Corollary 17.** *For any fixed $s, t \in \mathbb{N}$ and irreflexive pattern graph H, Max Partial H-Coloring can be solved in*

1. *$\{P_6, L_s, S_t\}$-free graphs in time $n^{\mathcal{O}(\sqrt{n})}$ (this in particular applies to P_5-free graphs),*
2. *$\{P_6, S_t\}$-free graphs in time $n^{\mathcal{O}\left(n^{t/(t+1)}\right)}$.*

6 Open problems

The following question, which originally motivated our work, still remains unresolved.

▶ **Question 1.** *Is* Odd Cycle Transversal *polynomial-time solvable in P_5-free graphs?*

Note that our work stops short of giving a positive answer to this question: we give an algorithm with running time $n^{\mathcal{O}(\omega(G))}$, a subexponential-time algorithm, and polynomial time algorithms for the cases when either a threshold graphs or a bull is additionally forbidden. Therefore, we are hopeful that the answer to the question is indeed positive.

One aspect of our work that we find particularly interesting is the possibility of treating the clique number $\omega(G)$ as a progress measure for an algorithm, which enables bounding the recursion depth in terms of $\omega(G)$. This approach naturally leads to algorithms with running time of the form $n^{f(\omega(G))}$ for some function f, that is, polynomial-time for every fixed clique number. By Lemma 16, having a polynomial function f in the above gives a subexponential-time algorithm, at least in the setting of Max Partial H-Coloring for irreflexive H. However, looking for algorithms with time complexity $n^{f(\omega(G))}$ seems to be another relaxation of the goal of polynomial-time solvability, somewhat orthogonal to subexponential-time algorithms [4, 6, 19] or approximation schemes [10]. Note that our work and the recent work of Brettell et al. [7] actually show two different methods of obtaining such algorithms: using direct recursion, or via dynamic programming on branch decompositions of bounded mim-width. It would be interesting to investigate this direction in the context of Maximum Independent Set in P_t-free graphs. A concrete question would be:

▶ **Question 2.** *Is there a polynomial-time algorithm for* Maximum Independent Set *in $\{P_t, K_t\}$-free graphs, for every fixed t?*

In all our algorithms, we state the time complexity assuming that the pattern graph H is fixed. This means that the constants hidden in the $\mathcal{O}(\cdot)$ notation in the exponent may – and do – depend on the size of H. In the language of parameterized complexity, this means that we give XP algorithms for the parameterization by the size of H. It is natural to ask whether this state of art can be improved to the existence of FPT algorithms, that is, with running time $f(H) \cdot n^c$ for some computable function f and universal constant c, independent of H. This is not known even for the case of k-Coloring P_5-free graphs, so let us re-iterate the old question of Hoàng et al. [22].

▶ **Question 3.** *Is there an* FPT *algorithm for k-*Coloring *in P_5-free graphs parameterized by k?*

While the above question seems hard, it is conceivable that FPT results could be derived in some more restricted settings considered in this work, for instance for $\{P_5, \text{bull}\}$-free graphs.

References

1 Tara Abrishami, Maria Chudnovsky, Marcin Pilipczuk, Paweł Rzążewski, and Paul Seymour. Induced subgraphs of bounded treewidth and the container method. *CoRR*, abs/2003.05185, 2020. arXiv:2003.05185.

2 Vladimir E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982. (in Russian).

3 Vladimir E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discret. Appl. Math.*, 135(1-3):3–16, 2004. doi:10.1016/S0166-218X(02)00290-1.

4 Gábor Bacsó, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Zsolt Tuza, and Erik Jan van Leeuwen. Subexponential-time algorithms for Maximum Independent Set in P_t-free and broom-free graphs. *Algorithmica*, 81(2):421–438, 2019. doi:10.1007/s00453-018-0479-5.

5 Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-coloring and list three-coloring of graphs without induced paths on seven vertices. *Combinatorica*, 38(4):779–801, 2018. doi:10.1007/s00493-017-3553-8.

6 Christoph Brause. A subexponential-time algorithm for the Maximum Independent Set problem in P_t-free graphs. *Discret. Appl. Math.*, 231:113–118, 2017. doi:10.1016/j.dam.2016.06.016.

7 Nick Brettell, Jake Horsfield, and Daniël Paulusma. Colouring $sP_1 + P_5$-free graphs: a mim-width perspective. *CoRR*, abs/2004.05022, 2020. arXiv:2004.05022.

8 Maria Chudnovsky, Jason King, Michal Pilipczuk, Paweł Rzążewski, and Sophie Spirkl. Finding large H-colorable subgraphs in hereditary graph classes. *CoRR*, abs/2004.09425, 2020. arXiv:2004.09425.

9 Maria Chudnovsky, Daniël Paulusma, and Oliver Schaudt. Graph colouring: from structure to algorithms (dagstuhl seminar 19271). *Dagstuhl Reports*, 9(6):125–142, 2019. doi:10.4230/DagRep.9.6.125.

10 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the Maximum Weight Independent Set problem in H-free graphs. In *Proceedings of the* 31^{st} *ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2260–2278. SIAM, 2020. doi:10.1137/1.9781611975994.139.

11 Maria Chudnovsky and Shmuel Safra. The Erdős-Hajnal conjecture for bull-free graphs. *J. Comb. Theory, Ser. B*, 98(6):1301–1310, 2008. doi:10.1016/j.jctb.2008.02.005.

12 Maria Chudnovsky, Oliver Schaudt, Sophie Spirkl, Maya Stein, and Mingxian Zhong. Approximately coloring graphs without long induced paths. *Algorithmica*, 81(8):3186–3199, 2019. doi:10.1007/s00453-019-00577-6.

13 Maria Chudnovsky and Vaidy Sivaraman. Odd holes in bull-free graphs. *SIAM J. Discrete Math.*, 32(2):951–955, 2018. doi:10.1137/17M1131301.

14 Maria Chudnovsky, Sophie Spirkl, and Mingxian Zhong. Four-coloring P_6-free graphs. In *Proceedings of the* 30^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1239–1256. SIAM, 2019. doi:10.1137/1.9781611975482.76.

15 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

16 Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, Giacomo Paesani, Daniël Paulusma, and Paweł Rzążewski. On cycle transversals and their connected variants in the absence of a small linear forest. *CoRR*, abs/1908.00491, 2019. Accepted to Algorithmica. arXiv:1908.00491.

17 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015. doi:10.1137/140964801.

18 Petr A. Golovach, Daniël Paulusma, and Jian Song. Closing complexity gaps for coloring problems on H-free graphs. *Inf. Comput.*, 237:204–214, 2014. doi:10.1016/j.ic.2014.02.004.

19 Carla Groenland, Karolina Okrasa, Paweł Rzążewski, Alex D. Scott, Paul D. Seymour, and Sophie Spirkl. H-colouring P_t-free graphs in subexponential time. *Discret. Appl. Math.*, 267:184–189, 2019. doi:10.1016/j.dam.2019.04.010.

20 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for Maximum Weight Independent Set on P_6-free graphs. In *Proceedings of the* 30^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1257–1271. SIAM, 2019. doi:10.1137/1.9781611975482.77.

21 Gregory Z. Gutin, Pavol Hell, Arash Rafiey, and Anders Yeo. A dichotomy for minimum cost graph homomorphisms. *Eur. J. Comb.*, 29(4):900–911, 2008. doi:10.1016/j.ejc.2007.11.012.

22 Chính T Hoàng, Marcin Kamiński, Vadim Lozin, Joe Sawada, and Xiao Shu. Deciding k-colorability of P_5-free graphs in polynomial time. *Algorithmica*, 57(1):74–81, 2010.

23 Shenwei Huang. Improved complexity results on k-coloring P_t-free graphs. *Eur. J. Comb.*, 51:336–346, 2016. doi:10.1016/j.ejc.2015.06.005.

24 Daniel Lokshtanov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5-free graphs in polynomial time. In *Proceedings of the* 25^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 570–581. SIAM, 2014. doi:10.1137/1.9781611973402.43.

25 Vadim V. Lozin and Martin Milanič. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *J. Discrete Algorithms*, 6(4):595–604, 2008. doi:10.1016/j.jda.2008.04.001.

26 George J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory, Ser. B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.

27 Jana Novotná, Karolina Okrasa, Michał Pilipczuk, Paweł Rzążewski, Erik Jan van Leeuwen, and Bartosz Walczak. Subexponential-time algorithms for finding large induced sparse subgraphs. In *Proceedings of the* 14^{th} *International Symposium on Parameterized and Exact Computation, IPEC 2019*, volume 148 of *LIPIcs*, pages 23:1–23:11. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.23.

28 Karolina Okrasa and Paweł Rzążewski. Subexponential algorithms for variants of the homomorphism problem in string graphs. *J. Comput. Syst. Sci.*, 109:126–144, 2020. doi:10.1016/j.jcss.2019.12.004.

29 Najiba Sbihi. Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics*, 29(1):53–76, 1980. (in French).

Compact Oblivious Routing in Weighted Graphs

Philipp Czerner
Department of Informatics, TU München, Germany
czerner@in.tum.de

Harald Räcke
Department of Informatics, TU München, Germany
raecke@in.tum.de

Abstract

The space-requirement for routing-tables is an important characteristic of routing schemes. For the cost-measure of minimizing the total network load there exist a variety of results that show tradeoffs between stretch and required size for the routing tables. This paper designs compact routing schemes for the cost-measure congestion, where the goal is to minimize the maximum relative load of a link in the network (the relative load of a link is its traffic divided by its bandwidth). We show that for arbitrary undirected graphs we can obtain oblivious routing strategies with competitive ratio $\tilde{\mathcal{O}}(1)$ that have header length $\tilde{\mathcal{O}}(1)$, label size $\tilde{\mathcal{O}}(1)$, and require routing-tables of size $\tilde{\mathcal{O}}(\deg(v))$ at each vertex v in the graph.

This improves a result of Räcke and Schmid who proved a similar result in *unweighted* graphs.

2012 ACM Subject Classification Theory of computation → Network flows; Networks → Network algorithms

Keywords and phrases Oblivious Routing, Compact Routing, Competitive Analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.36

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.02427`.

1 Introduction

Oblivious routing strategies choose routing paths independent of the traffic in the network and are therefore usually much easier to implement than adaptive routing solutions that might require centralized control and/or lead to frequent reconfigurations of traffic routes. Because of this simplicity a lot of research in recent years has been performed on the question whether the quality of route allocations performed by oblivious algorithms is comparable to that of adaptive solutions (see e.g. [2, 5, 6, 17, 19, 20, 21, 26]). For some cost-metrics this is indeed the case. For example for minimizing the total traffic in the network (a.k.a. total load), shortest path routing is a simple optimal oblivious strategy. When one aims to minimize the congestion, i.e., the maximum (relative) load of a network link, one can still obtain strategies with a competitive ratio of $\mathcal{O}(\log n)$, i.e., the congestion generated by these strategies is at most an $\mathcal{O}(\log n)$-factor than the best possible congestion [21].

However, another important aspect for implementing oblivious routing strategies on large networks is the size of the required routing tables. This aspect has been investigated thoroughly for the cost-measure total load (see e.g. [7, 9, 11, 18, 24, 25, 29]), and various trade-offs between competitive ratio (also called stretch for the total load scenario) and the table-size have been discovered.

If for example every vertex stores the next hop on a shortest path to a target one can obtain a stretch of 1 at the cost of having routing tables of size $\mathcal{O}(n \log n)$ per node. If one allows non-optimal solutions Thorup and Zwick [25] have shown how to obtain a stretch of $4k - 5$ for any $k > 2$ with routing tables of size $\tilde{\mathcal{O}}(n^{1/k})$. This routing scheme works for the so-called labeled scenario in which the designer of the routing-scheme is allowed to relabel the vertices of the network in order to make routing decisions easier. Of course, there is still a restriction on the label-size as otherwise the power of being able to assign labels to vertices could be abused.

In the (more difficult) so-called *name-independent* model the designer is not allowed to relabel the vertices. Abraham et al. [1] have shown that for general undirected graphs one can asymptotically match the bounds for the labeled variant. They obtain a stretch of $\mathcal{O}(k)$ and routing tables of size $\tilde{\mathcal{O}}(n^{1/k})$. If a famous conjecture due to Erdős [8] about the existence of low-girth graphs holds then there is also a lower bound that says that obtaining a stretch better than $2k+1$ requires routing tables of size $\Omega(n^{1/k})$. This means that for general undirected graphs the existing tradeoffs between stretch and space are fairly tight.

There exist many more results that analyze problem variants as e.g. obtained by restricting the graph representing the network (see e.g. [18, 7, 10, 12, 13, 24]); so the problem of designing compact routing schemes is very well studied for the cost-measure *total load*.

However, for the cost-measure congestion this is not the case. Räcke and Schmid [22] gave the first oblivious routing scheme that combines a guarantee w.r.t. the congestion with small routing-tables. They consider the labeled model and design an oblivious routing scheme that for a general undirected, unweighted graph G requires routing tables of size $\tilde{\mathcal{O}}(\deg(v))$ at each vertex v and obtains a competitive ratio of $\tilde{\mathcal{O}}(1)$ w.r.t. congestion.

There are important differences when comparing this result to its counter-parts for the total-load scenario. Firstly, the space used at a vertex v may depend on the degree of v. This is a reasonable assumption from a practical perspective as a node corresponds to a router in the network and it is reasonable to assume that the memory at a router (node) grows with the number of ports (number of incident edges). However, this assumption seems also crucial for getting any reasonable guarantees. In order to minimize congestion it is important to distribute the traffic among all network resources. It seems very difficult to do this if the routing table at a vertex is a lot smaller than the number of outgoing edges.

Another difference is that there is no tradeoff parameter k that gives a smooth transition from optimal routing with large tables to more compact routing. The reason is that for *congestion* the competitive ratio may be $\Omega(\log n)$ even for unlimited routing tables [3].

One important shortcoming of the result by Schmid and Räcke [22] is that it only applies to unweighted graphs (there is a straightforward generalization that obtains routing tables of size $\mathcal{O}(W \operatorname{polylog} n)$ where W is the largest weight of an edge, but this is undesirable). This restriction is due to the fact that the result by Schmid and Räcke uses *paths* to route within well-connected clusters, which they obtain by randomized rounding.

There are two major obstacles in generalizing this result to the weighted case:

(1) In unweighted graphs, low congestion also ensures a low number of paths using an edge. However, an edge of weight W might be used by W small paths, which cannot be stored in a compact manner.

(2) Even within a well-connected cluster, it is not sufficient to route a commodity using a small number of paths, if the nodes are connected by many low-weight edges (illustrated in Figure 1). Hence a source node may have to route (and store) W small paths.

In this paper we give a construction of an oblivious routing scheme that avoids both problems, by storing aggregate routing information for many paths at once, as well as distributing storage across nodes for commodities that need to spread out over multiple paths. In this manner we obtain a polylogarithmic competitive ratio with polylogarithmic space requirement per edge in the network. Our main result is the following.

▶ **Theorem 13.** *There is a compact oblivious routing scheme with competitive ratio at most* $\mathcal{O}(\log^6 n \log^3 W)$*, that uses routing tables of size* $\mathcal{O}(\log^5 n \log W \log^3(nW) \cdot \deg(v))$ *at a node* v*, packet headers of length* $\mathcal{O}(\log^3(nW))$*, and node labels of length* $\mathcal{O}(\log^2 n)$*.*

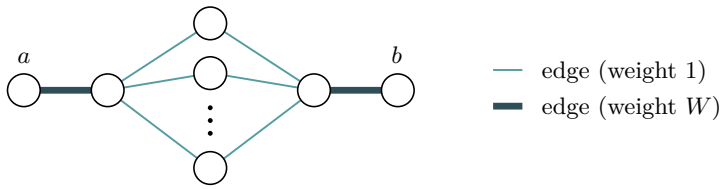


Figure 1 Routing a single demand over multiple edges. Sending data from a to b requires roughly W paths, but a has degree 1 and can store only $\tilde{\mathcal{O}}(1)$ bits.

In particular this result shows that if we can route some demand in a network with a multicommodity flow f of congestion C, then it is possible to route the demands *space-efficiently*, i.e., one can set up small routing tables so that packets follow a (maybe) different flow f' that routes the same demands with a slightly worse congestion. This question of space-efficiently routing demands in a network is orthogonal to oblivious routing and it is not clear by how much the performance (i.e., the congestion) degrades because of the space-requirement. The above theorem gives a polylogarithmic upper bound but to the best of our knowledge this problem has not been studied before.

1.1 Further Work

Oblivious routing with the goal of either minimizing the total load (or stretch), minimizing the congestion or a combination of both is a well studied problem. The research started with deterministic algorithms and it was shown by Borodin and Hopcroft [5] that on *any* bounded degree graph G for *any* deterministic routing scheme there exists a permutation routing instance that incurs congestion $\Omega(\sqrt{n}/\Delta^{3/2})$. This result was improved by Kaklamanis et al. [15] to a lower bound of $\Omega(\sqrt{n}/\Delta)$. As there exist bounded degree graphs that can route any permutation with small congestion this gives a large lower bound on the competitive ratio of deterministic oblivious routing schemes.

For randomized algorithms Valiant and Brebner [28] showed how to obtain a polylogarithmic competitive ratio for the hypercube by routing to random intermediate destinations (known as Valiant's trick).

Räcke [20] presented the first oblivious routing scheme with a polylogarithmic competitive ratio of $\mathcal{O}(\log^3 n)$ in general undirected networks. This routing scheme is based on a hierarchical decomposition of a graph and forms the basis for the compact routing schemes that we construct in this paper. The construction in [20] was not polynomial time. This drawback was independently addressed by Bienkowski et al. [4] and Harrelson et al. [14]. Both papers give a polynomial-time algorithm for constructing the hierarchical decomposition (and, hence, the routing scheme) – the first with a competitive ratio of $\mathcal{O}(\log^4 n)$, and the second with a competitive ratio of $\mathcal{O}(\log^2 n \log\log n)$.

In 2014 Räcke et al. [23] presented another construction of the hierarchy that runs in time $\mathcal{O}(m \operatorname{polylog} n)$ and guarantees a competitive ratio of $\mathcal{O}(\log^5 n)$ (however, going from the hierarchy to the actual routing scheme may require superlinear time).

The above oblivious routing schemes that are based on hierarchical tree decompositions do not give the best possible competitive ratio. In [21] Räcke presents an oblivious routing scheme that is based on embedding a convex combination of trees into the graph G. This scheme obtains a competitive ratio of $\mathcal{O}(\log n)$, which is optimal due to a lower bound of Bartal and Leonardi for online routing in grids [3].

However, the number of trees that are used in the above result [21] is fairly large ($\Theta(m)$). Therefore, it seems difficult to design a compact routing scheme based on the tree embedding approach, and, therefore we use the earlier results that are based on hierarchical decompositions (a single tree!) but only guarantee slightly weaker competitive ratios.

1.2 Preliminaries

Throughout the paper we use $G = (V, E, w)$ to denote an undirected weighted graph with n node and m edges. We will refer to the weight of an edge also as the *capacity* of the edge. Wlog. we assume that the minimum edge weight is 1, that edge-weights are a power of 2, and that the largest edge weight is W. We call an edge of capacity/weight 2^i a *class* i edge and use $N_{\text{class}} := 1 + \log_2 W$ to denote the total number of classes. Further, we use $\Gamma(v)$ to denote the neighborhood of a vertex v, i.e., $\Gamma(v) = \{u \in V \mid \{u, v\} \in E\}$.

The degree of a node v in the graph G will be referred to as $\deg_G(v)$, that is $\deg_G(v) := |\Gamma(v)|$. We apply that to directed graphs as well, where it refers to the number of outgoing edges.

While the edges E are undirected, it will be convenient to refer to a certain orientation of an edge, so we define $E_{\text{or}} := \{(u, v) \in V^2 : \{u, v\} \in E\}$. A mapping $f : E_{\text{or}} \to \mathbb{R}$ with $f((u, v)) = -f((v, u))$ for $(u, v) \in E_{\text{or}}$ is called a (single-commodity) flow. If $f(u, v) > 0$ for some edge $(u, v) \in E$, this indicates flow from u to v. The reverse flow of f is simply $-f$. For the sake of readability we omit double parentheses and write, e.g., $f(u, v)$ instead of $f((u, v))$.

A flow f may have multiple sources and sinks. The balance of a node $v \in V$ is denoted by $\text{bal}_f(v) := \sum_{u \in \Gamma(v)} f(u, v)$, so a positive balance indicates that the node is receiving more flow than sending out. A flow is *acyclic*, if there is no path $(p_0, ..., p_k)$ in G with $p_0 = p_k$ and $f(p_i, p_{i+1}) > 0$ for all i. Its congestion is the maximum ratio between the flow over an edge and its weight, denoted by $\text{cong}(f) := \max_{\{u,v\} \in E} |f(u, v)|/w(u, v)$. Given a multi-set of flows $F := \{f_1, f_2, ..., f_k\}$, its *total congestion* is $\text{cong}(F) := \max_{\{u,v\} \in E} \sum_k |f_k(u, v)|/w(u, v)$.

If a flow f has all flow originating at a single node s, i.e., $\text{bal}_f(s) \leq 0$ and $\text{bal}_f(u) \geq 0$ for $u \neq s$, we say that f is an s-flow. If additionally $\text{bal}_f(s) = -1$, we call f a *unit* s-flow. The set of all unit s-flows is denoted with $\text{flow}(s)$. If a flow f only sends from s to t, i.e., f is an s-flow and $-f$ is a t-flow we call f an s-t flow.

We use $\tilde{\mathcal{O}}$ to disregard logarithmic factors, so $g = \tilde{\mathcal{O}}(h)$ iff $g = \mathcal{O}(h \log^c(nW))$ for some constant c.

Oblivious Routing Scheme

Now we define the concept of an oblivious routing scheme. The idea is to fix a single flow between each pair of nodes (u, v), and then multiply that flow with the actual demand from u to v to get the route. This flow can be interpreted probabilistically or fractionally, so if we have $f(e) = \frac{1}{2}$ for some edge e it means that the probability of the packet being routed across edge e is $\frac{1}{2}$; or that half a packet travels along that edge. We will use both interpretations interchangeably.

▶ **Definition 1.** *An* oblivious routing scheme $S = (f_{u,v})_{u,v \in V}$ *consists of a unit u-v-flow for each pair of nodes $u, v \in V$. Given demands $d : V \times V \to \mathbb{R}_{\geq 0}$ the congestion of S w.r.t. d, denoted* $\text{cong}(S, d)$*, is the total congestion of the set of flows $\{d(u, v)f_{u,v} : u, v \in V\}$. The* competitive ratio *of S is* $\max_d \text{cong}(S, d)/\text{cong}_{\text{opt}}(d)$*, where* $\text{cong}_{\text{opt}}(d)$ *denotes the optimal congestion that can be obtained for demands d by any routing scheme.*

Defining a compact oblivious routing scheme formally is a bit more involved, as we have to clarify where information is stored and how it is used. Before we do so, we introduce the notion of a *routing algorithm*, which defines formally how packets are sent through the network. The intuition is that each packet carries a *packet header*, storing per-packet information. Each node stores a *routing table*, containing arbitrary information for the routing algorithm to use.

The routing algorithms forwards a packet in a local manner, meaning that it reads both the packet header and the routing table before choosing an outgoing edge on which to sent the packet. At the same time, it may modify the packet header. This procedure repeats, until routing algorithm indicates that the packet has reached its destination, by outputting no outgoing edge.

It remains to describe how the packet header is initialized. For oblivious routing schemes, we simply use the name of the target node as initial packet header. However, we will later define more general building blocks, namely transformation schemes. Hence a routing algorithm works with a set of abstract *input labels* as possible initial packet headers (and thus as input to the routing algorithm). For the purposes of a routing algorithm these are simply some set, but later definitions will describe their structure more concretely.

▶ **Definition 2.** *A* routing algorithm $A = (\mathcal{A}, \mathcal{L}, \mathcal{T})$ *is a tuple,* $\mathcal{L} \subset \{0,1\}^*$ *denoting a finite set of* input labels *and* $\mathcal{T} : V \to \{0,1\}^*$ *a* routing table *for each node. Additionally,* $\mathcal{A} : \mathcal{T}(V) \times \{0,1\}^* \to (E \cup \{\emptyset\}) \times \{0,1\}^*$ *is a (possibly randomized) algorithm, taking both a routing table and a packet header as input, which calculates both the outgoing edge (if any) and the new packet header.*

We remark that the outgoing edge given by $\mathcal{A}$ has to be encoded in some manner, and it must be adjacent to the node the routing table belongs to. The routing table $\mathcal{T}(v)$ for a node v can contain information about v, such as a list of adjacent nodes, so any straightforward encoding, e.g., the index in this list, will work.

Given a routing algorithm $A = (\mathcal{A}, \mathcal{L}, \mathcal{T})$, a start vertex $v \in V$, and an input label $l \in \mathcal{L}$, the above mechanism defines a process for probabilistically distributing packets from v to targets in the network. We use $A(v, l)$ to denote a flow that describes the associated routing paths. This is defined as follows: We inject a packet at v, with l as packet header and execute $\mathcal{A}$ until no outgoing edge is returned. Then $A(v, l)(e)$ is the probability that the packet is routed over e (note that $\mathcal{A}$ may be randomized).

A routing algorithm $A = (\mathcal{A}, \mathcal{L}, \mathcal{T})$ is *compact*, if packet headers and input labels have size $\tilde{\mathcal{O}}(1)$, and the routing table of a node $v \in V$ has size $\tilde{\mathcal{O}}(\deg(v))$.

Recall that an oblivious routing scheme corresponds to a routing algorithm where the input labels are names of nodes in the graph. Consequently, we say that such a scheme is compact if its routing algorithm is compact.

Formally, we assign a name to each vertex in the graph, which we call *node label*, i.e., we have some function $\text{node} : V \to \{0,1\}^*$. For an oblivious routing scheme $S = (f_{u,v})_{u,v \in V}$ we use the set of all node labels $\text{node}(V)$ as input labels, so the initial packet header is the node label of the target node.

We say that S is *compact* if there exists a compact routing algorithm $A = (\mathcal{A}, \text{node}(V), \mathcal{T})$ with $f_{u,v} = A(u, \text{node}(v))$. This definition matches the one used by Räcke and Schmid [22], although it is more explicit.

The main result of this paper is the existence of a compact oblivious routing scheme, with competitive ratio $\tilde{\mathcal{O}}(1)$.

Transformation Schemes

Our routing scheme will be composed out of several building blocks, which we call *transformation schemes*. Loosely speaking, they correspond to single-commodity flows which we are able to route.

We consider *distributions* or *weight functions* of the form $\mu : V \to \mathbb{R}_{\geq 0}$ that assign a non-negative weight to vertices in V. If we only specify a weight function on a subset $S \subseteq V$ we assume that it is 0 on $V \setminus S$. We use $\mu(S) := \sum_{v \in S} \mu(v)$ to denote the weight of a subset S, and $\mathbf{1}_v : V \to \mathbb{N}$ to denote the special weight function that has weight 1 on v and 0 elsewhere. For a distribution μ we use $\bar{\mu} := \frac{1}{\mu(V)}\mu$ to denote the corresponding *normalized distribution*.

▶ **Definition 3.** *A* (compact[1]) transformation scheme (TS) *is a compact routing algorithm with a single input label.*

The above definition is not very useful by itself. The underlying idea is that we view a transformation scheme TS as on operation to transform one distribution of packets into another, by executing the routing algorithm. More precisely, given P packets each packet follows the flow $TS(v)$ at its source location $v \in V$. This will send it to some target node (probabilistically, according to $TS(v)$).

We say that a transformation scheme routes from some *input distribution* μ_{in} to an *output distribution* μ_{out}, if the above process transforms a set of P packets that are distributed according to $\bar{\mu}_{\text{in}}$ (i.e., a node v has $\mu_{\text{in}}(v)/\mu_{\text{in}}(V) \cdot P$ packets in expectation) into a set of packets that are distributed according to $\bar{\mu}_{\text{out}}$, i.e., afterwards a node has $\mu_{\text{out}}/\mu_{\text{out}}(V) \cdot P$ packets in expectation.

In addition we associate a *demand* $d(TS)$ and *congestion* $\text{cong}(TS)$ with a transformation scheme TS. We say a tranformation scheme routes demand $d(TS)$ from μ_{in} to μ_{out} with congestion $\text{cong}(TS)$ if the expected load on an edge e for the above process is at most $\text{cong}(TS) \cdot w(e)$ when $P = d(TS)$ (we allow P to be non-integral).

Note that, of course, the input for a transformation scheme could be any packet distribution. However, the congestion of the scheme is stated w.r.t. some fixed input distribution μ_{in} (its *natural input distribution*) and some total demand $d(TS)$.

From the congestion-value for μ_{in} and its demand d, one can then deduce the congestion-value for other inputs. If we, e.g., use the transformation scheme on a demand d' that is distributed according to ν we experience congestion at most $\max_{v \in V} d'\bar{\nu}(v)/d\bar{\mu}_{\text{in}}(v)$.

To make our notation more concise, we write a statement like "*TS routes* μ_{in} *to* μ_{out} *with demand* d *and congestion at most* C" as "*TS routes* $\mu_{\text{in}} \xrightarrow{d} \mu_{\text{out}}$ with congestion (at most) C". We omit the demand d if it equals 1.

It may happen that for some transformation scheme TS we cannot exactly specify the output distribution that corresponds to its natural input distribution μ_{in}. We say TS routes $\mu_{\text{in}} \to \mu_{\text{out}}$ with *approximation* σ if the real output distribution μ'_{out} fulfills $\bar{\mu}_{\text{out}}(v)/\sigma \leq \bar{\mu}'_{\text{out}}(v) \leq \bar{\mu}_{\text{out}} \cdot \sigma$.

Finally, in some proofs we will view packets as discrete entities and specify that the transformation scheme does not split them up.

However, this collides with the fractional nature of the transformation scheme, which is caused by randomization. Therefore we introduce the following definition of a deterministic transformation scheme, that extracts this randomness and makes it explicit.

[1] As all of our transformation schemes are compact (the later variants of deterministic and concurrent transformation schemes will be as well), we may drop the 'compact' when appropriate.

▶ **Definition 4.** *A* (compact) deterministic transformation scheme $TS = (\mathcal{A}, \mathcal{P}(V), \mathcal{T})$ *is a compact routing algorithm where* $\mathcal{A}$ *is deterministic and* $\mathcal{P}(v) = \{1, ..., N_v\}$ *is a set of* path-ids *valid for a node* $v \in V$*.*

The idea of the above definition is that we can specify a"random seed" as input label, which will determine precisely how a packet is routed. The ordinary transformation scheme will correspond to choosing an input label u.a.r. from sets $\mathcal{P}(v)$. In this sense the above definition makes the random choices of a transformation scheme explicit.

Note that, technically, the definition of routing algorithms allows any path id in $\mathcal{P}(V)$ to be specified at a node v, not only the ones in $\mathcal{P}(v)$. We ensure that this does not occur.

As $\mathcal{A}$ is deterministic, the path id indeed determines the exact route a packet will take when starting at a certain node. More precisely, each flow $TS(v, l)$ for $v \in V, l \in \{1, ..., N_v\}$ is simply a path starting at v. Still, there is no guarantee that different path-ids send the packet to the same node.

We associate a transformation scheme with each deterministic TS, by choosing the path-id uniformly at random. In this fashion we extend the notions, such as congestion, input/output distributions, etc., that were defined above for ordinary transformation schemes to also cover deterministic transformation schemes.

Concurrent Transformation Schemes

While a transformation scheme can mix packets arbitrarily, often we want to distribute several commodities at the same time, with separate input and output distributions for each commodity. This allows us to analyze the congestion more precisely and aggregate the routing information for different commodities. Hence we define the notion of a *concurrent transformation scheme*, which executes multiple transformation schemes in parallel.

The idea is that we take a transformation scheme and additionally specify a commodity as input.

▶ **Definition 5.** *Let* I *denote a set of commodities. A* (compact) concurrent [deterministic] transformation scheme (CTS) *is a compact routing algorithm* $TS = (\mathcal{A}, I \times \mathcal{L}, \mathcal{T})$*, s.t.* $TS_i := (\mathcal{A}, \{i\} \times \mathcal{L}, \mathcal{T})$ *is a [deterministic] transformation scheme for each commodity* $i \in I$*.*

Note that transformation schemes have a single input label, in which case the $\mathcal{L}$ in the above definition is superfluous and the input to the concurrent transformation scheme is just the commodity. If it is deterministic, we need the path id as input, and $\mathcal{L}$ would be the set of possible path ids. Similar to before, any combination of commodity and path id may be specified at a node, according to the definition of a routing algorithm, but for our purpose only some of these make sense.

The congestion of such a concurrent transformation scheme is defined as follows. Let μ_i and d_i denote the input distribution and demand of TS_i, respectively. Let $X_i(e)$ denote the expected load on an edge e if we execute TS_i on d_i packets distributed according to μ_i. The congestion of the CTS D is defined as $\text{cong}(D) := \max_e \frac{1}{w(e)} \sum_i X_i(e)$.

As an input to the routing algorithm, a commodity $i \in I$ has to be encoded in some fashion. Often, the commodity is determined by the source node (i.e., for each node v at most one input distributions μ_i is nonzero) and does not need to be specified. Otherwise, we will explicitly describe the necessary encoding as a property of the CTS.

Analogous to transformation schemes, we write "TS *routes* $\mu_i \xrightarrow{d_i} \nu_i$ with congestion (at most) C" for each commodity $i \in I$ for a CTS, and extend this notation to deterministic CTS by considering the associated transformation schemes.

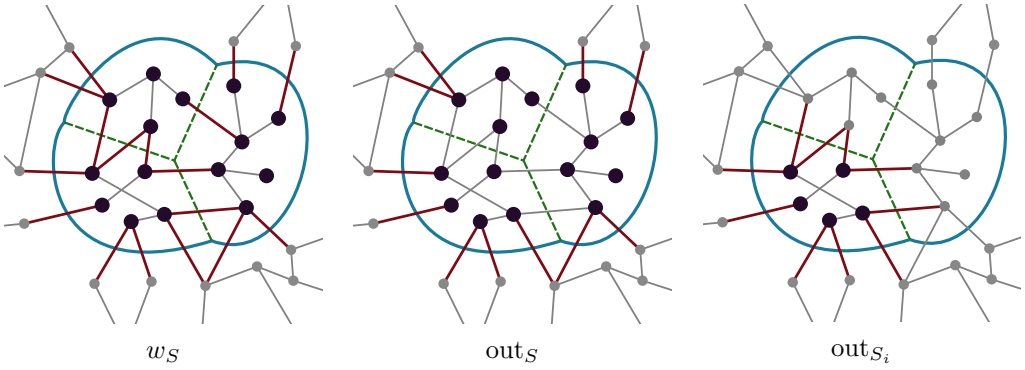


Figure 2 Distributions inside a cluster S with child S_i. The child clusters are separated by dashed lines. The respective distribution is nonzero only on the highlighted nodes and counts the total weight of highlighted edges adjacent to a node.

2 Overview

In this section we give a high-level overview of the most important steps in our construction. The first part gives a rough outline of the general approach of routing along a decomposition tree that forms the basis for some oblivious routing schemes (e.g. [20, 4, 14]), and has also been used by Räcke and Schmid [22] to obtain compact routing schemes.

2.1 The Decomposition Tree

The result by Räcke and Schmid [22] as well as our extension of it use a decomposition tree, in particular the one described in [20]. We refer the reader to these for a more detailed description and just briefly mention the key ideas here. We start with a single cluster containing all nodes, and then further refine that until all clusters consist of just a single node. Hence we get a tree T where nodes are subsets of V, which we call *clusters*. The tree T has root V, i.e., the cluster containing all nodes, and leaves $\{v\}$ for each $v \in V$. For a cluster S with children $S_1, ..., S_r$ we have $S = S_1 \dot\cup ... \dot\cup S_r$, so the children are a partition of the parent. We use $\text{height}(T)$ to denote the maximum distance from any leaf to the root, and $\deg(T)$ to denote the largest number of children of any cluster.

Now we introduce a number of distributions, which will be important for routing within the decomposition tree. For any cluster S we define the *border-weight* $\text{out}_S : S \to \mathbb{N}$ by $\text{out}_S(v) := \sum_{u \notin S} w(v,u)$ for $v \in S$, counting the total weight of edges leaving the cluster adjacent to a node. Additionally, for any cluster S with child clusters $S_1, ..., S_r$ we define the *cluster-weight* $w_S : S \to \mathbb{N}$ as $w_S := \sum_i \text{out}_{S_i}$, which also counts edges between children of S. These distributions are shown schematically in Figure 2.

The decomposition from [20] has two essential properties:

- For each cluster S we can solve the multi-commodity flow problem with demands $d(u,v) := w_S(u)w_S(v)/w_S(S)$ for $u, v \in S$ with congestion $C \in \mathcal{O}(\log^2 n)$ *within* S, and
- the tree has logarithmic height, i.e., $\text{height}(T) \in \mathcal{O}(\log n)$.

The essential idea for oblivious routing is that in order to route between two nodes s and t in the graph we determine the path $\{s\} = S_1, S_2, \ldots, S_k = \{t\}$ in the tree and then we route a packet successively along this path by routing it from distribution $\bar w_{S_i}$ to distribution $\bar w_{S_{i+1}}$ for $i = 1, \ldots, k-1$. Note that distribution $\bar w_{S_1} = \bar w_{S_{\{s\}}} = \mathbf{1}_s$ and distribution $\bar w_{S_k} = \bar w_{S_{\{t\}}} = \mathbf{1}_t$, i.e., we indeed route from s to t.

Now suppose that the optimal congestion for the given demand d is $C_{\text{opt}}(d)$. How much demand does the above process induce for routing from w_{S_i} to w_S for a child-cluster S_i of some cluster S (for all packets)? Each packet that uses the tree edge (S_i, S) in its path has to leave the cluster S_i and thus create a load of 1 in out_{S_i}. Conversely, OPT has congestion $C_{\text{opt}}(d)$, i.e., a load of at most $\text{out}_{S_i}(S_i) \cdot C_{\text{opt}}(d)$ on out_{S_i}. Therefore the total demand that has to be routed for $w_{S_i} \to w_S$ is at most $\text{out}_{S_i}(S_i) \cdot C_{\text{opt}}(d) = w_S(S_i) \cdot C_{\text{opt}}(d)$. An analogous argument holds for sending from w_S to w_{S_i}.

Now, we define a CTS for every cluster S that concurrently routes $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ for all child-clusters with small congestion. If these schemes have congestion at most C then the overall competitive ratio of the compact oblivious routing scheme is $\text{height}(T)C$ as an edge is contained in at most $\text{height}(T)$ many clusters. Hence, we can restate our goal as follows. For every cluster S find

Mixing CTS
A CTS that routes $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ for each child S_i, with congestion $\tilde{\mathcal{O}}(1)$.

Unmixing CTS
A CTS that routes $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ for each child S_i, with congestion $\tilde{\mathcal{O}}(1)$.

Here, we have to think about the encoding of the commodities, i.e., the indices of child clusters S_i. For our oblivious routing scheme we relabel the vertices so that the new name of a vertex v encodes the path from the root to the leaf $\{v\}$ in the decomposition tree. Then when we are given a packet with a source and a target node we can determine the path in the tree. For routing along an edge (S_i, S_{i+1}) of this path we extract the name of the child cluster and use this as commodity for the CTS. Furthermore, we will fix a specific name for each child cluster, incorporating a little bit of information for the CTS. (As in the scheme by Räcke and Schmid [22], the name will be the index in the list of child clusters, sorted by size.)

2.2 Constructing Transformation Schemes

In this section we give an overview of the steps for constructing transformation schemes that for some cluster S route $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ and $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ with small congestion. For this we use simplified versions of the main lemmata that are proven in the technical analysis in Section A. We mark these simplified version with a " $'$ ", so Lemma 3$'$ would be the simplified version of Lemma 3 in Section A.

Single-commodity flows

The first lemma that we show is how to construct a transformation scheme from a given flow, to route between *integral* distributions.

▶ **Lemma 3$'$** (Single-commodity flow routing). *Let f denote a flow with congestion at most $\mathcal{O}(\text{poly}(nW))$, and μ, μ' integral distributions with $\mu' - \mu = \text{bal}_f$.*[2] *Then there exists a compact, deterministic transformation scheme that routes $\mu \xrightarrow{\mu(V)} \mu'$ with congestion $\mathcal{O}(\text{cong}(f))$ and has $N_v := \mu(v)$ valid path-ids at node v.*

This means that if we are given a flow then we can construct a transformation scheme that allows us to send packets from sources (outgoing net-flow) to targets (incoming net-flow). Note that there is no guarantee which target a packet will be sent to if the flow contains several targets.

[2] We remark that due to flow conservation, $\mu(V) = \mu'(V)$ holds.

Product multicommodity flow

The second step of our approach is to obtain a concurrent transformation scheme that routes a *product multicommodity flow* (PMCF). Suppose that we are given a weight function $c : V \to \mathbb{N}$ on the vertices of the graph. We associate a multicommodity flow problem with this weight function by defining a demand $d(u, v) = c(u)c(v)/c(V)$ for any pair (u, v) of vertices. One can view this demand as each vertex u generating a flow of $c(u)$ and distributing it according to $\bar{c}$. Suppose that we can solve this multicommodity flow problem with some congestion $C = \mathcal{O}(\text{poly}(nW))$. We show that we can then obtain a CTS that routes a solution to the PMCF.

▶ **Lemma 6′** (PMCF-routing). *Given a graph G together with a weight function $w : V \to \mathbb{N}$ on the vertices there exists a compact, deterministic CTS that routes $\mathbf{1}_u \xrightarrow{c(u)} c$ for each $u \in V$ with approximation $1 + \mathcal{O}(n^{-1})$ and congestion $\tilde{\mathcal{O}}(C)$.*

We obtain this result by making use of the KRV-framework [16]. One way to view this framework is that it tries to embed an expander into a graph by solving a small number of single-commodity maximum flow problems. Each maximum flow solution gives rise to a matching. One can then route to a random vertex by following the "*matching random walk*", i.e., in the i-th step the packet takes the (embedded) matching edge with probability $1/2$.

We proceed slightly differently. Instead of decomposing the flow into matchings and then route along the matchings (which seems difficult to do with small routing tables) we simply use Lemma 3′ to route along the flow. This means in the i-th step we stay with probability $1/2$ or we route the packet along the flow to some target of the flow. More concretely, assume that the KRV-scheme uses a flow f between sources $S := \{v \in V \mid \text{bal}_f(v) < 0\}$ and targets $T := \{v \in V \mid \text{bal}_f(v) > 0\}$ in the i-th step. Then we construct two transformation schemes. Let

$$\mu(v) := \begin{cases} -\text{bal}_f(v) & v \in S \\ 0 & \text{otw.} \end{cases} \quad \text{and} \quad \mu'(v) := \begin{cases} \text{bal}_f(v) & v \in T \\ 0 & \text{otw.} \end{cases}$$

We use Lemma 3′ to construct a transformation scheme that routes $\mu \to \mu'$ and one that routes $\mu' \to \mu$. These then allow us to distribute packets in the described way. The guarantees of KRV still hold for this slightly modified scheme, which means that after performing a polylogarithmic number of such steps a packet is at a random location.

The above process and the transformation schemes for the individual iterations can be combined into a single concurrent transformation scheme. The id-sets M_v for this scheme contain bitstrings that encode for every iteration: (a) the id to be used in the transformation scheme for this iteration and (b) a bit that indicates whether to route along the flow or to stay. Note that the CTS is deterministic, i.e., after choosing the id the packet follows a fixed path in the network.

The result by Räcke and Schmid [22] also required a sub-routine for routing a product multicommodity flow. They used a randomized rounding approach on the multicommodity flow solution of the PMCF-instance, and crucially exploited the fact that the number of routing paths going through an edge in such a solution is $\tilde{\mathcal{O}}(C)$. As illustrated in Figure 3, we cannot do this in our scenario as the number of paths going through an edge might be as large as $\tilde{\mathcal{O}}(CW)$. Storing these paths would require large routing tables.

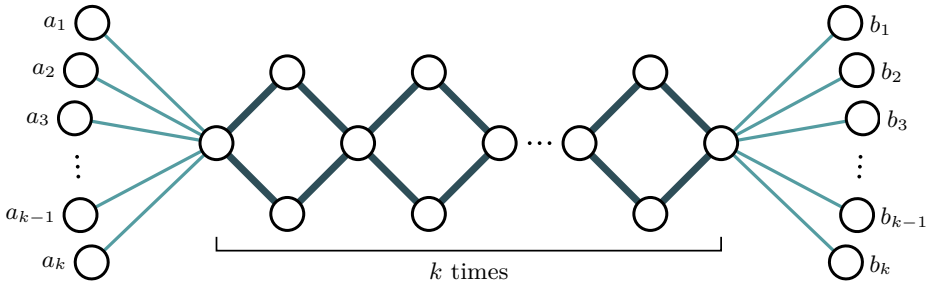


Figure 3 Routing multiple paths over a single edge. Each node a_i sends a packet to b_i. A single path needs k bits of information to encode, there are k paths and $n = \mathcal{O}(k)$ nodes, so on average each node needs to store $k^2/n = \Omega(n)$ bits if an arbitrary set of paths is chosen, which is too much. The same holds for paths chosen uniformly at random.

Routing arbitrary demands

The PMCF-scheme described above routes $\mathbf{1}_u \xrightarrow{w(u)} c$. If we choose $c := w_S$ for some cluster S then this scheme gives us the first part of our goal: we can concurrently route $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ with small congestion. To see this, observe that if we consider the demands in the PMCF-scheme for all commodities $u \in S_i$ combined, this is $\sum_u w_S(u)\mathbf{1}_u = w_S \restriction S_i = \mathrm{out}_{S_i}$. We can go from w_{S_i} to out_{S_i} *within* S_i using Lemma 3′, and then the PMCF-scheme distributes it according to w_S.

Hence, by simply merging commodities $u \in S_i$ into one we obtain our desired CTS.

However, for our oblivious routing scheme we also need to be able to route commodities $w_S \xrightarrow{w_S(S_i)} w_{S_i}$. This turns out to be much more involved. Note that we cannot simply "route in reverse" because a transformation scheme is inherently directed.

We do not directly construct a transformation scheme that routes $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ but we first embed an auxiliary graph into the cluster S (via a transformation scheme). This auxiliary graph is directed and must have small degree for the embedding to be compact.

▶ **Lemma 10′** (general graph embedding). *Let $G' = (S, A, d)$ denote a weighted, directed graph, where the total weight of incoming and outgoing edges of a node v is at most $w_S(v)$, and $\deg_{G'}(v) \in \tilde{\mathcal{O}}(\deg(v))$. Then there is a compact CTS that routes $\mathbf{1}_u \xrightarrow{d(u,v)} \mathbf{1}_v$ for each $(u, v) \in A$ with congestion $\tilde{\mathcal{O}}(C)$.*

This lemma is the main technical contribution of our work. A rough outline of the approach is as follows. The first observation is that one could combine the result for the PMCF-routing with Valiant's trick [28, 27] of routing to random intermediate destinations. Suppose that we want to route from x to y. Then we first apply the PMCF-scheme of Lemma 6′; this brings us to a node z chosen according to w_S. At z we choose a path id id_y that brings us to y, i.e., if we apply the transformation scheme starting from node z with id id_y the packet is delivered to y. We choose id_y uniformly at random from all path ids that will deliver the packet to y. This applies Valiant's trick and the standard analysis shows that the congestion of this approach will be $\tilde{\mathcal{O}}(C)$.

However, implementing this approach with small routing tables is problematic. The node z could store a table of path ids which can be used for routing to y but this is clearly not compact.

If all edges have weight 1, we can apply a suitable randomized rounding to the above path generation method. Then the number of paths that go from x to y are just $\tilde{\mathcal{O}}(\deg(x))$. This allows the node x to store the necessary information for every path. In the weighted setting, however, the randomized rounding approach leads to $\tilde{\mathcal{O}}(W \cdot \deg(x))$ many paths. The resulting tables would not be compact.

Instead we proceed as follows. We say a path p is a *class l path* if the smallest capacity edge of p is from class l. (Recall that we assume all capacities to be powers of two, and that a class l edge is one with capacity 2^l.)

A pair (x, y) is from class l if l is the most frequent class that occurs when generating x-y-paths by the above process.

In a first step we change the path generation process to only use class l paths for a class l pair. This only increases the congestion by a factor of N_{class} (the number of classes). For a randomized rounding approach to guarantee a good congestion we need to spread the traffic between a class l pair (x, y) over roughly $k := d(x, y)/2^l$ many class l paths.

For this we split the packet into k different parts, each with a different path. However, we cannot store information for each such path in x directly, as k may be large. Instead, we identify k many other nodes, each of which we use to store the information for just a single path. Of course, we cannot simply pick any nodes in the cluster — they have to be reachable from x with low congestion.

As it turns out, the set of k class l paths from x to y already contains an appropriate choice. Each class l path contains a class l edge, and we pick one of its two adjacent nodes as helper node. Now observe that each path transports 2^l flow, so there can only be a small number of paths using that edge, because we have low congestion. That means that we can use $\tilde{\mathcal{O}}(1)$ space in the helper node, for each path that uses it.

Now that we have found k helper nodes that can store our routing information, the packets still have to reach these nodes, to pick up that information. So now we send a single-commodity flow from *all* source nodes x' of class l pairs to their helper nodes, and then back. We set up a TS for both directions of the flow, using Lemma 3.

Note that this does not guarantee that a packet from source node x reaches "its" helper node, but this is not required — it only needs to reach *a* node in which to store its routing information. Similarly, the packet may not get back to x, but end up at a different x'.

We have the same packet distribution as before, meaning every x' has the same number of packets, but possibly different ones. However, each packet has picked up $\tilde{\mathcal{O}}(1)$ of information while passing its helper node.

Suppose that all packets now in x have a target y' s.t. (x, y') is a class $l' \geq l$ pair. Then we can pick a single path for each of these packets, and store the information for that path in the helper node. (Recall that paths are generated by the PMCF-scheme of Lemma 6′, so we only need to store their path ids.)

If that is not the case we split the packet further, by applying the scheme recursively.

Hypercube embedding

The previous lemma tells us that we can embed graphs with low degree. More precisely, we can embed any graph $G' = (S, A, d)$ which fulfills the following properties.

(1) The degree $\deg_{G'}(v)$ at a vertex is polylogarithmic.

(2) The capacity of incoming edges and the capacity of outgoing edges at a vertex is roughly equal to $w_S(v)$.

Now, in order to be able to construct an unmixing CTS, i.e., route $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ for a cluster S, we find some graph with the above properties where an unmixing CTS is easy to implement, and then embed that graph into G. In particular, we want a G' which has one additional property.

(3) There is a suitable numbering of the child clusters of S for which there exists a CTS for G' that routes $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ for each S_i with small congestion and commodity S_i encoded as integer i.

The result by Räcke and Schmidt [22] used a hypercube, where each node v received $w_S(v)$ hypercube ids. For weighted edges this would violate property (1).

Instead, we essentially embed several (disconnected) hypercubes, one for each class l. A node v then receives roughly $w_S^{(l)}(v)$ hypercube ids, at most one for each class l edge adjacent to it. The existence of a good CTS scheme for G' then follows from classical results about online routing on the hypercube [28].

References

1 Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. On space-stretch trade-offs: Upper bounds. In *Proc. 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 217–224, 2006. doi:10.1145/1148109.1148144.

2 Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Räcke. Optimal oblivious routing in polynomial time. *Journal of Computer and System Sciences*, 69(3):383–394, 2004. doi:10.1145/780542.780599.

3 Yair Bartal and Stefano Leonardi. On-line routing in all-optical networks. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 516–526. Springer, 1997.

4 Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 24–33, 2003. doi:10.1145/777412.777418.

5 Allan Borodin and John E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of computer and system sciences*, 30(1):130–145, 1985. doi:10.1145/800070.802209.

6 Marco Chiesa, Gábor Rétvári, and Michael Schapira. Oblivious routing in ip networks. *IEEE/ACM Transactions on Networking (TON)*, 26(3):1292–1305, 2018. doi:10.1109/TNET.2018.2832020.

7 Lenore J Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001. doi:10.1006/jagm.2000.1134.

8 Paul Erdős. Extremal problems in graph theory. In *Proceedings of the Symposium on Theory of Graphs and its Applications*, pages 29–36, 1963. doi:10.1002/jgt.3190010206.

9 Pierre Fraigniaud and Cyril Gavoille. Memory requirement for universal routing schemes. In *Proc. 14th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 223–230. ACM, 1995. doi:10.1145/224964.224989.

10 Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 757–772. Springer, 2001. doi:10.1007/3-540-48224-5_62.

11 Greg N Frederickson and Ravi Janardan. Designing networks with compact routing tables. *Algorithmica*, 3(1-4):171–190, 1988. doi:10.1007/BF01762113.

12 Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News*, 32(1):36–52, 2001. doi:10.1145/568438.568451.

13 Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In *Proc. 15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 125–133. ACM, 1996. doi:10.1145/248052.248075.

14 Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proc. 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 34–43, 2003. doi:10.1145/777412.777419.

15 Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas. Tight bounds for oblivious routing in the hypercube. *Mathematical Systems Theory*, 24(1):223–232, 1991. doi:10.1145/97444.97453.

16 Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM (JACM)*, 56(4):19, 2009. doi:10.1145/1538902.1538903.

17 M. Kodialam, T.V. Lakshman, J.B. Orlin, and S. Sengupta. Oblivious routing of highly variable traffic in service overlays and ip backbones. *IEEE/ACM Transactions on Networking (TON)*, 17(2):459–472, 2009. doi:10.1109/TNET.2008.927257.

18 Dmitri Krioukov, Kevin Fall, and Xiaowei Yang. Compact routing on internet-like graphs. In *Proc. IEEE INFOCOM*. IEEE, 2004. doi:10.1109/INFCOM.2004.1354495.

19 Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 157–170, Renton, WA, April 2018. USENIX Association. URL: https://www.usenix.org/conference/nsdi18/presentation/kumar.

20 Harald Racke. Minimizing congestion in general networks. In *Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 43–52. IEEE, 2002. doi:10.1109/SFCS.2002.1181881.

21 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 255–264. ACM, 2008. doi:10.1145/1374376.1374415.

22 Harald Räcke and Stefan Schmid. Compact oblivious routing. In *Proceedings of the 27th European Symposium on Algorithms (ESA)*, 2019. doi:10.4230/LIPIcs.ESA.2019.75.

23 Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 227–238. Society for Industrial and Applied Mathematics, 2014. doi:10.1137/1.9781611973402.17.

24 Gábor Rétvári, András Gulyás, Zalán Heszberger, Márton Csernai, and József J Bíró. Compact policy routing. *Distributed computing*, 26(5-6):309–320, 2013. doi:10.1007/s00446-012-0181-9.

25 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the 13th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, SPAA 01, pages 1–10, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/378580.378581.

26 Brian Towles and William J Dally. Worst-case traffic for oblivious routing functions. In *Proc. 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. ACM, 2002. doi:10.1109/L-CA.2002.12.

27 Leslie G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982. doi:10.1137/0211027.

28 Leslie G. Valiant and Gordon J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th ACM Symposium on Theory of Computing (STOC)*, pages 263–277, 1981. doi:10.1145/800076.802479.

29 Jan van Leeuwen and Richard B Tan. Compact routing methods: A survey. In *Proc. Colloquium on Structural Information and Communication Complexity (SICC)*, pages 99–109, 1995.

A Detailed Analysis

In this section we provide the details for constructing a mixing and an unmixing CTS for every cluster. This will then give the oblivious routing scheme.

Most of our lemmata related to the PMCF work for arbitrary weights c where the corresponding PMCF can be solved with congestion $C \in \mathcal{O}(\text{poly}(nW))$. We prove those without making further assumptions.

Note also that we can restrict the transformation schemes to route within a cluster: If a lemma is proven for arbitrary weights c, then it also works for weights w_S within the subgraph $G[S]$.

A.1 Constructing a mixing CTS

Single-commodity flows

To get started, we show how to use a single-commodity flow to construct a transformation scheme. The key idea is that we decompose the flow into paths with unique identifiers and store intervals for each edge, to encode the outgoing paths.

▶ **Lemma 3** (Flow routing). *Let f denote a flow with $\mathrm{cong}(f) \in \mathcal{O}(\mathrm{poly}(nW))$, and μ, μ' integral distributions with $\mu' - \mu = \mathrm{bal}_f$. Then there exists a deterministic TS that routes $\mu \xrightarrow{\mu(V)} \mu'$ with congestion $\mathcal{O}(\mathrm{cong}(f))$.*

The routing table of node v has size $\mathcal{O}(\deg(v)\log(nW))$, and packet headers have length $\mathcal{O}(\log(nW))$. At a node v there are $N_v := \mu(v)$ valid path ids.

Proof. First, we transform f to be integral and acyclic.

Let $F := \lceil \mathrm{cong}(f) \rceil$. We consider the single-commodity flow problem where we add a source s, a sink t and edges (s, v), (v, t) for $v \in V$ with respective capacities $\mu(v)$ and $\mu'(v)$. We retain the other edges, scaling their capacities by F. All capacities are integral and f is a solution with flow value $\mu(V)$, so there is also an integral, acyclic solution f', which by construction has $\mu' - \mu = \mathrm{bal}_{f'}$ and $\mathrm{cong}(f') \leq F$.

For each source $v \in V$ (i.e., a node that has $k := -\mathrm{bal}_{f'}(v) > 0$) we put k tokens into v. These have labels $a+1, a+2, ..., a+k$ where a is chosen s.t. the labels are disjoint for all nodes. We store a and k in v, which takes $\mathcal{O}(\log(nW))$ space.

Now we repeatedly move those tokens according to f', iteratively constructing the TS in the process. Each token represents a unit of flow.

As f' is acyclic, we iterate over the nodes based on the topological ordering given by f'. Let u denote the current node. We assume the invariant that no previous node contains any tokens, which is true in the beginning and will hold inductively for all other iterations.

Therefore, u has tokens precisely equal to the flow over incoming edges of u (plus $-\mathrm{bal}_{f'}(u)$ if u is a source), which, due to flow conservation, is the same as the flow over outgoing edges (plus $\mathrm{bal}_{f'}(u)$ if u is a sink). We distribute the tokens by sorting them and assign each outgoing edge (u, v) an amount of consecutive tokens, according to its flow $f'(u, v)$. These tokens are sent over that edge. Exactly $\max\{\mathrm{bal}_{f'}(v), 0\}$ tokens remain, which we remove from the graph. Hence we deleted all tokens from u and added tokens only to its successors, so our invariant still holds.

After the last iteration, all nodes are empty, so each token has been routed. To construct a TS, we encode the path of all tokens, by storing an *interval* $I(u, v)$ of ids for each edge (u, v), which contains the tokens which were routed over that edge. Note that the interval may be larger than the number of tokens that use e; all tokens that traverse u and are inside $I(u, v)$ use edge $e = (u, v)$, but the interval may contain tokens that do not traverse u. In total we just need to store two ids of length $\mathcal{O}(\log(nW))$ per incident edge. (Recall that the maximum load over any edge is at most $FW \in \mathcal{O}(\mathrm{poly}(nW))$.)

As we send exactly $|f'(e)|$ tokens over an edge e we get the same congestion as f'.

We want to construct a deterministic TS, so the input label at a node v contains a path id $l \in \{1, ..., \mu(v)\}$. As v stores the offset a from above, we can map the path id to $a + l$, one of the tokens starting at v. Afterwards, the routing algorithm simply needs to check which interval contains the token, to simulate their movement. This is deterministic. ◀

We use Lemma 3 typically to route between distributions which are 'close' to our weights c. As we can solve the PMCF with low congestion, this will have low congestion as well. The following lemma encapsulates that argument.

▶ **Lemma 4.** *Let $\mu_{\text{in}}, \mu_{\text{out}}$ denote distributions with $\mu_{\text{in}}, \mu_{\text{out}} \leq c$ and $\mu_{\text{in}}(V) = \mu_{\text{out}}(V)$. Then there is a flow f with $\text{bal}_f = \mu_{\text{out}} - \mu_{\text{in}}$ and $\text{cong}(f) \leq 2C$.*

Proof (omitted). The proof simply uses the PMCF together with Valiant's trick. It can be found in the full version of this paper. ◀

Routing the PMCF

Our first goal is to create a transformation scheme for the PMCF, for which we use the technique of cut-matching games introduced in [16]. We will not discuss it in detail, but instead encapsulate the properties of interest and refer to the original proofs. We need modify the technique slightly [3] , so for those parts we briefly show how the proofs can be adapted.

Consider the following game. We are given a finite set of nodes V', with $|V'|$ even. There are $N \in \mathcal{O}(\log^2 |V'|)$ rounds and two players. In round k,

- Player 1 (the"cut player") chooses a partition $A_1 \dot\cup A_2 = V'$ with $|A_1| = |A_2|$,
- Player 2 (the"matching player") chooses a bijection $M : V' \to V'$ *respecting the partition*, i.e., it maps A_1 to A_2 and vice versa, and
- Player 2 chooses a partition B of V'.

At the end of the game, we define a random walk on V' consisting of N steps. In step k, a packet

- moves from node v to either v or $M(v)$ with probability $\frac{1}{2}$, and then
- moves from the resulting node v' to a node in $B_{v'}$ uniformly at random, where $v' \in B_{v'} \in B$ is the group of v' in the partition B.

The game is won by Player 1 if this random walk is *mixing*, i.e., for any $u, v \in V'$ it moves from u to v with probability between $1/|V'| \pm \varepsilon$, where $\varepsilon = 1/|V'|^2$.

▶ **Lemma 5** (KRV)**.** *Player 1 has a winning strategy.*

Proof (omitted). See [16, Section 3.1]. The proofs have to be changed slightly to work here, a precise explanation can be found in the full version of this paper. ◀

These random walks can be made deterministic, by storing whether to switch sides at each step, provided that we can send packets along our chosen bijection M. Additionally, while we need a lot of nodes in V' to match the weights c, a single node v can simulate many in V' with only little bookkeeping. Here we use that "mixing" nodes of V' at each step is not problematic, a notion made precise by choosing the appropriate partition B. The bijections M will be stored implicitly using single-commodity flows.

In total we get short descriptions of the possible paths taken by the random walk, enabling us to circumvent one of the key problems arising in weighted graphs — the inability to store paths directly within the graph due to too many paths going over a single edge.

▶ **Lemma 6** (PMCF transformation scheme)**.** *There exists a deterministic CTS that routes $\mathbf{1}_v \xrightarrow{c(v)} c$ for each $v \in V$ with approximation $1 + \mathcal{O}(n^{-1})$ and congestion $\mathcal{O}(C \log^2 n)$.*

The routing table of node v has size $\mathcal{O}(\deg(v) \log^3(nW))$, while path ids and packet headers have length $\mathcal{O}(\log^3(nW))$.

[3] In particular, we use a bijection instead of a perfect matching, allow the matching player to choose subsets which will be shuffled randomly, and require a stronger bound on the error.

Proof. We want to play the game described above Lemma 5, so we define a set of"virtual" nodes $V' := \{1, ..., 2nc(V)\}$ [4] and choose an embedding $\varphi : V' \to V$ which assigns each node virtual nodes according to its weight, i.e. $|\varphi^{-1}(v)| = 2nc(v)$.

In each turn we have $A_1 \cup A_2 = V', |A_1| = |A_2|$ and can choose any bijection M respecting that partition. We want to simulate the random walk, so we also need a way to send packets according to M. In other words, we need a CTS that routes $\mathbf{1}_{\varphi(v')} \to \mathbf{1}_{M(\varphi(v'))}$ for each $v' \in V'$. It is difficult to construct such a CTS *given* a specific M, so instead we build the transformation scheme first, and then define M accordingly.

We construct two deterministic transformation schemes TS_1 and TS_2, routing from A_1 to A_2 and vice versa. Let us first consider TS_1.

A node v sends out a packet for each virtual node in A_1 assigned to it, so $\mu_1(v) := |\varphi^{-1}(v) \cap A_1|$ in total. Analogously, it receives $\mu_2(v) := |\varphi^{-1}(v) \cap A_2|$ packets. Hence we want to route $\mu_1 \xrightarrow{nc(V)} \mu_2$.

We have $\mu_1, \mu_2 \leq c$ and $\mu_1(V) = \mu_2(V) = nc(V)$, so Lemma 4 yields a flow f with $\mathrm{bal}_f = \mu_2 - \mu_1$ and $\mathrm{cong}(f) \leq 2nc(V)$. (We scale μ_1, μ_2 by $1/n$ and the resulting flow by n.)

Using f, we apply Lemma 3 to get a deterministic transformation scheme TS_1 that routes $\mu_1 \xrightarrow{nc(V)} \mu_2$ with congestion $\mathcal{O}(nC)$.

At each node v there are now $\mu_1(v)$ path ids to route a packet using TS_1. The transformation scheme is deterministic, so each path id corresponds to a single target node. Consider giving $\mu_1(v)$ packets to each node v, one for each path id, and routing them accordingly. Then, v will receive $\mu_2(v)$ packets. By mapping the outgoing packets to $\varphi^{-1}(v) \cap A_1$ and the incoming packets to $\varphi^{-1}(v) \cap A_2$ in some fashion, we get a bijection from A_1 to A_2.

We construct TS_2 in the same manner, and combine the two mappings to get the desired bijection M from V' to V'.

It remains to be shown that we can indeed route this bijection efficiently, i.e., without encoding any arbitrary mappings between virtual nodes and path ids.

Instead, we will simply choose a *random* path id for either TS_1 or TS_2, weighted such that each path id has the same probability. This corresponds to moving to a random virtual node assigned to v in each iteration, i.e. choosing our partition as $B := \{\varphi^{-1}(v) : v \in V\}$ in each round.

To summarize, we route the random walk as follows. In each iteration $k = 1, ..., N$ we flip a fair coin to decide whether we move from node v according to M or not. If yes, we pick a number i u.a.r. in $\{1, ..., 2nc(v)\}$. The first $\mu_1(v)$ numbers stand for path ids of TS_1, the others for path ids of TS_2. Then we route using the given transformation scheme and path id.

As we want our transformation scheme to be deterministic, these random choices will not be made while routing the packet, but encoded in the path id. There is a small technical issue in that the set $\{1, ..., 2nc(v)\}$ from which we sample i depends on v, but needs to be encoded in a path id chosen u.a.r. from some *fixed* range of integers. Instead, we will sample $i' \in \{1, ..., 2n^2c(V)N\}$ u.a.r and set $i := i' \pmod{2nc(v)}$. (Recall that N is the number of rounds.) So i is not quite uniform, but the probabilities differ by at most a factor of $1 + 1/nN$ in each round, and $1 + 1/n$ in total.

Thus we can store all random choices for the $\mathcal{O}(\log^2 nc(V)) = \mathcal{O}(\log^2(nW))$ iterations using $\mathcal{O}(\log^3(nW))$ bits. These are the path ids of our transformation scheme. As there is no randomness apart from the choices encoded in the path id, the transformation scheme is deterministic. The packet headers need to include the headers from Lemma 3, as well as our path ids; the length of the latter dominates.

[4] We would like to have $V' = \{1, ..., c(V)\}$, but we need to make sure that $|V'|$ is even and at least n.

Recall that we want to have a CDS that routes $\mathbf{1}_v \xrightarrow{c(v)} c$ for each $v \in V$. Hence, in aggregate the input distribution is c.

Let us now analyze the congestion. TS_1 routes $\mu_1 \xrightarrow{nc(V)} \mu_2$ and TS_2 does $\mu_2 \xrightarrow{nc(V)} \mu_1$, both with congestion $\mathcal{O}(nC)$. If we add them and scale the demand by $1/n$ we route $c \xrightarrow{c(V)} c$ with congestion $\mathcal{O}(C)$. So the distribution of packets does not change in an iteration, except for the factor of $1 + 1/nN$ above, and the total congestion is $\mathcal{O}(C \log^2(nW))$.

Due to Lemma 5, the random walk moves to any virtual node with probability between $1/2nc(V) \pm 1/|V'|^2$. We have $|V'|^2 \geq 2n^2 c(V)$, so scaling by the total amount of flow $c(V)$ yields a value in $1/2n \pm 1/2n^2$. A node $v \in V$ has $2nc(v)$ virtual nodes, so it receives between $c(v)(1 \pm 1/n)$ packets in the random walk, or between $c(v)(1 \pm 2/n)$ in the actual CTS. Hence the output distribution is c, with an approximation of $1 + \mathcal{O}(n^{-1})$. ◀

We remark that this CTS has input distribution $\mathbf{1}_v$ for commodity $v \in V$, which means that the source node of a packet already encodes its commodity.

Mixing CTS

To close out section we prove that we can implement the mixing step with the tools we have. To start, we need a small helper lemma.

▶ **Lemma 7** (Routing distributions similar to c)**.** *Let $\mu_{\text{in}}, \mu_{\text{out}}$ denote integral distributions with $\mu_{\text{in}}, \mu_{\text{out}} \leq c$ and set $M := \min\{\mu_{\text{in}}(V), \mu_{\text{out}}(V)\}$. Then there exists a deterministic TS that routes $\mu_{\text{in}}(V) \xrightarrow{M} \mu_{\text{out}}(V)$ with congestion $\mathcal{O}(C)$. The routing table of node v has size $\mathcal{O}(\deg(v) \log(nW))$, while path ids and packet headers have length $\mathcal{O}(\log(nW))$.*

Proof (omitted). The proof can be found in the full version of this paper. ◀

Now we fix a cluster S with children $S_1, ..., S_r$. As we will later change the numbering of children, it is important that the following lemma works for an arbitrary one.

▶ **Lemma 8** (Mixing CTS)**.** *There exists a CTS that routes $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ for each $i = 1, ..., r$ with congestion $\mathcal{O}(C \log^2 n)$ and approximation $1 + \mathcal{O}(n^{-1})$. The routing table of node v has size $\mathcal{O}(\deg(v) \log^3(nW))$, while path ids and packet headers have length $\mathcal{O}(\log^3(nW))$.*

Proof. For each S_i we route $w_{S_i} \xrightarrow{w_S(S_i)} \text{out}_{S_i}$ *within* S_i using Lemma 7 for weights $c := w_{S_i}$. This has congestion $\mathcal{O}(C)$, as $w_S(S_i) = \text{out}(S_i)$. It uses space only within S_i, so $\mathcal{O}(\deg(v) \log(nW))$ per node $v \in S$.

In total, the packets are now in distribution $\sum_i \text{out}_{S_i} = w_S$ and we apply Lemma 6 with $c := w_S$. (Here we do not use that Lemma 6 gives a deterministic CTS.) As all source nodes route to w_S concurrently, we route $\text{out}_{S_i} \xrightarrow{w_S(S_i)} w_S$ for each $i = 1, ..., r$ (with approximation $1 + \mathcal{O}(n^{-1})$). This has congestion $\mathcal{O}(C \log^2 n)$.

For the bounds on space, the costs of the latter step dominate. ◀

As for Lemma 6, the source node of a packet already determines the commodity, so there is no need to specify an encoding for it.

A.2 Constructing an Unmixing CTS

General Graph Embedding

Up until now, we have not used that we have only edges of distinct classes. The next two lemmas concern randomized rounding, which we use to select a small number of paths from a flow without increasing congestion. This uses a probabilistic argument to prove existence, but the choice of paths is fixed and not subject to randomness.

Consider some flow f sending k packets from a source node u to a node v with congestion 1, where the flow involves only edges of weight 1. It is obvious that taking a single path with weight k from that flow uniformly at random increases the congestion to k, while taking k paths with weight one should intuitively work quite well, giving a congestion $1 + \mathcal{O}(\log k)$. This intuition is correct, which we now prove formally.

We call a multi-set of u-v-paths a *path system*. The *class of a path* is the minimum class of its edges, and the *class of a path system* P is the minimum class amongst its paths. To send a packet using a path system P we choose a path uniformly at random. Therefore, if we have multiple path systems $P = \{P_1, ..., P_k\}$ with demands $d = \{d_1, ..., d_k\}$, then their total congestion is $\text{cong}(P, d) := \text{cong}(\{d_i p/|P_i| : i \in \{1, ..., k\}, p \in P_i\})$.

▶ **Lemma 9** (Randomized rounding). *Let $P = \{P_1, ..., P_k\}$ denote a set of path systems with demands d. Then there exists a set $P' = \{P'_1, ..., P'_k\}$, with $P'_i \subseteq P_i$ and $|P'_i| \leq \lceil 2^{-l} d_i \rceil$ for each P_i of class l. The congestion is $\text{cong}(P', d) \in \mathcal{O}(\text{cong}(P, d) + \log n)$.*

Proof (omitted). The proof can be found in the full version of this paper. ◀

Having the tool of randomized rounding at our disposal, we now turn to the most involved lemma in our construction. If we want to route small demands we can already do so using Lemmata 6 (to get a path system) and 9 (to pick a small number of paths to store). However, routing a demand of size, say, W from node u to v, we would have to pick $2^{-l}W$ paths from a path system $P_{u,v}$ connecting u and v, to ensure low congestion. Here, l is the class of $P_{u,v}$.

Hence, if l is small we would need to route a large number of paths. Instead, we find a cut consisting only of small (i.e., class l) edges separating each path in $P_{u,v}$. [5] These can be used to store routing information.

So we take all pairs (u_i, v_i) in the same situation, that is, connected by a class l path system, and take the union of all the cuts consisting of class l edges. Then we route a single-commodity flow from the nodes u_i to this cut. Of course, the packets from u may have ended up at an edge belonging to some other u_i, so it may not be possible to route to v directly. Instead we send the packets back through the single-commodity flow.

Again, the packets from u may now reside in a different node u'. However, on the way they passed through a class l edge, which we can use for storing the path from u' to v. (To be precise, we use one of the adjacent nodes for storage.) But now we have a new problem – while both $P_{u,v}$ and $P_{u',v'}$ are class l path systems, $P_{u',v}$ need not be. If it has class at least l, all is well and we can route the packet with a single path. Though if it has a class $l' < l$ we have to route using multiple paths again.

The fact that the class keeps decreasing allows us to solve this problem recursively. At each class we split the packet into smaller ones and find an edge to store the routing information for each of them. This stops when the packet is small enough to route directly, at the latest once it has reached size 1.

When we refer to storing the routing information for a node u in some previous node v on the path of a packet, we are using shorthand for a slightly elaborate transformation scheme, which we will refer to as *anticipative routing*. When the packet arrives at node v, the node checks the packet header and adds the stored routing information to it, before sending the packet on its way normally. Then, once the packet has reached the node u the routing information is extracted from the packet header and used.

[5] Note that this is not a cut of the *graph*, which might still be connected, but of the *path system*.

▶ **Lemma 10** (General graph embedding). *Let $G' = (V, A, d)$ denote a weighted, directed graph, where the total weight of incoming and outgoing arcs of a node v is at most $c(v)$, and $\deg_{G'}(v) \in \mathcal{O}(\deg(v) \log^2 n)$. Then there is a CTS that routes $\mathbf{1}_v \xrightarrow{d(u,v)} \mathbf{1}_v$ for each $(u, v) \in A$ with congestion $\mathcal{O}(C \log^2 n \log^2 W)$.*

The routing table of node v has size $\mathcal{O}(\deg(v) C \log^2 n \log W \log^3(nW))$, while packet headers have length $\mathcal{O}(\log^3(nW))$. Commodity $(u, v) \in A$ is encoded as $l \in \{1, ..., \deg_{G'}(u)\}$.

Proof. As mentioned above, the problematic demands are those which need multiple paths to route with low congestion. We will refer to those demands as *large*. More precisely, we call an arc $(u, v) \in A$ *l-large* if $d(u, v) > 2^l$ and l is the class of $P_{u,v}$ (defined below).

The proof will proceed in three parts.

(a) First, we construct path systems $P = \{P_{u,v} : u, v \in V\}$, s.t. all paths in $P_{u,v}$ have the same class and can be routed by storing a $\mathcal{O}(\log^3(nW))$ path id. For any demands d' where both the total incoming and outgoing demand of a node v are at most $c(v)$, we have $\text{cong}(P, d') \in \mathcal{O}(C \log^2 n \log W)$.

(b) Then we show that we can partially route arcs $a \in A$ which are l-large, replacing them with $2^{-l} d(u, v)$ arcs of weight 2^l. This does not change either the total outgoing or incoming demand of any node.

(c) Finally, we construct the CTS and derive the resulting bounds.

Part (a). We use Lemma 6 to construct a *deterministic* concurrent transformation scheme TS_1 routing the PMCF. Hence we can have $P'_{u,v}$ denote the path system containing the paths from u to v, one for each path id. Then we employ Valiant's trick and define $P^*_{u,v}$ as $\bigcup_{w \in V} P'_{u,w} \circ P'_{w,v}$, where $P \circ P'$ is a concatenation of path systems P, P' given by $P \circ P' := \{p \circ p' : p \in P, p' \in P'\}$. That means that we can split a path in $P^*_{u,v}$ into its first and second part.

Now consider some demands d', where the incoming or outgoing demand of any node v is at most $c(v)$. As $\bigcup_{w \in V} P'_{u,w}$ are all outgoing paths of u, sampling one u.a.r. is equivalent to sending a packet with TS_1 from u. So the first parts create the same congestion as TS_1, given that $\sum_v d(u, v) \leq c(v)$. To be precise, the congestion increases by a factor of $1 + \mathcal{O}(n^{-1})$, the approximation guaranteed by Lemma 6. This is only a constant factor, so we are going to disregard it.

The intermediate node w follows distribution c. The second parts, i.e., the paths from $P'_{w,v}$ then have weight $\sum_u c(w) d(u, v) \leq c(w) c(v)$. Routing a packet from w using TS_1 chooses a path from $P'_{w,v}$ with weight $\bar{c}(v)$, so we also bound the congestion based on TS_1.

In total we get $\text{cong}(P^*, d') \leq 2 \,\text{cong}(TS_1, c) \in \mathcal{O}(C \log^2 n)$. Finally, we want to modify $P^*_{u,v}$ so that it only contains paths of one class. We simply pick a class l with the maximum number of paths in $P^*_{u,v}$, and set $P_{u,v} := \{p \in P^*_{u,v} : p \text{ has class } l\}$. As there are N_{class} classes, we have $|P^*_{u,v}| \leq |P_{u,v}| N_{\text{class}}$ and the congestion increases by $\mathcal{O}(\log W)$.

Each path in $P_{u,v}$ is the concatenation of two paths from TS_1, so we can store two path ids of TS_1 to route it.

Part (b). We choose the highest class l where the set A' of l-large arcs is non-empty. Additionally, we introduce $\text{str} : A \to V$, which is the node that will be used to store the routing information for an arc. Initially, $\text{str}(u, v) = u$.

For any arc $a \in A'$ we define $d'(a)$ as the largest multiple of 2^l s.t. $d'(a) \leq d(a)$, and then set $d := d - d'$. So when routing a, a coin is flipped. With weight $d(a)$ we route using a (how precisely is yet to be determined), with weight $d'(a)$ we do the following procedure.

For all $(u, v) \in A'$ the path system $P_{u,v}$ has class l. Using Lemma 9 with demands d', we find a set of $d'(u, v)2^{-l}$ class l paths from u to v for $(u, v) \in A$, with congestion $\mathcal{O}(C \log^2 n \log W)$. We let M denote the set of prefixes of these paths, up to (and including) their first class l edge. By treating M them as a flow f, we can construct a transformation scheme TS, using Lemma 3, which has the same congestion.

Going back to the arc a we want to route, we send a packet using TS, with path id chosen uniformly at random. The necessary information for this is stored in $\text{str}(a)$. There are $N := d'(a)2^{-l}$ paths in M for a (and thus path ids of TS). If we put that number of tokens into the source of a and apply TS (one path id per token), they end up at nodes $z_1, ..., z_N$. A node z_i may receive tokens from other demands $a' \in A'$, but at most $\mathcal{O}(\deg(z_i)C \log^2 n \log W)$ in total, as each path ending in z_i in M induces a load of 2^l on a class l edge adjacent to z_i, i.e., a congestion of 1.

We also construct a transformation scheme based on the reverse flow $-f$, to send the tokens back. This does *not* use a random path id, instead a node z_i stores a mapping from incoming to outgoing path ids (any mapping is fine). We remark that the tokens of a may not end up where they started, as routing through a single-commodity flow mixes packets arbitrarily.

To summarize, an arc $a =: (u, v)$ has sent out N tokens, each of which corresponds to 2^l flow from $d'(a)$. Each token traversed an intermediate node z_i to end up at a node u'. Node z_i was passed by a low number of tokens in total. So now we add a new arc $a' := (u', v)$ to A, with demand $d(a') := 2^l$. Crucially, the routing information for a' is stored in z_i, i.e., $\text{str}(a') := z_i$.

As a technical detail we note that we allow for parallel arcs in G'. It is important that we do not merge multiple small demands into a larger one, as we have already ensured sufficient storage space for each, which would be lost.

The tokens are routed through f and then $-f$, so the number of tokens starting and ending at u is the same. This implies that both the total outgoing and incoming demand of any node remain unchanged.

As mentioned above, this procedure uses anticipative routing. For demand a we send N packets from u, each of which follows a deterministic path. So the intermediate node z_i assigns the packet the specific path id sending it to u' as well as the (yet to be determined) information on how to proceed from there. At u' the node does not have to look up the packet header in its routing table, but merely execute the information contained within.

Part (c). First, we apply (b) at most N_{class} times to eliminate all large arcs. Note that while (b) introduces new arcs, these have demand 2^l, where l is maximum class s.t. l-large arcs exist. So the new demands can only be l'-large for an $l' < l$.

Now we route the remaining arcs. Those are not large, so we can use Lemma 9 to pick a single path from $P_{u,v}$ for each $(u, v) \in A$. Based on our construction in (a), each path in $P_{u,v}$ can be routed using a $\mathcal{O}(\log^3(nW))$ path id. This will be stored in $\text{str}(u, v)$.

For the initial arcs, we store their path ids within their respective source nodes together with their (encoded) commodity.

Finally, we analyze the congestion and space requirements.

Each use of (b) creates congestion of $\mathcal{O}(C \log^2 n \log W)$, due to embedding two flows. Routing the non-large arcs at the end creates the same congestion (though only once). So in total we have a congestion of $\mathcal{O}(C \log^2 n \log^2 W)$.

In total, each node v is used at most $\deg_{G'}(v) \in \mathcal{O}(\deg(v) \log^2 n)$ times for storage due to our initial demands, and then at most $\cdot\mathcal{O}(C \log^2 n \log W)$ times for each adjacent class l edge when executing (b) for class l. Storing routing information for a large arc needs $\mathcal{O}(\log(nW))$

additional space to store the number of tokens and the range of path ids for them. This is dominated by the $\mathcal{O}(\log^3(nW))$ sized path id we need for both large and non-large arcs. (Recall that a large demand is first split into a fractional part and a multiple of 2^l.)

To embed the flows in (b) using Lemma 3, we need a total of $\mathcal{O}(\deg(v)\log(nW)\log W)$ space per node v for the routing tables, and transformation scheme in (a) from Lemma 6 uses $\mathcal{O}(\deg(v)\log^3(nW)))$ space. Summing everything up, we get $\mathcal{O}(\deg(v)C\log^2 n\log^2 W\log^3(nW))$.

Regarding packet headers, we need packet headers of Lemmata 3 and 6, as well as some additional space for our anticipative routing (at most $\mathcal{O}(\log^3(nW)))$. In total we get $\mathcal{O}(\log^3(nW))$. ◀

We want to remark on a slight technicality in the previous proof. Usually, scaling the routed distributions by some constant factor will scale the congestion by the same and nothing of importance has changed. However, the proof argues that there is a bound on the space used for each node, based on the congestion. Scaling the routed distribution to decrease congestion does actually affect this bound, so we could try scaling the congestion even lower. Though, as it turns out it is not possible to get a congestion below $\mathcal{O}(C\log^2 n\log W)$ as that is the minimum when fixing a single path provided by part (a). Using a fractional path would indeed have lower congestion, but not take up less space.

Hypercube embedding

Similar to Räcke and Schmid [22], we embed a hypercube. However, as mentioned earlier, we use a hypercube for each class l of edges and each hypercube id we assign has weight 2^l, i.e., a node $v \in S$ gets roughly $\mathrm{maj}_S^{(l)}(v)/2^l$ hypercube ids. The construction proceeds in a similar manner as the one of Räcke and Schmid up until the embedding of the hypercube edges, where we use Lemma 10 instead of simple randomized rounding.

The details can be found in the full version of this paper.

▶ **Lemma 11** (Hypercube embedding). *Let S be an arbitrary cluster with children $S_1, ..., S_r$. There exists a compact CTS that routes $\mathrm{maj}_S^{(l)} \xrightarrow{w_S(S_i)} \mathrm{out}_{S_i}^{(l)}$ for each S_i of class l with approximation 2 and congestion $\mathcal{O}(C\log^3 n\log^3 W)$.*

The routing table of node v has size $\mathcal{O}(\deg(v)C\log^2 n\log W\log^3(nW))$, while packet headers have length $\mathcal{O}(\log^3(nW))$. There exists a numbering of $S_1, ..., S_r$ s.t. commodity S_i is encoded as integer i.

Unmixing CTS

Given the hypercube embedding from the last lemma, we can now construct the unmixing CTS. At the beginning we need to ensure that we move to the distribution for the correct class, then we move through the (class specific) hypercube, and finally we go to the target distribution.

▶ **Lemma 12** (Unmixing CTS). *There exists a CTS that routes $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ for each $i = 1, ..., r$ with congestion $\mathcal{O}(C\log^3 n\log^3 W)$. The routing table of node v has size $\mathcal{O}(\deg(v)C\log^2 n\log W\log^3(nW))$, while packet headers have length $\mathcal{O}(\log^3(nW))$. There exists a numbering of $S_1, ..., S_r$ s.t. commodity S_i is encoded as integer i.*

Proof. The numbering of child clusters and our path ids are the same as for Lemma 11. Therefore we can determine the class l of S_i based on its index, as shown in the proof of that lemma.

For a child S_i with class l we want to route $w_S \to \mathrm{maj}_S^{(l)} \to \mathrm{out}_{S_i}^{(l)} \to \mathrm{out}_{S_i}$.

(1) For each class l let $M \subseteq S$ denote the union of class l child clusters. We route $w_S \xrightarrow{w_S(M)} \operatorname{maj}_S^{(l)}$ using Lemma 7 with congestion $C \cdot w_S(M)/\operatorname{maj}_S^{(l)}(M)$. This is at most CN_{class}, as $\operatorname{maj}_S^{(l)}(S_i) = \operatorname{out}_{S_i}^{(l)}(S_i) \geq w_S(S_i)/N_{\text{class}}$ for each child S_i with class l.

(2) We use Lemma 11 once, to route $\operatorname{maj}_S^{(l)} \xrightarrow{w_S(S_i)} \operatorname{out}_{S_i}^{(l)}$, with congestion $\mathcal{O}(C \log^3 n \log^3 W)$.

(3) For each S_i we route $\operatorname{out}_{S_i}^{(l)} \xrightarrow{w_S(S_i)} \operatorname{out}_{S_i}$ *within* S_i using Lemma 7. Here we have congestion $C \cdot w_S(S_i)/\operatorname{out}_{S_i}^{(l)}(S_i) \leq CN_{\text{class}}$.

Note that (1) has to be implemented on the whole cluster for each class, so its total congestion is $\mathcal{O}(C \log^2 W)$ (but still lower than step (2)). For the bounds on size of routing tables and length of packet headers, the costs of step (2) dominate. ◀

A.3 Combining the Results

Lemma 11 can be used directly as a drop-in replacement in the original result in [22].

The key idea is routing between two nodes u and v using the decomposition tree, spreading out a packet according to distribution w_S in each cluster. This ensures that routing within a cluster can be done with low congestion. Moving through the tree, the congestion is determined by the bottlenecks out_S. However, the optimal algorithm has to send the packets through these bottlenecks as well, so we remain competitive.

▶ **Theorem 13.** *There exists a compact oblivious routing scheme with competitive ratio* $\mathcal{O}(\log^6 n \log^3 W)$*, using a routing table of length* $\mathcal{O}(\deg(v) \log^5 n \log W \log^3(nW))$ *for a node* $v \in V$*, packet headers of length* $\mathcal{O}(\log^3(nW))$*, and node labels of length at most* $\mathcal{O}(\operatorname{height}(T) \log \deg(T))$*.*

Proof (omitted). The proof can be found in the full version of this paper. ◀

▶ **Corollary 14.** *Assume* $W \in \mathcal{O}(\operatorname{poly}(n))$*. Then there exists a compact oblivious routing scheme with competitive ratio* $\mathcal{O}(\log^9 n)$*, using a routing table of length* $\mathcal{O}(\deg(v) \log^9 n)$ *for a node* $v \in V$*, packet headers of length* $\mathcal{O}(\log^3 n)$ *and node labels of length* $\mathcal{O}(\log^2 n)$*.*

Approximation Algorithms for Clustering with Dynamic Points

Shichuan Deng
Institute for Interdisciplinary Information Sciences, Tsinghua University, China
dsc15@mails.tsinghua.edu.cn

Jian Li
Institute for Interdisciplinary Information Sciences, Tsinghua University, China
lijian83@mail.tsinghua.edu.cn

Yuval Rabani
The Rachel and Selim Benin School of Computer Science and Engineering,
The Hebrew University of Jerusalem, Israel
yrabani@cs.huji.ac.il

Abstract

In many classic clustering problems, we seek to sketch a massive data set of n points (a.k.a *clients*) in a metric space, by segmenting them into k categories or clusters, each cluster represented concisely by a single point in the metric space (a.k.a. the cluster's *center* or its *facility*). The goal is to find such a sketch that minimizes some objective that depends on the distances between the clients and their respective facilities (the objective is a.k.a. the *service cost*). Two notable examples are the k-center/k-supplier problem where the objective is to minimize the maximum distance from any client to its facility, and the k-median problem where the objective is to minimize the sum over all clients of the distance from the client to its facility.

In practical applications of clustering, the data set may evolve over time, reflecting an evolution of the underlying clustering model. Thus, in such applications, a good clustering must simultaneously represent the temporal data set well, but also not change too drastically between time steps. In this paper, we initiate the study of a dynamic version of clustering problems that aims to capture these considerations. In this version there are T time steps, and in each time step $t \in \{1, 2, \ldots, T\}$, the set of clients needed to be clustered may change, and we can move the k facilities between time steps. The general goal is to minimize certain combinations of the service cost and the facility movement cost, or minimize one subject to some constraints on the other. More specifically, we study two concrete problems in this framework: the Dynamic Ordered k-Median and the Dynamic k-Supplier problem. Our technical contributions are as follows:

- We consider the Dynamic Ordered k-Median problem, where the objective is to minimize the weighted sum of ordered distances over all time steps, plus the total cost of moving the facilities between time steps. We present one constant-factor approximation algorithm for $T = 2$ and another approximation algorithm for fixed $T \geq 3$.
- We consider the Dynamic k-Supplier problem, where the objective is to minimize the maximum distance from any client to its facility, subject to the constraint that between time steps the maximum distance moved by any facility is no more than a given threshold. When the number of time steps T is 2, we present a simple constant factor approximation algorithm and a bi-criteria constant factor approximation algorithm for the outlier version, where some of the clients can be discarded. We also show that it is NP-hard to approximate the problem with any factor for $T \geq 3$.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases clustering, dynamic points, multi-objective optimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.37

Related Version A full version of the paper is available at `https://arxiv.org/abs/2006.14403`.

Funding *Shichuan Deng* and *Jian Li*: Supported in part by the National Natural Science Foundation of China Grant 61822203, 61772297, 61632016, 61761146003, the Zhongguancun Haihua Institute for Frontier Information Technology, Turing AI Institute of Nanjing, and Xi'an Institute for Interdisciplinary Information Core Technology.
Yuval Rabani: Supported in part by ISF grant number 2553-17.

Acknowledgements We want to thank the reviewers for their insightful and constructive comments.

1 Introduction

Clustering a data set of points in a metric space is a fundamental abstraction of many practical problems of interest and has been subject to extensive study as a fundamental problem of both machine learning and combinatorial optimization. In particular, cluster analysis is one of the main methods of unsupervised learning, and clustering often models facility location problems.[1] More specifically, some of the most well-studied clustering problems involve the following generic setting. We are given a set C of points in a metric space, and our goal is to compute a set of k centers that optimizes a certain objective function which involves the distances between the points in C and the computed centers. Two prominent examples are the k-median problem and the k-center problem. They are formally defined as follows. Let S denote the computed set of k cluster centers, let $d(j,S) = \min_{i\in S} d(i,j)$ be the minimum distance from a point $j \in C$ to S, and let $D = (d(j,S))_{j\in C}$ be called the *service cost vector*. The k-median problem aims to minimize the L_1 objective $\|D\|_1 = \sum_{j\in C} d(j,S)$ over the choices of S, and the k-center aims to minimize the L_∞ objective $\|D\|_\infty = \max_{j\in C} d(j,S)$. In general metric spaces and when k is not a fixed constant, both problems are APX-hard and exhibit constant factor approximation algorithms [4, 6, 12, 23, 30]. An important generalization is the *ordered k-median* problem. Here, in addition to C and k, we are given also a non-increasing weight vector $w \in \mathbb{R}^{|C|}_{\geq 0}$. Letting $D^\downarrow$ denote the sorted version of D in non-increasing order, the objective of ordered k-median is to minimize $w \cdot D^\downarrow$. This problem generalizes both k-center and k-median and has attracted significant attention recently and several constant factor approximation algorithms have been developed [3, 7, 9, 10].

In this paper, we study several dynamic versions of the classical clustering problems, in which the points that need to be clustered may change for each time step, and we are allowed to move the cluster centers in each time step, either subject to a constraint on the distance moved, or by incurring a cost proportional to that distance. These versions are motivated in general by practical applications of clustering, where the data set evolves over time, reflecting an evolution of the underlying clustering model. Consider, for instance, a data set representing the active users of a web service, and a clustering representing some meaningful segmentation of the user base. The segmentation should be allowed to change over time, but if it is changed drastically between time steps, then it is probably meaningless. For a more concrete example, consider the following application scenario. There is a giant construction company with several construction teams working in a city. The company has k movable wireless base stations for their private radio communication, and each construction team also has a terminal device. The teams need to put their devices at a certain energy level, in order to maintain the communication channel between the device and the nearest base station. Some construction team may finish some project and move to another place at

[1] In the facility location literature, points are called clients and centers are called facilities, and we will use these terms interchangeably.

some time. Note that the wireless base stations are also movable at a certain expense. Our high level objective is to have all teams covered by the base stations at all times, meanwhile minimizing the energy cost of all teams plus the cost of moving these base stations.

We study two problems of this flavor. The first problem, a dynamic version of the *ordered k-median* problem, is a very general model that captures a wide range of dynamic clustering problems where the objective is to minimize the sum of service cost and movement cost. In particular, it generalizes dynamic versions of k-center and k-median. The problem is defined as follows. We are given a metric space and there are T time steps. In each time step t, there is a set C_t of clients that we need to serve. In each time step, we can also choose the locations for k movable facilities to serve the clients (each client is served by its closest facility). Our goal is to minimize the total ordered service distance (i.e., the ordered k-median objective), summed over all times steps, plus the total distances traveled by the k movable facilities. We define the problem formally as follows.

▶ **Definition 1** (Dynamic Ordered k-Median). *We are given a metric space (X, d). An instance of Dynamic Ordered k-Median is specified by $(\{C_t\}_{t=1}^T, \{F_t\}_{t=1}^T, \{w_t \in \mathbb{R}_{\geq 0}^{|C_t|}\}_{t=1}^T, \gamma > 0, k \in \mathbb{N}_+)$, where $T \geq 2$ is a constant integer, $C_t \subset X$ is the set of clients for time t, $F_t \subset X$ is the set of candidate locations where we can place facilities. For a vector v, denote by $v^{\downarrow}$ the vector derived from v by sorting its entries in non-increasing order. Also denote by $m(X, Y) = \min_{M_0 \in M(X,Y)} \sum_{(i,i') \in M_0} d(i, i')$ the total weight of minimum-weight perfect matching between two equal-sized multi-sets X, Y. We are required to compute a sequence of multi-sets of facilities $\{A_t\}_{t=1}^T$ with $A_t \subset F_t, |A_t| = k$, so that the following sum of ordered service cost and movement cost is minimized:*

$$\sum_{t=1}^{T} w_t \cdot (d(j, A_t))_{j \in C_t}^{\downarrow} + \gamma \cdot \sum_{t=1}^{T-1} m(A_t, A_{t+1}). \tag{1}$$

It is also natural to formulate dynamic clustering problems where the objective is to minimize just the service cost, subject to a constraint on the movement cost. This turns out to be technically very different from Dynamic Ordered k-Median. Our second problem, which we call Dynamic k-Supplier, is such a concrete problem, motivated by the above-mentioned construction company application. In this problem the service cost is the k-supplier objective, i.e. the maximum client service distance over all time steps, and the constraints are that any facility cannot be moved further than a fixed value $B > 0$ between any two consecutive time steps. More formally:

▶ **Definition 2** (Dynamic k-Supplier). *We are given a metric space (X, d). An instance of Dynamic k-Supplier is specified by $(\{C_t\}_{t=1}^T, \{F_t\}_{t=1}^T, B > 0, k \in \mathbb{N}_+)$, where $T \geq 2$ is the number of time steps, $C_t \subset X$ is the set of clients for time t, $F_t \subset X$ is the set of candidate locations where we can place facilities. We are required to compute a sequence of multi-sets of facilities $\{A_t\}_{t=1}^T$, with $A_t \subset F_t$, $|A_t| = k$, minimizing the maximum service cost of any client $\max_t \max_{j \in C_t} d(j, A_t)$, subject to the constraint that there must exist a one-to-one matching between A_t and A_{t+1} for any t, such that the distance between each matched pair is at most B.*

In the outlier version, we are additionally given the outlier constraints $\{l_t \in \mathbb{N}\}_{t=1}^T$. We are asked to identify a sequence of multi-sets of facilities $\{A_t\}_{t=1}^T$, as well as a sequence of subsets of served clients $\{S_t \subset C_t\}_{t=1}^T$. The goal is to minimize the maximum service cost of any served client $\max_t \max_{j \in S_t} d(j, A_t)$, with the constraints that $A_t \subset F_t, |A_t| = k, |S_t| \geq l_t$, and there must exist a one-to-one matching between A_t and A_{t+1} for any t, such that the distance between each matched pair is at most B.

Note. The solutions for both Dynamic Ordered k-Median and Dynamic k-Supplier may be multi-sets, since we allow multiple centers to travel to the same location.

1.1 Our Results

We first study Dynamic Ordered k-Median. We assume the number of time steps T is a constant and all entries of the weight vector are larger than some small constant $\epsilon > 0$. We present a polynomial-time approximation on general metrics. Moreover, if $T = 2$ we present a constant-factor approximation algorithm without the assumption on the entries of the weight vectors.

▶ **Theorem 3.** 1. *If $T = 2$, for any constant $\delta > 0$ there exists a polynomial-time $(48 + 20\sqrt{3} + \delta)$-approximation for Dynamic Ordered k-Median.*
2. *If $T \geq 3$ is a constant and all entries in $\{w_t\}_{t=1}^{T}$ are at least $\epsilon > 0$, for any constant $\delta > 0$ there exists a polynomial-time $(48 + 20\sqrt{3} + \delta + 6\gamma/\epsilon)$-approximation algorithm for Dynamic Ordered k-Median.*

Our techniques. The key idea in our algorithm is to design a surrogate relaxed LP as in [7] and embed the fractional LP solution in a network flow instance. We then proceed to round the fractional flow to an integral flow, thus obtaining the integral solution. The network is constructed based on a filtering process introduced by Charikar and Li [12]. We also adapt the oblivious clustering arguments by Byrka et al. [7], but with a slight increase in approximation factors due to the structure of our network flow.

Our approach can also give a constant approximation to the facility-weighted minimum total movement mobile facility location problem (facility-weighted TM-MFL), with a simpler analysis than the previously known local-search based algorithm [1], which achieves an approximation factor of $3 + O(\sqrt{\log \log p / \log p})$ for a p-Swap algorithm. The following result is also an improvement over the previously proven factor 499 in [1] when $p = 1$. For more details, we direct the interested readers to the full version of the paper and [1].

▶ **Theorem 4.** *There exists a polynomial-time 10-approximation algorithm for facility-weighted minimum total movement mobile facility location problem.*

As a second result, we consider Dynamic k-Supplier and its outlier version. We show that if $T \geq 3$, it is not possible to obtain efficient approximation algorithms for Dynamic k-Supplier with *any* approximation factor, unless P = NP, via a simple reduction from perfect 3D matching [27]. However, for the case of $T = 2$, we present a flow-based 3-approximation, which is the best possible factor since vanilla k-supplier is NP-hard to approximate within a factor of $(3 - \epsilon)$ for any constant $\epsilon > 0$ [24].

▶ **Theorem 5.**
1. *There exists a 3-approximation for Dynamic k-Supplier when $T = 2$.*
2. *There is no polynomial time algorithm for solving Dynamic k-Supplier with any approximation factor if $T \geq 3$, unless $P = NP$.*

We also study the outlier version of the problem for $T = 2$. In the outlier version, we can exclude a certain fraction of the clients as outliers in each time step. We obtain a bi-criteria approximation for the problem.

▶ **Theorem 6.** *For any constant $\epsilon > 0$, there exists a bi-criteria 3-approximation algorithm for Dynamic k-Supplier with outliers when $T = 2$, that outputs a solution which covers at least $(1 - \epsilon)l_t$ clients within radius $3R^*$ at time t, where $t = 1, 2$ and R^* is the optimal radius.*

Our techniques. We first guess a constant-size portion of facilities in the optimal solution, remove these facilities and solve the LP relaxation of the remaining problem. This guessing step is standard as in multi-objective optimizations in [20]. From the LP solution, we form clusters as in Harris et al. [22], cast the outlier constraints as budget constraints over the LP solution, and finally round the fractional LP solution to an integral solution using the budgeted optimization methods by Grandoni et al. [20]. Note that since our outlier constraints translate naturally to budget lower bounds, and our optimization goal is minimization, we are only able to achieve bi-criteria approximations instead of pure approximations. For more details, please refer to the full version of this paper.

1.2 Other Related Work

The ordered k-median problem generalizes a number of classic clustering problems like k-center, k-median, k-facility l-centrum, and has been studied extensively in the literature. There are numerous approximation algorithms known for its special cases. We survey here only the results most relevant to our work (ignoring, for instance, results regarding restricted metric spaces or fixed k). Constant approximations for k-median can be obtained via local search, Lagrangian relaxations and the primal-dual schema, or LP-rounding [4, 6, 25, 26]. Constant approximations for k-center are obtained via greedy algorithms [23]. Aouad and Segev [3] employ the idea of surrogate models and give the first $O(\log n)$-approximation for ordered k-median. Later, Byrka et al. [7], Chakrabarty and Swamy [9] both successfully design constant-factor approximations for k-facility l-centrum and ordered k-median. Chakrabarty and Swamy [10] subsequently improve the approximation factor for ordered k-median to $(5+\epsilon)$, using deterministic rounding in a unified framework.

The outlier setting of clustering problems, specifically for center-type clustering problems, was introduced by Charikar et al. [11] and later further studied by Chakrabarty et al. [8]. Many other variants of different clustering constraints are also extensively studied, including matroid and knapsack center with outliers [13], and fair center-type problems with outliers [22].

Our problems are closely related to the mobile facility location problems (MFL), introduced by Demaine et al. [16]. In these problems, a static set of clients has to be served by a set of facilities that are given initial locations and can be moved to improve the service cost at the expense of incurring a facility movement cost. For the minimum total movement MFL problem (TM-MFL), Friggstad and Salavatipour [18] give an 8-approximation using LP-rounding, where all facilities have unit weights. Ahmadian et al. [1] give a local search algorithm for TM-MFL with weighted facilities using p-swaps with an approximation ratio of $3+O(\sqrt{\log\log p/\log p})$, and specifically show that the approximation ratio is at most 499 for $p=1$.

The dynamic formulations of our problems are closely related to the facility location problem with evolving metrics, proposed by Eisenstat et al. [17]. In this problem, there are also T time steps, while the facilities and clients are fixed, and the underlying metric is changing. The total cost is the sum of facility-opening cost, client-serving cost and additional switching costs for each client. The switching cost is paid whenever a client switches facility between adjacent time steps. In comparison, our problem Dynamic k-Supplier considers the cost of moving facilities instead of opening costs, and allows the number of clients to change over time. Eisenstat et al. [17] consider the problem when the open facility set A is fixed, and give a $O(\log(nT))$-approximation, where n is the number of clients. They also show a hardness result on $o(\log T)$-approximations. An et al. [2] consider the case when the open facilities are allowed to evolve as well, and give a 14-approximation.

Our problem is also related to stochastic k-server [15] and the page migration problem [5, 32]. Dehghani et al. [15] first study the *stochastic k-server* problem. In this problem, we also have T time steps, and the distributions $\{P_t\}_{t\in[T]}$ are given *in advance*. The t-th client is drawn from P_t, and we can use k movable servers. One variant they consider is that, after a client shows up, its closest server goes to the client's location and comes back, and the optimization objective is the total distance travelled by all servers. They provide an $O(\log n)$-approximation for general metrics, where n is the size of the distribution support. In expectation, their objective is the same as in Dynamic Ordered k-Median, if we consider non-ordered weighted clients and total weights sum up to 1 for each time slot. However, we note that our result does not imply a constant approximation for their problem. The difficulty is that if one maps the stochastic k-server problem to our problem, the corresponding weight coefficient γ is not necessarily a constant and our approximation ratio is proportional to γ. Obtaining a constant factor approximation algorithm for stochastic k-server is still an interesting open problem.

2 A Constant Approximation for Dynamic Ordered k-Median

We devise an LP-based algorithm, which generalizes the oblivious-clustering argument by Byrka et al. [7]. At the center of our algorithm, a network flow method is used, where an integral flow is used to represent our solution.

2.1 Flow-based Rounding of LP Solution

We first formulate the LP relaxation. By adding a superscript to every variable to indicate the time step, we denote $x_{ij}^{(t)} \in [0,1]$ the partial assignment of client j to facility i and $y_i^{(t)} \in [0,1]$ the extent of opening facility location i at time step t. Moreover, denote $z_{ii'}^{(t)}$ the fractional movement from facility i to facility i', between neighboring time steps t and $t+1$.

The following surrogate LP is designed using the cost reduction trick by Byrka et al. [7]. When the reduced cost functions are exactly guessed, the LP relaxation has an objective value at most the total cost of the optimal solution, denoted by OPT. Call $d' : X \times X \to R_{\geq 0}$ a reduced cost function (not necessarily a metric) of distance function d, if for any $x, y \in X$, $d'(x,y) \geq 0$, $d'(x,y) = d'(y,x)$, and $d(x_1,y_1) \leq d(x_2,y_2) \Rightarrow d'(x_1,y_1) \leq d'(x_2,y_2)$. For a sequence of reduced cost functions $\mathfrak{D} = \{d^t\}_{t=1}^T$ of d, the modified LP relaxation is defined as follows.

$$\text{minimize}: \sum_{t=1}^{T}\sum_{j\in C_t}\sum_{i\in F_t} d^t(i,j)x_{ij}^{(t)} + \gamma\sum_{t=1}^{T-1}\sum_{i\in F_t}\sum_{i'\in F_{t+1}} d(i,i')z_{ii'}^{(t)} \qquad \mathsf{LP}(\mathfrak{D})$$

$$\text{subject to}: \sum_{i\in F_t} x_{ij}^{(t)} = 1, \quad \forall j \in C_t, t \in [T] \tag{2}$$

$$\sum_{i\in F_t} y_i^{(t)} = k, \quad \forall t \in [T] \tag{3}$$

$$0 \leq x_{ij}^{(t)} \leq y_i^{(t)}, \quad \forall i \in F_t, j \in C_t, t \in [T] \tag{4}$$

$$\sum_{i'\in F_{t+1}} z_{ii'}^{(t)} = y_i^{(t)}, \quad \forall i \in F_t, t \in [T-1] \tag{5}$$

$$\sum_{i\in F_t} z_{ii'}^{(t)} = y_{i'}^{(t+1)}, \quad \forall i' \in F_{t+1}, t \in [T-1] \tag{6}$$

Suppose we have solved the corresponding surrogate $\mathsf{LP}(\mathfrak{D})$. In the optimal solution (x, y, z), we assume that whenever $x_{ij}^{(t)} > 0$, we have $x_{ij}^{(t)} = y_i^{(t)}$, via the standard duplication technique of facility locations (for example, see [12]). Denote $\mathsf{Ball}^o(j, R) = \{x \in X : d(x, j) < R\}$ the open ball centered at j with radius R, and $E_j^{(t)} = \{i \in F_t : x_{ij}^{(t)} > 0\}$ the relevant facilities for client j. For any specific time step t, denote $d_{\mathsf{av}}^{(t)}(j) = \sum_{i \in F_t} d(i, j) x_{ij}^{(t)}$ the average *unweighted* service cost of client j and $y^{(t)}(S) = \sum_{i \in S} y^{(t)}(i)$ the amount of fractional facilities in $S \subset F_t$. We perform a filtering-and-matching algorithm (see the full version of this paper) to obtain a subset $C_t' \subset C_t$ for each t, a bundle $\mathcal{U}_j^{(t)} \subset F_t$ for each $j \in C_t'$, as well as P_t a partition of C_t', where

1. C_t' is a subset of "well-separated" clients of C_t, such that for any client in $C_t \backslash C_t'$, there exists another relatively close client in C_t'. To be more precise, for any $j \neq j'$ in C_t', $d(j, j') \geq 4 \max\{d_{\mathsf{av}}^{(t)}(j), d_{\mathsf{av}}^{(t)}(j')\}$, and for any $j'' \in C_t \backslash C_t'$, there exists $j''' \in C_t'$ such that $d_{\mathsf{av}}^{(t)}(j''') \leq d_{\mathsf{av}}^{(t)}(j''), d(j'', j''') \leq 4 \max\{d_{\mathsf{av}}^{(t)}(j''), d_{\mathsf{av}}^{(t)}(j''')\}$;
2. $\mathcal{U}_j^{(t)}$ is a subset of fractionally open facility locations that are relatively close to client j;
3. P_t is a judiciously created partition of C_t', where every subset contains either a pair of clients, or a single client. Each pair $\{j, j'\}$ in P_t is chosen such that either j or j' is the closest neighbor of the other, and we guarantee to open a facility location in $\mathcal{U}_j^{(t)}$ or $\mathcal{U}_{j'}^{(t)}$.

The filtering-and-matching algorithm is fairly standard in several LP-based methods for median-type problems (see e.g. [7, 10, 12]). It is worth noting that, while we define the objective value of $\mathsf{LP}(\mathfrak{D})$ using reduced cost functions $\mathfrak{D}$ with respect to the weights, the filtering algorithm is completely oblivious of the weights and only uses the underlying metric d.

Network construction. We construct an instance of network flow $\mathcal{N}$, and embed the LP solution as a fractional flow $\tilde{f}$. The network $\mathcal{N}$ consists of a source s, a sink t and $6T$ intermediate layers $L_1, L_2, \ldots, L_{6T}$ arranged in a linear fashion.

For each time step $t \in [T]$, we create two nodes for every pair $p \in P_t$, every bundle $\mathcal{U}_j^{(t)}$ and every candidate facility location $i \in F_t$. All these nodes are contained in the layers $L_{6t-5}, \ldots, L_{6t}$. To distinguish between the two mirror nodes, we use $\mathscr{L}(\cdot)$ and $\mathscr{R}(\cdot)$ to represent the nodes in $\{L_{6t-5}, L_{6t-4}, L_{6t-3}\}$ and the nodes in $\{L_{6t-2}, L_{6t-1}, L_{6t}\}$, respectively. The network is constructed as follows. An example figure is shown in Figure 1.

1. For all $t \in [T]$, add $\mathscr{L}(i)$ to L_{6t-5} and $\mathscr{R}(i)$ to L_{6t} for each $i \in F_t$.
2. For all $t \in [T]$, add $\mathscr{L}(\mathcal{U}_j^{(t)})$ to L_{6t-4} and $\mathscr{R}(\mathcal{U}_j^{(t)})$ to L_{6t-1} for each $\mathcal{U}_j^{(t)}$.
3. For all $t \in [T]$, add $\mathscr{L}(p)$ to L_{6t-3} and $\mathscr{R}(p)$ to L_{6t-2} for each $p \in P_t$.
4. For all $t \in [T], j \in C_t', p \in P_t$ such that $j \in p$, connect $(\mathscr{L}(\mathcal{U}_j^{(t)}), \mathscr{L}(p)), (\mathscr{R}(p), \mathscr{R}(\mathcal{U}_j^{(t)}))$ *in neighboring layers* with an edge of capacity $\left[\lfloor y^{(t)}(\mathcal{U}_j^{(t)}) \rfloor, \lceil y^{(t)}(\mathcal{U}_j^{(t)}) \rceil\right]$. Let their initial fractional flow values be $\tilde{f}(\mathscr{L}(\mathcal{U}_j^{(t)}), \mathscr{L}(p)) = \tilde{f}(\mathscr{R}(p), \mathscr{R}(\mathcal{U}_j^{(t)})) = y^{(t)}(\mathcal{U}_j^{(t)})$. The capacity is either $[0, 1]$ or $\{1\}$.
5. For all $t \in [T], p \in P_t$, connect $(\mathscr{L}(p), \mathscr{R}(p))$ with an edge of capacity $\left[\lfloor y^{(t)}(p) \rfloor, \lceil y^{(t)}(p) \rceil\right]$, and define $\tilde{f}(\mathscr{L}(p), \mathscr{R}(p)) = y^{(t)}(p) = \sum_{j \in p} y^{(t)}(\mathcal{U}_j^{(t)})$. If p is a normal pair, the capacity is either $[1, 2]$ or $\{1\}$ or $\{2\}$; if p is a singleton pair, the capacity is either $[0, 1]$ or $\{1\}$.
6. For all $t \in [T], j \in C_t'$ and $i \in \mathcal{U}_j^{(t)}$, connect $(\mathscr{L}(i), \mathscr{L}(\mathcal{U}_j^{(t)})), (\mathscr{R}(\mathcal{U}_j^{(t)}), \mathscr{R}(i))$ *in neighboring layers* with an edge of unit capacity. Let the initial fractional flows be $\tilde{f}(\mathscr{L}(i), \mathscr{L}(\mathcal{U}_j^{(t)})) = \tilde{f}(\mathscr{R}(\mathcal{U}_j^{(t)}), \mathscr{R}(i)) = y_i^{(t)}$.

7. For all $t \in [T]$ but $i \in F_t - \bigcup_{j \in C'_t} \mathcal{U}_j^{(t)}$, connect $(\mathscr{L}(i), \mathscr{R}(i))$ with an edge of unit capacity (*across intermediate layers* $L_{6t-4}, \ldots, L_{6t-1}$). Let its initial fractional flow be $\tilde{f}(\mathscr{L}(i), \mathscr{R}(i)) = y_i^{(t)}$.
8. For all $z_{ii'}^{(t)}, i \in F_t, i' \in F_{t+1}$, connect $(\mathscr{R}(i), \mathscr{L}(i'))$ with an edge of unit capacity. Let its initial fractional flow be $\tilde{f}(\mathscr{R}(i), \mathscr{L}(i')) = z_{ii'}^{(t)}$.

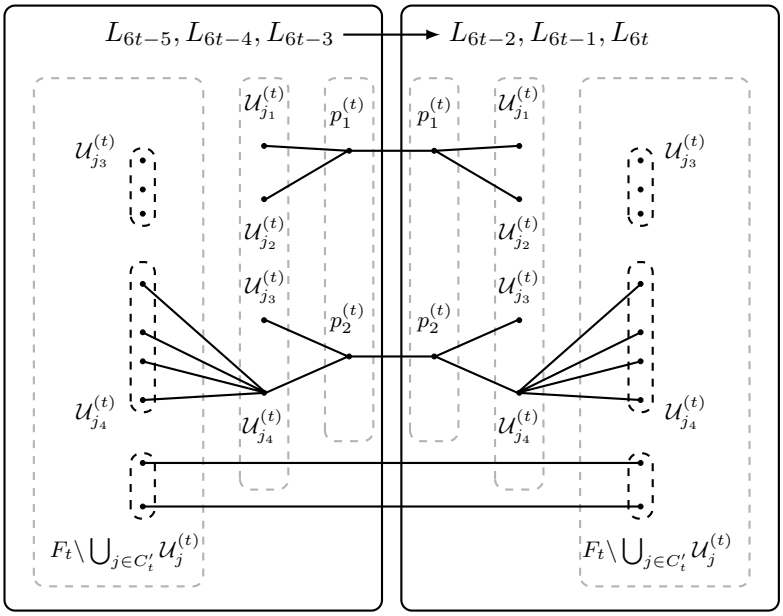


Figure 1 Some intermediate layers of $\mathcal{N}$ representing a single time step t.

Notice $\tilde{f}$ is naturally a flow with value k. Since the flow polytope is defined by a totally unimodular matrix, and our capacity constraints are all integers, it is a well-known result (see e.g. [19]) that we can efficiently and stochastically round $\tilde{f}$ to an integral flow $\bar{f}$, such that $\bar{f}$ is guaranteed to have value k, and $\mathbb{E}[\bar{f}] = \tilde{f}$. Next, given the integral flow $\bar{f}$, we deterministically construct the facilities to open $\{A_t\}_{t \in [T]}$ as follows.

- If $T = 2$, there are 12 layers $L_1, L_2, \ldots, L_{12}$ in the network. For each link $e = (\mathscr{R}(i_1), \mathscr{L}(i_2))$ between L_6 and L_7 such that $\bar{f}(e) = 1$, we add the original facility corresponding to i_1 to A_1, and the original facility of i_2 to A_2.
- If $T \geq 3$, the integral flow $\bar{f}$ may enter L_{6t-5} and exit from L_{6t} at sets of different facility locations. For illustration, denote $A_{t,1}$ the set in L_{6t-5} and $A_{t,2}$ the set in L_{6t}. Notice it may happens that $|A_{t,1} \bigcup A_{t,2}| > k$ and we cannot open them both, so we design an algorithm to find $A_t \subseteq A_{t,1} \bigcup A_{t,2}$ and $|A_t| = k$, and open the facilities in A_t for time t. The algorithm looks at each pair $(j_1, j_2) = p \in P_t$, and consider the 1 or 2 units of flow $\bar{f}$ on the link $(\mathscr{L}(p), \mathscr{R}(p))$. For a facility i, if there is one unit of flow through $\mathscr{L}(i)$ or $\mathscr{R}(i)$, we call the facility i *activated*. But if $\mathscr{L}(i_1)$ and $\mathscr{R}(i_2)$ are activated and $i_1, i_2 \in \mathcal{U}_{j_1}, i_1 \neq i_2$, we only open one of them. The same is true when $i_1 \in \mathcal{U}_{j_1}, i_2 \in \mathcal{U}_{j_2}, i_1 \neq i_2$.
  For each unit flow, our algorithm either always choose i_1 to open where $\mathscr{L}(i_1)$ is activated, or always choose the facility in $\mathcal{U}_{j_2}$ if j_1 is not the closest neighbor of j_2. As a result, we give the following lemma estimating the movement cost, and the detailed algorithm can be found in the full version of this paper.

▶ **Lemma 7.** *Let $d(A, A')$ denote the cost of minimum weight matching between A, A'. If $T = 2$, the expected movement cost of solution $\{A_1, A_2\}$ satisfies*

$$\mathbb{E}[d(A_1, A_2)] = \sum_{i \in F_1} \sum_{i' \in F_2} d(i, i') z^{(1)}_{ii'}.$$

If $T \geq 3$, the expected movement cost of solution $\{A_t\}_{t=1}^T$ after rerouting satisfies

$$\mathbb{E}\left[\sum_{t \in [T-1]} d(A_t, A_{t+1})\right] \leq \sum_{t \in [T-1]} \sum_{i \in F_t} \sum_{i' \in F_{t+1}} d(i, i') z^{(t)}_{ii'} + 6 \sum_{t=1}^{T} \sum_{j \in C_t} d^{(t)}_{\mathsf{av}}(j).$$

2.2 From Rectangular to General Cases

We first provide a lemma to bound the stochastic k-facility l-centrum cost of A_t for any fixed time t. Consequently, the ordered cost can be nicely bounded as well. The proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 8** (adapted from [7]). *Fix $t \in [T]$ and let $m \in \mathbb{N}_+, h > 0$. Define $\mathsf{rect}(a, b)$ the rectangular vector of length b, where the first a elements are 1s and the rest are 0s. For A_t as the (random) set of activated locations returned by our algorithm, and $d(C_t, A_t) = (d(j, A_t))_{j \in C_t}$ as the service cost vector, we have*

$$\mathbb{E}_{A_t}[\mathsf{rect}(m, |C_t|) \cdot d(C_t, A_t)^{\downarrow}] \leq (24 + 10\sqrt{3}) m \cdot h + (24 + 10\sqrt{3}) \sum_{j \in C_t} d^{-h^{(t)}}_{\mathsf{av}}(j),$$

where $d^{-h}(j, j') = 0$ if $d(j, j') < h$ and $d^{-h}(j, j') = d(j, j')$ otherwise. Similar to $d_{\mathsf{av}}(j)$, the average clipped service cost $d^{-h^{(t)}}_{\mathsf{av}}(j)$ is defined as $d^{-h^{(t)}}_{\mathsf{av}}(j) = \sum_{i \in F_t} d^{-h}(i, j) x^{(t)}_{ij}$.

Finally we turn to the generally-weighted case, where the weight vectors $w_t, t \in [T]$ are not necessarily rectangular ones like $\mathsf{rect}(m, |C_t|)$. The guessing of underlying reduced cost functions $\mathfrak{D}$ is exactly the same as in Byrka et al. [7], thus omitted here. We solve $\mathsf{LP}(\mathfrak{D})$ using these induced reduced cost functions and proceed accordingly. The following lemma is similar to that of Lemma 5.1 in Byrka et al. [7], and the proof can be found in the full version of this paper.

▶ **Lemma 9.** *When $T = 2$, the procedure described above is a $(48 + 20\sqrt{3})$-approximation for Dynamic Ordered k-Median.*

If $T \geq 3$ is a constant and the smallest entry in $\{w_t\}_{t=1}^T$ is at least some constant $\epsilon > 0$, the above-described procedure is a $(48 + 20\sqrt{3} + 6\gamma/\epsilon)$-approximation for Dynamic Ordered k-Median.

In both cases, the procedure makes $O\left(\prod_{t=1}^T (|F_t| \cdot |C_t|)^{N_t}\right)$ calls to its subroutines, where N_t is the number of distinct entries in the weight vector w_t, $t \in [T]$.

Fix some positive parameter $\delta > 0$ and recall the distance bucketing trick by Aouad and Segev [3]. When T is a constant, it is possible to guess the largest service distance for each time step by paying a polynomial factor in the running time. Then we make logarithmically many buckets for each time step to hold the service cost values of clients. For each bucket, its average weight is also guessed up to a small multiplicative error $(1 + \delta)$. Since there are at most $O\left(\log_{1+\delta}\left(\frac{n}{\delta}\right)\right) = O\left(\frac{1}{\delta}\log\left(\frac{n}{\delta}\right)\right)$ buckets for each time step, where $n = |F| + |C|$, guessing a non-increasing sequence of the average weights only causes another polynomial factor $\exp\left(O\left(\frac{1}{\delta}\log\left(\frac{n}{\delta}\right)\right)\right) = n^{O(1/\delta)}$. Finally, because T is a constant, the overall number of guesses is still bounded by a polynomial. For more details, see [3, 7].

▶ **Theorem 10.** *When $T = 2$, for any $\delta > 0$ there exists a $(48+20\sqrt{3})(1+\delta)$-approximation algorithm for Dynamic Ordered k-Median, with running time $(|F_1| + |C_1|)^{O(1/\delta)} \cdot (|F_2| + |C_2|)^{O(1/\delta)}$.*

When $T \geq 3$ is a constant, and the smallest entry in $\{w_t\}_{t=1}^T$ is at least some constant $\epsilon > 0$, for any $\delta > 0$ there exists a $\left(48 + 20\sqrt{3} + 6\gamma/\epsilon\right)(1+\delta)$-approximation algorithm for Dynamic Ordered k-Median, with running time $\prod_{t\in[T]} (|F_t| + |C_t|)^{O(1/\delta)}$.

Proof. This is almost a direct consequence of Theorem 5.2 in [7], with the constant factor replaced by our $\mu = 24+10\sqrt{3}$. Notice that we need to slightly modify the way of constructing rounded weights $\{w_t^*\}_{t=1}^T$ in the following way,

$$\forall t \in [T], r \in [|C_t|], w_{tr}^* = \begin{cases} w_{t1} & r = 1, \\ \min\left\{(1+\delta)^{\lceil \log_{1+\delta} w_{tr} \rceil}, w_{t1}\right\} & w_{tr} \geq \epsilon w_{t1}/|C_t|, r \neq 1, \\ \epsilon w_{t1}/|C_t| & w_{tr} < \epsilon w_{t1}/|C_t|, \end{cases}$$

so that the perturbed weight vectors are rounded larger, but at most $(1+\delta)$ times larger in terms of the overall objective, and there are $O(\log_{1+\delta}(|C_t|/\delta))$ different values in w_t^*.

Plugging in the approximations of individual time steps does not affect the analysis of movement costs in the proof of Lemma 9, hence the approximation factor follows. We omit the technical details here due to space limit. They can be be found in Appendix D of [7]. ◀

3 Approximating Dynamic k-Supplier

We present a flow-based algorithm that gives a 3-approximation for Dynamic k-Supplier when $T = 2$, and show it is NP-hard to obtain polynomial-time approximation algorithms for Dynamic k-Supplier with any approximation factor when $T \geq 3$. We also briefly introduce our bi-criteria approximation algorithm for Dynamic k-Supplier with outliers and $T = 2$, while the detailed algorithm and analysis can be found in the full version of this paper.

3.1 A 3-Approximation for Dynamic k-Supplier, $T = 2$

In contrast to the NP-hardness of approximating Dynamic k-Supplier for $T \geq 3$, we consider Dynamic k-Supplier when $T = 2$ on general metrics and present a simple flow-based constant approximation. Suppose we are given the client sets C_1, C_2 and F_1, F_2 as candidate facility locations and the movement constraint is $B > 0$.

First, since the optimal radius R^* is obviously the distance between some client and some facility location, we assume we have successfully guessed the optimal radius R^* (using binary search). Next, we construct the following network flow instance $\mathcal{G}(\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ consists of 4 layers of vertices (two layers $\mathcal{L}^{11}, \mathcal{L}^{12}$ for $t = 1$, two layers $\mathcal{L}^{21}, \mathcal{L}^{22}$ for $t = 2$), a source s and sink t. We define the layers and links in $\mathcal{G}$ as follows:

- For each $i \in F_1$, add a vertex in $\mathcal{L}^{12}$. For $i' \in F_2$, add a vertex in $\mathcal{L}^{21}$.
- Repeatedly pick an arbitrary client $j \in C_1$ and remove from C_1 every client within distance $2R^*$ from j. Denote these clients a new cluster corresponding to j. Since we have guessed the optimal radius R^*, it is easy to see we can get at most k such clusters. And if there are less than k clusters, we create some extra dummy clusters to obtain exactly k clusters, while dummy clusters do not correspond to any client. For each cluster, add a vertex to $\mathcal{L}^{11}$. Repeat this for C_2 and form $\mathcal{L}^{22}$.
- The four layers are arranged in order as $\mathcal{L}^{11}, \mathcal{L}^{12}, \mathcal{L}^{21}, \mathcal{L}^{22}$. With a slight abuse of notation, for $u \in \mathcal{L}^{11}, v \in \mathcal{L}^{12}$, connect them using a link with unit capacity if $d(u, v) \leq R^*$; for $w \in \mathcal{L}^{21}, z \in \mathcal{L}^{22}$, connect them using a link with unit capacity if $d(w, z) \leq R^*$. For $v \in \mathcal{L}^{12}, w \in \mathcal{L}^{21}$, connect them using a link with *unbounded* capacity if $d(v, w) \leq B$.

- Connect every dummy cluster in $\mathcal{L}^{11}$ with every facility location vertex in $\mathcal{L}^{12}$. Connect every dummy cluster in $\mathcal{L}^{22}$ with every facility location vertex in $\mathcal{L}^{21}$. Every such link has unit capacity.
- Finally, the source $\mathtt{s}$ is connected to every vertex in $\mathcal{L}^{11}$ and the sink $\mathtt{t}$ is connected to every vertex in $\mathcal{L}^{22}$, with every edge having unit capacity.

▶ **Lemma 11.** *$\mathcal{G}(\mathcal{V}, \mathcal{E})$ admits a flow of value k. Moreover, we can obtain a feasible solution of cost at most $3R^*$ from an integral flow of value k in $\mathcal{G}(\mathcal{V}, \mathcal{E})$.*

Proof. As an optimal solution with objective R^*, there exist two multi-sets $A_1 \subset F_1, A_2 \subset F_2$ such that $|A_1| = |A_2| = k$ and there exists a perfect matching between them. For any $i \in F_1, i' \in F_2$, if the pair (i, i') appears m times in the perfect matching, define a flow value $f(i, i') = m$ over link (i, i').

Consider the first time step. For any facility location i and clusters j, j', either $d(i, j)$ or $d(i, j')$ is larger than R^*, otherwise $d(j, j') \leq 2R^*$, contradicting with our construction. Because A_1 also covers all $j \in C_1$ with radius R^*, for every $j \in \mathcal{L}^{11}$, we can always find a different element $i \in A_1$ such that $d(i, j) \leq R^*$, and we add a unit flow as $f(j, i) = 1$. The same process is repeated for $\mathcal{L}_{22}$ and A_2.

The total flow between $\mathcal{L}^{12}$ and $\mathcal{L}^{21}$ is now obviously k, since the perfect matching between A_1 and A_2 has size k. After the construction of unit flows for non-dummy clusters, we arbitrarily direct the remaining flows from facility locations to the dummy clusters, one unit each time. Finally, for any cluster with unit flow, define the flow between it and the source/sink to be 1. This completes an integral flow of value k on $\mathcal{G}$.

For the second part, suppose we have an integral flow $\bar{f}$ of value k on $\mathcal{G}$. For any facility location $i \in F_1$, denote $g(i)$ the total flow through i. We place $g(i)$ facilities at location i, and repeat the same procedures for $i' \in F_2$. If $\bar{f}(i, i') = m$ for $i \in F_1, i' \in F_2$, move m facilities from i to i' in the transition between 2 time steps.

For any $j' \in C_1$, if j is the cluster center it belongs to, there exists a facility at most $d(j', i) \leq d(j', j) + d(j, i) \leq 3R^*$ away. ◀

▶ **Theorem 12.** *There exists a 3-approximation for Dynamic k-Supplier when $T = 2$.*

Proof. Consider the network flow instance we construct. It only has integer constraints and the coefficient matrix is totally unimodular. Moreover, there exists a flow of value k due to Lemma 11, hence we can efficiently compute an integral solution $\bar{f}$ of value k, thus obtaining a 3-approximation solution. ◀

3.2 The Hardness of Approximating Dynamic k-Supplier, $T \geq 3$

We show it is NP-hard to design approximation algorithms for Dynamic k-Supplier with any approximation factor when $T \geq 3$. The proof is via reduction from the perfect 3D matching problem, which is known to be NP-Complete [27].

▶ **Theorem 13.** *There is no polynomial time algorithm for solving Dynamic k-Supplier with any approximation factor if $T \geq 3$, unless $P = NP$.*

Proof. We reduce an arbitrary instance of perfect 3D-matching to Dynamic k-Supplier to show the NP-hardness. Recall for an instance of perfect 3D-matching, we are given three finite sets A, B, C with $|A| = |B| = |C|$, and a triplet set $\mathcal{T} \subset A \times B \times C$. Suppose $|A| = n$ and $|\mathcal{T}| = m$, and we are asked to decide whether there exists a subset $\mathcal{S} \subset \mathcal{T}$, such that $|\mathcal{S}| = n$, and each element in A, B, C appears exactly once in some triplet in $\mathcal{S}$. We construct the following graph $G = (V, E)$, where V, E are initially empty.

- For each triplet $g = (a, b, c) \in \mathcal{T}$, add three new vertices a_g, b_g, c_g to V correspondingly. Connect a_g, b_g with an edge of length α. Connect b_g, c_g with an edge of length α.
- Denote V_A all the vertices that correspond to vertices in A. Similarly for V_B and V_C.
- For any two vertices in V_A corresponding to the same element $a \in A$, connect them with an edge of length 1. Repeat the same procedure for V_B, V_C.

Assume we are able to solve Dynamic k-Supplier for $T = 3$ with an approximation factor α. We solve Dynamic k-Supplier for G on its graph metric d_G, with $k = n$ and the movement constraint $B = \alpha$, where the client sets are $\{V_A, V_B, V_C\}$ and facility sets are $\{V_A, V_B, V_C\}$ for the three time steps, respectively.

It is easy to see that the reduced Dynamic k-Supplier instance has covering radius $R^* = 1$ if and only if there exists a perfect 3D-matching, otherwise the covering radius is at least $2\alpha+1$. Since our approximation factor is α, this concludes the NP-hardness of approximation algorithms with any factor for Dynamic k-Supplier when $T \geq 3$. ◀

3.3 A Bi-criteria Approximation for Dynamic k-Supplier with Outliers

Lastly, we present our bi-criteria approximation algorithm that solves Dynamic k-Supplier, when $T = 2$ and outliers are allowed. As a useful ingredient, let us first briefly review the *m-budgeted bipartite matching* problem. The input consists of a bipartite graph $G = (V, E)$, and each edge $e \in E$ is associated with a weight $w(e) \geq 0$ and m types of lengths $f_i(e) \geq 0, i = 1, \ldots, m$. The problem asks for a maximum weight matching M with m budget constraints, where the ith constraint is that the sum of all f_i lengths in M is no more than L_i, i.e. $\sum_{e \in M} f_i(e) \leq L_i$. When the number of constraints m is a constant, a pure $(1 - \epsilon)$-approximation algorithm for any constant $\epsilon > 0$ is devised by Grandoni et al. [20].

Sketch. Due to space limit, we provide a sketch here and defer the details to the full version of this paper. Consider Dynamic k-Supplier with outliers and $T = 2$. In the solution, we place k facilities for time $t = 1$, serving in total at least l_1 clients in C_1, then move each of these facilities for a distance at most B to serve at least l_2 clients in C_2, and the maximum service distance is our minimization goal. Clearly, the optimal solution R^* only has a polynomial number of possible values and can be guessed efficiently, so we assume that R^* is known to us in the following analysis.

For a fixed R^*, denote c_i the number of clients that facility location i can serve within distance R^*. We assign two lengths $f_1(e) = c_i, f_2(e) = c_{i'}$ and weight $w(e) = 1$ for every candidate edge $e = (i, i')$, where $i \in F_1, i' \in F_2$. By duplicating each possible facility location in F_1 and F_2 and only allowing vertices within distance B to be matched, the required solution can be fully represented by a k-cardinality matching M between F_1 and F_2. Let us temporarily assume that any client miraculously contribute only once to the total number of clients served. Then the problem naturally translates to deciding whether there exists a bipartite matching M between F_1 and F_2 (with candidate facility locations duplicated) with weight k, such that the sum of all f_1 lengths in M is at least l_1, and the sum of all f_2 lengths in M is at least l_2.

This new problem is very similar to 2-*budgeted bipartite matching*, but there are still some major differences. In the approximation algorithm in [20], every integral matching M is obtained by first finding a *feasible* fractional matching M', which has at most $2m$ edges being fractional, and then dropping these fractionally-matched edges completely. Back to our problem where $m = 2$. If we obtain such a fractional solution M' which satisfies the constraints and only has at most 4 fractional edges, we would like to find an integral

matching M in a way that uses more "budget" instead of using less, so as to cover at least as many clients as M' does and not violate any budget constraint (in other words, outlier constraints), and we have to drop these fractional edges again from M'.

Contrary to 2-*budgeted bipartite matching*, we want to control the portion of budget dropped in this case. We achieve this by guessing a constant number of edges, which has either the top-θ f_1 lengths or top-θ f_2 lengths in the optimal solution, using a suitably chosen constant $\theta > 0$. We are able to devise a bi-criteria approximation algorithm that violates both budget constraints by any small constant ϵ-portion. The bi-criteria method is developed in line with the multi-criteria approximation schemes in [20].

To fully avoid counting any served client multiple times, whenever we duplicate a facility location, we make sure that only one copy induces non-zero lengths on edges that reside on it. We also use a greedy algorithm to remove some facility locations in F_1, F_2 and form client clusters around the remaining ones. Now, instead of defining c_i as the number of clients that facility location i can serve within distance R^*, we change c_i to the number of clients that are gathered around i. More specifically, for client set C_t and facility location set F_t, we find a subset $F'_t \subset F_t$ and a corresponding sub-partition $\{K_i\}_{i \in F'_t}$ of C_t (i.e., K_is are pair-wise disjoint and their union is a subset of C_t), such that $\forall j \in K_i, d(i,j) \leq 3R^*$ and we define $c_i = |K_i|$ for $i \in F'_t$, $c_i = 0$ for $i \in F_t \backslash F'_t$. Using this method, every client is counted at most once in all c_is, hence its contribution to the total number is always at most 1. The same filtering process can be found in [22]. See the full version of the paper for more details.

4 Future Directions

We list some interesting future directions and open problems.

1. It would be very interesting to remove the dependency of γ (the coefficient of movement cost) and ϵ (the lower bound of the weight) from the approximation factor for Dynamic Ordered k-Median in Theorem 10, or show such dependency is inevitable. We leave it as an important open problem. We note that a constant approximation factor for Dynamic Ordered k-Median without depending on γ would imply a constant approximation for stochastic k-server, for which only a logarithmic-factor approximation algorithm is known [15].
2. Our approximation algorithm for Dynamic Ordered k-Median is based on the technique developed in Byrka et al. [7]. The original ordered k-median problem has subsequently seen improved approximation results in [9, 10]. We did not try hard to optimize the constant factors. Nevertheless, it is an interesting future direction to further improve the constant approximation factors by leveraging the techniques from [9, 10] or other ideas.
3. From Theorem 13, we can see that Dynamic k-Supplier is hard to approximate when $T \geq 3$. However, it makes sense to relax the hard constraint B (we allow the distance a facility can move be at most αB for some constant α).

It is possible to formulate other concrete problems that naturally fit into the dynamic clustering theme and are well motivated by realistic applications, but not yet considered in the paper. For example, one can use the k-median objective for the service cost and the maximum distance of any facility movement as the movement cost. One can also consider combining the cost in more general fashion like in [10], or extending the problems to the fault-tolerant version [21, 28, 31] or the capacitated version [14, 29].

References

1 Sara Ahmadian, Zachary Friggstad, and Chaitanya Swamy. Local-search based approximation algorithms for mobile facility location problems. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1607–1621, 2013. doi:10.1137/1.9781611973105.115.

2 Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic facility location via exponential clocks. *ACM Transactions on Algorithms*, 13(2):1–20, 2017. doi:10.1145/2928272.

3 Ali Aouad and Danny Segev. The ordered k-median problem: surrogate models and approximation algorithms. *Mathematical Programming*, 177(1-2):55–83, 2019. doi:10.1007/s10107-018-1259-3.

4 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004. doi:10.1137/S0097539702416402.

5 David L Black, Anoop Gupta, and Wolf-Dietrich Weber. Competitive management of distributed shared memory. *COMPCON Spring 89*, pages 184–185, 1989. doi:10.1109/CMPCON.1989.301925.

6 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Transactions on Algorithms*, 13(2):1–31, 2017. doi:10.1145/2981561.

7 Jarosław Byrka, Krzysztof Sornat, and Joachim Spoerhase. Constant-factor approximation for ordered k-median. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 620–631, 2018. doi:10.1145/3188745.3188930.

8 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The non-uniform k-center problem. In *43rd International Colloquium on Automata, Languages, and Programming*, volume 55, page 67, 2016. doi:10.4230/LIPIcs.ICALP.2016.67.

9 Deeparnab Chakrabarty and Chaitanya Swamy. Interpolating between k-median and k-center: Approximation algorithms for ordered k-median. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107, page 29, 2018. doi:10.4230/LIPIcs.ICALP.2018.29.

10 Deeparnab Chakrabarty and Chaitanya Swamy. Approximation algorithms for minimum norm and ordered optimization problems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 126–137, 2019. doi:10.1145/3313276.3316322.

11 Moses Charikar, Samir Khuller, David M Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 642–651, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365555.

12 Moses Charikar and Shi Li. A dependent LP-rounding approach for the k-median problem. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming-Volume Part I*, pages 194–205, 2012. doi:10.1007/978-3-642-31594-7_17.

13 Danny Z Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and knapsack center problems. *Algorithmica*, 75(1):27–52, 2016. doi:10.1007/s00453-015-0010-1.

14 Julia Chuzhoy and Yuval Rabani. Approximating k-median with non-uniform capacities. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 952–958, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070569.

15 Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Stochastic k-server: How should Uber work? In *44th International Colloquium on Automata, Languages, and Programming*, volume 80, page 126, 2017. doi:10.4230/LIPIcs.ICALP.2017.126.

16 Erik D Demaine, MohammadTaghi Hajiaghayi, Hamid Mahini, Amin S Sayedi-Roshkhar, Shayan Oveisgharan, and Morteza Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms*, 5(3):30, 2009. doi:10.1145/1541885.1541891.

17 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In *International Colloquium on Automata, Languages, and Programming*, pages 459–470, 2014. doi:10.1007/978-3-662-43951-7_39.

18 Zachary Friggstad and Mohammad R Salavatipour. Minimizing movement in mobile facility location problems. *ACM Transactions on Algorithms*, 7(3):28, 2011. doi:10.1145/1978782.1978783.

19 Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM*, 53(3):324–360, 2006. doi:10.1145/1147954.1147956.

20 Fabrizio Grandoni, R Ravi, Mohit Singh, and Rico Zenklusen. New approaches to multi-objective optimization. *Mathematical Programming*, 146(1-2):525–554, 2014. doi:10.1007/s10107-013-0703-7.

21 Mohammadtaghi Hajiaghayi, Wei Hu, Jian Li, Shi Li, and Barna Saha. A constant factor approximation algorithm for fault-tolerant k-median. *ACM Transactions on Algorithms*, 12(3):36, 2016. doi:10.1145/2854153.

22 David G Harris, Thomas Pensyl, Aravind Srinivasan, and Khoa Trinh. A lottery model for center-type problems with outliers. *ACM Transactions on Algorithms*, 15(3):1–25, 2019. doi:10.1145/3311953.

23 Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985. doi:10.1287/moor.10.2.180.

24 Dorit S Hochbaum and David B Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986. doi:10.1145/5925.5933.

25 Kamal Jain and Vijay V Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *40th Annual Symposium on Foundations of Computer Science*, pages 2–13, 1999. doi:10.1109/SFFCS.1999.814571.

26 Kamal Jain and Vijay V Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001. doi:10.1145/375827.375845.

27 Richard M Karp. Reducibility among combinatorial problems. *50 Years of Integer Programming 1958-2008*, pages 219–241, 2010. doi:10.1007/978-3-540-68279-0_8.

28 Samir Khuller, Robert Pless, and Yoram J Sussmann. Fault tolerant k-center problems. *Theoretical Computer Science*, 242(1-2):237–245, 2000. doi:10.1016/S0304-3975(98)00222-9.

29 Shi Li. Approximating capacitated k-median with $(1+\varepsilon)k$ open facilities. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 786–796, 2016. doi:10.1137/1.9781611974331.ch56.

30 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the forty-fifth annual ACM symposium on Theory of Computing*, pages 901–910, 2013. doi:10.1145/2488608.2488723.

31 Chaitanya Swamy and David B Shmoys. Fault-tolerant facility location. *ACM Transactions on Algorithms*, 4(4):1–27, 2008. doi:10.1145/1383369.1383382.

32 Jeffery Westbrook. Randomized algorithms for multiprocessor page migration. *SIAM Journal on Computing*, 23(5):951–965, 1994. doi:10.1137/S0097539791199796.

A Sub-Linear Time Framework for Geometric Optimization with Outliers in High Dimensions

Hu Ding
School of Computer Science and Technology, University of Science and Technology of China, Anhui, China
http://staff.ustc.edu.cn/~huding/index.html
huding@ustc.edu.cn

Abstract

Many real-world problems can be formulated as geometric optimization problems in high dimensions, especially in the fields of machine learning and data mining. Moreover, we often need to take into account of outliers when optimizing the objective functions. However, the presence of outliers could make the problems to be much more challenging than their vanilla versions. In this paper, we study the fundamental minimum enclosing ball (MEB) with outliers problem first; partly inspired by the core-set method from Bădoiu and Clarkson, we propose a sub-linear time bi-criteria approximation algorithm based on two novel techniques, the Uniform-Adaptive Sampling method and Sandwich Lemma. To the best of our knowledge, our result is the first sub-linear time algorithm, which has the sample size (i.e., the number of sampled points) independent of both the number of input points n and dimensionality d, for MEB with outliers in high dimensions. Furthermore, we observe that these two techniques can be generalized to deal with a broader range of geometric optimization problems with outliers in high dimensions, including flat fitting, k-center clustering, and SVM with outliers, and therefore achieve the sub-linear time algorithms for these problems respectively.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases minimum enclosing ball, outliers, shape fitting, high dimensions, sub-linear time

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.38

Related Version A full version of the paper is available at https://arxiv.org/abs/2004.10090.

Acknowledgements The author wants to thank the anonymous reviewers for their helpful comments and suggestions for improving the paper.

1 Introduction

Geometric optimization is a fundamental topic that has been extensively studied in the community of computational geometry [1]. The *minimum enclosing ball (MEB)* problem is one of the most popular geometric optimization problems who has attracted a lot of attentions in past years, where the goal is to compute the smallest ball covering a given set of points in the Euclidean space [11, 50, 32]. Though its formulation is very simple, MEB has a number of applications in real world, such as classification [68, 20, 21], preserving privacy [59, 31], and quantum cryptography [38]. A more general geometric optimization problem is called *flat fitting* that is to compute the smallest slab (centered at a low-dimensional flat) to cover the input data [42, 60, 71]. Another closely related important topic is the *k-center clustering* problem, where the goal is to find $k > 1$ balls to cover the given input data and minimize the maximum radius of the balls [35]; the problem has been widely applied to many areas, such as facility location [18] and data analysis [67]. Moreover, some geometric optimization problems are trying to maximize their size functions. As an example, the well known classification technique *support vector machine (SVM)* [17] is to maximum the margin separating two differently labeled point sets in the space.

Real-world datasets are often noisy and contain outliers. Moreover, outliers could seriously affect the final optimization results. For example, it is easy to see that even one outlier could make the MEB arbitrarily large. In particular, as the rapid development of machine learning, the field of *adversarial machine learning* concerning about the potential vulnerabilities of the algorithms has attracted a great amount of attentions [45, 51, 9, 36]. A small set of outliers could be added by some adversarial attacker to make the decision boundary severely deviate [8, 48]. Furthermore, the presence of outliers often results in a quite challenging combinatorial optimization problem; as an example, if m of the input n data items are outliers ($m < n$), we have to consider an exponentially large number $\binom{n}{m}$ of different possible cases when optimizing the objective function. Therefore, the design of efficient and robust optimization algorithms is urgently needed to meet these challenges.

1.1 Our Contributions

In big data era, the data size could be so large that we cannot even afford to read the whole dataset once. In this paper, we consider to develop sub-linear time algorithms for several geometric optimization problems involving outliers. We study the aforementioned MEB with outliers problem first. Informally speaking, given a set of n points in $\mathbb{R}^d$ and a small parameter $\gamma \in (0, 1)$, the problem is to find the smallest ball covering at least $(1 - \gamma)n$ points from the input. We are aware of several existing sub-linear time bi-criteria approximation algorithms based on uniform sampling for MEB and k-center clustering with outliers [7, 46, 29], where the "bi-criteria" means that the ball (or the union of the k balls) is allowed to exclude a little more points than the pre-specified number of outliers. Their ideas are based on the theory of VC dimension [69]. But the sample size usually depends on the dimensionality d, which is roughly $O\big(\frac{1}{\delta^2\gamma}kd \cdot polylog(\frac{kd}{\delta\gamma})\big)$, if allowing to discard $(1 + \delta)\gamma n$ outliers with $\delta \in (0, 1)$ ($k = 1$ in the complexity for the MEB with outliers problem). A detailed overview on previous works is shown in Section 1.2.

Since many optimization problems in practice need to consider high-dimensional datasets, especially in the fields of machine learning and data mining, the above sample size from [7, 46, 29] could be very large. Partly inspired by the core-set method from Bădoiu and Clarkson [11] for computing MEB in high dimensions, **we are wondering that whether it is possible to remove the dependency on d in the sample size for MEB with outliers and other related high dimensional geometric optimization problems**. Given a parameter $\epsilon \in (0, 1)$, the method of [11] is a simple greedy algorithm that selects $\frac{2}{\epsilon}$ points (as the core-set) for constructing a $(1 + \epsilon)$-approximate MEB, where the resulting radius is at most $1 + \epsilon$ times the optimal one. A highlight of their method is that the core-set size $\frac{2}{\epsilon}$ is independent of d. However, there are several substantial challenges when applying their method to design sub-linear time algorithm for MEB with outliers. First, we need to implement the "greedy selection" step by a random sampling manner, but it is challenging to guarantee the resulting quality especially when the data is mixed with outliers. Second, the random sampling approach often yields a set of candidates for the ball center (e.g., we may need to repeatedly run the algorithm multiple times for boosting the success probability, and each time generates a candidate solution), and thus it is necessary to design an efficient strategy to determine which candidate is the best one in sub-linear time.

To tackle these challenges, we propose two key techniques, the novel "**Uniform-Adaptive Sampling**" method and "**Sandwich Lemma**". Roughly speaking, the Uniform-Adaptive Sampling method can help us to bound the error induced in each "randomized greedy selection" step; the Sandwich Lemma enables us to estimate the objective value of each candidate and select the best one in sub-linear time. To the best of our knowledge, our result

is the first sub-linear time approximation algorithm for the MEB with outliers problem with sample size being independent of the number of points n and the dimensionality d, which significantly improves the time complexities of existing algorithms.

Moreover, we observe that our proposed techniques can be used to solve a broader range of geometric optimization problems. We define a general optimization problem called **minimum enclosing "x" (MEX) with Outliers**, where the "x" stands for a specified kind of shape (e.g., the shape is a ball for MEB with outliers). We prove that it is able to generalize the Uniform-Adaptive Sampling method and Sandwich Lemma to adapt the shape "x", as long as it satisfies several properties. In particular we focus on the MEX with outlier problems including flat fitting, k-center clustering, and SVM with outliers; a common characteristic of these problems is that each of them has an iterative algorithm based on greedy selection for its vanilla version (without outliers) that is similar to the MEB algorithm of [11]. Though these problems have been widely studied before, the research in terms of their sub-linear time algorithms is still quite limited.

1.2 Related Work

Sub-linear time algorithms. The research on sub-linear time algorithms design has a long history [63, 25]. For example, a number of sub-linear time clustering algorithms have been studied in [47, 56, 57, 24]. Another important application of sub-linear time algorithms is property testing on graphs or probability distributions [34].

As mentioned before, the uniform sampling idea can be used to design sub-linear time algorithms for the problems of MEB and k-center clustering with outliers [7, 46, 29], but the sample size depends on the dimensionality d that could be very large in practice. Note that Alon et al. [7] presented another sub-linear time algorithm, which has the sample size independent of d, to test whether an input point set can be covered by a ball with a given radius; however, it is difficult to apply their method to solve the MEB with outliers problem as the algorithm relies on some nice properties of minimum enclosing ball, but these properties are not easy to be utilized when inliers and outliers are mixed. In [26], we proposed a notion of stability for MEB and developed the sub-linear time MEB algorithms for stable instance. Clarkson et al. [21] developed an elegant perceptron framework for solving several optimization problems arising in machine learning, such as MEB. For a set of n points in $\mathbb{R}^d$, their framework can solve the MEB problem in $\tilde{O}(\frac{n}{\epsilon^2} + \frac{d}{\epsilon})$ [1] time. Based on a stochastic primal-dual approach, Hazan et al. [44] provided an algorithm for solving the SVM problem in sub-linear time.

MEB and k-center clustering with outliers. *Core-set* is a popular technique to reduce the time complexities for many optimization problems [2, 61]. The core-set idea has also been used to compute approximate MEB in high dimensional space [20, 13, 50, 60, 49]. Bădoiu and Clarkson [11] showed that it is possible to find a core-set of size $\lceil 2/\epsilon \rceil$ that yields a $(1+\epsilon)$-approximate MEB. There are also several exact and approximation algorithms for MEB that do not rely on core-sets [32, 64, 6]. Streaming algorithms for computing MEB were also studied before [4, 16].

Bădoiu et al. [13] extended their core-set idea to the problems of MEB and k-center clustering with outliers, and achieved linear time bi-criteria approximation algorithms (if k is assumed to be a constant). Several algorithms for the low dimensional MEB with

[1] The asymptotic notation $\tilde{O}(f) = O\big(f \cdot polylog(\frac{nd}{\epsilon})\big)$.

outliers problem have been also developed [5, 30, 39, 54]. A 3-approximation algorithm for k-center clustering with outliers in arbitrary metrics was proposed by Charikar et al. [18]; Chakrabarty et al. [15] proposed a 2-approximation algorithm for k-center clustering with outliers. These algorithms often have high time complexities (e.g., $\Omega(n^2d)$). Recently, Ding et al. [29] provided a linear time greedy algorithm for k-center clustering with outliers based on the idea of the Gonzalez's algorithm [35]. Furthermore, there exist a number of works on streaming and distributed algorithms, such as [19, 55, 72, 53, 37, 14, 52].

Flat fitting with outliers. Given an integer $j \geq 0$ and a set of points in $\mathbb{R}^d$, the flat fitting problem is to find a j-dimensional flat having the smallest maximum distance to the input points [41]; obviously, the MEB problem is a special case with $j = 0$. In high dimensions, Har-Peled and Varadarajan [42] provided a linear time algorithm if j is assumed to be fixed; their running time was further reduced by Panigrahy [60] based on a core-set approach. There also exist several methods considering flat fitting with outliers but only for low-dimensional case [43, 3].

SVM with outliers. Given two point sets P_1 and P_2 in $\mathbb{R}^d$, the problem of *Support Vector Machine (SVM)* is to find the largest margin to separate P_1 and P_2 (if they are separable) [17]. SVM can be formulated as a quadratic programming problem, and a number of efficient techniques have been developed in the past, such as the soft margin SVM [22, 62], ν-SVM [65, 23], and Core-SVM [68, 20]. There also exist a number of works on designing robust algorithms of SVM with outliers [70, 40, 66, 28].

2 Definitions and Preliminaries

In this paper, we let $|A|$ denote the number of points of a given point set A in $\mathbb{R}^d$, and $||x - y||$ denote the Euclidean distance between two points x and y in $\mathbb{R}^d$. We use $\mathbb{B}(c, r)$ to denote the ball centered at a point c with radius $r > 0$. Below, we give several definitions used throughout this paper.

▶ **Definition 1** (Minimum Enclosing Ball (MEB)). *Given a set P of n points in $\mathbb{R}^d$, the MEB problem is to find a ball with minimum radius to cover all the points in P. The resulting ball and its radius are denoted by $MEB(P)$ and $Rad(P)$, respectively.*

A ball $\mathbb{B}(c, r)$ is called a λ-approximation of $MEB(P)$ for some $\lambda \geq 1$, if the ball covers all points in P and has radius $r \leq \lambda Rad(P)$.

▶ **Definition 2** (MEB with Outliers). *Given a set P of n points in $\mathbb{R}^d$ and a small parameter $\gamma \in (0, 1)$, the MEB with outliers problem is to find the smallest ball that covers $(1 - \gamma)n$ points. Namely, the task is to find a subset of P with size $(1 - \gamma)n$ such that the resulting MEB is the smallest among all possible choices of the subset. The obtained ball is denoted by $MEB(P, \gamma)$.*

For convenience, we use P_{opt} to denote the optimal subset of P with respect to $MEB(P, \gamma)$. That is, $P_{\text{opt}} = \arg_Q \min \left\{ Rad(Q) \mid Q \subset P, |Q| = (1 - \gamma)n \right\}$. From Definition 2, we can see that the main issue is to determine the subset of P. Actually, solving such combinatorial problems involving outliers are often challenging. In Section A.1, we present an example to show that it is impossible to achieve an approximation factor less than 2 for MEB with outliers, if the time complexity is required to be independent of n. Therefore, we consider finding the bi-criteria approximation. Actually, it is also a common way for solving other

optimization problems with outliers. For example, Mount et al. [58] and Meyerson et al. [56] studied the bi-criteria approximation algorithms respectively for the problems of linear regression and k-median clustering with outliers before.

▶ **Definition 3** (Bi-criteria Approximation). *Given an instance (P, γ) for MEB with outliers and two small parameters $0 < \epsilon, \delta < 1$, a $(1+\epsilon, 1+\delta)$-approximation of (P, γ) is a ball that covers at least $\big(1-(1+\delta)\gamma\big)n$ points and has radius at most $(1+\epsilon)Rad(P_{opt})$.*

When both ϵ and δ are small, the bi-criteria approximation is very close to the optimal solution with only slight changes on the number of covered points and the radius.

We also extend Definition 2 to the problem called **minimum enclosing "x" (MEX) with Outliers**, where the "x" could be any specified shape. To keep the structure of our paper more compact, we state the formal definition of MEX with outliers and the corresponding results in Section 5.

2.1 A More Careful Analysis for Core-set Construction in [11]

Before presenting our main results, we first revisit the core-set construction algorithm for MEB of Bădoiu and Clarkson [11], since their method will be used in our algorithms for MEB with outliers.

Let $0 < \epsilon < 1$. The algorithm in [11] yields an MEB core-set of size $2/\epsilon$ (for convenience, we always assume that $2/\epsilon$ is an integer). However, there is a small issue in their paper. The analysis assumes that the exact MEB of the core-set is computed in each iteration, but in fact one may only compute an approximate MEB. Thus, an immediate question is whether the quality is still guaranteed with such a change. Kumar et al. [50] fixed this issue, and showed that computing a $(1+O(\epsilon^2))$-approximate MEB for the core-set in each iteration still guarantees a core-set with size $O(1/\epsilon)$, where the hidden constant is larger than 80. Clarkson [20] systematically studied the *Frank-Wolfe* algorithm [33], and showed that the greedy core-set construction algorithm of MEB, as a special case of the Frank-Wolfe algorithm, yields a core-set with size slightly larger than $4/\epsilon$. Note that there exist several other methods yielding even lower core-set size [12, 49], but their construction algorithms are more complicated and thus not applicable to our problems. Increasing the core-set size from $2/\epsilon$ to α/ϵ (for some $\alpha > 2$) is neglectable in asymptotic analysis. But in Section 4, we will show that it could cause serious issue if outliers exist. Hence, a core-set of size $2/\epsilon$ is still desirable. **For this purpose, we provide a new analysis which is also interesting in its own right.**

For the sake of completeness, we first briefly introduce the idea of the core-set construction algorithm in [11]. Given a point set $P \subset \mathbb{R}^d$, the algorithm is a simple iterative procedure. Initially, it selects an arbitrary point from P and places it into an initially empty set T. In each of the following $2/\epsilon$ iterations, the algorithm updates the center of $MEB(T)$ and adds to T the farthest point from the current center of $MEB(T)$. Finally, the center of $MEB(T)$ induces a $(1+\epsilon)$-approximation for $MEB(P)$. The selected set of $2/\epsilon$ points (i.e., T) is called the core-set of MEB. To ensure the expected improvement in each iteration, [11] showed that the following two inequalities hold if the algorithm always selects the farthest point to the current center of $MEB(T)$:

$$r_{i+1} \geq (1+\epsilon)Rad(P) - L_i; \qquad r_{i+1} \geq \sqrt{r_i^2 + L_i^2}, \tag{1}$$

where r_i and r_{i+1} are the radii of $MEB(T)$ in the i-th and $(i+1)$-th iterations, respectively, and L_i is the shifting distance of the center of $MEB(T)$ from the i-th to $(i+1)$-th iteration.

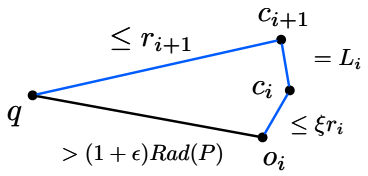


Figure 1 An illustration of (2).

As mentioned earlier, we often compute only an approximate $MEB(T)$ in each iteration. In the i-th iteration, we let c_i and o_i denote the centers of the exact and the approximate $MEB(T)$, respectively. Suppose that $||c_i - o_i|| \leq \xi r_i$, where $\xi \in (0, \frac{\epsilon}{1+\epsilon})$ (we will see why this bound is needed later). Using another algorithm proposed in [11], one can obtain the point o_i in $O(\frac{1}{\xi^2}|T|d)$ time. Note that we only compute o_i rather than c_i in each iteration. Hence we can only select the farthest point (say q) to o_i. If $||q - o_i|| \leq (1+\epsilon)Rad(P)$, we are done and a $(1+\epsilon)$-approximation of MEB is already obtained. Otherwise, we have

$$(1+\epsilon)Rad(P) < ||q - o_i|| \leq ||q - c_{i+1}|| + ||c_{i+1} - c_i|| + ||c_i - o_i|| \leq r_{i+1} + L_i + \xi r_i \quad (2)$$

by the triangle inequality (see Figure 1). In other words, we should replace the first inequality of (1) by $r_{i+1} > (1+\epsilon)Rad(P) - L_i - \xi r_i$. Also, the second inequality of (1) still holds since it depends only on the property of the exact MEB (see Lemma 2.1 in [11]). Thus, we have

$$r_{i+1} \geq \max\left\{\sqrt{r_i^2 + L_i^2}, (1+\epsilon)Rad(P) - L_i - \xi r_i\right\}. \quad (3)$$

This leads to the following theorem whose proof can be found in Section A.2.

▶ **Theorem 4.** *In the core-set construction algorithm of [11], if one computes an approximate MEB for T in each iteration and the resulting center o_i has the distance to c_i less than $\xi r_i = s\frac{\epsilon}{1+\epsilon}r_i$ for some $s \in (0,1)$, the final core-set size is bounded by $z = \frac{2}{(1-s)\epsilon}$. Also, the bound could be arbitrarily close to $2/\epsilon$ when s is small enough.*

▶ Remark 5. We want to emphasize a simple observation on the above core-set construction procedure, which will be used in our algorithms and analysis later on. The algorithm always selects the farthest point to o_i in each iteration. However, this is actually not necessary. As long as the selected point has distance at least $(1+\epsilon)Rad(P)$, the inequality (2) always holds and the following analysis is still true. If no such a point exists (i.e., $P \setminus \mathbb{B}\big(o_i, (1+\epsilon)Rad(P)\big) = \emptyset$), a $(1+\epsilon)$-approximate MEB (i.e., $\mathbb{B}\big(o_i, (1+\epsilon)Rad(P)\big)$) has already been obtained.

3 Two Key Lemmas for Handling Outliers

In this section, we introduce two important techniques, Lemma 6 and 7, for solving the problem of MEB with outliers in sub-linear time; the proofs are placed in Section 3.1 and 3.2, respectively. The algorithms are presented in Section 4. Moreover, these techniques can be generalized to solve a broader range of optimization problems, and we show the details in Section 5.

To shed some light on our ideas, consider using the core-set construction method in Section 2.1 to compute a bi-criteria $(1+\epsilon, 1+\delta)$-approximation for an instance (P, γ) of MEB with outliers. Let o_i be the obtained ball center in the current iteration, and Q be the set of $(1+\delta)\gamma n$ farthest points to o_i from P. A key step for updating o_i is finding a point in the set $P_{opt} \cap Q$ (the formal analysis is given in Section 4). Actually, this can be done

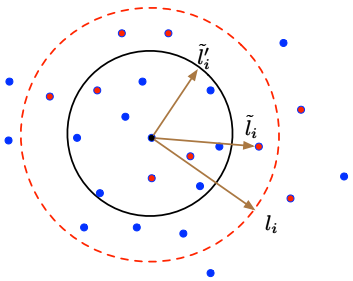


Figure 2 The red points are the sampled n'' points in Lemma 7, and the $\big((1+\delta)^2\gamma n''+1\big)$-th farthest point is in the ring bounded by the spheres $\mathbb{B}(o_i,\tilde{l}'_i)$ and $\mathbb{B}(o_i,l_i)$.

by performing a random sampling from Q. However, it requires to compute the set Q in advance, which takes an $\Omega(nd)$ time complexity. To keep the running time to be sub-linear, we need to find a point from $P_{opt}\cap Q$ by a more sophisticated way.

Since P_{opt} is mixed with outliers in the set Q, simple uniform sampling cannot realize our goal. To remedy this issue, we propose a "two level" sampling procedure which is called "**Uniform-Adaptive Sampling**" (see Lemma 6). Roughly speaking, we take a random sample A of size n' first (i.e., the uniform sampling step), and then randomly select a point from Q', the set of the farthest $\frac{3}{2}(1+\delta)\gamma n'$ points from A to o_i (i.e., the adaptive sampling step). According to Lemma 6, with probability at least $(1-\eta_1)\frac{\delta}{3(1+\delta)}$, the selected point belongs to $P_{opt}\cap Q$; more importantly, the sample size n' is independent of n and d. The key to prove Lemma 6 is to show that the size of the intersection $Q'\cap\big(P_{opt}\cap Q\big)$ is large enough. By setting an appropriate value for n', we can prove a lower bound of $|Q'\cap\big(P_{opt}\cap Q\big)|$.

▶ **Lemma 6** (Uniform-Adaptive Sampling). *Let $\eta_1\in(0,1)$. If we sample $n'=O(\frac{1}{\delta\gamma}\log\frac{1}{\eta_1})$ points independently and uniformly at random from P and let Q' be the set of farthest $\frac{3}{2}(1+\delta)\gamma n'$ points to o_i from the sample, then, with probability at least $1-\eta_1$, the following holds*

$$\frac{\Big|Q'\cap\big(P_{opt}\cap Q\big)\Big|}{|Q'|}\geq\frac{\delta}{3(1+\delta)}. \tag{4}$$

The Uniform-Adaptive Sampling procedure will result in a "side-effect". To boost the overall success probability, we have to repeatedly run the algorithm multiple times and each time the algorithm will generate a candidate solution (i.e., the ball center). Consequently we have to select the best one as our final solution. With a slight abuse of notation, we still use o_i to denote a candidate ball center; to achieve a $(1+\epsilon,1+\delta)$-approximation, its radius should be the $\big((1+\delta)\gamma n+1\big)$-th largest distance from P to o_i, which is denoted as l_i. A straightforward way is to compute the value "l_i" in linear time for each candidate and return the one having the smallest l_i. In this section, we propose the "**Sandwich Lemma**" to estimate l_i in sub-linear time (see Lemma 7). Let B be the set of n'' sampled points from P in Lemma 7, and $\tilde{l}_i$ be the $\big((1+\delta)^2\gamma n''+1\big)$-th largest distance from B to o_i. The key idea is to prove that the ball $\mathbb{B}(o_i,\tilde{l}_i)$ is "sandwiched" by two balls $\mathbb{B}(o_i,\tilde{l}'_i)$ and $\mathbb{B}(o_i,l_i)$, where $\tilde{l}'_i$ is a carefully designed value satisfying (i) $\tilde{l}'_i\leq\tilde{l}_i\leq l_i$ and (ii) $\Big|P\setminus\mathbb{B}(o_i,\tilde{l}'_i)\Big|\leq(1+O(\delta))\gamma n$. See Figure 2 for an illustration. These two conditions of $\tilde{l}'_i$ can imply the inequalities (5) and (6) of Lemma 7. Further, the inequalities (5) and (6) jointly imply that $\tilde{l}_i$ is a qualified estimation of l_i: if $\mathbb{B}(o_i,l_i)$ is a $(1+\epsilon,1+\delta)$-approximation, the ball $\mathbb{B}(o_i,\tilde{l}_i)$ should be a $(1+\epsilon,1+O(\delta))$-approximation. Similar to Lemma 6, the sample size n'' is also independent of n and d.

▶ **Lemma 7** (Sandwich Lemma). *Let $\eta_2 \in (0,1)$ and assume $\delta < 1/3$. If we sample $n'' = O\big(\frac{1}{\delta^2\gamma}\log\frac{1}{\eta_2}\big)$ points independently and uniformly at random from P and let $\tilde{l}_i$ be the $\big((1+\delta)^2\gamma n''+1\big)$-th largest distance from the sample to o_i, then, with probability $1-\eta_2$, the following holds*

$$\tilde{l}_i \leq l_i; \tag{5}$$

$$\Big|P \setminus \mathbb{B}(o_i, \tilde{l}_i)\Big| \leq (1+O(\delta))\gamma n. \tag{6}$$

3.1 Proof of Lemma 6

Let A denote the set of sampled n' points from P. First, we know $|Q| = (1+\delta)\gamma n$ and $|P_{opt} \cap Q| \geq \delta\gamma n$ (since there are at most γn outliers in Q). For ease of presentation, let $\lambda = \frac{|P_{opt}\cap Q|}{n} \geq \delta\gamma$. Let $\{x_i \mid 1 \leq i \leq n'\}$ be n' independent random variables with $x_i = 1$ if the i-th sampled point of A belongs to $P_{opt} \cap Q$, and $x_i = 0$ otherwise. Thus, $E[x_i] = \lambda$ for each i. Let σ be a small parameter in $(0,1)$. By using the Chernoff bound, we have $\mathbf{Pr}\Big(\sum_{i=1}^{n'} x_i \notin (1\pm\sigma)\lambda n'\Big) \leq e^{-O(\sigma^2\lambda n')}$. That is,

$$\mathbf{Pr}\Big(|A \cap \big(P_{opt}\cap Q\big)| \in (1\pm\sigma)\lambda n'\Big) \geq 1 - e^{-O(\sigma^2\lambda n')}. \tag{7}$$

Similarly, we have

$$\mathbf{Pr}\Big(|A\cap Q| \in (1\pm\sigma)(1+\delta)\gamma n'\Big) \geq 1 - e^{-O(\sigma^2(1+\delta)\gamma n')}. \tag{8}$$

Note that $n' = O(\frac{1}{\delta\gamma}\log\frac{1}{\eta_1})$. By setting $\sigma < 1/2$ in (7) and (8), we have

$$\Big|A \cap \big(P_{opt}\cap Q\big)\Big| > \frac{1}{2}\delta\gamma n' \quad \text{and} \quad \Big|A\cap Q\Big| < \frac{3}{2}(1+\delta)\gamma n' \tag{9}$$

with probability $1-\eta_1$. Note that Q contains all the farthest $(1+\delta)\gamma n$ points to o_i. Denote by l_i the $\big((1+\delta)\gamma n+1\big)$-th largest distance from P to o_i. Thus

$$A\cap Q = \{p \in A \mid ||p-o_i|| > l_i\}. \tag{10}$$

Also, since Q' is the set of the farthest $\frac{3}{2}(1+\delta)\gamma n'$ points to o_i from A, there exists some $l_i' > 0$ such that

$$Q' = \{p \in A \mid ||p-o_i|| > l_i'\}. \tag{11}$$

(10) and (11) imply that either $(A\cap Q) \subseteq Q'$ or $Q' \subseteq (A\cap Q)$. Since $\Big|A\cap Q\Big| < \frac{3}{2}(1+\delta)\gamma n'$ and $|Q'| = \frac{3}{2}(1+\delta)\gamma n'$, we know $\Big(A\cap Q\Big) \subseteq Q'$. Therefore,

$$\Big(A\cap\big(P_{opt}\cap Q\big)\Big) = \Big(P_{opt}\cap\big(A\cap Q\big)\Big) \subseteq Q'. \tag{12}$$

Obviously,

$$\Big(A\cap\big(P_{opt}\cap Q\big)\Big) \subseteq \big(P_{opt}\cap Q\big). \tag{13}$$

The above (12) and (13) together imply

$$\Big(A\cap\big(P_{opt}\cap Q\big)\Big) \subseteq \Big(Q'\cap\big(P_{opt}\cap Q\big)\Big). \tag{14}$$

Moreover, since $Q' \subseteq A$, we have

$$\Big(Q'\cap\big(P_{opt}\cap Q\big)\Big) \subseteq \Big(A\cap\big(P_{opt}\cap Q\big)\Big). \tag{15}$$

Consequently, (14) and (15) together imply $Q' \cap \big(P_{opt} \cap Q\big) = A \cap \big(P_{opt} \cap Q\big)$ and hence

$$\frac{\Big|Q' \cap \big(P_{opt} \cap Q\big)\Big|}{|Q'|} = \frac{\Big|A \cap \big(P_{opt} \cap Q\big)\Big|}{|Q'|} \geq \frac{\delta}{3(1+\delta)}, \tag{16}$$

where the final inequality comes from the first inequality of (9) and the fact $|Q'| = \frac{3}{2}(1+\delta)\gamma n'$.

3.2 Proof of Lemma 7

Let B denote the set of sampled n'' points from P. For simplicity, let $t = (1+\delta)\gamma n$. Assume $\tilde{l}'_i > 0$ is the value such that $\Big|P \setminus \mathbb{B}(o_i, \tilde{l}'_i)\Big| = \frac{(1+\delta)^2}{1-\delta}\gamma n$. Recall that l_i is the $\big(t+1\big)$-th largest distance from P to o_i. Since $(1+\delta)\gamma n < \frac{(1+\delta)^2}{1-\delta}\gamma n$, it is easy to know $\tilde{l}'_i \leq l_i$. Below, we aim to prove that the $\big((1+\delta)^2\gamma n'' + 1\big)$-th farthest point from B is in the ring bounded by the spheres $\mathbb{B}(o_i, \tilde{l}'_i)$ and $\mathbb{B}(o_i, l_i)$ (see Figure 2).

Again, using the Chernoff bound (let $\sigma = \delta/2$) and the same idea for proving (9), since $|B| = n'' = O\big(\frac{1}{\delta^2\gamma}\log\frac{1}{\eta_2}\big)$, we have

$$\Big|B \setminus \mathbb{B}(o_i, \tilde{l}'_i)\Big| \geq (1-\delta/2)\frac{(1+\delta)^2}{1-\delta}\gamma n'' > (1-\delta)\frac{(1+\delta)^2}{1-\delta}\gamma n'' = (1+\delta)^2\gamma n''; \tag{17}$$

$$\Big|B \cap Q\Big| \leq (1+\delta/2)\frac{t}{n}n'' < (1+\delta)\frac{t}{n}n'' = (1+\delta)^2\gamma n'', \tag{18}$$

with probability $1-\eta_2$. Suppose that (17) and (18) both hold. Recall that $\tilde{l}_i$ is the $\big((1+\delta)^2\gamma n'' + 1\big)$-th largest distance from the sampled points B to o_i, so $\Big|B \setminus \mathbb{B}(o_i, \tilde{l}_i)\Big| = (1+\delta)^2\gamma n''$, and thus $\tilde{l}_i \geq \tilde{l}'_i$ by (17).

The inequality (18) implies that the $\big((1+\delta)^2\gamma n'' + 1\big)$-th farthest point (say q_x) from B to o_i is not in Q. Then, we claim that $\mathbb{B}(o_i, \tilde{l}_i) \cap Q = \emptyset$. Otherwise, let $q_y \in \mathbb{B}(o_i, \tilde{l}_i) \cap Q$. Then we have

$$||q_y - o_i|| \leq \tilde{l}_i = ||q_x - o_i||. \tag{19}$$

Note that Q is the set of farthest t points to o_i of P. So $q_x \notin Q$ implies

$$||q_x - o_i|| < \min_{q \in Q} ||q - o_i|| \leq ||q_y - o_i|| \tag{20}$$

which is in contradiction to (19). Therefore, $\mathbb{B}(o_i, \tilde{l}_i) \cap Q = \emptyset$. Further, since $\mathbb{B}(o_i, l_i)$ excludes exactly the farthest t points (i.e., Q), $\mathbb{B}(o_i, \tilde{l}_i) \cap Q = \emptyset$ implies $\tilde{l}_i \leq l_i$.

Overall, we have $\tilde{l}_i \in [\tilde{l}'_i, l_i]$, i.e., the $\big((1+\delta)^2\gamma n'' + 1\big)$-th farthest point from B locates in the ring bounded by the spheres $\mathbb{B}(o_i, \tilde{l}'_i)$ and $\mathbb{B}(o_i, l_i)$ as shown in Figure 2. Also, $\tilde{l}_i \geq \tilde{l}'_i$ implies

$$\Big|P \setminus \mathbb{B}(o_i, \tilde{l}_i)\Big| \leq \Big|P \setminus \mathbb{B}(o_i, \tilde{l}'_i)\Big| = \frac{(1+\delta)^2}{1-\delta}\gamma n = (1+O(\delta))\gamma n, \tag{21}$$

where the last equality comes from the assumption $\delta < 1/3$. So (5) and (6) are true in Lemma 7.

4 Sub-linear Time Algorithm of MEB with Outliers

Recall the remark following Theorem 4. As long as the selected point has a distance to the center of $MEB(T)$ larger than $(1+\epsilon)$ times the optimal radius, the expected improvement will always be guaranteed. Following this observation, we investigate the following approach.

Suppose we run the core-set construction procedure decribed in Theorem 4 (we should replace P by P_{opt} in our following analysis). In the i-th step, we add an arbitrary point from $P_{\text{opt}} \setminus \mathbb{B}(o_i, (1+\epsilon)Rad(P_{opt}))$ to T. We know that a $(1+\epsilon)$-approximation is obtained after at most $\frac{2}{(1-s)\epsilon}$ steps, that is, $P_{opt} \subset \mathbb{B}\big(o_i, (1+\epsilon)Rad(P_{opt})\big)$ for some $i \leq \frac{2}{(1-s)\epsilon}$.

However, we need to solve two key issues in order to implement the above approach: **(i)** how to determine the value of $Rad(P_{opt})$ and **(ii)** how to correctly select a point from $P_{\text{opt}} \setminus \mathbb{B}(o_i, (1+\epsilon)Rad(P_{opt}))$. Actually, we can implicitly avoid the first issue via replacing $(1+\epsilon)Rad(P_{opt})$ by the t-th largest distance from the points of P to o_i, where we set $t = (1+\delta)\gamma n$ for achieving a $(1+\epsilon, 1+\delta)$-approximation in the following analysis. For the second issue, we randomly select one point from the farthest t points of P to o_i, and show that it belongs to $P_{\text{opt}} \setminus \mathbb{B}(o_i, (1+\epsilon)Rad(P_{opt}))$ with a certain probability.

Based on the above idea, we present a sub-linear time $(1+\epsilon, 1+\delta)$-approximation algorithm in this section. To better understand the algorithm, we show a linear time algorithm first (Algorithm 1 in Sections 4.1). Note that Bădoiu et al. [13] also achieved a $(1+\epsilon, 1+\delta)$-approximation algorithm but with a higher complexity. **Please see more details in our analysis on the running time at the end of Sections 4.1**. More importantly, we improve the running time of Algorithm 1 to be sub-linear. For this purpose, we need to avoid computing the farthest t points to o_i, since this operation will take linear time. Also, Algorithm 1 generates a set of candidates for the solution and we need to select the best one. This process also costs linear time. By using the techniques proposed in Section 3, we can remedy these issues and develop a sub-linear time algorithm that has the sample complexity independent of n and d, in Section 4.2.

4.1 A Linear Time Algorithm

Algorithm 1 $(1+\epsilon, 1+\delta)$-approximation Algorithm for MEB with Outliers.

Input: A point set P with n points in $\mathbb{R}^d$, the fraction of outliers $\gamma \in (0,1)$, and the parameters $0 < \epsilon, \delta < 1$, $z \in \mathbb{Z}^+$.

1: Let $t = (1+\delta)\gamma n$.
2: Initially, randomly select a point $p \in P$ and let $T = \{p\}$.
3: $i = 1$; repeat the following steps until $i > z$:
   (1) Compute the approximate MEB center o_i of T.
   (2) Let Q be the set of farthest t points from P to o_i; denote by l_i the $(t+1)$-th largest distance from P to o_i.
   (3) Randomly select a point $q \in Q$, and add it to T.
   (4) $i = i + 1$.
4: Output the ball $\mathbb{B}(o_{\hat{i}}, l_{\hat{i}})$ where $\hat{i} = \arg_i \min\{l_i \mid 1 \leq i \leq z\}$.

In this section, we present our linear time $(1+\epsilon, 1+\delta)$-approximation algorithm for MEB with outliers (see Algorithm 1). For convenience, denote by c_i and r_i the exact center and radius of $MEB(T)$ respectively in the i-th round of Step 3 of Algorithm 1. In Step 3(1), we compute the approximate center o_i with a distance to c_i of less than $\xi Rad(T) = s\frac{\epsilon}{1+\epsilon}Rad(T)$, where $s \in (0,1)$ as described in Theorem 4 (we will determine the value of s in our following analysis on the running time). The following theorem shows the success probability of Algorithm 1.

▶ **Theorem 8.** *If the input parameter $z = \frac{2}{(1-s)\epsilon}$ (we assume it is an integer for convenience), then with probability $(1-\gamma)(\frac{\delta}{1+\delta})^z$, Algorithm 1 outputs a $(1+\epsilon, 1+\delta)$-approximation for the MEB with outliers problem.*

Before proving Theorem 8, we present the following two lemmas first.

▶ **Lemma 9.** *With probability* $(1-\gamma)(\frac{\delta}{1+\delta})^z$*, after running* z *rounds in Step 3, the set* $T \subset P_{opt}$ *in Algorithm 1.*

Proof. Initially, because $|P_{opt}|/|P| = 1-\gamma$, the first selected point in Step 2 belongs to P_{opt} with probability $1-\gamma$. In each of the z rounds in Step 3, the selected point belongs to P_{opt} with probability $\frac{\delta}{1+\delta}$, since

$$\frac{|P_{opt} \cap Q|}{|Q|} = 1 - \frac{|Q \setminus P_{opt}|}{|Q|} \geq 1 - \frac{|P \setminus P_{opt}|}{|Q|} = 1 - \frac{\gamma n}{(1+\delta)\gamma n} = \frac{\delta}{1+\delta}. \tag{22}$$

Therefore, $T \subset P_{opt}$ with probability $(1-\gamma)(\frac{\delta}{1+\delta})^z$. ◀

▶ **Lemma 10.** *In the* i*-th round of Step 3 for* $1 \leq i \leq z$*, at least one of the following two events happens: (1)* o_i *is the ball center of a* $(1+\epsilon, 1+\delta)$*-approximation; (2)* $r_{i+1} > (1+\epsilon)Rad(P_{opt}) - ||c_i - c_{i+1}|| - \xi r_i$*.*

Proof. If $l_i \leq (1+\epsilon)Rad(P_{opt})$, then we are done. That is, the ball $\mathbb{B}(o_i, l_i)$ covers $(1-(1+\delta)\gamma)n$ points with radius $l_i \leq (1+\epsilon)Rad(P_{opt})$ (the first event happens). Otherwise, $l_i > (1+\epsilon)Rad(P_{opt})$ and we consider the second event. Let q be the point added to T in the i-th round. Using the triangle inequality, we have

$$||o_i - q|| \leq ||o_i - c_i|| + ||c_i - c_{i+1}|| + |c_{i+1} - q|| \leq \xi r_i + ||c_i - c_{i+1}|| + r_{i+1}. \tag{23}$$

Since $l_i > (1+\epsilon)Rad(P_{opt})$ and q lies outside of $\mathbb{B}(o_i, l_i)$, i.e, $||o_i - q|| \geq l_i > (1+\epsilon)Rad(P_{opt})$, (23) implies that the second event happens and the proof is completed. ◀

Proof of Theorem 8. Suppose that the first event of Lemma 10 never happens. As a consequence, we obtain a series of inequalities for each pair of radii r_{i+1} and r_i, i.e., $r_{i+1} > (1+\epsilon)Rad(P_{opt}) - ||c_i - c_{i+1}|| - \xi r_i$. Assume that $T \subset P_{opt}$ in Lemma 9, i.e., each time the algorithm correctly adds a point from P_{opt} to T. Using the almost identical idea for proving Theorem 4 in Section 2.1, we know that a $(1+\epsilon)$-approximate MEB of P_{opt} is obtained after at most z rounds. The success probability directly comes from Lemma 9. Overall, we obtain Theorem 8. ◀

Moreover, Theorem 8 implies the following corollary.

▶ **Corollary 11.** *If one repeatedly runs Algorithm 1* $O(\frac{1}{1-\gamma}(1+\frac{1}{\delta})^z)$ *times, with constant probability, the algorithm outputs a* $(1+\epsilon, 1+\delta)$*-approximation for the problem of MEB with outliers.*

Running time. In Theorem 8, we set $z = \frac{2}{(1-s)\epsilon}$ and $s \in (0,1)$. To keep z small, according to Theorem 4, we set $s = \frac{\epsilon}{2+\epsilon}$ so that $z = \frac{2}{\epsilon} + 1$ (only larger than the lower bound $\frac{2}{\epsilon}$ by 1). For each round of Step 3, we need to compute an approximate center o_i that has a distance to the exact one less than $\xi r_i = s\frac{\epsilon}{1+\epsilon} r_i = O(\epsilon^2) r_i$. Using the algorithm proposed in [11], this can be done in $O(\frac{1}{\xi^2}|T|d) = O(\frac{1}{\epsilon^5}d)$ time. Also, the set Q can be obtained in linear time by the algorithm in [10]. In total, the time complexity for obtaining a $(1+\epsilon, 1+\delta)$-approximation in Corollary 11 is

$$O(\frac{C}{\epsilon}(n + \frac{1}{\epsilon^5})d), \tag{24}$$

where $C = O(\frac{1}{1-\gamma}(1+\frac{1}{\delta})^{\frac{2}{\epsilon}+1})$. As mentioned before, Bădoiu et al. [13] also achieved a linear time bi-criteria approximation. However, the hidden constant of their running time is exponential in $\Theta(\frac{1}{\epsilon\mu})$ (where μ is defined in [13], and should be $\delta\gamma$ to ensure a $(1+\epsilon, 1+\delta)$-approximation) that is much larger than $\frac{2}{\epsilon} + 1$.

4.2 Improvement on Running Time

In this section, we show that the running time of Algorithm 1 can be further improved to be independent of the number of points n. First, we observe that it is not necessary to compute the set Q of the farthest t points in Step 3(2) of the algorithm. Actually, as long as the selected point q is part of $P_{opt} \cap Q$ in Step 3(3), a $(1+\epsilon, 1+\delta)$-approximation is still guaranteed. The Uniform-Adaptive Sampling procedure proposed in Section 3 can help us to obtain a point $q \in P_{opt} \cap Q$ without computing the set Q. Moreover, in Lemma 7, we show that the radius of each candidate solution can be estimated via random sampling. Overall, we achieve a sub-linear time algorithm (Algorithm 2). Following the analysis in Section 4.1, we set $s = \frac{\epsilon}{2+\epsilon}$ so that $z = \frac{2}{(1-s)\epsilon} = \frac{2}{\epsilon} + 1$. We present the results in Theorem 12 and Corollary 13. Comparing with Theorem 8, we have an extra $(1-\eta_1)(1-\eta_2)$ in the success probability in Theorem 12, due to the probabilities from Lemmas 6 and 7.

Algorithm 2 Sub-linear Time $(1+\epsilon, 1+O(\delta))$-approximation Algorithm for MEB with Outliers.

Input: A point set P with n points in $\mathbb{R}^d$, the fraction of outliers $\gamma \in (0,1)$, and the parameters $\epsilon, \eta_1, \eta_2 \in (0,1)$, $\delta \in (0, 1/3)$, and $z \in \mathbb{Z}^+$.

1: Let $n' = O(\frac{1}{\delta\gamma} \log \frac{1}{\eta_1})$, $n'' = O\big(\frac{1}{\delta^2\gamma} \log \frac{1}{\eta_2}\big)$, $t' = \frac{3}{2}(1+\delta)\gamma n'$, and $t'' = (1+\delta)^2\gamma n''$.

2: Initially, randomly select a point $p \in P$ and let $T = \{p\}$.

3: $i = 1$; repeat the following steps until $j = z$:

- **(1)** Compute the approximate MEB center o_i of T.
- **(2)** Sample n' points uniformly at random from P, and let Q' be the set of farthest t' points to o_i from the sample.
- **(3)** Randomly select a point $q \in Q'$, and add it to T.
- **(4)** Sample n'' points uniformly at random from P, and let $\tilde{l}_i$ be the $(t''+1)$-th largest distance from the sampled points to o_i.
- **(5)** $i = i + 1$.

4: Output the ball $\mathbb{B}(o_{\hat{i}}, \tilde{l}_{\hat{i}})$ where $\hat{i} = \arg_i \min\{\tilde{l}_i \mid 1 \le i \le z\}$.

▶ **Theorem 12.** *If the input parameter $z = \frac{2}{\epsilon} + 1$, then with probability $(1-\gamma)\big((1-\eta_1)(1-\eta_2)\frac{\delta}{3(1+\delta)}\big)^z$, Algorithm 2 outputs a $(1+\epsilon, 1+O(\delta))$-approximation for the problem of MEB with outliers.*

To boost the success probability in Theorem 12, we need to repeatedly run Algorithm 2 and output the best candidate. However, we need to be careful on setting the parameters. The success probability in Theorem 12 consists of two parts, $\mathcal{P}_1 = (1-\gamma)\big((1-\eta_1)\frac{\delta}{3(1+\delta)}\big)^z$ and $\mathcal{P}_2 = (1-\eta_2)^z$, where $\mathcal{P}_1$ indicates the probability that $\{o_1, \cdots, o_z\}$ contains a qualified candidate, and $\mathcal{P}_2$ indicates the success probability of Lemma 7 over all the z rounds. Therefore, if we run Algorithm 2 $N = O(\frac{1}{\mathcal{P}_1})$ times, with constant probability (by taking the union bound), the set of all the generated candidates contains at least one that yields a $(1+\epsilon, 1+O(\delta))$-approximation; moreover, to guarantee that we can correctly estimate the resulting radii of all the candidates via the Sandwich Lemma with constant probability, we need to set $\eta_2 = O(\frac{1}{zN})$ (because there are $O(zN)$ candidates).

▶ **Corollary 13.** *If one repeatedly runs Algorithm 2 $N = O\Big(\frac{1}{1-\gamma}\big(\frac{1}{1-\eta_1}(3+\frac{3}{\delta})\big)^z\Big)$ times with setting $\eta_2 = O(\frac{1}{zN})$, with constant probability, the algorithm outputs a $(1+\epsilon, 1+O(\delta))$-approximation for the problem of MEB with outliers.*

The calculation of running time is similar to (24) in Section 4.1. We just replace n by $\max\{n', n''\} = O\big(\frac{1}{\delta^2\gamma}\log\frac{1}{\eta_2}\big) = O\big(\frac{1}{\delta^2\gamma}\log(zN)\big) = \tilde{O}\big(\frac{1}{\delta^2\gamma\epsilon}\big)$ [2], and change the value of C to be $O\Big(\frac{1}{1-\gamma}\big(\frac{1}{1-\eta_1}(3+\frac{3}{\delta})\big)^{\frac{2}{\epsilon}+1}\Big)$. So the total running time is independent of n. Also, to covert the result from $(1+\epsilon, 1+O(\delta))$-approximation to $(1+\epsilon, 1+\delta)$-approximation, we just need to reduce the value of δ in the input of Algorithm 2 appropriately.

5 The Extension: MEX with Outliers

In this section, we extend Definition 2 and define a more general problem called **minimum enclosing "x" (MEX) with Outliers**. We observe that the ideas of Lemma 6 and 7 can be further extended to deal with MEX with outliers problems, as long as the shape "x" satisfies several properties. **To describe a shape "x", we need to clarify three basic concepts: center, size, and distance function.**

Let $\mathcal{X}$ be the set of specified shapes in $\mathbb{R}^d$. In this paper, we require that each shape $x \in \mathcal{X}$ is uniquely determined by the following two components: "$c(x)$", the **center** of x, and "$s(x) \geq 0$", the **size** of x. For any two shapes $x_1, x_2 \in \mathcal{X}$, $x_1 = x_2$ if and only if $c(x_1) = c(x_2)$ and $s(x_1) = s(x_2)$. Moreover, given a center o_0 and a value $l_0 \geq 0$, we use $x(o_0, l_0)$ to denote the shape x with $c(x) = o_0$ and $s(x) = l_0$. For different shapes, we have different definitions for the center and size. For example, if x is a ball, $c(x)$ and $s(x)$ should be the ball center and the radius respectively; given $o_0 \in \mathbb{R}^d$ and $l_0 \geq 0$, $x(o_0, l_0)$ should be the ball $\mathbb{B}(o_0, l_0)$. As a more complicated example, consider the k-center clustering with outliers problem, which is to find k balls to cover the input point set excluding a certain number of outliers and minimize the maximum radius (*w.l.o.g.,* we can assume that the k balls have the same radius). For this problem, the shape "x" is a union of k balls in $\mathbb{R}^d$; the center $c(x)$ is the set of the k ball centers and the size $s(x)$ is the radius.

For any point $p \in \mathbb{R}^d$ and any shape $x \in \mathcal{X}$, we also need to define a **distance function** $f(c(x), p)$ between the center $c(x)$ and p. For example, if x is a ball, $f(c(x), p)$ is simply equal to $||p - c(x)||$; if x is a union of k balls with the center $c(x) = \{c_1, c_2, \cdots, c_k\}$, the distance should be $\min_{1\leq j\leq k} ||p - c_j||$. Note that the distance function is only for ranking the points to $c(x)$, and not necessary to be non-negative (e.g., we define a distance function $f(c(x), p) \leq 0$ for SVM). By using this distance function, we can define the set "Q" and the value "l_i" when generalizing Lemma 6 and 7 below. To guarantee their correctnesses, we also require $\mathcal{X}$ to satisfy the following three properties.

▶ **Property 1.** *For any two shapes $x_1 \neq x_2 \in \mathcal{X}$, if $c(x_1) = c(x_2)$, then*

$$s(x_1) \leq s(x_2) \Longleftrightarrow x_1 \text{ is covered by } x_2, \tag{25}$$

where "x_1 is covered by x_2" means "for any point $p \in \mathbb{R}^d$, $p \in x_1 \Rightarrow p \in x_2$".

▶ **Property 2.** *Given any shape $x \in \mathcal{X}$ and any point $p_0 \in x$, the set*

$$\{p \mid p \in \mathbb{R}^d \text{ and } f(c(x), p) \leq f(c(x), p_0)\} \subseteq x. \tag{26}$$

▶ **Property 3.** *Given any shape center o_0 and any point $p_0 \in \mathbb{R}^d$, they together determine a value $r_0 = \min\{r \mid r \geq 0, p_0 \in x(o_0, r)\}$; that is, $p_0 \in x(o_0, r_0)$ and $p_0 \notin x(o_0, r)$ for any $r < r_0$. (**Note:** usually the value r_0 is just the distance from p_0 to the shape center o_0; but for some cases, such as the SVM problem, the shape size and distance function have different meanings).*

[2] The asymptotic notation $\tilde{O}(f) = O\big(f \cdot polylog(\frac{1}{\eta_1\delta(1-\gamma)})\big)$.

Intuitively, Property 1 shows that $s(x)$ defines an order of the shapes sharing the same center $c(x)$. Property 2 shows that the distance function f defines an order of the points to a given shape center $c(x)$. Property 3 shows that a center o_0 and a point p_0 can define a shape just "touching" p_0. We can take $\mathcal{X} = \{\text{all } d\text{-dimensional balls}\}$ as an example. For any two concentric balls, the smaller one is always covered by the larger one (Property 1); if a point p_0 is inside a ball x, any point p having the distance $||p - c(x)|| \leq ||p_0 - c(x)||$ should be inside x too (Property 2); also, given a ball center o_0 and a point p_0, $p_0 \in \mathbb{B}(o_0, ||p_0 - o_0||)$ and $p_0 \notin \mathbb{B}(o_0, r)$ for any $r < ||p_0 - o_0||$ (Property 3).

Now, we introduce the formal definitions of the MEX with outliers problem and its bi-criteria approximation.

▶ **Definition 14** (MEX with Outliers)**.** *Suppose the shape set $\mathcal{X}$ satisfies Property 1, 2, and 3. Given a set P of n points in $\mathbb{R}^d$ and a small parameter $\gamma \in (0, 1)$, the MEX with outliers problem is to find the smallest shape $x \in \mathcal{X}$ that covers $(1-\gamma)n$ points. Namely, the task is to find a subset of P with size $(1-\gamma)n$ such that its minimum enclosing shape of $\mathcal{X}$ is the smallest among all possible choices of the subset. The obtained solution is denoted by $MEX(P, \gamma)$.*

▶ **Definition 15** (Bi-criteria Approximation)**.** *Given an instance (P, γ) for MEX with outliers and two small parameters $0 < \epsilon, \delta < 1$, a $(1+\epsilon, 1+\delta)$-approximation of (P, γ) is a solution $x \in \mathcal{X}$ that covers at least $\big(1-(1+\delta)\gamma\big)n$ points and has the size at most $(1+\epsilon)s(x_{opt})$, where x_{opt} is the optimal solution.*

It is easy to see that Definition 2 of MEB with outliers actually is a special case of Definition 14. Similar to MEB with outliers, we still use P_{opt}, where $P_{opt} \subset P$ and $|P_{opt}| = (1-\gamma)n$, to denote the subset covered by the optimal solution of MEX with outliers.

Now, we provide the generalized versions of Lemma 6 and 7. Similar to the core-set construction method in Section 2.1, we assume that there exists an iterative algorithm Γ to compute MEX (without outliers); actually, this is an important prerequisite to design the sub-linear time algorithms under our framework (we will discuss the iterative algorithms for the MEX with outliers problems considered in our paper, including flat fitting, k-center clustering, and SVM, in the appendix). In the i-th iteration of Γ, it maintains a shape center o_i. Also, let Q be the set of $(1+\delta)\gamma n$ farthest points from P to o_i with respect to the distance function f. First, we need to define the value "l_i" by Q in the following claim.

▷ Claim 16. There exists a value $l_i \geq 0$ satisfying $P \setminus x(o_i, l_i) = Q$.

Proof. The points of P can be ranked based on their distances to o_i. Without loss of generality, let $P = \{p_1, p_2, \cdots, p_n\}$ with $f(o_i, p_1) > f(o_i, p_2) > \cdots > f(o_i, p_n)$ (for convenience, we assume that any two distances are not equal; if there is a tie, we can arbitrarily decide their order to o_i). Therefore, the set $Q = \{p_j \mid 1 \leq j \leq (1+\delta)\gamma n\}$. Moreover, from Property 3, we know that each point $p_j \in P$ corresponds to a value r_j that $p_j \in x(o_i, r_j)$ and $p_j \notin x(o_i, r)$ for any $r < r_j$. Denote by x_j the shape $x(o_i, r_j)$. We select the point p_{j_0} with $j_0 = (1+\delta)\gamma n + 1$. From Property 2, we know that $p_j \in x_{j_0}$ for any $j \geq j_0$, i.e., **(a)** $P \setminus Q \subseteq x_{j_0}$. We also need to prove that $p_j \notin x_{j_0}$ for any $j < j_0$. Assume there exists some $p_{j_1} \in x_{j_0}$ with $j_1 < j_0$. Then we have $r_{j_1} < r_{j_0}$ and thus $p_{j_0} \notin x_{j_1}$ (by Property 3). By Property 2, $p_{j_0} \notin x_{j_1}$ implies $f(o_i, p_{j_0}) > f(o_i, p_{j_1})$, which is in contradiction to the fact $f(o_i, p_{j_0}) < f(o_i, p_{j_1})$. So we have **(b)** $Q \cap x_{j_0} = \emptyset$.

From the above **(a)** and **(b)**, we know $P \setminus x_{j_0} = Q$. Therefore, we can set the value $l_i = r_{j_0}$ and then $P \setminus x(o_i, l_i) = Q$. ◁

▶ **Lemma 17** (Generalized Uniform-Adaptive Sampling). *Let $\eta_1 \in (0,1)$. If we sample $n' = O(\frac{1}{\delta\gamma}\log\frac{1}{\eta_1})$ points independently and uniformly at random from P and let Q' be the set of farthest $\frac{3}{2}(1+\delta)\gamma n'$ points to o_i from the sample, then, with probability at least $1-\eta_1$, the following holds*

$$\frac{\left|Q' \cap \left(P_{opt} \cap Q\right)\right|}{|Q'|} \geq \frac{\delta}{3(1+\delta)}. \tag{27}$$

Proof. Let A denote the set of sampled n' points from P. Similar to (9), we have

$$\left|A \cap \left(P_{opt} \cap Q\right)\right| > \frac{1}{2}\delta\gamma n' \quad \text{and} \quad \left|A \cap Q\right| < \frac{3}{2}(1+\delta)\gamma n' \tag{28}$$

with probability $1-\eta_1$. Similar to (10), we have

$$A \cap Q = \{p \in A \mid f(o_i, p) > f(o_i, p_{j_0})\}, \tag{29}$$

where p_{j_0} is the point selected in the proof of Claim 16. By using the same manner of Claim 16, we also can select a point $p_{j_0'} \in A$ with

$$Q' = \{p \in A \mid f(o_i, p) > f(o_i, p_{j_0'})\}. \tag{30}$$

Then, we can prove

$$\left(A \cap \left(P_{opt} \cap Q\right)\right) = \left(Q' \cap \left(P_{opt} \cap Q\right)\right). \tag{31}$$

by using the same idea of (14). Hence,

$$\frac{\left|Q' \cap \left(P_{opt} \cap Q\right)\right|}{|Q'|} = \frac{\left|A \cap \left(P_{opt} \cap Q\right)\right|}{|Q'|} \geq \frac{\delta}{3(1+\delta)}, \tag{32}$$

where the final inequality comes from the first inequality of (28) and the fact $|Q'| = \frac{3}{2}(1+\delta)\gamma n'$. ◀

▶ **Lemma 18** (Generalized Sandwich Lemma). *Let $\eta_2 \in (0,1)$ and assume $\delta < 1/3$. l_i is the value from Claim 16. We sample $n'' = O\left(\frac{1}{\delta^2\gamma}\log\frac{1}{\eta_2}\right)$ points independently and uniformly at random from P and let q be the $\left((1+\delta)^2\gamma n'' + 1\right)$-th farthest one from the sampled points to o_i. If $\tilde{l}_i = \min\{r \mid r \geq 0, q \in x(o_i, r)\}$ (similar to the way defining "r_0" in Property 3), then, with probability $1-\eta_2$, the following holds*

$$\tilde{l}_i \leq l_i; \tag{33}$$

$$\left|P \setminus x(o_i, \tilde{l}_i)\right| \leq (1+O(\delta))\gamma n. \tag{34}$$

Proof. Let B denote the set of sampled n'' points from P. By using the same manner of Claim 16, we know that there exists a value $\tilde{l}_i' > 0$ satisfying $\left|P \setminus x(o_i, \tilde{l}_i')\right| = \frac{(1+\delta)^2}{1-\delta}\gamma n$. Similar to the proof of Lemma 7, we can prove that $\tilde{l}_i \in [\tilde{l}_i', l_i]$. Due to Property 1, we know that $x(o_i, \tilde{l}_i)$ is "sandwiched" by the two shapes $x(o_i, \tilde{l}_i')$ and $x(o_i, l_i)$. Further, since $x(o_i, \tilde{l}_i')$ is covered by $x(o_i, \tilde{l}_i)$, we have

$$\left|P \setminus x(o_i, \tilde{l}_i)\right| \leq \left|P \setminus x(o_i, \tilde{l}_i')\right| = \frac{(1+\delta)^2}{1-\delta}\gamma n = (1+O(\delta))\gamma n, \tag{35}$$

where the last equality comes from the assumption $\delta < 1/3$. So (33) and (34) are true. ◀

Due to the space limit, we place other parts to the full version of our paper [27]. We propose the sub-linear time bi-criteria approximation algorithms for three different MEX with outlier problems: k-center clustering, flat fitting, and SVM with outliers. All of these problems have important applications in real world, and our results significantly reduce their time complexities comparing with existing approaches.

References

1 Pankaj K. Agarwal, Esther Ezra, and Kyle Fox. Geometric optimization revisited. In Bernhard Steffen and Gerhard J. Woeginger, editors, *Computing and Software Science - State of the Art and Perspectives*, volume 10000 of *Lecture Notes in Computer Science*, pages 66–84. Springer, 2019.

2 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. *Combinatorial and Computational Geometry*, 52:1–30, 2005.

3 Pankaj K. Agarwal, Sariel Har-Peled, and Hai Yu. Robust shape fitting via peeling and grating coresets. *Discrete & Computational Geometry*, 39(1-3):38–58, 2008.

4 Pankaj K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. *Algorithmica*, 72(1):83–98, 2015.

5 Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *Journal of algorithms*, 12(1):38–56, 1991.

6 Zeyuan Allen Zhu, Zhenyu Liao, and Yang Yuan. Optimization algorithms for faster computational geometry. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 53:1–53:6, 2016.

7 Noga Alon, Seannie Dar, Michal Parnas, and Dana Ron. Testing of clustering. *SIAM Journal on Discrete Mathematics*, 16(3):393–417, 2003.

8 Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

9 Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

10 Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

11 Mihai Bădoiu and Kenneth L. Clarkson. Smaller core-sets for balls. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–802, 2003.

12 Mihai Bădoiu and Kenneth L. Clarkson. Optimal core-sets for balls. *Computational Geometry*, 40(1):14–22, 2008.

13 Mihai Bădoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 250–257, 2002.

14 Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. Solving k-center clustering (with outliers) in mapreduce and streaming, almost as accurately as sequentially. *PVLDB*, 12(7):766–778, 2019.

15 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The non-uniform k-center problem. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 67:1–67:15, 2016.

16 Timothy M. Chan and Vinayak Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. *Comput. Geom.*, 47(2):240–247, 2014.

17 Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2(3), 2011.

18 Moses Charikar, Samir Khuller, David M Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 642–651. Society for Industrial and Applied Mathematics, 2001.

19 Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39. ACM, 2003.

20 Kenneth L. Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Transactions on Algorithms*, 6(4):63, 2010.

21 Kenneth L. Clarkson, Elad Hazan, and David P. Woodruff. Sublinear optimization for machine learning. *J. ACM*, 59(5):23:1–23:49, 2012.

22 Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273, 1995.

23 David J. Crisp and Christopher J. C. Burges. A geometric interpretation of v-SVM classifiers. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *NIPS*, pages 244–250. The MIT Press, 1999.

24 Artur Czumaj and Christian Sohler. Sublinear-time approximation for clustering via random sampling. In *International Colloquium on Automata, Languages, and Programming*, pages 396–407. Springer, 2004.

25 Artur Czumaj and Christian Sohler. Sublinear-time algorithms. In *Property Testing - Current Research and Surveys*, pages 41–64, 2010.

26 Hu Ding. Minimum enclosing ball revisited: Stability and sub-linear time algorithms. *CoRR*, abs/1904.03796, 2019.

27 Hu Ding. A sub-linear time framework for geometric optimization with outliers in high dimensions. *CoRR*, abs/2004.10090, 2020. `arXiv:2004.10090`.

28 Hu Ding and Jinhui Xu. Random gradient descent tree: A combinatorial approach for svm with outliers. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 2561–2567, 2015.

29 Hu Ding, Haikuo Yu, and Zixiu Wang. Greedy strategy works for k-center clustering with outliers and coreset construction. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.*, pages 40:1–40:16, 2019.

30 Alon Efrat, Micha Sharir, and Alon Ziv. Computing the smallest k-enclosing circle and related problems. *Computational Geometry*, 4(3):119–136, 1994.

31 Dan Feldman, Chongyuan Xiang, Ruihao Zhu, and Daniela Rus. Coresets for differentially private k-means clustering and applications to privacy in mobile sensor networks. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN 2017, Pittsburgh, PA, USA, April 18-21, 2017*, pages 3–15, 2017.

32 Kaspar Fischer, Bernd Gärtner, and Martin Kutz. Fast smallest-enclosing-ball computation in high dimensions. In *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, pages 630–641, 2003.

33 Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

34 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

35 Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

36 Ian J. Goodfellow, Patrick D. McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Commun. ACM*, 61(7):56–66, 2018.

37 Sudipto Guha, Yi Li, and Qin Zhang. Distributed partial clustering. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 143–152. ACM, 2017.

38 Laszlo Gyongyosi and Sandor Imre. Geometrical analysis of physically allowed quantum cloning transformations for quantum cryptography. *Information Sciences*, 285:1–23, 2014.

39 Sariel Har-Peled and Soham Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005.

40 Sariel Har-Peled, Dan Roth, and Dav Zimak. Maximum margin coresets for active and noise tolerant learning. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 836–841, 2007.

41 Sariel Har-Peled and Kasturi R. Varadarajan. Approximate shape fitting via linearization. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 66–73, 2001.

42 Sariel Har-Peled and Kasturi R. Varadarajan. High-dimensional shape fitting in linear time. *Discret. Comput. Geom.*, 32(2):269–288, 2004.

43 Sariel Har-Peled and Yusu Wang. Shape fitting with outliers. *SIAM Journal on Computing*, 33(2):269–285, 2004.

44 Elad Hazan, Tomer Koren, and Nati Srebro. Beating SGD: learning svms in sublinear time. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 1233–1241, 2011.

45 Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58, 2011.

46 Lingxiao Huang, Shaofeng Jiang, Jian Li, and Xuan Wu. Epsilon-coresets for clustering (with outliers) in doubling metrics. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 814–825, 2018.

47 Piotr Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 428–434, 1999.

48 Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 19–35, 2018.

49 Michael Kerber and R. Sharathkumar. Approximate čech complex in low and high dimensions. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, pages 666–676, 2013.

50 Piyush Kumar, Joseph S. B. Mitchell, and E. Alper Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *ACM Journal of Experimental Algorithmics*, 8, 2003.

51 Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint*, 2016. `arXiv:1611.01236`.

52 Shi Li and Xiangyu Guo. Distributed k-clustering for data with heavy noise. In *Advances in Neural Information Processing Systems*, pages 7849–7857, 2018.

53 Gustavo Malkomes, Matt J Kusner, Wenlin Chen, Kilian Q Weinberger, and Benjamin Moseley. Fast distributed k-center clustering with outliers on massive data. In *Advances in Neural Information Processing Systems*, pages 1063–1071, 2015.

54 Jiří Matoušek. On enclosing k points by a circle. *Information Processing Letters*, 53(4):217–221, 1995.

55 Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 165–178. Springer, 2008.

56 Adam Meyerson, Liadan O'Callaghan, and Serge A. Plotkin. A k-median algorithm with running time independent of data size. *Machine Learning*, 56(1-3):61–87, 2004.

57 Nina Mishra, Dan Oblinger, and Leonard Pitt. Sublinear time approximate clustering. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 439–447. Society for Industrial and Applied Mathematics, 2001.

58 David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. On the least trimmed squares estimator. *Algorithmica*, 69(1):148–183, 2014.

59 Kobbi Nissim, Uri Stemmer, and Salil P. Vadhan. Locating a small cluster privately. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 413–427, 2016.

60 Rina Panigrahy. Minimum enclosing polytope in high dimensions. *arXiv preprint*, 2004. `arXiv:cs/0407020`.

61 Jeff M. Phillips. Coresets and sketches. *Computing Research Repository*, 2016.

62 J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

63 Ronitt Rubinfeld. Sublinear time algorithms. In *Survey*. Citeseer, 2006.

64 Ankan Saha, S. V. N. Vishwanathan, and Xinhua Zhang. New approximation algorithms for minimum enclosing convex shapes. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1146–1160, 2011.

65 B. Scholkopf, A. J. Smola, K. R. Muller, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

66 Shinya Suzumura, Kohei Ogawa, Masashi Sugiyama, and Ichiro Takeuchi. Outlier path: A homotopy algorithm for robust svm. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1098–1106, 2014.

67 Pang-Ning Tan, Michael S. Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

68 Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005. URL: `http://jmlr.org/papers/v6/tsang05a.html`.

69 Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

70 Linli Xu, Koby Crammer, and Dale Schuurmans. Robust support vector machine training via convex outlier ablation. In *AAAI*, pages 536–542. AAAI Press, 2006.

71 Hai Yu, Pankaj K. Agarwal, Raghunath Poreddy, and Kasturi R. Varadarajan. Practical methods for shape fitting and kinetic data structures using coresets. *Algorithmica*, 52(3):378–402, 2008.

72 Hamid Zarrabi-Zadeh and Asish Mukhopadhyay. Streaming 1-center with outliers in high dimensions. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, pages 83–86, 2009.

A Appendix

A.1 The Lower Bound of Sample Size for MEB with Outliers

Actually, it is easy to obtain a sub-linear time randomized 2-approximation algorithm for MEB with outliers (if only returning the center): one can randomly select one point $p \in P$, and it belongs P_{opt} with probability $1-\gamma$; thus, with probability $1-\gamma$, the point p yields a 2-approximation by using the triangle inequality. Below, we consider an example and show that it is impossible to achieve a single-criterion λ-approximation for any $\lambda < 2$, if the time complexity is required to be independent of n.

Let q_a, q_b, and q_c be three colinear points in $\mathbb{R}^d$, where $||q_a - q_b|| = x$ and $||q_b - q_c|| = y$ (see Figure 3). Let $\gamma \in (0,1)$ and the point set $P = P_a \cup P_b \cup P_c$, where P_a contains a single point located at q_a, P_b contains $(1-\gamma)n - 1$ points overlapping at q_b, and P_c contains γn points overlapping at q_c. Consider the instance (P, γ) for the problem of MEB with outliers. Suppose $x \ll y$. Consequently, the optimal subset P_{opt} should be $P_a \cup P_b$ and the optimal radius is $x/2$.

Figure 3 $||q_a - q_b|| = x$ and $||q_b - q_c|| = y$.

If we take a random sample S of size $m = o(n)$ from P, with high probability, $P_a \cap S = \emptyset$ (even if we repeat our sampling a constant number of times, the single point P_a will still be missing with high probability). Therefore, S only contains the points from P_b and P_c. If we run an existing algorithm on S, the resulting ball center, say o, should always be inside the convex hull of S, i.e., the segment $\overline{q_b q_c}$. Let $\mathbb{B}(o, r)$ be the ball covering $(1-\gamma)n$ points of P. We consider two cases: (1) $q_a \in \mathbb{B}(o, r)$ and (2) $q_a \notin \mathbb{B}(o, r)$. For case (1), since $o \in \overline{q_b q_c}$, it is easy to know that the radius $r \geq ||q_a - q_b|| = x$ and therefore the approximation ratio is at least 2. For case (2), since $|P_b| = (1-\gamma)n - 1$, $\mathbb{B}(o, r)$ must cover some point from P_c and therefore $r = y/2$; because $x \ll y$, the approximation ratio is also larger than 2.

A.2 Proof of Theorem 4

Similar to the analysis in [11], we let $\lambda_i = \frac{r_i}{(1+\epsilon)Rad(P)}$. Because r_i is the radius of $MEB(T)$ and $T \subset P$, we know $r_i \leq Rad(P)$ and then $\lambda_i \leq 1/(1+\epsilon)$. By simple calculation, we know that when $L_i = \frac{\big((1+\epsilon)Rad(P) - \xi r_i\big)^2 - r_i^2}{2\big((1+\epsilon)Rad(P) - \xi r_i\big)}$ the lower bound of r_{i+1} in (3) achieves the minimum value. Plugging this value of L_i into (3), we have

$$\lambda_{i+1}^2 \geq \lambda_i^2 + \frac{\big((1-\xi\lambda_i)^2 - \lambda_i^2\big)^2}{4(1-\xi\lambda_i)^2}. \tag{36}$$

To simplify inequality (36), we consider the function $g(x) = \frac{(1-x)^2 - \lambda_i^2}{1-x}$, where $0 < x < \xi$. Its derivative $g'(x) = -1 - \frac{\lambda_i^2}{(1-x)^2}$ is always negative, thus we have

$$g(x) \geq g(\xi) = \frac{(1-\xi)^2 - \lambda_i^2}{1-\xi}. \tag{37}$$

Because $\xi < \frac{\epsilon}{1+\epsilon}$ and $\lambda_i \leq 1/(1+\epsilon)$, we know that the right-hand side of (37) is always non-negative. Using (37), the inequality (36) can be simplified to

$$\begin{aligned} \lambda_{i+1}^2 &\geq \lambda_i^2 + \frac{1}{4}\big(g(\xi)\big)^2 \\ &= \lambda_i^2 + \frac{\big((1-\xi)^2 - \lambda_i^2\big)^2}{4(1-\xi)^2}. \end{aligned} \tag{38}$$

(38) can be further rewritten as

$$\begin{aligned} \Big(\frac{\lambda_{i+1}}{1-\xi}\Big)^2 &\geq \frac{1}{4}\Big(1 + (\frac{\lambda_i}{1-\xi})^2\Big)^2 \\ \Longrightarrow \frac{\lambda_{i+1}}{1-\xi} &\geq \frac{1}{2}\Big(1 + (\frac{\lambda_i}{1-\xi})^2\Big). \end{aligned} \tag{39}$$

Now, we can apply a similar transformation of λ_i which was used in [11]. Let $\gamma_i = \frac{1}{1 - \frac{\lambda_i}{1-\xi}}$.

We know $\gamma_i > 1$ (note $0 \leq \lambda_i \leq \frac{1}{1+\epsilon}$ and $\xi < \frac{\epsilon}{1+\epsilon}$). Then, (39) implies that

$$\begin{aligned} \gamma_{i+1} &\geq \frac{\gamma_i}{1-\frac{1}{2\gamma_i}} \\ &= \gamma_i\big(1+\frac{1}{2\gamma_i}+(\frac{1}{2\gamma_i})^2+\cdots\big) \\ &> \gamma_i+\frac{1}{2}, \end{aligned} \tag{40}$$

where the equation comes from the fact that $\gamma_i > 1$ and thus $\frac{1}{2\gamma_i} \in (0, \frac{1}{2})$. Note that $\lambda_0 = 0$ and thus $\gamma_0 = 1$. As a consequence, we have $\gamma_i > 1 + \frac{i}{2}$. In addition, since $\lambda_i \leq \frac{1}{1+\epsilon}$, that is, $\gamma_i \leq \frac{1}{1-\frac{1}{(1+\epsilon)(1-\xi)}}$, we have

$$i < \frac{2}{\epsilon-\xi-\epsilon\xi} = \frac{2}{(1-\frac{1+\epsilon}{\epsilon}\xi)\epsilon}. \tag{41}$$

Consequently, we obtain the theorem.

Practical Performance of Space Efficient Data Structures for Longest Common Extensions

Patrick Dinklage
Department of Computer Science, Technical University of Dortmund, Germany
patrick.dinklage@tu-dortmund.de

Johannes Fischer
Department of Computer Science, Technical University of Dortmund, Germany
johannes.fischer@cs.tu-dortmund.de

Alexander Herlez
Department of Computer Science, Technical University of Dortmund, Germany
alexander.herlez@tu-dortmund.de

Tomasz Kociumaka
Department of Computer Science, Bar-Ilan Unviersity, Ramat Gan, Israel
kociumaka@mimuw.edu.pl

Florian Kurpicz
Department of Computer Science, Technical University of Dortmund, Germany
florian.kurpicz@tu-dortmund.de

Abstract

For a text $T[1,n]$, a Longest Common Extension (LCE) query $\mathrm{lce}_T(i,j)$ asks for the length of the longest common prefix of the suffixes $T[i,n]$ and $T[j,n]$ identified by their starting positions $1 \le i,j \le n$. A classic problem in stringology asks to preprocess a static text $T[1,n]$ over an alphabet of size σ so that LCE queries can be efficiently answered on-line. Since its introduction in the 1980's, this problem has found numerous applications: in suffix sorting, edit distance computation, approximate pattern matching, regularities finding, string mining, and many more. Text-book solutions offer $O(n)$ preprocessing time and $O(1)$ query time, but they employ memory-heavy data structures, such as suffix arrays, in practice several times bigger than the text itself.

Very recently, more space efficient solutions using $O(n \log \sigma)$ bits of total space or even only $O(\log n)$ bits of extra space have been proposed: string synchronizing sets [Kempa and Kociumaka, STOC'19, and Birenzwige et al., SODA'20] and in-place fingerprinting [Prezza, SODA'18]. The goal of this article is to present well-engineered implementations of these new solutions and study their practicality on a commonly agreed text corpus. We show that both perform extremely well in practice, with space consumption of only around 10% of the input size for string synchronizing sets (around 20% for highly repetitive texts), and essentially no extra space for fingerprinting. Interestingly, our experiments also show that both solutions become much faster than naive scanning even for finding common prefixes of moderate length, contradicting a common belief that sophisticated data structures for LCE queries are not competitive with naive approaches [Ilie and Tinta, SPIRE'09].

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases text indexing, longest common prefix, space efficient data structures

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.39

Supplementary Material `https://github.com/herlez/lce-test`

Funding *Patrick Dinklage*: Supported by the DFG SPP 1736 "Algorithms for Big Data".
Tomasz Kociumaka: Supported by ISF grants no. 1278/16 and 1926/19, by a BSF grant no. 2018364, and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064).
Florian Kurpicz: Supported by the DFG SPP 1736 "Algorithms for Big Data".

Acknowledgements Part of this work was carried out during the Dagstuhl Seminar 19241, "25 Years of the Burrows–Wheeler Transform".

1 Introduction and Related Work

The longest common extension (LCE) problem is formally defined as follows:

Given: A text $T[1,n]$ with n symbols from an alphabet of size σ.

Construct: A data structure that allows subsequent on-line queries of the form

$$\mathrm{lce}_T(i,j) := \max\{\ell \geq 0 \ : \ T[i,i+\ell) = T[j,j+\ell)\}.$$

This is a fundamental problem in stringology, with numerous applications in (sparse) suffix sorting [12,19,24], edit distance computation [35,37], approximate pattern matching [2,22,36], regularities finding [4,5,13], string mining [21], and many more. The problem has a classic solution with $O(1)$-time queries based either on the suffix tree of T combined with lowest common ancestor queries [15,26,37], or using the inverse suffix array of T and the longest common prefix array with a data structure for range minimum queries [18,30]. The latter variant is more space efficient in practice but still several times larger than the text itself.

Recently, two notable methods appeared that also partly address the LCE problem: Prezza's *in-place fingerprinting* data structure [44] using a clever replacement of the original text by well-chosen Karp–Rabin fingerprints [31], and data structures relying on local consistency techniques [33,39,46] to construct so-called *string synchronizing sets* (SSS) [12,32].

Although the LCE problem is also of high practical importance, we are only aware of a few experimental papers on the LCE problem [18,29]. However, these studies are now somewhat dated, and naturally none of them includes the recent theoretical advances on the problem. As a result, practitioners (e.g., in bioinformatics) still use those old (slow and/or memory-heavy) data structures (see, e.g., [43,45,47,50]), thereby limiting themselves to problem sizes much smaller than they could actually handle.

Our Contributions and Outline. The goal of this paper is to engineer and evaluate LCE data structures based on the recent theoretical advances in this field [12,32,44]: in-place fingerprinting and string synchronizing sets. We first describe the theory behind these two approaches in Sect. 3, after having introduced some general background in Sect. 2. Sect. 4 describes further details of the implementations, including a simple but practical data structure for the well-known predecessor/successor queries [49]. Finally, in Sect. 5, we evaluate our implementations on a well-established benchmarking set, showing that

(1) our implementations based on SSS are always among the fastest to answer queries and

(2) in-place fingerprinting is very fast in practice and useful to answer queries that have long results.

Further Related Work. Compressed suffix trees with LCA functionality [20] can also be used for LCE queries, but they are certainly too powerful (and hence too space consuming) for the singular problem considered in this paper; this is confirmed experimentally in Sect. 5.

Time-space tradeoffs for LCE queries were considered by many authors [7,9,10,12,24,48]. Kosolobov [34] showed that the product of extra space (in bits) and LCE query time must be at least $\Omega(n \log n)$ under certain assumptions. Furthermore, there are data structures for LCE queries in compressed [6,8,27,28,42,48] and dynamic texts [1,23,39,42].

2 Preliminaries

2.1 Karp–Rabin Fingerprints

The goal of Karp–Rabin fingerprints [31] is to hash substrings to small integers in order to achieve fast comparisons for equality. To construct Karp–Rabin fingerprints, we choose a random prime number $q = \Theta(n^c)$ for some constant $c > 1$. The Karp–Rabin fingerprint $\phi(x, y)$ of the substring $T[x, y]$ is then defined as

$$\phi(x, y) = \left(\sum_{z=x}^{y} T[z] \cdot \sigma^{y-z} \right) \bmod q \ . \tag{1}$$

Observe that the Karp–Rabin fingerprints of matching substrings are equal, i.e., for every integer $\ell \geq 0$, if $T[x, x + \ell] = T[y, y + \ell]$, then $\phi(x, x + \ell) = \phi(y, y + \ell)$. On the other hand, if two substrings do not match, their fingerprints will be different with high probability. Specifically, if $T[x, x+\ell] \neq T[y, y+\ell]$, then $\mathrm{Prob}[\phi(x, x+\ell) = \phi(y, y+\ell)] = O\big(\frac{\ell \log \sigma}{n^c}\big)$. Thus, by choosing large enough constant $c > 1$, we can control the probability of false positives when comparing fingerprints instead of the underlying substrings.

2.2 Static Successor Data Structures

Let U be a universe of u subsequent integers and let $S[0, n-1]$ be a sequence of n integers from U sorted in ascending order. For any $x \in U$, we call $\mathrm{succ}_S(x) := \min\{y \in S \mid y \geq x\}$ the *successor* of x in S, i.e., the smallest value in S larger than or equal to x. Without loss of generality, we assume that $x > S[0]$ (otherwise, its successor is $S[0]$) and $x \leq S[n-1]$ (otherwise, it has no successor). Furthermore, in the following, we are interested only in the *position* i of the successor such that $S[i] = \mathrm{succ}_S(x)$.

The most straightforward ways to find $\mathrm{succ}_S(x)$ are by doing a linear search in time $O(n)$ or, since S is sorted, by doing a binary search in time $O(\log n)$. Both require $O(\log n)$ bits of extra space for keeping track of the current position or the search interval and pivot, respectively. Alternatively, we can construct a bit vector B_S of size $u' = S[n-1] - S[0] + 1$ bits, in which we set $B_S[x - S[0]] := 1$ if and only if $x \in S$. Enhancing B_S to support constant-time rank queries (see [41, p. 75]) then allows for constant-time successor queries, because $\mathrm{succ}_S(x) = \mathrm{rank}_1(B_S, x - S[0])$. However, this method requires $u' + o(u')$ bits of memory and only works for static sets. Matsuoka et al. [38, Corollary 1] showed how to achieve the same space bounds in a dynamic setting, raising the query time to $O(\log \log u)$.

One can also interpret the numbers from S as binary strings and store them in a trie; some nodes of that trie can then be sampled to speed up the successor search. Such *universe-based sampling* has already been used in the *van Emde Boas tree* [49], which answers successor queries in time $O(\log \log u)$ and requires $O(n \log u)$ bits of memory using hashing. Dementiev et al. [14] described a careful implementation of this idea for 32-bit universes, called *STree*. However, the STree is designed for *dynamic* sequences and contains components to support insertion and deletion into the set, which are not needed in the static case.

3 LCE Data Structures

3.1 In-Place Fingerprinting

Prezza [44] showed how to store the Karp–Rabin fingerprints from Sect. 2.1 succinctly so that the fingerprints *replace* the original text characters, but the original characters can still be recovered.

To facilitate explanation, let us assume a byte-alphabet ($\sigma = 256$) and a computer with word size $w := 64$ bits. We group characters from T into blocks of size $\tau := \Theta(w/\log\sigma)$; in our case, $\tau = 8$. Call the resulting array $B[1, n/\tau]$, with $B[i] := T[(i-1)\tau + 1, i\tau]$ for all $i = 1, \ldots, n/\tau$ (assume for simplicity that τ divides n). Note that τ characters fit into a computer word, and can hence be transferred to/from memory and arithmetically manipulated in constant time. We also choose a random prime q such that $\frac{1}{2}\sigma^\tau \leq q < \sigma^\tau$. We then compute the fingerprints $\phi(1, i\tau)$ for all $i = 1, \ldots, n/\tau$. (The original description actually computes the values $(\phi(1, i\tau) + \bar{s}) \bmod q$, where $0 \leq \bar{s} < q$ is a random seed, which we omit in the following.) Since $D[i] := \lfloor B[i]/q \rfloor \in \{0, 1\}$, all we need to recover the full contents of block $B[i]$ are the two fingerprints $\phi(1, (i-1)\tau)$ and $\phi(1, i\tau)$, and the value $D[i]$. Prezza then shows how to choose q such that w.h.p. the fingerprints $\phi(1, i\tau)$ all have their most significant bit (MSB) set to 0; thus, the array D can be stored instead of those MSBs. (If this doesn't hold, one can recompute the data structure with a different value of q until it holds.) This also allows us to compute the fingerprints for *arbitrary* substrings of T (not necessarily aligned with the block boundaries), also in $O(1)$ time. Prezza shows that with the above choices of q and τ, the fingerprints are collision free with high probability $1 - n^{-\Omega(1)}$.

The advantage of replacing the original text characters with fingerprints is that arbitrarily long substrings of the text can be tested for equality in constant time, using the fingerprints of those substrings. This can be applied for longest common extension queries as follows: To compute $\ell := \mathrm{lce}_T(i, j)$ for any $1 \leq i, j \leq n$, we do an exponential search by comparing $\phi(i, i + 2^k)$ with $\phi(j, j + 2^k)$ for increasing exponents k until the fingerprints mismatch; the actual position of the first mismatch between $T[i, n]$ and $T[j, n]$ is then found by a further binary search on an interval of size $O(\ell)$. The whole process takes $O(\log \ell)$ time, assuming that we have access to a precomputed table of all necessary powers of σ modulo q, which adds another $O(w \log n)$ bits of space (negligible in practice).

3.2 String Synchronizing Sets

The idea behind a *string synchronizing set* (SSS for short) is to designate a (hopefully small) set of positions from T such that this choice of positions is *locally consistent*, meaning that in sufficiently long matching substrings of T, the chosen positions are the same (relative to the beginnings of these substrings). We use the following simplified definition of SSS that is sufficient for our purposes:

▶ **Definition 1.** *For a positive integer $\tau \leq n/2$, the τ-synchronizing set of T is defined as*

$$S = \{i \in [1, n-2\tau+1] : \min\{\phi(j, j+\tau-1) : j \in [i, i+\tau]\} \in \{\phi(i, i+\tau-1), \phi(i+\tau, i+2\tau-1)\}\}.$$

The τ-synchronizing set satisfies the following *consistency property*: for all $i, j \in [1, n-2\tau+1]$, if $T[i, i+2\tau-1] = T[j, j+2\tau-1]$, then $i \in S \iff j \in S$. The original definition [32] also includes a *density property* that is necessary to guarantee a small SSS (of size $O(n/\tau)$) even for highly repetitive parts of T, which we do not consider in this paper, as it would make the data structure and the query procedure significantly more complicated. (Omitting the density property can only make our data structure larger, but it remains correct).

We can use SSS for LCE data structures as follows. We first build a successor data structure on S. Let $n' := |S|$ denote the size of the SSS, and let $s_1 < s_2 < \cdots < s_{n'}$ denote the positions of S in increasing order. Define a new text T' of length n' by setting $T'[i] := T[s_i, s_i + 3\tau - 1]$ for all $1 \leq i \leq n'$. Then, build a data structure for constant-time LCE queries on T', e.g., the RMQ-based solution mentioned in the introduction [18].

In order to answer an $\mathrm{lce}_T(i,j)$ query for arbitrary $1 \leq i,j \leq n$, first compare at most $3\tau+1$ characters from $T[i,i+3\tau]$ and $T[j,j+3\tau]$. If this comparison yields a mismatch, we are done. If not, compute $s_{i'} := \mathrm{succ}_S(i)$, $s_{j'} := \mathrm{succ}_S(j)$, and $\ell' := \mathrm{lce}_{T'}(i',j')$. Then,

$$\mathrm{lce}_T(i,j) = s_{i'+\ell'} - i + \mathrm{lce}_T(s_{i'+\ell'}, s_{j'+\ell'}) \ , \tag{2}$$

and the latter lce_T value can again be computed naively as it is at most 3τ due to $T'[i'+\ell'] \neq T'[j'+\ell']$. Overall, we get $O(\tau)$ query time. (The original description [32] sets $\tau = \Theta(\log_\sigma n)$ and uses the bit-vector approach from Sect. 2.2 for successor queries, as well as word-packing and bit-fiddling techniques, to achieve $O(1)$ query time, still in $O(n \log \sigma)$ bits of space.)

An advantage of this LCE data structure is that it naturally combines fast naive scanning for small LCE values (up to 3τ) with more sophisticated data structures guaranteeing a good worst-case performance. Taken together with the fact that the data structure is easily tunable by adjusting τ, it is an excellent candidate for a practical LCE data structure.

4 Implementation

In this section, we describe our `C++` implementations of the data structures from Sect. 3. The code is optimized for byte-alphabets ($\sigma = 256$) and can handle texts of arbitrary length, with the restriction for the algorithms described in Sect. 4.4 that the synchronizing set S can contain at most $|S| < 2^{32}$ elements.

4.1 A Simple but Fast Static Successor Data Structure

We first describe our implementation of a fast practical data structure for successor queries on a static sorted integer sequence $S[0,n-1]$ of length n over a universe U of size $u = 2^w$. To the best of our knowledge, there is no systematic study of such data structures. Our data structure is a simple index that combines universe-based sampling, binary search, and linear search to find the successor $\mathrm{succ}_S(x)$ for any given $x \in U$.

Let $k < w$ be a trade-off parameter and let c be the number of elements from U that fit into a cache line. We build an index P over S such that we can, in constant time, look up an interval in S of length at most 2^k+1 in which $\mathrm{succ}_S(x)$ is guaranteed to be contained. In this interval, we proceed with a binary search up to the point when the size of the current search interval becomes at most c. In this final small interval, which fully fits into a cache line, we look for $\mathrm{succ}_S(x)$ using linear search. This successor query takes $O(k+c)$ time.

It remains to construct P. Let $\mathrm{hi}_k(x) = \lfloor x/2^k \rfloor$. We define an array I_P of $2^{w-k} = u/2^k$ intervals such that, for every non-negative integer $q < u/2^k$, the entry $I_P[q]$ is defined as

$$[\max\{i \in [0,n-1] \ : \ \mathrm{hi}_k(S[i]) < q\} + 1, \ \max\{j \in [0,n-2] \ : \ \mathrm{hi}_k(S[j]) \leq q\} + 1] \ .$$

If any maximum does not exist, we set the corresponding boundary to 0. As $\mathrm{hi}_k(s) \leq u/2^k$ for each $s \in S$, the right boundary of the last interval is always $n-1$. Informally, we split the universe U into 2^{w-k} intervals of size 2^k, find the corresponding boundaries in S, and store them in I_P with the right boundary shifted forward by one. The successor of x is then guaranteed to be contained in the interval $I_P[\mathrm{hi}_k(x)]$ of S, which is of length at most 2^k+1.

Since, for $1 < q < u/2^k$, the right boundary of $I_P[q-1]$ equals the left boundary of $I_P[q]$, in our final index P, we only store the left boundaries and append $n-1$ (the right boundary of the last interval) at the end. Then, the interval of S in which the successor of x is contained can be expressed as $[P[\mathrm{hi}_k(x)], \ P[\mathrm{hi}_k(x)+1]]$. We can construct P in time $O(n + u/2^k)$ with one scan over S; observe that P takes $(u/2^k+1)\lceil \lg n \rceil$ bits of space.

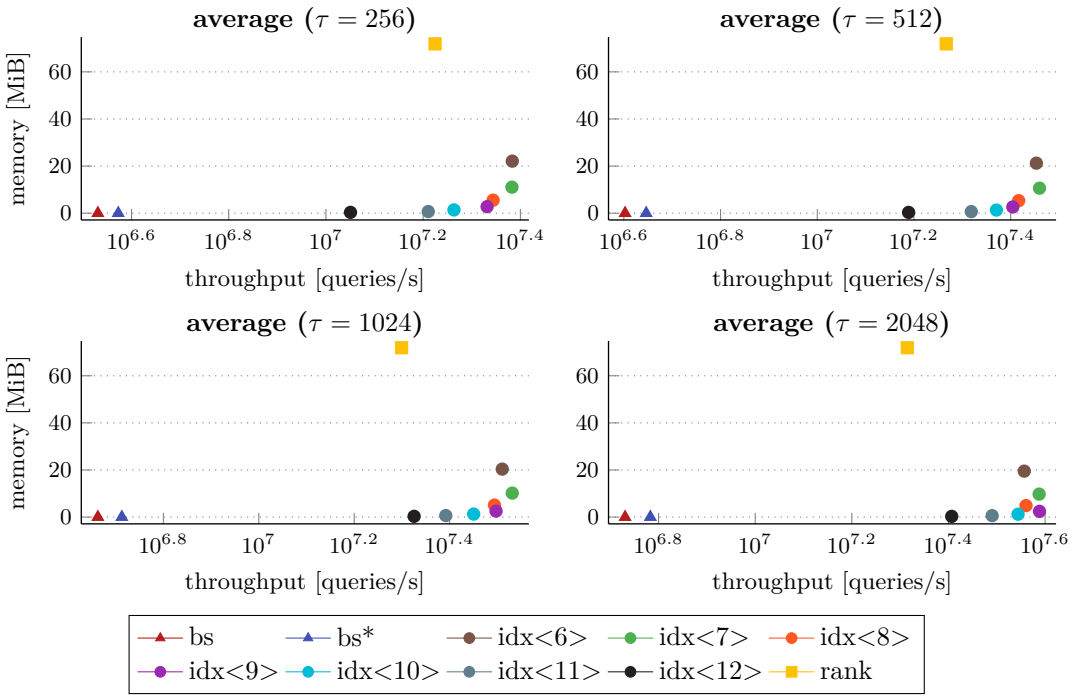


Figure 1 Query throughput versus space usage for static successor data structures built on top of string synchronizing sets with various values of τ, averaged over all texts in the corpus (Tbl. 1).

Note that the space requirement is not optimal in theory and very wasteful if $u \gg n$. However, in our use case of τ-synchronizing sets, the universe is relatively small and gaps between two subsequent integers in S are of length at most τ, making our index practical.

In a preliminary experiment, we looked for the best configuration of the parameter k for our successor data structure. For this experiment, we constructed the string synchronizing sets S for our input texts (see Tbl. 1) and compared the following successor data structures:

1. simple binary search (*bs*),
2. binary search that switches to linear search (*bs**) for intervals of length at most c,
3. the simple data structure from Sect. 2.2, using a bit vector supporting constant-time *rank* queries, and
4. our index P for different parameters k (*idx<k>*).

We measured the memory consumption of the successor data structures and the median query throughput for ten million successor queries over five iterations. The results in Figure 1 indicate that $k := 7$ yields the most beneficial trade-off between query throughput and memory consumption. Therefore, we used this configuration in further experiments. Appendix A shows a detailed breakdown of the results for all input texts.

4.2 Naive LCE Queries

Let us now turn our attention to LCE data structures. An **ultra naive** LCE query algorithm compares characters one by one until a mismatch is detected. This can be sped up significantly by using `uint_128` variables – nowadays supported by most CPUs – to compare blocks of 16 characters at once. Once we have a mismatch, we compare the last block character by character. This algorithm is called **naive**.

4.3 In-Place Fingerprinting

We group 8 consecutive bytes of the input text into a block, reverse the order of the characters in each block by simply swapping the characters (to speed up arithmetic operations when querying the data structure), compute the fingerprint up to the end of the block using Eq. (1), and overwrite the block with this fingerprint. Reversing the order of characters has no significant effect on the construction time, as this construction algorithm is still the fastest on all tested inputs (ignoring the naive approaches that do not require any construction at all), see Sect. 5. We use the smallest prime $q > 2^{63}$, which is $q = 2^{63} + 29$; this worked without any collision detected and with MSB $= 0$ for all stored fingerprints across all tested texts. We also use `uint_128` and the appropriate extended processor instructions for multiplying two 64-bit fingerprints.

Answering LCE queries using the procedure explained in Sect. 2.1 is done block-wise, i.e., in steps by 8, 16, ... characters. Within a block, instead of reconstructing and comparing fingerprints of short substrings, we first translate the fingerprint of the block back to its original contents (characters from T) and scan these characters naively. Because this is faster for LCE values smaller than 256, we also do this at the start of every LCE query. That means that we only start the exponential search after scanning 8 blocks of 8 characters each. We do the same when the final binary search reaches an interval shorter than 256 characters.

4.4 String Synchronizing Sets

While constructing a synchronizing set S, we only need to keep the fingerprints of a window of size 2τ in memory. For reasons of speed, we do so by using a ring buffer of size $2^{\lceil \lg 2\tau \rceil}$: the power of two allows us to compute the positions in the buffer using fast bit operations.

When sorting the suffixes of T', we first use sequential radix sort (`bingmann_msd_CI3_sb`) by Bingmann et al. [11] to reduce the alphabet size of T' (its characters are length-3τ substrings of T). Then, we compute the suffix array using SAIS-LITE [40], because it is the fastest for integer alphabets [3]. According to Sect. 3.2, we now have to compute the LCP array of T'. However, since in the end we are interested only in the LCP values within the original text T, we deviate slightly from Eq. (2) and compute a *sparse* LCP array of T, using the positions from S as indexed positions. For this, we rely on [32, Fact 5.1] that the lexicographic order of suffixes of T' coincides with the order of the corresponding suffixes of T. Apart from eliminating the need to map positions from T' back to T, this also saves us from performing the 3τ naive character comparisons at the end of the query in Eq. (2).

For the actual LCE computation for positions in S, we also build an RMQ data structure on the sparse LCP array; we apply a data structure by Ferrada and Navarro [16] for this purpose. We use a trick to speed up RMQ computations: since we know the length of the interval on which an RMQ is performed, we can make use of the fact that scanning small intervals is faster than an RMQ [29]. In our implementation, we only use the complicated RMQ machinery for intervals larger than 1024; this value yielded the fastest query times. Smaller intervals are simply scanned for the minimum. We implemented two variants of LCE queries:

Prefer-short corresponds to the description in Sect. 3.2: First, scan 3τ characters from T naively; if they are equal, compute the successors, and then apply Eq. (2) (with the modification described in the preceding paragraph). This variant should be used when *small* LCE values are expected, as the procedure is likely to stop already in the initial scan (before computing the successors).

Prefer-long, on the other hand, is optimized for a setting where *large* LCE values are expected. Here, the important observation is that the initial naive scan could reach synchronized positions much earlier than after 3τ comparisons. From that point on, one could immediately resort to LCE computation on T'. We therefore swap the computation of the successors ($s_{i'}$ and $s_{j'}$) and the naive scan. The naive scan then only has to verify that $T[i, s_{i'})$ and $T[j, s_{j'})$ indeed match.

5 Experimental Evaluation

We tested the data structures from Sect. 4: **ultra_naive** and **naive** (Sect. 4.2), **our-rk** (fingerprinting from Sect. 4.3), and **sss**$_\tau$ (string synchronizing sets from Sect. 4.4 with $\tau = 256, 512, 1024, 2048$) in the both variants (*pl* denotes the prefer-long variant). Our implementations are available at `https://github.com/herlez/lce-test`. We also compared our implementations with existing implementations: **prezza-rk** (`https://github.com/nicolaprezza/rk-lce`) and data structures using the compressed suffix trees (CST) **sada** and **sct3** contained in SDSL [25] (`https://github.com/simongog/sdsl-lite`). Other existing implementations were excluded due to their much higher space consumption.

Our experiments were conducted on a computer with two Intel Xeon E5-2640v4 CPUs (each with 10 cores at 2.4 GHz base frequency (3.4 GHz maximum turbo frequency) and cache sizes: 32 KB L1, 256 KB L2, 25.6 MB L3 cache) and 64 GB RAM. All algorithms are sequential and only use a single core, and the results are averages of five runs. The code was compiled with GCC 9.2.0 and compiler flags `-O3`, `-ffast-math`, and `-march=native`.

We ran all algorithms on the input shown in Tbl. 1. The texts were taken from the well-established benchmark suite Pizza&Chili [17]. Data sets from the top half of the table are from the *regular corpus*, whereas those from the bottom half are from the *repetitive corpus*.

Apart from several characteristics of the input texts, like length and alphabet size, Tbl. 1 also shows the sizes of the resulting string synchronizing sets for different values of τ. These numbers confirm our claim from Sect. 3.2 that the additional measures for keeping the SSS small are not necessary in practice, as it can be observed that the growth of SSS size is almost always perfectly proportional to the decrease of the values τ (growth rate only slightly less than 1). The only notable exception is the data set "cere", one of the highly repetitive texts, which contains long runs (of the character `N`). There, for example, when halving τ from 512 to 256, the SSS grows only by a factor of ≈ 1.21 (instead of the expected ≈ 2).

Table 1 Additional information about inputs used in evaluation: name, size n, alphabet size σ, and sizes of the corresponding synchronizing sets $|S_\tau|$ for $\tau = 256$, 512, 1024, and 2048.

name	n	σ	$\lvert S_{256}\rvert$	$\lvert S_{512}\rvert$	$\lvert S_{1024}\rvert$	$\lvert S_{2048}\rvert$
dblp.xml	296 135 874	97	2 304 012	1 153 799	578 703	288 758
dna	403 927 746	16	3 141 655	1 575 668	788 113	393 823
english.1024MB	1 073 741 824	239	8 354 560	4 187 042	2 095 466	1 047 924
proteins	1 184 051 855	27	9 215 429	4 616 809	2 341 374	1 155 364
sources	210 866 607	230	1 640 498	821 450	417 457	205 792
cere	461 286 644	5	31 619 034	26 205 215	20 013 847	12 699 848
coreutils	205 281 778	236	1 596 863	800 826	422 577	205 539
Escherichia_Coli	112 689 515	15	876 378	439 647	222 292	109 797
einstein.de.txt	92 758 441	117	721 365	361 616	181 080	90 702
einstein.en.txt	467 626 544	139	3 640 253	1 823 454	998 765	547 184
influenza	154 808 555	15	1 204 011	604 046	301 601	151 142
kernel	257 961 616	160	2 008 714	1 006 209	505 080	252 391

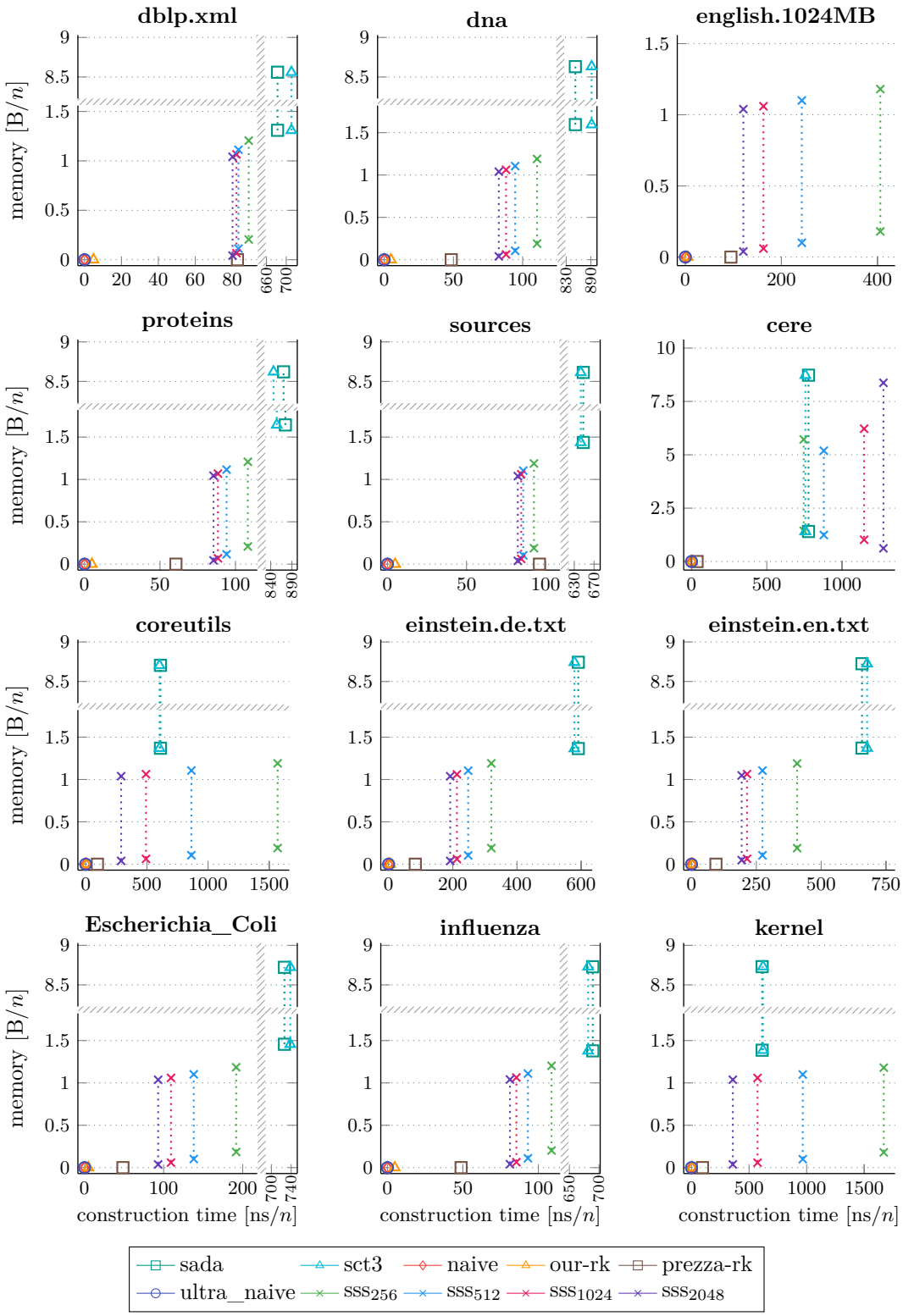


Figure 2 Construction time in nanoseconds and memory consumption in bytes per character of the input for the LCE data structures. The upper mark is the memory peak during construction, and the lower mark is the memory required for the final data structure. (Needs colors for viewing.)

5.1 Construction and Space Usage

Fig. 2 shows the construction times and the space usage of the data structures. For the SSS and CST data structures, that figure shows two numbers: the final size of the data structure (bottom mark) and the memory peak during construction time (top mark). The difference of these two numbers is therefore the additional space at construction time, which results from the intermediate steps using additional data structures (as described in Sect. 4.4 for SSS).

The other data structures need no extra space during construction, as they are either in-place ("prezza-rk" and "our-rk") or do not do any preprocessing at all ("ultra_naive" and "naive"). Also, "our-rk" is significantly faster to build than "prezza-rk" on all inputs; it is 19.76 times faster on "sources". The CST could not be computed for "english.1024MB". Overall, the CST data structures require the most memory. They are also the slowest to construct on all texts but "cere", "coreutils", and "kernel", where SSS is slower for some τ.

The first thing to note is that both time and space requirements grow for SSS with decreasing parameter τ. For space, this is what one would expect immediately, whereas for the running time, this needs further explanation: the times for *sorting* the characters of T', which are substrings of T, depend both on their length 3τ and their number $|S|$. One could now conjecture that, in all cases, roughly the same amount of characters has to be sorted, resulting in construction times mostly independent of τ. However, as our string sorting procedure is *prefix aware* (it only considers the strings up to their distinguishing prefixes), the 3τ-long substrings are usually not inspected in full. This implies that the number of strings can have a higher impact on the running times than their total length, despite the fact that the total length of the strings remains the same (e.g., "english.1024MB"). For other texts (like "dblp.xml"), the construction times are very similar for different values of τ, but are generally faster with growing τ. The notable exception is again the data set "cere", where the order on the time-axis is reversed (but not on the space-axis). This can be explained as follows: we already said before that the growth of the size of the SSS is much less pronounced with decreasing values of τ than for the other data sets. The string sorter is likely to inspect a number of characters proportional to the length of the strings. Therefore, the sorting times for the cere-substrings are influenced more by their total lengths than for the other data sets.

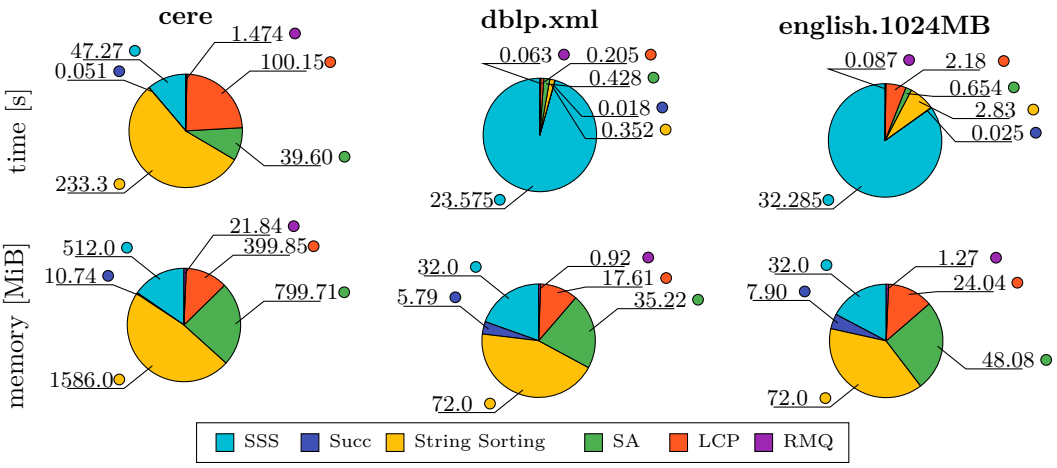


Figure 3 Snapshots of construction time and memory usage during construction for different phases of sss_{512} (same as $\mathrm{sss}^{\mathrm{pl}}_{512}$).

The construction of the SSS structure can be broken down into several steps. Fig. 3 shows examples of how much time is needed for the individual components (building SSS, sorting the length-3τ substrings, building the successor data structure, the suffix array, the LCP array, and the RMQ data structure). Some components differ significantly from text to text: for "cere", the string sorting takes almost half of the total time, whereas for "dblp.xml" (and partly also "english"), the construction of the SSS itself takes almost all of the time. The charts also indicate if and where further engineering efforts can pay off: for example, improving the successor data structure further will most likely neither speed up the final construction times significantly nor improve the space usage. On the contrary, speeding up string sorting or SSS construction (e.g., by parallelization) could result in an overall speedup.

5.2 Query Times

Now, we are interested in query times for different LCE values. To this end, we precomputed queries whose results are in the interval $[2^k, 2^{k+1})$ for all k, allowing us to get a detailed look at query times depending on the LCE value and use the same queries for all data structures.

We now first evaluate the query throughput of the different SSS data structures (with varying values for τ). The results are shown in Fig. 4, grouped by the lengths of the actual LCEs. It can be observed that the initial scanning of characters generally results in a visible drop of the throughput for longer LCEs; smaller values of τ lead to faster query times; the prefer-long variants are indeed faster for longer LCEs, whereas they are slower for shorter ones; and the throughput of all variants does not drop below a threshold for very long LCEs. Indeed, for many data sets, very long LCEs become faster, which can be explained by our method for answering RMQs: for very long LCEs, the corresponding LCP values are likely to be close together in the LCP array, so our query procedure is likely to use the fast scanning for the minimum instead of invoking the heavy $O(1)$-time machinery. Given that the results for $\tau = 512$ and $\tau = 256$ are almost equally fast (in particular for "prefer-long"), we choose the larger of the two values for the following tests, as it results in a smaller data structure.

Fig. 5 shows the query throughput of all data structures (for SSS only with $\tau = 512$). We observe that "ultra_naive" is always slower than "naive" (this comes at no surprise); for short LCEs (roughly up to 2^8), "ultra_naive", "naive", and "sss_{512}" are fastest due to the simple initial scanning; for longer LCEs (greater than 2^{12}), "$\text{sss}^{\text{pl}}_{512}$" becomes the fastest data structure (also faster than "sss_{512}", which becomes faster than the naive approaches for LCEs longer than $\approx 2^{12}$); "our-rk" is slower than Prezza's original implementation "prezza-rk" except for medium sized LCEs around 2^5–2^{10}; and SSS's are much faster than the fingerprinting or naive methods for LCEs longer than $\approx 2^{12}$. The CST data structures "sada" and "sct3" are of similar speed and faster than "ultra_naive" only for long LCEs, around 2^{16} on most inputs. On the repetitive texts "einstein.de.txt" and "einstein.en.txt" both are faster than the fingerprint data structures for medium-size LCEs around $\approx 2^{13}$.

6 Conclusions

We conclude from the experiments that string synchronizing sets (SSS) are the method of choice if their 10–20% overhead on top of the original text size fits into RAM, as they naturally combine the best of two worlds: they answer short LCEs equally fast as naive scanning methods, but are much faster for long LCEs and have a guaranteed worst-case query time. This threshold is much earlier than previously conjectured: even for LCE values larger than $\approx 2^9$, the additional effort for constructing the data structure pays off. If even the little extra space for the SSS is too much and worst case query times have to be guaranteed for long LCEs, then one should use an in-place fingerprinting method.

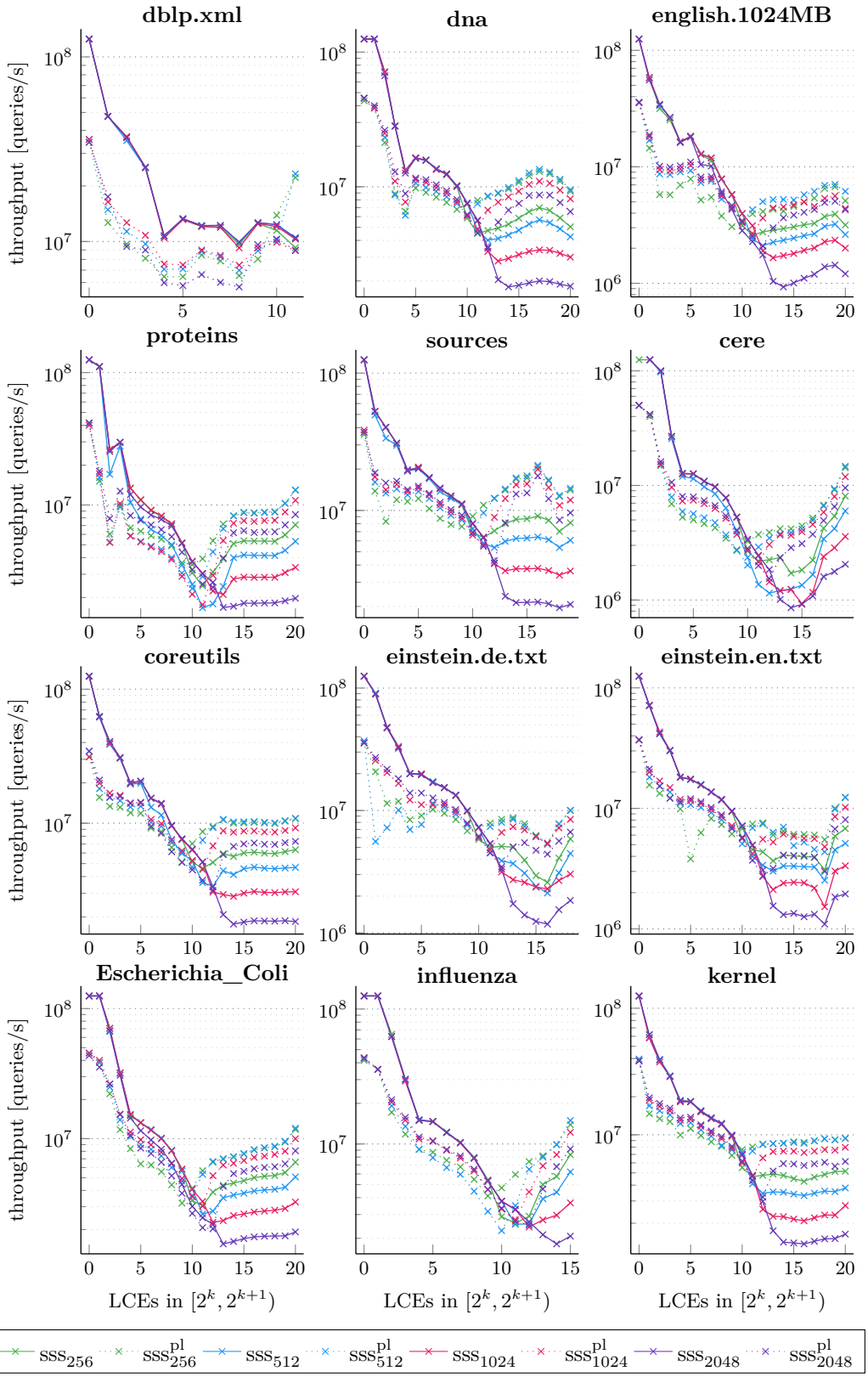


Figure 4 Comparing query throughput of our SSS LCE data structures.

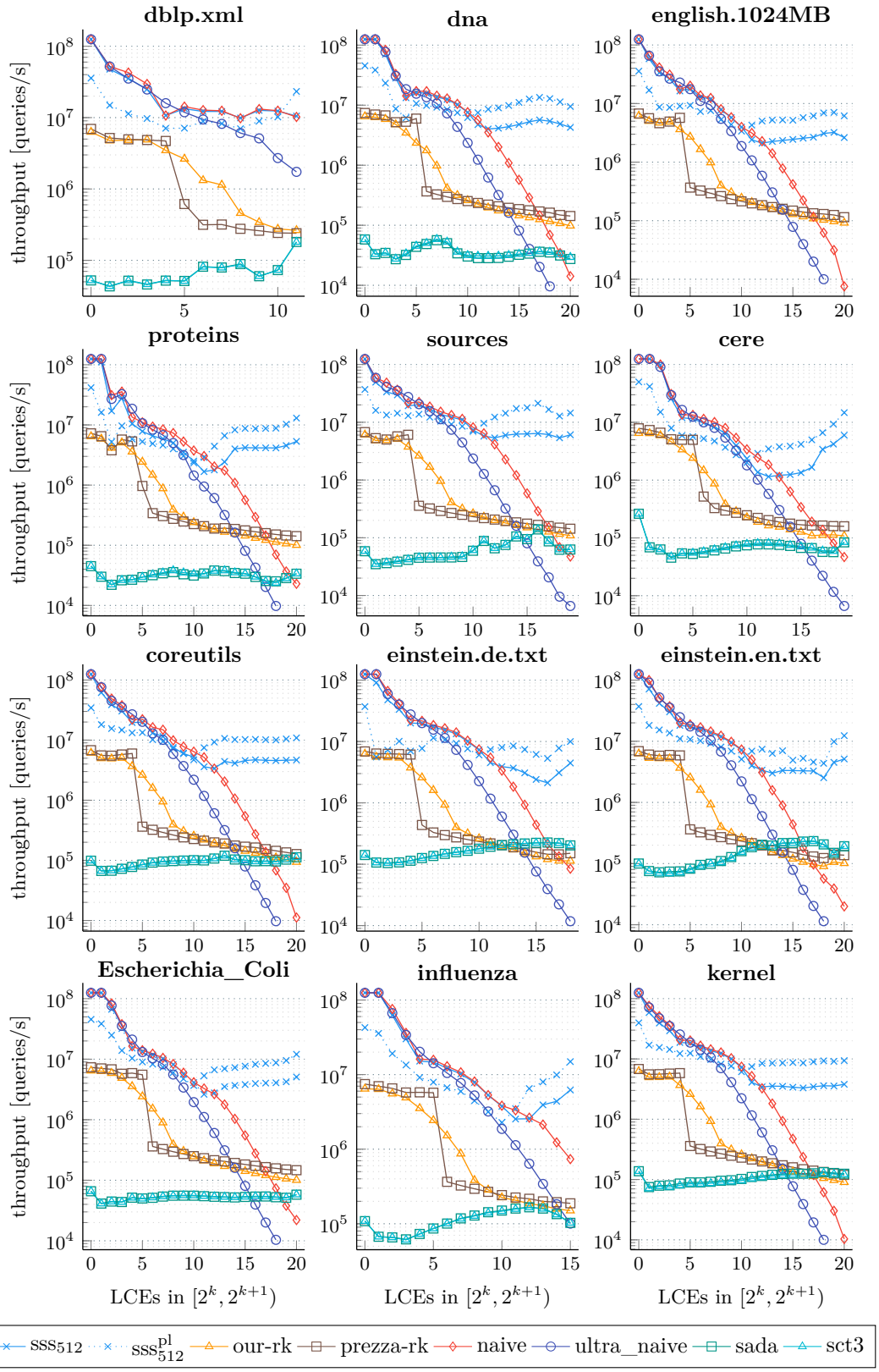


Figure 5 Query throughput of the LCE data structures, only including the fastest SSS ($\tau = 512$).

References

1. Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In *11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 819–828. ACM/SIAM, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338645.
2. Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.
3. Johannes Bahne, Nico Bertram, Marvin Böcker, Jonas Bode, Johannes Fischer, Hermann Foot, Florian Grieskamp, Florian Kurpicz, Marvin Löbel, Oliver Magiera, Rosa Pink, David Piper, and Christopher Poeplau. Sacabench: Benchmarking suffix array construction. In *26th International Symposium on String Processing and Information Retrieval (SPIRE)*, volume 11811 of *Lecture Notes in Computer Science*, pages 407–416. Springer, 2019. doi:10.1007/978-3-030-32686-9_29.
4. Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
5. Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.22.
6. Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz. Finger search in grammar-compressed strings. *Theory of Computing Systems*, 62(8):1715–1735, 2018. doi:10.1007/s00224-017-9839-9.
7. Philip Bille, Johannes Fischer, Inge Li Gørtz, Tsvi Kopelowitz, Benjamin Sach, and Hjalte Wedel Vildhøj. Sparse text indexing in small space. *ACM Transactions on Algorithms*, 12(3):39:1–39:19, 2016. doi:10.1145/2836166.
8. Philip Bille, Inge Li Gørtz, Patrick Hagge Cording, Benjamin Sach, Hjalte Wedel Vildhøj, and Søren Vind. Fingerprints in compressed strings. *Journal of Computer and System Sciences*, 86:171–180, 2017. doi:10.1016/j.jcss.2017.01.002.
9. Philip Bille, Inge Li Gørtz, Mathias Bæk Tejs Knudsen, Moshe Lewenstein, and Hjalte Wedel Vildhøj. Longest common extensions in sublinear space. In *26th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 9133 of *Lecture Notes in Computer Science*, pages 65–76. Springer, 2015. doi:10.1007/978-3-319-19929-0_6.
10. Philip Bille, Inge Li Gørtz, Benjamin Sach, and Hjalte Wedel Vildhøj. Time-space trade-offs for longest common extensions. *Journal of Discrete Algorithms*, 25:42–50, 2014. doi:10.1016/j.jda.2013.06.003.
11. Timo Bingmann, Andreas Eberle, and Peter Sanders. Engineering parallel string sorting. *Algorithmica*, 77(1):235–286, 2017. doi:10.1007/s00453-015-0071-1.
12. Or Birenzwige, Shay Golan, and Ely Porat. Locally consistent parsing for text indexing in small space. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 607–626. SIAM, 2020. doi:10.1137/1.9781611975994.37.
13. Maxime Crochemore, Roman Kolpakov, and Gregory Kucherov. Optimal bounds for computing α-gapped repeats. *Information and Computation*, 268, 2019. doi:10.1016/j.ic.2019.104434.
14. Roman Dementiev, Lutz Kettner, Jens Mehnert, and Peter Sanders. Engineering a sorted list data structure for 32 bit key. In *Sixth Workshop on Algorithm Engineering and Experiments (ALENEX) and the First Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 142–151. SIAM, 2004.
15. Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *Journal of the ACM*, 47(6):987–1011, 2000. doi:10.1145/355541.355547.
16. Héctor Ferrada and Gonzalo Navarro. Improved range minimum queries. *Journal of Discrete Algorithms*, 43:72–80, 2017. doi:10.1016/j.jda.2016.09.002.

17 Paolo Ferragina and Gonzalo Navarro. Pizza&Chili corpus: Compressed indexes and their testbeds. URL: http://pizzachili.dcc.uchile.cl/.

18 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to LCA and LCE. In *17th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2006. doi:10.1007/11780441_5.

19 Johannes Fischer, Tomohiro I, and Dominik Köppl. Deterministic sparse suffix sorting on rewritable texts. In *12th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 9644 of *Lecture Notes in Computer Science*, pages 483–496. Springer, 2016. doi:10.1007/978-3-662-49529-2_36.

20 Johannes Fischer, Veli Mäkinen, and Gonzalo Navarro. Faster entropy-bounded compressed suffix trees. *Theoretical Computer Science*, 410(51):5354–5364, 2009. doi:10.1016/j.tcs.2009.09.012.

21 Johannes Fischer, Veli Mäkinen, and Niko Välimäki. Space efficient string mining under frequency constraints. In *8th IEEE International Conference on Data Mining (ICDM)*, pages 193–202. IEEE Computer Society, 2008. doi:10.1109/ICDM.2008.32.

22 Zvi Galil and Raffaele Giancarlo. Improved string matching with k mismatches. *SIGACT News*, 17(4):52–54, 1986. doi:10.1145/8307.8309.

23 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1509–1528. SIAM, 2018. doi:10.1137/1.9781611975031.99.

24 Paweł Gawrychowski and Tomasz Kociumaka. Sparse suffix tree construction in optimal time and space. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 425–439. SIAM, 2017. doi:10.1137/1.9781611974782.27.

25 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In Joachim Gudmundsson and Jyrki Katajainen, editors, *13th International Symposium on Experimental Algorithms, SEA 2014*, volume 8504 of *LNCS*, pages 326–337. Springer, 2014. doi:10.1007/978-3-319-07959-2_28.

26 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

27 Tomohiro I. Longest common extensions with recompression. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 78 of *LIPIcs*, pages 18:1–18:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.18.

28 Tomohiro I, Wataru Matsubara, Kouji Shimohira, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda, Kazuyuki Narisawa, and Ayumi Shinohara. Detecting regularities on grammar-compressed strings. *Information and Computation*, 240:74–89, 2015. doi:10.1016/j.ic.2014.09.009.

29 Lucian Ilie and Liviu Tinta. Practical algorithms for the longest common extension problem. In *16th International Symposium on String Processing and Information Retrieval (SPIRE)*, volume 5721 of *Lecture Notes in Computer Science*, pages 302–309. Springer, 2009. doi:10.1007/978-3-642-03784-9_30.

30 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.

31 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

32 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.

33 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.

34 Dmitry Kosolobov. Tight lower bounds for the longest common extension problem. *Information Processing Letters*, 125:26–29, 2017. doi:10.1016/j.ipl.2017.05.003.

35 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998. doi:10.1137/S0097539794264810.

36 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986. doi:10.1016/0304-3975(86)90178-7.

37 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

38 Yoshiaki Matsuoka, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Semi-dynamic compact index for short patterns and succinct van Emde Boas tree. In *26th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 9133 of *Lecture Notes in Computer Science*, pages 355–366. Springer, 2015. doi:10.1007/978-3-319-19929-0_30.

39 Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997. doi:10.1007/BF02522825.

40 Yuta Mori. sais: An implementation of the induced sorting algorithm. URL: https://sites.google.com/site/yuta256/.

41 Gonzalo Navarro. *Compact Data Structures: A Practical Approach*. Cambridge University Press, 2016. doi:10.1017/cbo9781316588284.

42 Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 58 of *LIPIcs*, pages 72:1–72:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.72.

43 Cinzia Pizzi. Missmax: alignment-free sequence comparison with mismatches through filtering and heuristics. *Algorithms for Molecular Biology*, 11:6, 2016. doi:10.1186/s13015-016-0072-x.

44 Nicola Prezza. In-place sparse suffix sorting. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1496–1508. SIAM, 2018. doi:10.1137/1.9781611975031.98.

45 Wei Quan, Bo Liu, and Yadong Wang. SALT: a fast, memory-efficient and snp-aware short read alignment tool. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1774–1779. IEEE, 2019. doi:10.1109/BIBM47256.2019.8983162.

46 Süleyman Cenk Sahinalp and Uzi Vishkin. On a parallel-algorithms method for string matching problems. In *Second Italian Conference on Algorithms and Complexity (CIAC)*, volume 778 of *LNCS*, pages 22–32. Springer, 1994. doi:10.1007/3-540-57811-0_3.

47 Avi Srivastava, Hirak Sarkar, Nitish Gupta, and Robert Patro. Rapmap: a rapid, sensitive and accurate tool for mapping rna-seq reads to transcriptomes. *Bioinformatics*, 32(12):192–200, 2016. doi:10.1093/bioinformatics/btw277.

48 Yuka Tanimura, Tomohiro I, Hideo Bannai, Shunsuke Inenaga, Simon J. Puglisi, and Masayuki Takeda. Deterministic sub-linear space LCE data structures with efficient construction. In *27th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 54 of *LIPIcs*, pages 1:1–1:10. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.1.

49 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.

50 Chen Zhou, Hao Chi, Leheng Wang, You Li, Yan-Jie Wu, Yan Fu, Ruixiang Sun, and Si-Min He. Speeding up tandem mass spectrometry-based database searching by longest common prefix. *BMC Bioinformatics*, 11:577, 2010. doi:10.1186/1471-2105-11-577.

A Detailed Results for Predecessor Queries

(a) $\tau = 256$.

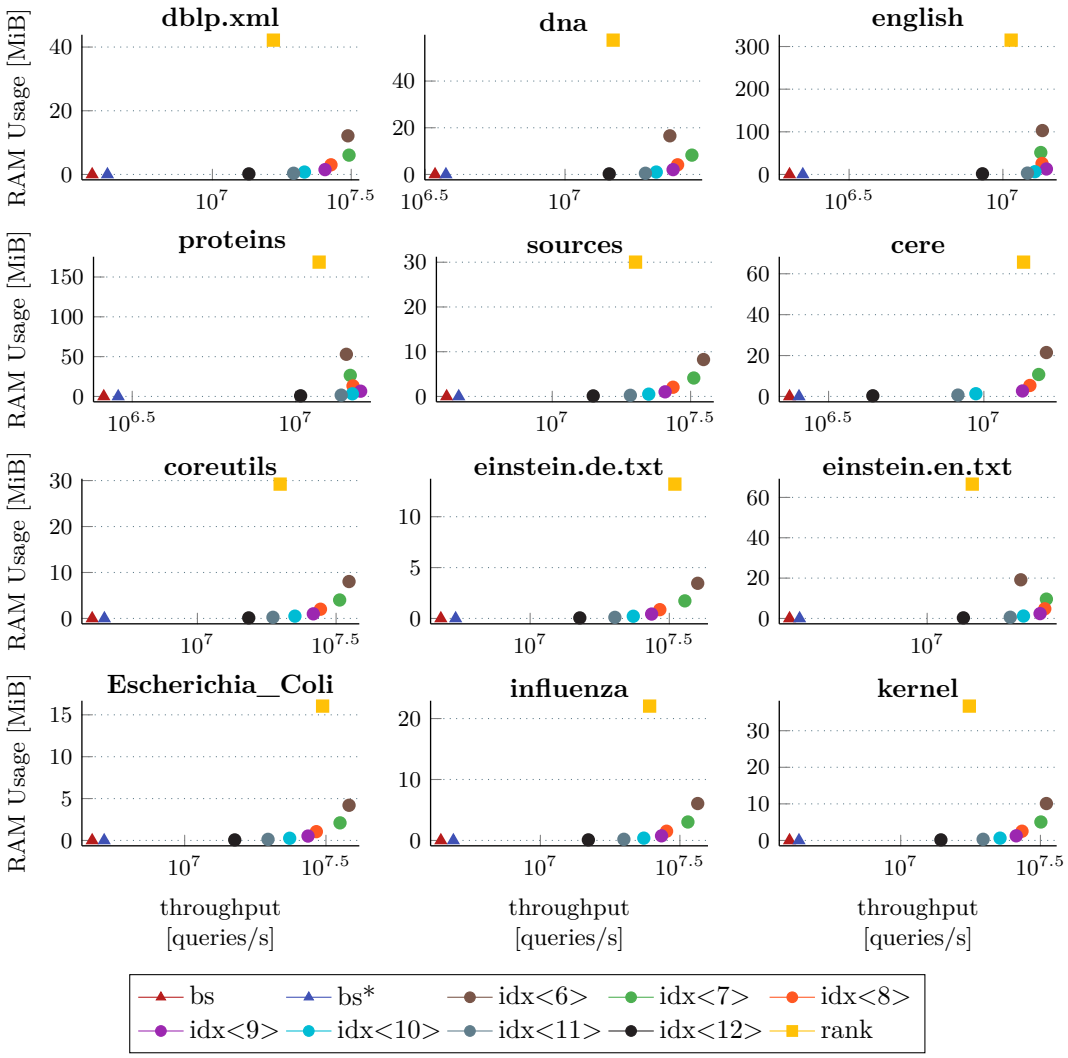


Figure 6 Query throughput and space usage for static successor data structures on SSS's.

(a) $\tau = 512$.

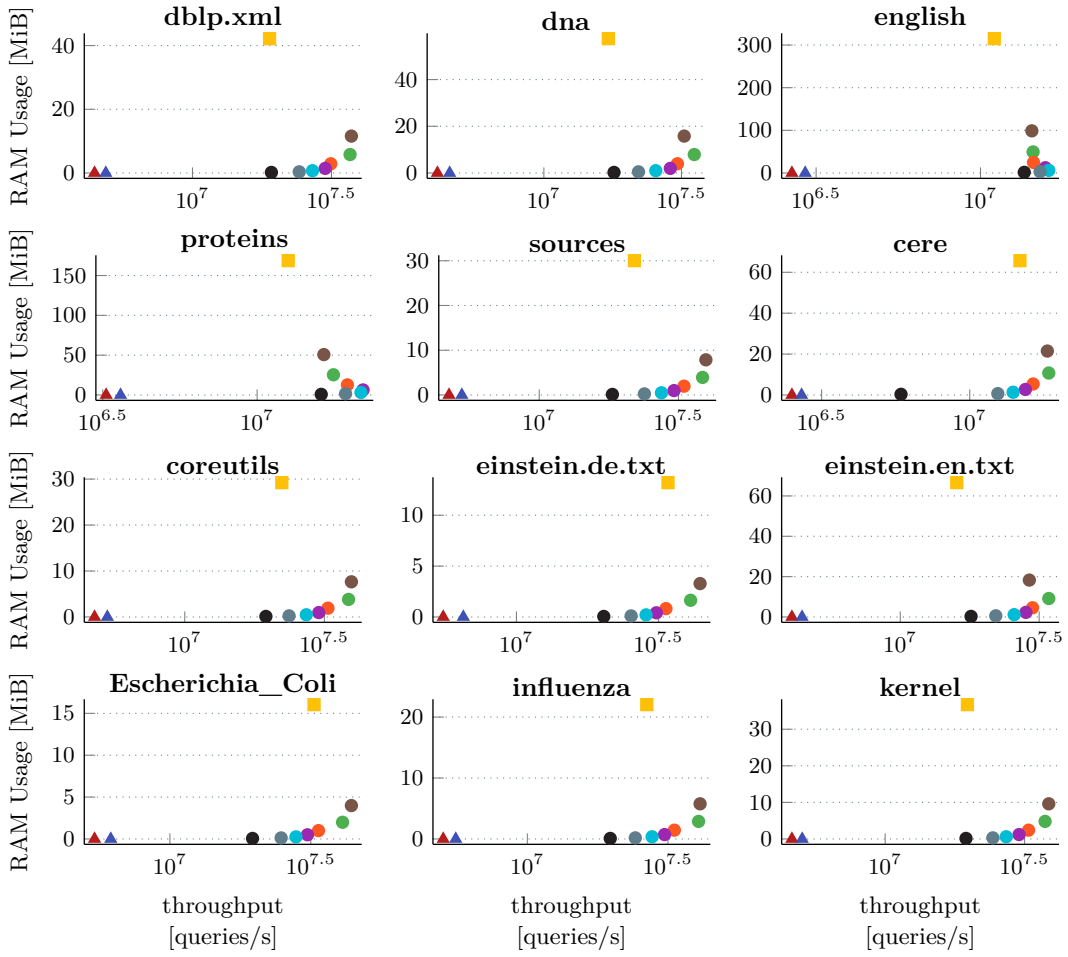


Figure 7 Query throughput and space usage for static successor data structures on SSS's (cont.).

(a) $\tau = 1024$.

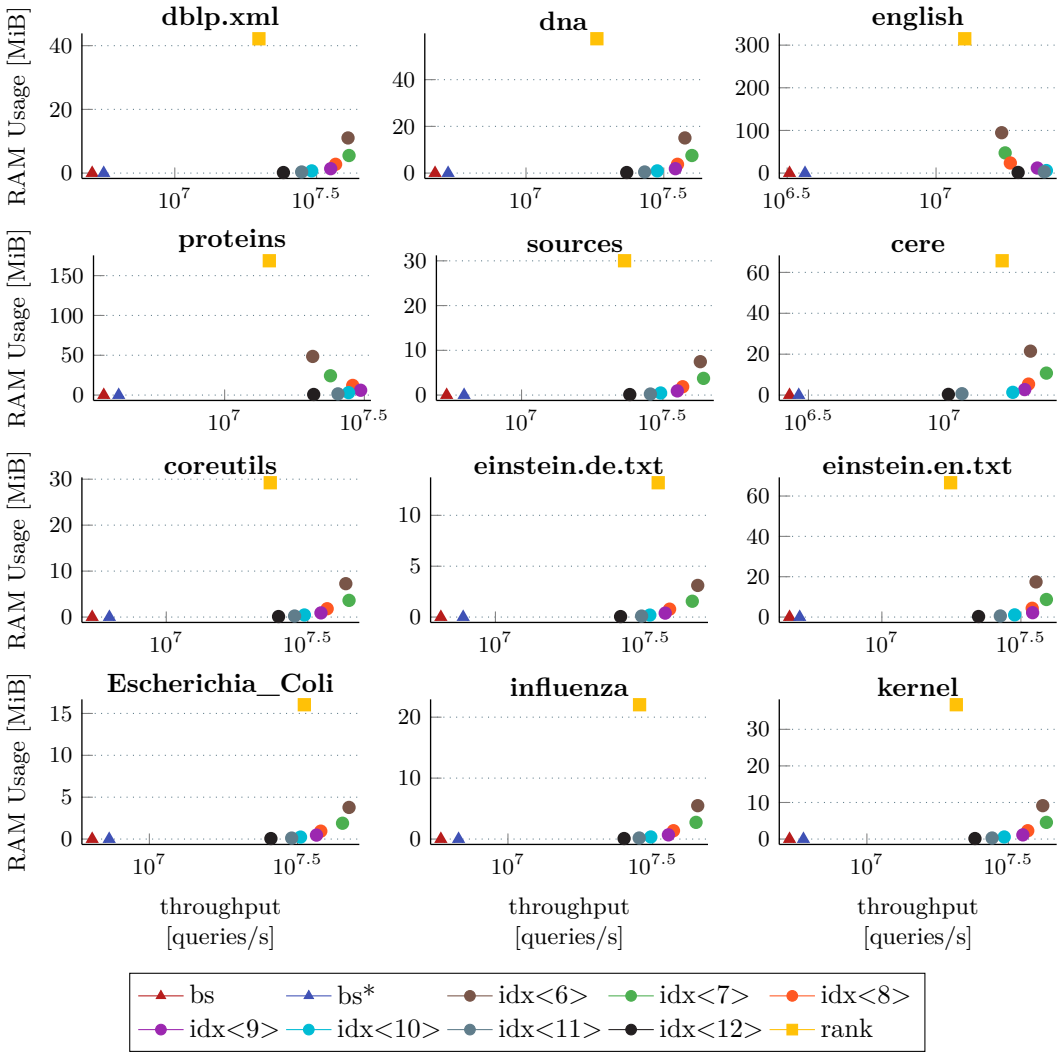


Figure 8 Query throughput and space usage for static successor data structures on SSS's (cont.).

(a) $\tau = 2048$.

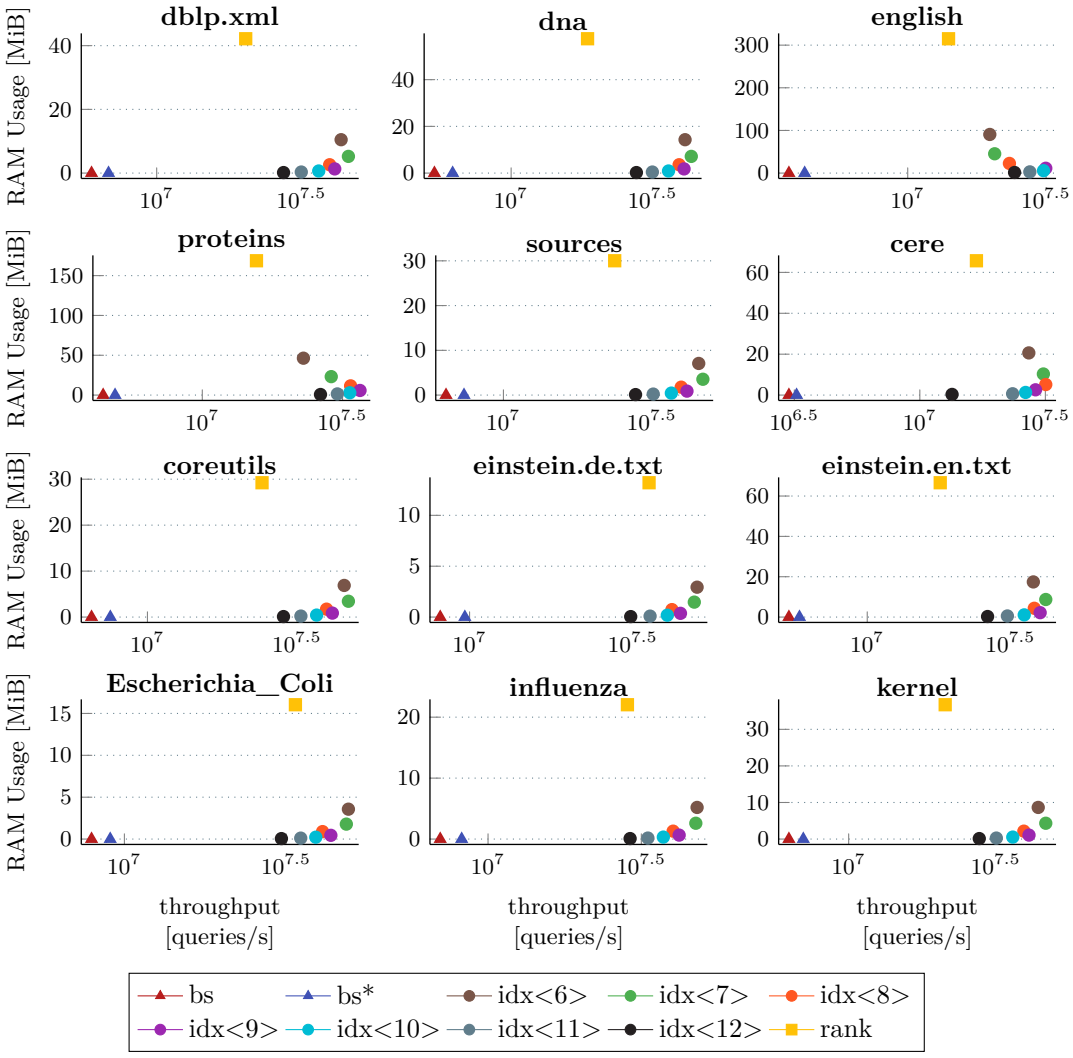


Figure 9 Query throughput and space usage for static successor data structures on SSS's (cont.).

First-Order Model-Checking in Random Graphs and Complex Networks

Jan Dreier
Department of Computer Science, RWTH Aachen University, Germany
https://tcs.rwth-aachen.de/~dreier
dreier@cs.rwth-aachen.de

Philipp Kuinke
Department of Computer Science, RWTH Aachen University, Germany
https://tcs.rwth-aachen.de/~kuinke
kuinke@cs.rwth-aachen.de

Peter Rossmanith
Department of Computer Science, RWTH Aachen University, Germany
https://tcs.rwth-aachen.de
rossmani@cs.rwth-aachen.de

Abstract

Complex networks are everywhere. They appear for example in the form of biological networks, social networks, or computer networks and have been studied extensively. Efficient algorithms to solve problems on complex networks play a central role in today's society. Algorithmic meta-theorems show that many problems can be solved efficiently. Since logic is a powerful tool to model problems, it has been used to obtain very general meta-theorems. In this work, we consider all problems definable in first-order logic and analyze which properties of complex networks allow them to be solved efficiently.

The mathematical tool to describe complex networks are random graph models. We define a property of random graph models called α-power-law-boundedness. Roughly speaking, a random graph is α-power-law-bounded if it does not admit strong clustering and its degree sequence is bounded by a power-law distribution with exponent at least α (i.e. the fraction of vertices with degree k is roughly $O(k^{-\alpha})$).

We solve the first-order model-checking problem (parameterized by the length of the formula) in almost linear FPT time on random graph models satisfying this property with $\alpha \geq 3$. This means in particular that one can solve every problem expressible in first-order logic in almost linear expected time on these random graph models. This includes for example preferential attachment graphs, Chung–Lu graphs, configuration graphs, and sparse Erdős–Rényi graphs. Our results match known hardness results and generalize previous tractability results on this topic.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases random graphs, average case analysis, first-order model-checking

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.40

Related Version A full version of the paper is available at https://arxiv.org/abs/2006.14488.

Funding This research has been partially supported by the German Science Foundation (DFG) under grant DFG RO 927/15-1.

1 Introduction

Complex networks, as they occur in society, biology and technology, play a central role in our everyday lives. Even though these networks occur in vastly different contexts, they are structured and evolve according to a common set of underlying principles. Over the last two

decades, with the emergence of the field of network science, there has been an explosion in research to understand these fundamental laws. One well observed property is the *small-world phenomenon*, which means that distances between vertices are very small. This has been verified for the internet and many other networks [1, 56]. Furthermore, many real networks tend to be *clustered*. They contain groups of vertices that are densely connected [67]. If two vertices share a common neighbor, then there is a high chance that there is also an edge between them. A network can be considered clustered if the ratio between the number of triangles and the number of paths with three vertices is non-vanishing. This is formalized by the clustering coefficient, which is high for many networks [72]. A third important property is a *heavy tailed degree distribution*. While most vertices have a low number of connections, there are a few hubs with a high degree. Experiments show that the degrees follow for example a power-law or log-normal distribution. In a power-law distribution, the fraction of vertices with degree k is proportional to $k^{-\alpha}$ (usually with α between 2 and 3). This behavior makes complex networks highly inhomogeneous [65, 58, 10, 15].

One important goal of theoretical computer science has always been to explore what kinds of inputs allow or forbid us to construct efficient algorithms. In this context, *algorithmic meta-theorems* [52] are of particular interest. They are usually theorems stating that problems definable in a certain logic can be solved efficiently on graph classes that satisfy certain properties. Logic is a powerful tool to model problems and therefore has been used to obtain very general meta-theorems. A well-known example is Courcelle's theorem [16], which states that every problem expressible in counting monadic second-order logic can be solved in linear time on graph classes with bounded treewidth. It has been further generalized to graph classes with bounded cliquewidth [17]. To obtain results for larger graph classes one has to consider weaker logics. The languages of relational database systems are based on first-order logic. In this logic, one is allowed to quantify over vertices and to test equality and adjacency of vertices. With k existential quantifiers, one may ask for the existence of a fixed graph with k vertices (k-subgraph isomorphism), a problem relevant to motif-counting [57, 29]. On the other hand, connectivity properties cannot be expressed in first-order logic. We define for every graph class $\mathcal{G}$ the parameterized first-order model-checking problem $p\text{-MC}(\text{FO}, \mathcal{G})$ [45].

$p\text{-MC}(\text{FO}, \mathcal{G})$	
Input:	A graph $G \in \mathcal{G}$ and a first-order sentence φ
Parameter:	The number of symbols in φ, denoted by $\lvert\varphi\rvert$
Problem:	Does φ hold on G (i.e. $G \models \varphi$)?

The aim is to show for a given graph class $\mathcal{G}$ that $p\text{-MC}(\text{FO}, \mathcal{G})$ is *fixed parameter tractable* (FPT), i.e., can be decided in time $f(\lvert\varphi\rvert)n^{O(1)}$ for some function f (see for example [18] for an introduction to fixed parameter tractability). Since input graphs may be large, a linear dependence on n is desirable. If one is successful, then every problem expressible in first-order logic can be solved on $\mathcal{G}$ in linear time.

For the class of all graphs $\mathfrak{G}$, $p\text{-MC}(\text{FO}, \mathfrak{G})$ is AW[$*$]-complete [25] and therefore most likely not fpt. Over time, tractability of $p\text{-MC}(\text{FO}, \mathcal{G})$ has been shown for more and more sparse graph classes $\mathcal{G}$: bounded vertex degree [69], forbidden minors [35], bounded local treewidth [34], and further generalizations [19, 30, 68]. Grohe, Kreutzer and Siebertz prove that $p\text{-MC}(\text{FO}, \mathcal{G})$ can be solved in almost linear FPT time $f(\lvert\varphi\rvert, \varepsilon)n^{1+\varepsilon}$ for all $\varepsilon > 0$ if $\mathcal{G}$ is a nowhere dense graph class [46]. On the other hand if $\mathcal{G}$ is a monotone somewhere dense graph class, $p\text{-MC}(\text{FO}, \mathcal{G})$ is AW[$*$]-hard [46]. Nowhere dense graph classes were introduced by Nešetřil and Ossona de Mendez as those graph classes where for every $r \in \mathbf{N}$ the size of all r-shallow clique minors of all graphs in the graph class is bounded by a function of r

(Section 4.3). A graph class is somewhere dense if it is not nowhere dense. The tractability of the model-checking problem on monotone graph classes is completely characterized with a dichotomy between nowhere dense and somewhere dense graph classes. These very general results come at a cost: Frick and Grohe showed that the dependence of the run time on φ is non-elementary [38]. We want to transfer this rich algorithmic theory to complex networks. But what is the right abstraction to describe complex networks?

Network scientists observed that the chaotic and unordered structure of real networks can by captured using *randomness*. There is a vast body of research using random processes to create graphs that mimic the fundamental properties of complex networks. The most prominent ones are the preferential attachment model [3, 64], Chung–Lu model [12, 13], configuration model [60, 59], Kleinberg model [50, 51], hyperbolic graph model [53], and random intersection graph model [47, 66]. All these are random models. It has been thoroughly analyzed how well they predict various properties of complex networks [43].

When it comes to algorithmic meta-theorems on random graph models "even the most basic questions are wide open," as Grohe puts it [45]. By analyzing which models of complex networks and which values of the model-parameters allow for efficient algorithms, we aim to develop an understanding how the different properties of complex networks control their algorithmic tractability.

In this work we show for a wide range of models, including the well known preferential attachment model, that one can solve the parameterized first-order model-checking problem in almost linear FPT time. This means in particular that one can solve every problem expressible in first-order logic efficiently on these models. Our original goal was to obtain efficient algorithms only for preferential attachment graphs, but we found an abstraction that transfers these results to many other random graph models. Roughly speaking, the following two criteria are sufficient for efficiently solving first-order definable problems on a random graph model:

- The model needs to be unclustered. In particular the expected number of triangles needs to be subpolynomial.
- For every k, the fraction of vertices with degree k is roughly $O(k^{-3})$. In other words, the degree sequence needs to be bounded by a power-law distribution with exponent 3 or higher.

Models satisfying these properties include sparse Erdős–Rényi graphs, preferential attachment graphs as well as certain Chung–Lu and configuration graphs. On the other hand, the Kleinberg model, the hyperbolic random graph model, or the random intersection graph model do not satisfy these properties. Our results generalize previous results [44, 22] and match known hardness results: The model-checking problem has been proven to be hard on power-law distributions with exponent smaller than 3 [28]. We therefore identify the threshold for tractability to be a power-law coefficient of 3. It is also a big open question whether the model-checking problem can also be solved on clustered random graph models, especially since real networks tend to be clustered. Furthermore, significant engineering challenges need to be overcome to make our algorithms applicable in practice.

1.1 Average Case Complexity

Average-case complexity analyzes the typical run time of algorithms on random instances (see [6] for a survey), based on the idea that a worst-case analysis often is too pessimistic as for many problems hard instances occur rarely in the real world. Since models of complex networks are probability distributions over graphs, we analyze the run time of algorithms under average-case complexity. However, there are multiple notions and one needs to be careful which one to choose.

Assume a random graph model is asymptotically almost surely (a.a.s.) nowhere dense, i.e., a random graph from the model with n vertices belongs with probability $1-\delta(n)$ to a nowhere dense graph class, where $\lim_{n\to\infty}\delta(n)=0$ (Section 4.3). Then the first-order model-checking problem can be efficiently solved with a probability converging to one [46]. However, with probability $\delta(n)$ the run time can be arbitrarily high and the rate of convergence of $\delta(n)$ to zero can be arbitrarily slow. These two missing bounds are undesirable from an algorithmic standpoint and the field of average-case complexity has established a theory on how the run time needs to be bounded with respect to the fraction of inputs that lead to this run time.

This is formalized by the well-established notion of *average polynomial run time*, introduced by Levin [54]. An algorithm has average polynomial run time with respect to a random graph model if there is an $\varepsilon>0$ and a polynomial p such that for every n,t the probability that the algorithm runs longer than t steps on an input of size n is at most $p(n)/t^{\varepsilon}$. This means there is a polynomial trade-off between run time and fraction of inputs. This notion has been widely studied [6, 2] and is considered from a complexity theoretic standpoint the right notion of polynomial run time on random inputs. It is closed under invoking polynomial subroutines.

In our work, however, we wish to explicitly distinguish linear time. While Levin's complexity class is a good analogy to the class P, it is not suited to capture algorithms with average linear run time. For this reason, we turn to the expected value of the run time, a stronger notion than average polynomial time. In fact, using Markov's inequality we see that if an algorithm has expected linear run time, all previous measures of average tractability are also bounded. Their relationship is as follows.

$$\text{expected linear} \;\Rightarrow\; \text{expected polynomial} \;\Rightarrow\; \text{average polynomial} \;\Rightarrow\; \text{a.a.s. polynomial}$$

With this in mind we can present our notion of algorithmic tractability. A labeled graph is a graph where every vertex can have (multiple) labels. First-order formulas can have unary predicates for each type of label. These predicates test whether a vertex has a label of a certain type. We define $\mathfrak{G}$ to be the class of all graphs, and $\mathfrak{G}_{lb}$ to be the class of all vertex-labeled graphs. A function $L\colon \mathfrak{G}\to\mathfrak{G}_{lb}$ is an *l-labeling function* for $l\in\mathbf{N}$ if for every $G\in\mathfrak{G}$, $L(G)$ is a labeling of G with up to l classes of labels (see Section 4 for details). Furthermore, a *random graph model* is a sequence $\mathcal{G}=(\mathcal{G}_n)_{n\in\mathbf{N}}$, where $\mathcal{G}_n$ is a probability distribution over unlabeled simple graphs with n vertices.

▶ **Definition 1.** *We say* p-MC$(\mathrm{FO},\mathfrak{G}_{lb})$ *can be decided on a random graph model* $(\mathcal{G}_n)_{n\in\mathbf{N}}$ *in* expected time $f(|\varphi|,n)$ *if there exists a deterministic algorithm* $\mathcal{A}$ *which decides* p-MC$(\mathrm{FO},\mathfrak{G}_{lb})$ *on input* G, φ *in time* $t_{\mathcal{A}}(G,\varphi)$ *and if for all* $n\in\mathbf{N}$, *all first-order sentences* φ *and all* $|\varphi|$*-labeling functions* L, $\mathrm{E}_{G\sim\mathcal{G}_n}\big[t_{\mathcal{A}}(L(G),\varphi)\big]\le f(|\varphi|,n)$. *We say* p-MC$(\mathrm{FO},\mathfrak{G}_{lb})$ *on a random graph model can be decided in* expected FPT time *if it can be decided in expected time* $g(|\varphi|)n^{O(1)}$ *for some function* g.

In particular, this definition implies efficient average run time according to Levin's notion (which is closed under polynomial subroutines). We choose to include labels into our notion of average-case hardness for two reasons: First, it makes our algorithmic results stronger, as the expected run time is small, even in the presence of an adversary that labels the vertices of the graph. Secondly, it matches known hardness results that require adversary labeling.

1.2 Previous Work

There have been efforts to transfer the results for classical graph classes to random graph models by showing that a graph sampled from some random graph model belongs with high probability to a certain algorithmically tractable graph class. For most random graph

models the treewidth is polynomial in the size of the graph [41, 4]. Therefore, people have considered more permissive graph measures than treewidth, such as low degree [44], or bounded expansion [22, 33]. Demaine et al. showed that some Chung–Lu and configuration graphs have bounded expansion and provided empirical evidence that some real-world networks do, too [22]. However, this technique is still limited, as many random graph models (such as the preferential attachment model [22, 27]) are not known to be contained in any of the well-known tractable graph classes.

The previous tractability results presented in this section all use the following technique: Assume we have a formula φ and sample a graph of size n from a random graph model. If the sampled graph belongs to the tractable graph class, an efficient model-checking algorithm for the graph class can solve the instance in FPT time. If the graph does not belong to the graph class, the naive model-checking algorithm can still solve the instance in time $O(n^{|\varphi|})$. Assume we can show that the second case only happens with probability $\delta(n)$ converging to zero faster than any polynomial. Then $\delta(n)O(n^{|\varphi|})$ converges to zero and the expected run time remains bounded by an FPT function.

Let $p(n)$ be a function with $p(n) = O(n^\varepsilon/n)$ for all $\varepsilon > 0$. Grohe showed that one can solve p-MC(FO, $\mathfrak{G}_{lb}$) on Erdős–Rényi graphs $G(n, p(n))$ in expected time $f(|\varphi|, \varepsilon)n^{1+\varepsilon}$ for every $\varepsilon > 0$ [44]. This result was obtained by showing that with high probability the maximum degree of the random graph model is $O(n^\varepsilon)$ for every $\varepsilon > 0$ and then using a model-checking algorithm for low degree graphs. Later Demaine et al. and Farrell et al. showed that certain Chung–Lu and configuration graphs whose degrees follow a power-law distribution with exponent $\alpha > 3$ [22] as well as certain random intersection graphs [33] belong with high probability to a graph class with bounded expansion. While they do not mention it explicitly, the previous argument implies that one can solve p-MC(FO, $\mathfrak{G}_{lb}$) in expected time $f(|\varphi|)n$ on these random graph models.

There further exist some average-case hardness results for the model-checking problem. It has been shown that one cannot decide p-MC(FO, $\mathfrak{G}_{lb}$) on Erdős–Rényi graphs $G(n, 1/2)$ or $G(n, p(n))$ with $p(n) = n^\varepsilon/n$ for some $0 < \varepsilon < 1$, $\varepsilon \in \mathbf{Q}$, in expected FPT time unless AW[$*$] $\subseteq$ FPT/poly [28]. The same holds for Chung–Lu graphs with exponent $2.5 < \alpha < 3$, $\alpha \in \mathbf{Q}$. These hardness results fundamentally require the adversary labeling of Definition 1. It is a big open question whether they can be transferred to model-checking without labels.

Another thing to keep in mind when considering logic and random graphs [70] are *zero-one laws*. They state that in many Erdős–Rényi graphs every first-order formula holds in the limit either with probability zero or one [70, 42, 32]. Not all random graph models satisfy a zero-one law for first-order logic (e.g. the limit probability of the existence of a K_4 in a Chung–Lu graph with weights $w_i = \sqrt{n/i}$ is neither zero nor one).

2 Our Results

We define a property called *α-power-law-boundedness*. This property depends on a parameter α and captures many unclustered random graph models for which the fraction of vertices with expected degree $d \in \mathbf{N}$ is roughly $O(d^{-\alpha})$. Our main contribution is solving the model-checking problem efficiently on all α-power-law-bounded random graph models with $\alpha \geq 3$. This includes preferential attachment graphs, Chung–Lu graphs, Erdős–Rényi graphs, and other random graph models. Note that graphs do not need to have a power-law degree distribution to be α-power-law-bounded. Our results hold for arbitrary labelings of the random graph and are based on a novel decomposition technique for local regions of random graphs. While all previous algorithms work by placing the random graph model with high probability in a sparse graph class, our technique also works for some a.a.s. somewhere dense random graphs (e.g. preferential attachment graphs [27]).

2.1 Power-Law-Boundedness

We start by formalizing our property. Since it generalizes the Chung–Lu model, we define this model first. A Chung–Lu graph with exponent α and vertices $v_1, \dots, v_n$ is defined such that two vertices v_i and v_j are adjacent with probability $\Theta(w_i w_j/n)$ where $w_i = (n/i)^{1/(\alpha-1)}$ [12]. Furthermore all edges are independent, which means that the probability that a set of edges occurs equals the product over the probabilities of each individual edge. In our model the probability of a set of edges can be a certain factor larger than the product of the individual probabilities, which allows edges to be moderately dependent.

▶ **Definition 2.** *Let $\alpha > 2$. We say a random graph model $(\mathcal{G}_n)_{n\in\mathbf{N}}$ is α-power-law-bounded if for every $n \in \mathbf{N}$ there exists an ordering $v_1, \dots, v_n$ of $V(\mathcal{G}_n)$ such that for all $E \subseteq \binom{\{v_1,\dots,v_n\}}{2}$*

$$\Pr\big[E \subseteq E(\mathcal{G}_n)\big] \leq \prod_{v_i v_j \in E} \frac{(n/i)^{1/(\alpha-1)}(n/j)^{1/(\alpha-1)}}{n} \cdot \begin{cases} 2^{O(|E|^2)} & \text{if } \alpha > 3 \\ \log(n)^{O(|E|^2)} & \text{if } \alpha = 3 \\ O(n^\varepsilon)^{|E|^2} \text{ for every } \varepsilon > 0 & \text{if } \alpha < 3. \end{cases}$$

The probability that a set of edges E occurs may be up to a factor $2^{O(|E|^2)}$ or $\log(n)^{O(|E|^2)}$ or $O(n^\varepsilon)^{|E|^2}$ (depending on α) larger than the probability in the corresponding Chung–Lu graph. For conditional probabilities this means the following: The probability bound for an edge under the condition that some set of l edges is already present may be up to a factor $2^{O(l)}$ or $\log(n)^{O(l)}$ or $O(n^\varepsilon)^l$ larger than the unconditional probability. This lets power-law-bounded random graphs capture moderate dependence between edges. The factor undergoes a phase transition at $\alpha = 3$. The smaller factor $2^{O(|E|^2)}$ for $\alpha > 3$ was chosen to guarantee linear FPT run time of our model-checking algorithm (Theorem 4) if $\alpha > 3$. The slightly larger factor of $\log(n)^{O(|E|^2)}$ for $\alpha = 3$ was chosen to capture preferential attachment graphs while still maintaining a quasilinear FPT run time of our algorithm.

The parameter α of an α-power-law-bounded random graph model controls the degree distribution. Note that if a graph class is α-power-law-bounded it is also α'-power-law-bounded for all $2 < \alpha' < \alpha$. It can be easily seen that a vertex v_i has expected degree at most $O(n^\varepsilon)(n/i)^{1/(\alpha-1)}$ for every $\varepsilon > 0$. This means the expected degree sequence of an α-power-law-bounded random graph model is not power-law distributed with exponent smaller than α. The gap is often tight: For example, Chung–Lu graphs with a power-law degree distribution exponent α are α-power-law-bounded and preferential attachment graphs have a power-law degree distribution with exponent 3 and are 3-power-law-bounded. For the interesting case $\alpha = 3$, the inequality in Definition 2 simplifies to

$$\Pr\big[E \subseteq E(\mathcal{G}_n)\big] \leq \log(n)^{O(|E|^2)} \prod_{v_i v_j \in E} \frac{1}{\sqrt{ij}}.$$

2.2 Model Checking

We now present our model-checking algorithm for α-power-law-bounded graphs. We express its run time relative to the term

$$\tilde{d}_\alpha(n) = \begin{cases} O(1) & \alpha > 3 \\ \log(n)^{O(1)} & \alpha = 3 \\ O(n^{3-\alpha}) & \alpha < 3. \end{cases}$$

This term is related to an established property of degree distributions, namely the *second order average degree* [12]. If a random graph with n vertices has expected degrees $w_1, \dots, w_n$ then the second order average degree is defined as $\sum_{i=1}^{n} w_i^2 / \sum_{k=1}^{n} w_k$. In graphs with a power-law degree distribution α we have $w_i = \Theta((n/i)^{1/(\alpha-1)})$. The second order average degree then is $\Theta\big(\sum_{i=1}^{n}(n/i)^{2/(\alpha-1)} / \sum_{k=1}^{n}(n/k)^{1/(\alpha-1)}\big)$. For $\alpha > 3$, this term is constant, for $\alpha = 3$ it is logarithmic, and for $\alpha < 3$ it is polynomial in n [12]. Thus, we can interpret $\tilde{d}_\alpha(n)$ as an estimate of the second order average degree. We prove that the model-checking problem can be solved efficiently if $\tilde{d}_\alpha(n)$ is small.

▶ **Theorem 3.** *There exists a function f such that one can solve p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ on every α-power-law-bounded random graph model in expected time $\tilde{d}_\alpha(n)^{f(|\varphi|)}n$.*

The term $\tilde{d}_\alpha(n)$ naturally arises in our proofs and is not a consequence of how we defined the multiplicative factor (i.e., $2^{O(|E|^2)}$, $\log(n)^{O(|E|^2)}$, $O(n^\varepsilon)^{|E|^2}$) in Definition 2. In fact the dependence goes the other way: We defined the factor for each α as large as possible such that it does not dominate the run time of the algorithm. Next we specify exactly those values of α where the previous theorem leads to FPT run times. (In the third case $\varepsilon > 0$ can be chosen arbitrarily small since we require α to be arbitrarily close to 3.)

▶ **Theorem 4.** *Let $(\mathcal{G}_n)_{n\in\mathbf{N}}$ be a random graph model and $\varepsilon > 0$. There exists a function f such that one can solve p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ in expected time*

- $f(|\varphi|)n$ *if $(\mathcal{G}_n)_{n\in\mathbf{N}}$ is α-power-law-bounded for some $\alpha > 3$,*
- $\log(n)^{f(|\varphi|)}n$ *if $(\mathcal{G}_n)_{n\in\mathbf{N}}$ is α-power-law-bounded for $\alpha = 3$,*
- $f(|\varphi|, \varepsilon)n^{1+\varepsilon}$ *for all $\varepsilon > 0$* *if $(\mathcal{G}_n)_{n\in\mathbf{N}}$ is α-power-law-bounded for every $2 < \alpha < 3$.*

This solves the model-checking problem efficiently on a wide range of random graph models. These tractability results are matched by previous intractability results. (Note that the third case of Theorem 4 requires power-law-boundedness for *every* $2 < \alpha < 3$ and thus does not contradict Proposition 5.)

▶ **Proposition 5** ([28] and [26, Lemma 10.3])**.** *For every $2 < \alpha < 3$ there exists an α-power-law-bounded random graph model $(\mathcal{G}_n)_{n\in\mathbf{N}}$ such that one cannot solve p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ on $(\mathcal{G}_n)_{n\in\mathbf{N}}$ in expected* FPT *time unless* AW[$*$] $\subseteq$ FPT/poly*.*

We observe a phase transition in tractability at power-law exponent $\alpha = 3$. Also the run time of our algorithm cannot be linear in n for $\alpha \leq 3$ as a 3-power-law-bounded random graph can have for example $n\log(n)$ edges in expectation. We discuss some of the algorithmic implications of our result for some well-known random graph models in Section 9. More details can be found in Section 10 of [26].

2.3 Structure

Many algorithmic results are based on *structural decompositions*. For example, bidimensionality theory introduced by Demaine et al. [20, 21] is based on the grid minor theorem, which is itself based on a structural decomposition into a clique-sum of almost-embeddable graphs developed by Robertson and Seymour [61]. The model-checking algorithm for graph classes with bounded expansion by Dvořak, Král, and Thomas [30] relies on a structural decomposition of bounded expansion graph classes by Nešetřil and Ossona de Mendez called low tree-depth colorings [63]. Our algorithms are based on a structural decomposition of α-power-law-bounded random graph models.

All algorithms prior to this work rely on showing that a certain graph model is with high probability contained in a certain well-known tractable graph class (for example bounded expansion) and then use the structural decompositions [63] of said graph class. However,

these decompositions were not originally designed with random graphs in mind and therefore may not provide the optimal level of abstraction for random graphs. Our algorithms are based on a specially defined structural decomposition. This direct approach helps us capture random graph models that could otherwise not be captured such as the a.a.s. somewhere dense preferential attachment model. By focusing on α-power-law-bounded random graph models, we obtain structural decompositions for a wide range of models.

We observe that α-power-law-bounded random graphs have mostly an extremely sparse structure with the exception of a part whose size is bounded by the second order average degree. However, this denser part can be separated well from the remaining graph. We show that local regions consist of a core part, bounded in size by the second order average degree, to which trees and graphs of constant size are attached by a constant number of edges. This decomposition is similar to so called *protrusion decompositions*, which have been used by Bodlaender et al. to obtain meta-theorems on kernelization [5]. Our structural decomposition is valid for all graphs that fit into the framework of α-power-law-boundedness, such as preferential attachment graphs or Chung–Lu graphs. We define an approximation of the second order average degree of the degree distribution as $\hat{d}_\alpha(n) = 2$ for $\alpha > 3$, $\hat{d}_\alpha(n) = \log(n)$ for $\alpha = 3$ and $\hat{d}_\alpha(n) = n^{3-\alpha}$ for $\alpha < 3$ (similarly to $\tilde{d}_\alpha(n)$ without O-notation).

▶ **Theorem 6.** *Let $(\mathcal{G}_n)_{n\in\mathbf{N}}$ be an α-power-law-bounded random graph model. There exist constants c, r_0 such that for every $r \geq r_0$ a.a.s. for every r-neighborhood H of $\mathcal{G}_n$ one can partition $V(H)$ into three (possibly empty) sets X, Y, Z with the following properties.*

- *$|X| \leq \hat{d}_\alpha(n)^{cr^2}$.*
- *Every connected component of $H[Y]$ has size at most cr and at most c neighbors in X.*
- *Every connected component of $H[Z]$ is a tree with at most one edge to $H[X \cup Y]$.*

Removing a few vertices makes the local neighborhoods even sparser:

▶ **Corollary 7.** *Let $(\mathcal{G}_n)_{n\in\mathbf{N}}$ be an α-power-law-bounded random graph model. There exist constants c, r_0 such that for every $r \geq r_0$ a.a.s. one can remove $\hat{d}_\alpha(n)^{cr^2}$ vertices from $\mathcal{G}_n$ such that every r-neighborhood has treewidth at most 26.*

Further structural results that may be interesting beyond the purpose of model-checking as well as proofs of the results outlined here can be found in Section 9 of [26]. We now discuss how we use the decomposition of Theorem 6 for our algorithms and why decompositions similar to Corollary 7 are not sufficient for our purposes.

3 Techniques and Outline

A first building block of our algorithm is Gaifman's locality theorem [39]. It implies that in order to solve the first-order model-checking problem on a graph, it is sufficient to solve the problem on all r-neighborhoods of the graph for some small r. We can therefore restrict ourselves to the model-checking problem on the neighborhoods of random graphs. With this in mind, we want to obtain structural decompositions of these neighborhoods.

One important thing to note is that a decomposition according to Corollary 7 is not sufficient. Let us focus on the interesting case $\alpha = 3$ where efficient model-checking is still possible. Corollary 7 then states that the removal of polylogarithmically many vertices yields neighborhoods with treewidth at most 26. While we could easily solve the model-checking problem on graphs with treewidth at most 26 via Courcelle's theorem [16], we cannot solve it on graphs where we need to remove a set X of $\log(n)$ vertices to obtain a treewidth of at most 26. Every vertex not in X may have an arbitrary subset of X as neighborhood. Since

there are $2^{|X|} = n$ possible neighborhoods, we can encode a large complicated structure into this graph by stating that two vertices $i, j \in \mathbf{N}$ are adjacent if and only if there is a vertex whose neighborhood in X represents a binary encoding of the edge ij (omitting some details). Because of this, the model-checking problem on this graph class is as hard as on general graphs. We need the additional requirement that X is only loosely connected to the remaining graph. The decomposition in Theorem 6 fulfills this requirement. Every component of $H \setminus X$ has at most a constant number of neighbors in X.

Let us assume we have decompositions of the neighborhoods of a graph according to Theorem 6 where the sets X are chosen as small as possible. We can now use a variant of the Feferman–Vaught theorem [48] for each r-neighborhood to prune the protrusions and thereby construct a smaller graph that satisfies the same (short) first-order formulas as the original graph, We call this smaller graph the *kernel*. The size of this kernel will be some function of $|X|$. We then use the brute-force model-checking algorithm on the kernel.

For the first steps of the algorithm (decomposition into neighborhoods, kernelization using Feferman–Vaught) one can easily show that they always take FPT time. However, the run time of the last step requires a careful analysis. One can check a formula φ on a graph of size x in time $O(x^{|\varphi|})$ by brute force. Thus, checking the formula on the kernel of all n many r-neighborhoods of a random graph takes expected time at most $n \sum_{x=1}^{n} p_x O(x^{|\varphi|})$, where p_x is the probability that the kernelization procedure on an r-neighborhood of a random graph yields a kernel of size x. In order to guarantee a run time of the form $\log(n)^{f(|\varphi|)} n$ for some function f, p_x should be of order $\log(n)^{f(|\varphi|)} x^{-|\varphi|}$.

Earlier, we discussed that the size of the kernel will be some function of $|X|$ and that we choose X as small as possible. It is therefore sufficient to bound the probability that the set X of the decomposition of a neighborhood exceeds a certain size. Parameterizing the decomposition by two values (denoted by b and μ later on) gives us enough control to guarantee such a bound on p_x. A large part of this work is devoted to proving a good trade-off between the size of the set X of the decomposition and the probability that X is of minimal size. Furthermore, computing the set X is computationally hard, so the whole procedure has to work without knowing the set X, but only its existence.

Our proofs are structured as follows. First, we show in Section 5 that α-power-law-bounded random graph models have the following structure with high probability: They can be partitioned into sets A, B, C, where $A \cup B$ is small, $B \cup C$ is sparse and A and C locally share only few edges. This is done by characterizing this structure by a collection of small forbidden edge-sets and then excluding these edge-sets using the union bound and Definition 2. Then in Section 6 we show that the partition into A, B, C implies the protrusion decomposition of Theorem 6. In Section 7, we partially recover the protrusion decomposition from a given input, and use it to kernelize each r-neighborhood into an equivalent smaller graph. At last, in Section 8, we combine Gaifman's locality theorem with the previous algorithms and probability bounds to obtain our algorithm and bound its run time. Some proofs are quite tedious, but the nature of this problem seems to stop us from using simpler methods.

3.1 Missing Proofs

Many proofs of the results presented in this paper have been omitted. In particular Section 2.3, 5 – 9 sketch only the main ideas behind our results. All missing proofs can be found in the corresponding full version of this paper [26].

4 Notations and Definitions

4.1 Graph Notation

We use common graph theory notation [23]. The *length* of a path equals its number of edges. The *distance* between two vertices u and v ($\mathrm{dist}(u,v)$) equals the length of a shortest path between u and v. For a vertex v let $N_r^G(v)$ be the set of vertices that have distance at most r to v in G. The *radius* of a graph is the minimum among all maximum distances from one vertex to all other vertices. An *r-neighborhood* in G is an induced subgraph of G with radius at most r. The *order* of a graph is $|G| = |V(G)|$. The *size* of a graph is $\|G\| = |V(G) + E(G)|$. The *edge-excess* of a graph G is $|E(G)| - |V(G)|$.

In this work we obtain results for *labeled graphs* [45]. A labeled graph is a tuple $G = (V(G), E(G), P_1(G), \ldots, P_l(G))$ with $P_i(G) \subseteq V(G)$. We call $P_1(G), \ldots, P_l(G)$ the *labels* of G. We say a vertex v is labeled with label $P_i(G)$ if $v \in P_i(G)$. A vertex may have multiple labels. We say the unlabeled simple graph $G' = (V(G), E(G))$ is the *underlying graph* of G and G is a *labeling* of G'. All notations for graphs extend to labeled graphs as expected. The union of two labeled graphs G and H, $(G \cup H)$, is obtained by setting $V(G \cup H) = V(G) \cup V(H)$, $E(G \cup H) = E(G) \cup E(H)$ and for each label $P_i(G \cup H) = P_i(G) \cup P_i(H)$.

For a graph class $\mathcal{G}$, we define $\mathcal{G}_{lb}$ to be the class of all labelings of $\mathcal{G}$. We define $\mathfrak{G}$ to be the class of all simple graphs and $\mathfrak{G}_{lb}$ to be the class of all labeled simple graphs.

4.2 Probabilities and Random Graph Models

We denote probabilities by $\Pr[*]$ and expectation by $\mathrm{E}[*]$. We consider a random graph model to be a sequence of probability distributions. For every $n \in \mathbf{N}$ a random graph model describes a probability distribution on unlabeled simple graphs with n vertices. In order to speak of probability distributions over graphs we fix a sequence of vertices $(v_i)_{i \geq 1}$ and require that a graph with n vertices has the vertex set $\{v_1, \ldots, v_n\}$. A random graph model is a sequence $\mathcal{G} = (\mathcal{G}_n)_{n \in \mathbf{N}}$, where $\mathcal{G}_n$ is a probability distribution over all unlabeled simple graphs G with $V(G) = \{v_1, \ldots, v_n\}$. Even though some random processes naturally lead to graphs with multi-edges or self-loops, we interpret them as simple graphs by removing all self-loops and replacing multiple edges with one single edge. In slight abuse of notation, we also write $\mathcal{G}_n$ for the random variable which is distributed according to $\mathcal{G}_n$. This way, we can lift graph notation to notation for random variables of graphs: For example edge sets and neighborhoods of a random graph $\mathcal{G}_n$ are represented by random variables $E(\mathcal{G}_n)$ and $N_r^{\mathcal{G}_n}(v)$.

4.3 Sparsity

At first, we define nowhere and somewhere density as a property of *graph classes* and then lift the notation to *random graph models*. There are various equivalent definitions and we use the most common definition based on shallow topological minors.

▶ **Definition 8** (Shallow topological minor [63]). *A graph H is an r-shallow topological minor of G if a graph obtained from H by subdividing every edge up to $2r$ times is isomorphic to a subgraph of G. The set of all r-shallow topological minors of a graph G is denoted by $G \widetilde{\nabla} r$. We define the maximum clique size over all shallow topological minors of G as*

$$\omega(G \widetilde{\nabla} r) = \max_{H \in G \widetilde{\nabla} r} \omega(H).$$

▶ **Definition 9** (Nowhere dense [62]). *A graph class $\mathcal{G}$ is nowhere dense if there exists a function f, such that for all $r \in \mathbf{N}$ and all $G \in \mathcal{G}$, $\omega(G \widetilde{\nabla} r) \leq f(r)$.*

▶ **Definition 10** (Somewhere dense [62]). *A graph class $\mathcal{G}$ is somewhere dense if for all functions f there exists an $r \in \mathbf{N}$ and a $G \in \mathcal{G}$, such that $\omega(G \widetilde{\nabla} r) > f(r)$.*

Observe that a graph class is somewhere dense if and only if it is not nowhere dense. We lift these notions to random graph models using the following two definitions.

▶ **Definition 11** (a.a.s. nowhere dense). *A random graph model $\mathcal{G}$ is a.a.s. nowhere dense if there exists a function f such that for all $r \in \mathbf{N}$*

$$\lim_{n\to\infty} \Pr[\omega(\mathcal{G}_n \widetilde{\nabla} r) \leq f(r)] = 1.$$

▶ **Definition 12** (a.a.s. somewhere dense). *A random graph model $\mathcal{G}$ is a.a.s. somewhere dense if for all functions f there is an $r \in N$ such that*

$$\lim_{n\to\infty} \Pr[\omega(\mathcal{G}_n \widetilde{\nabla} r) > f(r)] = 1.$$

While for graph classes the concepts are complementary, a random graph model can both be *neither* a.a.s. somewhere dense *nor* a.a.s. nowhere dense (e.g., if the random graph model is either the empty or the complete graph, both with a probability of 1/2).

4.4 First-Order Logic

We consider only first-order logic over labeled graphs. We interpret a labeled graph $G = (V, E, P_1, \ldots, P_l)$, as a structure with universe V and signature $(E, P_1, \ldots, P_l)$. The binary relation E expresses adjacency between vertices and the unary relations $P_1, \ldots, P_l$ indicate the labels of the vertices. Other structures can be easily converted into labeled graphs. We write $\varphi(x_1, \ldots, x_k)$ to indicate that a formula φ has *free variables* $x_1, \ldots, x_k$. The *quantifier rank* of a formula is the maximum nesting depth of quantifiers in the formula. Two labeled graphs G_1, G_2 with the same signature are *q-equivalent* ($G_1 \equiv_q G_2$) if for every first-order sentence φ with quantifier rank at most q and matching signature holds $G_1 \models \varphi$ if and only if $G_2 \models \varphi$. Furthermore, $|\varphi|$ is the number of symbols in φ. There exists a simple algorithm which decides whether $G \models \varphi$ in time $O(|G|^{|\varphi|})$.

4.5 Model-Checking

With all definitions in place, we can now properly restate the model-checking problem and what it means to solve it efficiently on a random graph model. The model-checking problem on labeled graphs is defined as follows.

p-MC(FO, $\mathfrak{G}_{lb}$)	
Input:	A graph $G \in \mathfrak{G}_{lb}$ and a first-order sentence φ
Parameter:	$\lvert\varphi\rvert$
Problem:	$G \models \varphi$?

Under worst-case complexity, p-MC(FO, $\mathfrak{G}_{lb}$) is AW[$*$]-complete [25] (and PSPACE-complete when unparameterized [71]). We want average case algorithms for p-MC(FO, $\mathfrak{G}_{lb}$) to be efficient for all possible labelings of a random graph model. A function $L : \mathfrak{G} \to \mathfrak{G}_{lb}$ is called a *l-labeling function* for $l \in \mathbf{N}$ if for every $G \in \mathfrak{G}$, $L(G)$ is a labeling of G with up to l labels.

▶ **Definition 1.** *We say* p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ *can be decided on a random graph model* $(\mathcal{G}_n)_{n\in\mathbf{N}}$ *in* expected time $f(|\varphi|, n)$ *if there exists a deterministic algorithm* $\mathcal{A}$ *which decides* p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ *on input* G, φ *in time* $t_{\mathcal{A}}(G, \varphi)$ *and if for all* $n \in \mathbf{N}$, *all first-order sentences* φ *and all* $|\varphi|$*-labeling functions* L, $\mathrm{E}_{G\sim\mathcal{G}_n}\left[t_{\mathcal{A}}(L(G), \varphi)\right] \leq f(|\varphi|, n)$. *We say* p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ *on a random graph model can be decided in* expected FPT time *if it can be decided in expected time* $g(|\varphi|)n^{O(1)}$ *for some function* g.

5 Structure Theorem for Power-Law-Bounded Random Graph Models

The goal of this section is to partition α-power-law-bounded random graph models. We show in Theorem 14 that their vertices can with high probability be partitioned into sets A, B, C with the following properties: The sets A and B are small, the graph $G[B \cup C]$ is locally almost a tree, i.e., has locally only a small edge-excess, and the set B almost separates A from C, i.e., every neighborhood in $G[C]$ has only a small number of edges to A. We call (A, B, C) a b-r-μ-partition. We state the formal definition.

▶ **Definition 13** (b-r-μ-partition)**.** *Let* $b, r, \mu \in \mathbf{N}^+$. *Let* G *be a graph. A tuple* (A, B, C) *is called a* b-r-μ*-partition of* G *if*

1. *the sets* A, B, C *are pairwise disjoint and their union is* $V(G)$,
2. $|A| \leq b$ *and* $|B| \leq b^{\mu}$,
3. *every* $40\mu r$*-neighborhood in* $G[B \cup C]$ *has an edge-excess of at most* μ^2, *and*
4. *for every* $20\mu r$*-neighborhood in* $G[C]$ *there are at most* μ *edges incident to both the neighborhood and to* A.

A graph for which a b-r-μ*-partition exists is called* b-r-μ*-partitionable.*

In summary, B and C are well behaved and the large set C is almost separated from A. Note that the properties of a b-r-μ-partition depend on three parameters b, r, μ. The results of this section imply that our random graphs are asymptotically almost surely b-r-μ-partitionable for $b = \tilde{d}_{\alpha}(n)^{\Omega(1)}$ and constant r, μ. It therefore helps to assume that b is a slowly growing function in n, such as $\log(n)$ and r, μ are constants. Higher values of μ boost the probability of a random graph being b-r-μ-partitionable. The parameter μ is therefore crucial for the design of efficient algorithms. The main result of this section is the following.

▶ **Theorem 14.** *Let* $(\mathcal{G}_n)_{n\in\mathbf{N}}$ *be an* α*-power-law-bounded random graph model and let* $b, r, \mu, n \in \mathbf{N}^+$ *with* $\mu \geq 5$. *The probability that* $\mathcal{G}_n$ *is not* b-r-μ*-partitionable is at most* $\tilde{d}_{\alpha}(n)^{O(\mu^6 r^2)} b^{-\mu^2/10}$.

For proofs of the results in this section, we refer the reader to Section 6 of the full version of this paper [26]. In the following we only sketch the main ideas.

For an α-power-law-bounded random graph model $(\mathcal{G}_n)_{n\in\mathbf{N}}$, we always assume the vertices of $\mathcal{G}_n$ to be $v_1, \ldots, v_n$, ordered as in Definition 2. We will choose $A = \{v_1, \ldots, v_b\}$, $B = \{v_{b+1}, \ldots, v_{b^{\mu}}\}$, $C = \{v_{b^{\mu}+1}, \ldots, v_n\}$ and show that the probability is low that (A, B, C) does not form a b-r-μ-partition. To do so, we first define $\mathcal{H}_n(b, r, \mu)$ to be a set of graphs over the vertex set $\{v_1, \ldots, v_n\}$.

▶ **Definition 15.** *Let* $b, r, \mu, n \in \mathbf{N}^+$. *We define* $\mathcal{H}_n(b, r, \mu)$ *to be the set of*

- *all graphs with vertex set* $V \subseteq \{v_{b+1}, \ldots, v_n\}$ *such that* $|V| \leq 200r\mu^3$, *all vertices have degree at least two, and the graph has an edge-excess of* μ^2, *and*

- *all graphs* $(V_1 \cup V_2, E)$ *such that* $V_1 \subseteq \{v_1, \dots, v_b\}$, $V_2 \subseteq \{v_{b^\mu+1}, \dots, v_n\}$, $|V_1 \cup V_2| \leq 25r\mu^2$, $|E| \leq 25r\mu^2$, $|V_1| \leq \mu$, *all vertices in* V_2 *have degree at least two, and the summed degree of* V_2 *is* $2|V_2| - 2 + \mu$.

We show that if (A, B, C) is not a b-r-μ-partition then the complete edge-set of some graph in $\mathcal{H}_n(b, r, \mu)$ is present in the graph.

▶ **Definition 16.** *Let* G *be a graph and* $\mathcal{H}$ *be a set of graphs over* $V(G)$. *We say* $\mathcal{H} \sqsubseteq G$ *if for some* $H \in \mathcal{H}$, $E(H) \subseteq E(G)$.

▶ **Lemma 17.** *Let* $b, r, \mu, n \in \mathbf{N}^+$. *If a graph* G *with vertex set* $\{v_1, \dots, v_n\}$ *is not* b-r-μ-*partitionable, then* $\mathcal{H}_n(b, r, \mu) \sqsubseteq G$.

For a fixed graph $H \in \mathcal{H}_n(b, r, \mu)$, α-power-law-boundedness (Definition 2) immediately gives us a good bound on the probability $\Pr\big[E(H) \subseteq E(\mathcal{G}_n)\big]$ that its edge-set occurs. Using the union bound over all graphs in $\mathcal{H}_n(b, r, \mu)$ and some tedious calculations (Lemmas 5.6, 5.7, 5.8, 5.9 in [26]) we can bound the probability of the edge-set of any graph from $\mathcal{H}_n(b, r, \mu)$ being present in the random graph model, proving the main result Theorem 14.

At last, we present one more result in which we bound the sum of the expected sizes of all r-neighborhoods in an α-power-law-bounded graph class. This is needed in Section 8 to bound the expected run time of an algorithm that iterates over all r-neighborhoods of a graph.

▶ **Lemma 18.** *Let* $(\mathcal{G}_n)_{n \in \mathbf{N}}$ *be an* α-*power-law-bounded random graph model. Let* $r, \mu, n \in \mathbf{N}^+$ *with* $\mu \geq 5$. *Let* A_b *be the event that* $b \in \mathbf{N}^+$ *is the minimal value such that* $\mathcal{G}_n$ *is* b-r-μ-*partitionable. Then*

$$\mathrm{E}\Big[\sum_{v \in V(\mathcal{G}_n)} \|\mathcal{G}_n[N_r^{\mathcal{G}_n}(v)]\| \;\Big|\; A_b\Big] \Pr[A_b] \leq (r\mu)^{O(r)} \tilde{d}_\alpha(n)^{O(\mu^6 r^2)} b^{-\mu^2/10} n.$$

6 Protrusion Decompositions of Neighborhoods

In this section, we show that local neighborhoods of power-law-bounded graph classes are likely to have the following nice structure: They consist of a (small) core graph to which so called *protrusions* are attached. Protrusions are (possibly large) subgraphs with small treewidth and boundary. The *boundary* of a subgraph is the size of its neighborhood in the remaining graph. Protrusions were introduced by Bodlaender et al. for very general kernelization results in graph classes with bounded genus [5].

Earlier, (Theorem 14) we showed that α-power-law-bounded random graph models are (for certain values of α, b, r, μ) likely to be b-r-μ-partitionable. It is therefore sufficient to show that r-neighborhoods of b-r-μ-partitionable graphs have such a nice protrusion structure.

However, in general it is not easy to find protrusions in a graph [49]. As we later need to be able to find them, we define special protrusion decompositions, called b-r-μ-local-protrusion-partitions in which (most of) the protrusions can be efficiently identified. The main and only result of this section is the following theorem.

▶ **Theorem 19.** *Let* $b, r, \mu \in \mathbf{N}^+$ *and let* G *be a* b-r-μ-*partitionable graph. Let* G^r *be an* r-*neighborhood in* G. *Then* G^r *is* $O(\mu^{17} r^3 b)$-r-$O(\mu)$-*locally-protrusion-partitionable.*

It remains to define what a b-r-μ-local-protrusion-partition of a graph G^r with radius at most r is. The definition has to strike the right balance: It needs to be permissive enough such that neighborhoods of power-law-bounded graph classes are likely to have this structure

and it needs to be restrictive enough to admit efficient algorithms. Informally speaking, a b-r-μ-local-protrusion-partition of a graph G^r is a partition (X, Y, Z) of the vertices of G^r such that X has small size and the connected components of $G^r[Y \cup Z]$ are protrusions. In order to be able to efficiently identify the protrusions, we further require that the components of $G^r[Y]$ have bounded size and the components of $G^r[Z]$ are trees. This is formalized in the following definition.

▶ **Definition 20** (b-r-μ-local-protrusion-partition)**.** *Let $b, r, \mu \in \mathbf{N}^+$. Let G^r be a graph with radius at most r. A tuple (X, Y, Z) is called a b-r-μ-local-protrusion-partition of G^r if*

1. *the sets X, Y, Z are pairwise disjoint and their union is $V(G^r)$,*
2. *$|X| \leq b^\mu$,*
3. *every connected component of $G^r[Y]$ has size at most $r\mu^7$ and at most μ neighbors in X,*
4. *every connected component of $G^r[Z]$ is a tree with at most one edge to $G^r[X \cup Y]$,*
5. *for a subgraph H of $G^r[Y \cup Z]$ we say $N^{G^r}(V(H)) \cap X$ is the boundary of H. The connected components of $G^r[Y]$ may have at most b^μ distinct boundaries, i.e., $|\{N^{G^r}(V(H)) \cap X \mid H \text{ connected component of } G^r[Y \cup Z]\}| \leq b^\mu$.*

A graph for which a b-r-μ-local-protrusion-partition exists is called b-r-μ-locally-protrusion-partitionable.

Property 3 and 4 enforce that the components of $G^r[Y \cup Z]$ are protrusions. Later, we will transform b-r-μ-local-protrusion-partitions into equivalent graphs of bounded size by replacing the protrusions with small graphs. Thus, Property 2 and 5 are there to ensure that the resulting kernelized graph will have size roughly b^μ (without Property 5 we could only guarantee a size of roughly b^{μ^2} which is too large for our purposes).

Proving Theorem 19 involves multiple pages of proofs for which we refer to the full version of this paper [26]. Here, we only sketch the construction of an $O(\mu^{17}r^3b)$-r-$O(\mu)$-local-protrusion-partition. Let a graph G and $b, r, \mu \in \mathbf{N}^+$ be fixed. We further assume G to be b-r-μ-partitionable and we fix a b-r-μ-partition (A, B, C) of G. Let further G^r be an r-neighborhood in G and let $A^r = A \cap V(G^r)$, $B^r = B \cap V(G^r)$, $C^r = C \cap V(G^r)$. We construct an $O(\mu^{17}r^3b)$-r-$O(\mu)$-local-protrusion-partition (X, Y, Z) of G^r by placing all vertices from $A^r \cup B^r$ into X. In order to distribute the vertices C^r to the sets X, Y, and Z, we define so called *ties*.

▶ **Definition 21** (Tie)**.** *Let $W \subseteq B^r \cup C^r$. We say (u_1, u_2, v) is a W-*tie *if $u_1, u_2 \in W$ and v lies on a walk p with the following properties: Every inner vertex of p is contained in C^r and has at least two neighbors in p; u_1 and u_2 are contained only as endpoints of p; and p is contained in a $20\mu r$-neighborhood in $G[B^r \cup C^r]$.*

We use this notion to partition the set C^r. We distinguish vertices connected to A^r, vertices connected to B^r (but not to A^r), those which are connected to neither but lie on a tie, and the rest. We set

- $C^r_A = N(A^r) \cap C^r$,
- $C^r_B = (N(B^r) \setminus N(A^r)) \cap C^r$,
- $C^r_Y = \{v \mid v \in C^r \setminus (C^r_A \cup C^r_B)$ and there exist $u_1, u_2 \in C^r_A \cup C^r_B$ such that (u_1, u_2, v) is a $(C^r_A \cup C^r_B)$-tie $\}$,
- $C^r_Z = C^r \setminus (C^r_A \cup C^r_B \cup C^r_Y)$.

Finally, we define X to be the union of A^r, B^r and all vertices from $C^r_A \cup C^r_B \cup C^r_Y$ which are in a connected component of $G[C^r]$ with more than one edge to B^r. We define Y to be the vertices from $C^r_A \cup C^r_B \cup C^r_Y$ which are in a connected component of $G[C^r]$ with at most one edge to B^r, and we define $Z = C^r_Z$. The fact that (X, Y, Z) is an $O(\mu^{17}r^3b)$-r-$O(\mu)$-local-protrusion-partition is proved in Section 6 of [26].

7 Compressing Neighborhoods

In this section, we kernelize b-r-μ-partitionable graphs. This means we replace the protrusions with subgraphs of bounded size that retain the same boundary. This yields a smaller graph which is q-equivalent to the original graph. The same technique has been used for obtaining small kernels in larger graph classes, e.g., in graphs that exclude a fixed minor [36]. The main result of this section is the following theorem.

▶ **Theorem 22.** *There exists an algorithm that takes $q, r, \mu \in \mathbf{N}^+$ and a connected labeled graph G with radius at most r and at most q labels as input, runs in time at most $f(q,r,\mu)\|G\|$ for some function $f(q,r,\mu)$, and computes a labeled graph $G^* \equiv_q G$. If G is b-r-μ-locally-protrusion-partitionable for some $b \in \mathbf{N}^+$ then $|G^*| \leq f(q,r,\mu)b^\mu$.*

This kernelization procedure and its run time bound is independent in b but the size of the output kernel is not: If b is small, then the output is small. The result is obtained by replacing protrusions with the help of the Feferman–Vaught theorem [48]. However, in order to replace the protrusions, one first has to identify them. The main complication in this section lies in partitioning a graph such that the relevant protrusions can be easily identified. It is crucial that we obtain the size bound $|G^*| \leq f(q,r,\mu)b^\mu$ in Theorem 22. Weaker bounds are easier to obtain but would not be sufficient for our purposes.

We use a variant of the Feferman–Vaught theorem [48] to replace a protrusion by a q-equivalent boundaried graph of minimal size. This size depends only on q and the size of the boundary. The original Feferman–Vaught theorem states that the validity of FO-formulas on the disjoint union or Cartesian product of two graphs is uniquely determined by the value of FO-formulas on the individual graphs. Makowsky adjusted the theorem for algorithmic use [55] in the context of MSO model-checking. The following proposition contains the Feferman–Vaught theorem in a very accessible form. There is also a nice and short proof in [45]. The notation is borrowed from [45], too. At first, we need to define so called q-types.

▶ **Definition 23** ([45]). *Let G be a labeled graph and $\bar{v} = (v_1, \ldots, v_k) \in V(G)^k$, for some nonnegative integer k. The* first-order q-type of $\bar{v}$ in G *is the set $\operatorname{tp}_q^{\mathrm{FO}}(G, \bar{v})$ of all first-order formulas $\psi(x_1, \ldots x_k)$ of rank at most q such that $G \models \psi(v_1, \ldots, v_k)$.*

A q-type could be an infinite set, but one can reduce them to a finite set by syntactically normalizing formulas, so that there are only finitely many normalized formulas of fixed quantifier rank and with a fixed set of free variables. These finitely many formulas can be enumerated. For a tuple $\bar{u} = (u_1, \ldots, u_k)$, we write $\{\bar{u}\}$ for the set $\{u_1, \ldots, u_k\}$. The following is a variant of the Feferman–Vaught theorem [48].

▶ **Proposition 24** ([45, Lemma 2.3]). *Let G, H be labeled graphs and $\bar{u} \in V(G)^k$, such that $V(G) \cap V(H) = \{\bar{u}\}$. Then for all $q \geq 0$, $\operatorname{tp}_q^{\mathrm{FO}}(G \cup H, \bar{u})$ is determined by $\operatorname{tp}_q^{\mathrm{FO}}(G, \bar{u})$ and $\operatorname{tp}_q^{\mathrm{FO}}(H, \bar{u})$.*

We use this proposition to define a q-type preserving protrusion replacement procedure. Assume we identify a protrusion H of a graph with boundary $\bar{u}$. Using Courcelle's theorem [17], we can compute $\operatorname{tp}_q^{\mathrm{FO}}(H, \bar{u})$ by checking all representing formulas. By then enumerating all graphs in ascending order by their size we can find a small graph H' with $\operatorname{tp}_q^{\mathrm{FO}}(H', \bar{u}) = \operatorname{tp}_q^{\mathrm{FO}}(H', \bar{u})$. Proposition 24 now states that we can replace H with H' to obtain a smaller q-equivalent graph. We repeat this for every protrusion we identify. In the full version of this paper [26], this procedure and the proof of Theorem 22 is presented in detail (Section 7).

8 Model-Checking

In this section, we finally obtain the main result of this paper, namely that for certain values of α one can perform model-checking on α-power-law-bounded random graph models in efficient expected time.

An important tool in this section is Gaifman's locality theorem [39]. It states that first-order formulas can express only local properties of graphs. It is a well established tool for the design of model-checking algorithms (e.g. [44, 45, 37]). We use it to reduce the model-checking problem on a graph to the model-checking problem on neighborhoods of said graph [26, Lemma 8.2]. This technique is described well by Grohe [45, section 5].

To illustrate our approach, consider the following thought experiment: Let X be a non-negative random variable with $\Pr[X = b] = \Theta(b^{-10})$ for all $b \in \mathbf{N}$. Assume an algorithm that gets an integer $b \in \mathbf{N}$ as input and runs in time $t(b)$. Its expected run time on input X is $\sum_{b\in\mathbf{N}} \Theta(b^{-10})t(X)$. If $t(b) = b^{10}$ then the expected run time is infinite. If $t(b) = b^8$ then the expected run time is $\Theta(1)$. Thus, small polynomial differences in the run time can have a huge impact on the expected run time. We notice that the run time on an input has to grow slower than the inverse of the probability that the input occurs.

Let us fix a formula φ and let r and μ be constants depending on φ. In this section we provide a model-checking algorithm whose run time on a graph G depends on the minimal value $b \in \mathbf{N}$ such that G is b-r-μ-partitionable. This means, we need to solve the model-checking problem on b-r-μ-partitionable graphs faster than the inverse of the probability that b is minimal.

Section 5 states that a graph from power-law-bounded graph classes is for some b not b-r-μ-partitionable with probability approximately $b^{-\mu^2}$ (we ignore the terms in r, μ and $\tilde{d}_\alpha(n)$ for now). Thus, the probability that a value b is minimal is approximately $b^{-\mu^2}$.

Let G be a graph and a be the minimal value such that G is b-r-μ-partitionable. In Section 6 we showed that all its r-neighborhoods are $O(\mu^{17}r^3b)$-r-$O(\mu)$-locally-protrusion-partitionable. The kernelization result from Section 7 states that such r-neighborhoods can be converted in linear time into $|\varphi|$-equivalent graphs of size approximately b^μ (we again ignore the factors independent of b for now). This means, using the naive model-checking algorithm, one can decide for an r-neighborhood G^r of G whether $G^r \models \varphi$ in time approximately $\|G\|b^{\mu|\varphi|}$. Thus, one can perform model-checking on all r-neighborhoods of G in time approximately $b^{\mu|\varphi|}\sum_v \|N_r^G(v)\|$. Using Gaifman's locality theorem, this (more or less) yields the answer to the model checking problem in the *whole* graph. Let G be a graph from a power-law-bounded random graph model. In summary, we have for every $b \in \mathbf{N}$:

- $b \in \mathbf{N}$ is the minimal value such that a graph is b-r-μ-partitionable with probability approximately $b^{-\mu^2}$.
- If $b \in \mathbf{N}$ is the minimal value such that G is b-r-μ-partitionable then we can decide whether $G \models \varphi$ in time approximately $b^{\mu|\varphi|}\sum_v \|N_r^G(v)\|$.

In this example one may choose $\mu = |\varphi|^2$ such that the run time grows slower than the inverse of the probability. We changed some numbers in these examples to simplify our arguments. Thus, in reality, μ needs to be chosen slightly differently.

The proofs of this section proceed as follows. We first use slightly nonstandard version of Gaifman locality [26, Lemma 8.2] and the kernelization result in Theorem 22 to solve the model-checking in b-r-μ-partitionable graphs, obtaining the following result (see [26, Lemma 8.4] for the proof).

▶ **Lemma 25.** *Let $\mu \in \mathbf{N}^+$. There exist functions ρ and f such that for every first-order sentence φ and every labeled graph G with at most $|\varphi|$ labels one can decide whether $G \models \varphi$ in time $f(\rho(|\varphi|), \mu) b^{\mu\rho(|\varphi|)} \sum_{v \in V(G)} \|G[N^G_{\rho(|\varphi|)}(v)]\|$, where $b \in \mathbf{N}^+$ is the minimal value such that G is b-$\rho(r)$-μ-partitionable.*

The run time of this algorithm depends not only on b but also on the sum of the sizes of all neighborhoods in a graph, which might be quadratic in the worst case. In order to get almost linear expected run time, we bound the expectation of this value in Lemma 18. We can now prove our main result.

▶ **Theorem 3.** *There exists a function f such that one can solve p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ on every α-power-law-bounded random graph model in expected time $\tilde{d}_\alpha(n)^{f(|\varphi|)} n$.*

Proof. Let $(\mathcal{G}_n)_{n \in \mathbf{N}}$ be an α-power-law-bounded random graph model and φ be a first-order formula. We fix a $|\varphi|$-labeling function L and $n \in \mathbf{N}$. We consider labeled graphs with vertices $V(\mathcal{G}_n)$ whose underlying graph is distributed according to $\mathcal{G}_n$, and analyze the expected run time of the model-checking algorithm from Lemma 25 on these graphs.

Let ρ be the function from Lemma 25 and let $r = \rho(|\varphi|)$ and $\mu = \rho(|\varphi|)^2 + 100$. For every graph G there exists a value $b \in \mathbf{N}^+$ such that G is b-r-μ-partitionable (i.e., by setting $b = |V(G)|$, $A = V(G)$). Let A_b be the event that $b \in \mathbf{N}^+$ is the minimal value such that $\mathcal{G}_n$ is b-r-μ-partitionable and let R be the expected run time of the model-checking algorithm from Lemma 25. The expected run time of the algorithm is exactly $\sum_{b=1}^{\infty} \mathrm{E}[R \mid A_b] \Pr[A_b]$. We use Lemma 25 and 18 to bound

$$
\begin{aligned}
&\sum_{b=1}^{\infty} \mathrm{E}[R \mid A_b] \Pr[A_b] \\
\leq &\sum_{b=1}^{\infty} \mathrm{E}\Big[f'(r,\mu) b^{r\mu} \sum_{v \in V(\mathcal{G}_n)} \|\mathcal{G}_n[N^{\mathcal{G}_n}_r(v)]\| \mid A_b\Big] \Pr[A_b] \\
= &\sum_{b=1}^{\infty} f'(r,\mu) b^{r\mu} \,\mathrm{E}\Big[\sum_{v \in V(\mathcal{G}_n)} \|\mathcal{G}_n[N^{\mathcal{G}_n}_r(v)]\| \mid A_b\Big] \Pr[A_b] \\
\leq &\sum_{b=1}^{\infty} f'(r,\mu) b^{r\mu} (200 r \mu^3)^{O(r)} \tilde{d}_\alpha(n)^{O(\mu^6 r^2)} b^{-\mu^2/10} n \\
= &f'(r,\mu)(200 r \mu^3)^{O(r)} \tilde{d}_\alpha(n)^{O(\mu^6 r^2)} n \sum_{b=1}^{\infty} b^{-\mu^2/10 + r\mu}.
\end{aligned}
$$

Note that for $\mu = \rho(|\varphi|)^2 + 100$ and $r = \rho(|\varphi|)$ we have $\sum_{b=1}^{\infty} b^{-\mu^2/10 + r\mu} \leq \sum_{b=1}^{\infty} b^{-2} = O(1)$. This yields a run time of $\tilde{d}_\alpha(n)^{f(|\varphi|)} n$ for some function f. ◀

By substituting the values of $\tilde{d}_\alpha(n)$ and distinguishing three cases we obtain the following alternative form of our main result.

▶ **Theorem 4.** *Let $(\mathcal{G}_n)_{n \in \mathbf{N}}$ be a random graph model and $\varepsilon > 0$. There exists a function f such that one can solve p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ in expected time*

- $f(|\varphi|) n$ *if $(\mathcal{G}_n)_{n \in \mathbf{N}}$ is α-power-law-bounded for some $\alpha > 3$,*
- $\log(n)^{f(|\varphi|)} n$ *if $(\mathcal{G}_n)_{n \in \mathbf{N}}$ is α-power-law-bounded for $\alpha = 3$,*
- $f(|\varphi|, \varepsilon) n^{1+\varepsilon}$ *for all $\varepsilon > 0$ if $(\mathcal{G}_n)_{n \in \mathbf{N}}$ is α-power-law-bounded for every $2 < \alpha < 3$.*

9 Implications for Various Graph Models

A wide range of unclustered random graph models are α-power-law-bounded. In this section, we discuss the implications of our result for preferential attachment, Chung–Lu, and Erdős–Rényi graphs. A more detailed discussion with proofs can be found in Section 10 in the full version [26].

9.1 Preferential Attachment Model

The preferential attachment model [3, 64] may be the best-known model for complex networks. In this model, graphs are created by a random process that iteratively adds new vertices and randomly connects them to already existing ones, where the attachment probability is proportional to the current degree of a vertex. The preferential attachment process exhibits small world behavior [24] and has been widely recognized as a reasonable explanation of the heavy tailed degree distribution of complex networks [7]. It has a vanishing clustering coefficient [8], but there exist extensions of the model that admit clustering [73].

Recent efficient model-checking algorithms on random graph models only worked on random graph models that asymptotically almost surely (a.a.s.) are nowhere dense [44, 22]. It is known that preferential attachment graphs are not a.a.s. nowhere dense [22] and even a.a.s. somewhere dense [27], thus previous techniques do not work.

We define G^n_m to be the preferential attachment graph with n vertices and m edges per vertex. Usually, the parameter m is considered to be constant. We obtain efficient algorithms even if we allow m to be a function of the size of the network. For a function $m(n)\colon \mathbf{N} \to \mathbf{N}$ we define $(G^n_{m(n)})_{n\in\mathbf{N}}$ to be the random graph model where the number of edges per vertex grows according to $m(n)$. Previous work [29] implies that this model is α-power-law-bounded ([26, Lemma 10.1]), which immediately implies the following model checking result.

▶ **Corollary 26.** *Let $m\colon \mathbf{N} \to \mathbf{N}$. There exists a function f such that one can solve p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ on the preferential attachment model $(G^n_{m(n)})_{n\in\mathbf{N}}$ in expected time*

- $\log(n)^{f(|\varphi|)}n$ *if* $m(n) = \log(n)^{O(1)}$,
- $f(|\varphi|,\varepsilon)n^{1+\varepsilon}$ *for every* $\varepsilon > 0$ *if* $m(n) = O(n^\varepsilon)$ *for every* $\varepsilon > 0$.

9.2 Chung–Lu Model

This model generates random graphs that fit a certain degree sequence and has been studied extensively [12, 13, 14]. The degree sequence is defined by a power-law distribution with exponent α. One can easily show that this model is α-power-law-bounded [26, Lemma 10.3]. We can therefore characterize the tractability of the labeled model-checking problem on Chung–Lu graphs based on α.

▶ **Corollary 27.** *Let $\mathcal{G}$ be the Chung–Lu random graph model with exponent α. There exists a function f such that one can solve p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ on $\mathcal{G}$ in expected time*

- $f(|\varphi|)n$ *if* $\alpha > 3$,
- $\log(n)^{f(|\varphi|)}n$ *if* $\alpha = 3$.

Furthermore, if $2.5 \leq \alpha < 3$, $\alpha \in \mathbf{Q}$ then one cannot solve p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ on $\mathcal{G}$ in expected FPT *time unless* AW$[*] \subseteq$ FPT/poly.

Previously, the model-checking problem has been known to be tractable on Chung–Lu graphs with exponent $\alpha > 3$, and hard on Chung–Lu graphs with exponent $2.5 \leq \alpha < 3$. The important case $\alpha = 3$ was open. Furthermore, the previous tractability result assumes

the maximum expected degree of a Chung–Lu graph with exponent α to be at most $O(n^{1/\alpha})$, while in the canonical definition of Chung–Lu graphs it is $\Theta(n^{1/(\alpha-1)})$. Our results hold for the canonical definition. The missing case $\alpha < 2.5$ is still open. We believe it can be proven to be hard with similar techniques as for $2.5 \leq \alpha < 3$.

9.3 Erdős–Rényi Model

One of the earliest and most intensively studied random graph models is the Erdős–Rényi model [31]. We say $G(n, p(n))$ is a random graph with n vertices where each pair of vertices is connected independently uniformly at random with probability $p(n)$. Many properties of Erdős–Rényi graphs are well studied, including but not limited to, threshold phenomena, the sizes of components, diameter, and length of paths [9]. With a three-line argument [26, Lemma 10.8], we obtain a fine grained picture over the tractability of the model-checking problem on sparse Erdős–Rényi graphs.

▶ **Corollary 28.** *There exists a function f such that one can solve p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ on $G(n, p(n))$ in expected time*

- $f(|\varphi|)n$ *if* $p(n) = O(1/n)$,
- $\log(n)^{f(|\varphi|)}n$ *if* $p(n) = \log(n)^{O(1)}/n$,
- $f(|\varphi|, \varepsilon)n^{1+\varepsilon}$ *for every* $\varepsilon > 0$ *if* $p(n) = O(n^{\varepsilon}/n)$ *for every* $\varepsilon > 0$.

The third case has been shown previously by Grohe [44]. Furthermore, under reasonable assumptions (AW[$*$] $\not\subseteq$ FPT/poly) we know that p-MC$(\mathrm{FO}, \mathfrak{G}_{lb})$ cannot be decided in expected FPT time on denser Erdős–Rényi graphs with $p(n) = n^{\delta}/n$ for some $0 < \delta < 1$, $\delta \in \mathbf{Q}$ [28].

10 Conclusion

We define α-power-law-bounded random graphs which generalize many unclustered random graphs models. We provide a structural decomposition of neighborhoods of these graphs and use it to obtain a meta-algorithm for deciding first-order properties in the the preferential attachment-, Erdős–Rényi-, Chung–Lu- and configuration random graph model.

There are various factors to consider when evaluating the practical implications of this result. The degree distribution of most real world networks is similar to a power-law distribution with exponent between two and three [15], but our algorithm is only fast for exponents at least three. This leaves many real world networks where our algorithm is slow. However, it has been shown that the model-checking problem (with labels) becomes hard on these graphs if we assume independently distributed edges [28].

So far, we do not know whether the model-checking problem is hard or tractable on clustered random graphs. If a random graph model is 3-power-law-bounded then one can show that the expected number of triangles is polylogarithmic (via union bound of all possible embeddings of a triangle). Therefore, random models with clustering, such as the Kleinberg model [50], the hyperbolic random graph model [53, 11], or the random intersection graph model [47], which have a high number of triangles currently do not fit into our framework (see [26, Section 10.5] for a proof that random intersection graphs are not α-power-law-bounded for any α). This is unfortunate, since clustering is a key aspect of real networks [72]. In the future, we hope to extend our results to clustered random graph models. We observe that some clustered random graph models can be expressed as *first-order transductions* of α-power-law-bounded random graph models. For example the random intersection graph model is a transduction of a sparse Erdős–Rényi graph. We believe this connection can be

used to transfer tractability results to clustered random graphs. If we can efficiently compute for a clustered random graph model $\mathcal{G}$ a pre-image of a transduction that is distributed like an α-power-law-bounded random graph then we can efficiently solve p-MC(FO, $\mathfrak{G}_{lb}$) on $\mathcal{G}$. The same idea is currently being considered for solving the model checking problem for transductions of sparse graph classes (e.g. structurally bounded expansion classes) [40].

In our algorithm, we use Gaifman's locality theorem to reduce our problem to r-neighborhoods of the input graph. In this construction the value of r can be exponential in the length of the formula [39]. On the other hand, the small world property states that the radius of real networks is rather small. This means, even for short formulas our neighborhood-based approach may practically be working on the whole graph instead of neighborhoods. It would be interesting to analyze for which values of r practical protrusion decompositions according to Theorem 6 exist in the real world.

At last, a big problem with all parameterized model-checking algorithms is their large run time dependence on the length of the formula. Grohe and Frick showed that already on trees every first-order model-checking algorithm takes worst-case time at least $f(|\varphi|)n$ where f is a non-elementary tower function [38]. So far, it is unclear whether this also holds in the average-case setting. The results presented in this paper have a non-elementary dependence on the length of the formula. We are curious whether one can find average-case model-checking algorithms with elementary expected FPT run time. In summary, many more obstacles need to be overcome to obtain a truly practical general purpose meta-algorithm for complex networks.

References

1 Réka Albert, Hawoong Jeong, and Albert-László Barabási. Internet: Diameter of the world-wide web. *Nature*, 401(6749):130, 1999.

2 Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, 2009.

3 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

4 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Hyperbolic random graphs: Separators and treewidth. In *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

5 Hans L Bodlaender, Fedor V Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M Thilikos. (Meta) kernelization. *Journal of the ACM (JACM)*, 63(5):44, 2016.

6 Andrej Bogdanov and Luca Trevisan. Average-Case Complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1):1–106, 2006.

7 Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. The degree sequence of a scale-free random graph process. *Random Structures & Algorithms*, 18(3):279–290, May 2001.

8 Béla Bollobás and Oliver M Riordan. Mathematical results on scale-free random graphs. *Handbook of graphs and networks: from the genome to the internet*, pages 1–34, 2003.

9 Béla Bollobás. *Random Graphs*. Cambridge University Press, 2nd edition, 2001.

10 Anna D. Broido and Aaron Clauset. Scale-free networks are rare. *Nature communications*, 10(1):1017, 2019.

11 Elisabetta Candellero and Nikolaos Fountoulakis. Clustering and the hyperbolic geometry of complex networks. *Internet Mathematics*, 12(1-2):2–53, 2016.

12 Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proc. of the National Academy of Sciences*, 99(25):15879–15882, 2002.

13 Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6(2):125–145, 2002.

14 Fan Chung and Linyuan Lu. *Complex graphs and networks*, volume 107. American Math. Soc., 2006.

15 Aaron Clauset, Cosma Rohilla Shalizi, and Mark E. J. Newman. Power-Law Distributions in Empirical Data. *SIAM Review*, 51(4):661–703, 2009.

16 Bruno Courcelle. The monadic second-order logic of graphs I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

17 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

18 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

19 Anuj Dawar, Martin Grohe, and Stephan Kreutzer. Locally Excluding a Minor. In *Proceedings of the 22nd Symposium on Logic in Computer Science*, pages 270–279, 2007.

20 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, November 2005. doi:10.1145/1101821.1101823.

21 Erik D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

22 Erik D. Demaine, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, Somnath Sikdar, and Blair D. Sullivan. Structural sparsity of complex networks: Bounded expansion in random models and real-world graphs. *J. Comput. Syst. Sci.*, 105:199–241, 2019. doi:10.1016/j.jcss.2019.05.004.

23 R. Diestel. *Graph Theory*. Springer, Heidelberg, 2010.

24 Sander Dommers, Remco van der Hofstad, and Gerard Hooghiemstra. Diameters in preferential attachment models. *Journal of Statistical Physics*, 139(1):72–107, 2010.

25 Rod G. Downey, Michael R. Fellows, and Udayan Taylor. The Parameterized Complexity of Relational Database Queries and an Improved Characterization of W[1]. *DMTCS*, 96:194–213, 1996.

26 Jan Dreier, Philipp Kuinke, and Peter Rossmanith. First-order model-checking in random graphs and complex networks, 2020. arXiv:2006.14488.

27 Jan Dreier, Philipp Kuinke, and Peter Rossmanith. Maximum shallow clique minors in preferential attachment graphs have polylogarithmic size. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 176 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

28 Jan Dreier and Peter Rossmanith. Hardness of FO model-checking on random graphs. In *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.11.

29 Jan Dreier and Peter Rossmanith. Motif counting in preferential attachment graphs. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.13.

30 Zdenek Dvořak, Daniel Král, and Robin Thomas. Deciding First-Order Properties for Sparse Graphs. In *Proceedings of the 51st Conference on Foundations of Computer Science*, pages 133–142, 2010.

31 P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

32 Ronald Fagin. Probabilities on finite models 1. *The Journal of Symbolic Logic*, 41(1):50–58, 1976.

33 Matthew Farrell, Timothy D Goodrich, Nathan Lemons, Felix Reidl, Fernando Sánchez Villaamil, and Blair D Sullivan. Hyperbolicity, degeneracy, and expansion of random intersection graphs. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 29–41. Springer, 2015.

34 Jörg Flum, Markus Frick, and Martin Grohe. Query Evaluation via Tree-Decompositions. *Journal of the ACM (JACM)*, 49(6):716–752, 2002.

35 Jörg Flum and Martin Grohe. Fixed-Parameter Tractability, Definability, and Model-Checking. *SIAM Journal on Computing*, 31(1):113–145, 2001.

36 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M Thilikos. Bidimensionality and kernels. In *Proc. of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 503–510, 2010.

37 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM (JACM)*, 48(6):1184–1206, 2001.

38 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of pure and applied logic*, 130(1-3):3–31, 2004.

39 Haim Gaifman. On local and non-local properties. In *Studies in Logic and the Foundations of Mathematics*, volume 107, pages 105–135. Elsevier, 1982.

40 Jakub Gajarský, Petr Hliněný, Jan Obdrzálek, Daniel Lokshtanov, and M. S. Ramanujan. A new perspective on FO model checking of dense graph classes. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 176–184, 2016. doi:10.1145/2933575.2935314.

41 Yong Gao. Treewidth of Erdős–Rényi random graphs, random intersection graphs, and scale-free random graphs. *Discrete Applied Mathematics*, 160(4-5):566–578, 2012.

42 Yu V Glebskii, DI Kogan, MI Liogon'kii, and VA Talanov. Range and degree of realizability of formulas in the restricted predicate calculus. *Cybernetics and Systems Analysis*, 5(2):142–154, 1969.

43 Anna Goldenberg, Alice X. Zheng, Stephen E. Fienberg, Edoardo M. Airoldi, et al. A survey of statistical network models. *Foundations and Trends in Machine Learning*, 2(2):129–233, 2010.

44 Martin Grohe. Generalized model-checking problems for first-order logic. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 12–26. Springer, 2001.

45 Martin Grohe. Logic, graphs, and algorithms. *Logic and Automata*, 2:357–422, 2008.

46 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3), 2017.

47 Michał Karoński, Edward R. Scheinerman, and Karen B. Singer-Cohen. On random intersection graphs: The subgraph problem. *Combinatorics, Probability and Computing*, 8(1-2):131–159, 1999.

48 Carol Karp. The first order properties of products of algebraic systems. fundamenta mathematicae. *Journal of Symbolic Logic*, 32(2):276–276, 1967. doi:10.2307/2271704.

49 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms (TALG)*, 12(2):21, 2016.

50 Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd Symposium on Theory of Computing*, pages 163–170, 2000.

51 Jon M. Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, 2000.

52 Stephan Kreutzer. Algorithmic meta-theorems. In *International Workshop on Parameterized and Exact Computation*, pages 10–12. Springer, 2008.

53 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.

54 Leonid A. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

55 Johann A. Makowsky. Algorithmic uses of the feferman–vaught theorem. *Annals of Pure and Applied Logic*, 126(1-3):159–213, 2004.

56 Stanley Milgram. The small world problem. *Psychology Today*, 2(1):60–67, 1967.

57 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

58 Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 29–42. ACM, 2007.

59 M. Molloy and B. A. Reed. The size of the giant component of a random graph with a given degree sequence. *Combin., Probab. Comput.*, 7(3):295–305, 1998.

60 Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random Structures & Algorithms*, 6(2-3):161–180, 1995.

61 Paul D. Seymour N. Robertson. Graph minors XVI. Excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89:43–76, 2003.

62 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity*. Springer, 2012.

63 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.

64 Derek de Solla Price. A general theory of bibliometric and other cumulative advantage processes. *Journal of the American society for Information science*, 27(5):292–306, 1976.

65 Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007.

66 Katarzyna Rybarczyk. Diameter, connectivity, and phase transition of the uniform random intersection graph. *Discrete Mathematics*, 311(17):1998–2019, 2011.

67 Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

68 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 151–163. ACM, 2018. doi:10.1145/3196959.3196971.

69 Detlef Seese. Linear time computable problems and first-order descriptions. *Math. Struct. in Comp. Science*, 6:505–526, 1996.

70 Joel Spencer. *The strange logic of random graphs*, volume 22. Springer Science & Business Media, 2013.

71 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

72 Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998.

73 Konstantin Zuev, Marián Boguná, Ginestra Bianconi, and Dmitri Krioukov. Emergence of soft communities from geometric preferential attachment. *Scientific reports*, 5:9421, 2015.

Optimally Handling Commitment Issues in Online Throughput Maximization

Franziska Eberle
Department for Mathematics and Computer Science, University of Bremen, Germany
feberle@uni-bremen.de

Nicole Megow
Department for Mathematics and Computer Science, University of Bremen, Germany
nicole.megow@uni-bremen.de

Kevin Schewior
Universität zu Köln, Department of Mathematics and Computer Science, Germany
kschewior@gmail.com

Abstract

We consider a fundamental online scheduling problem in which jobs with processing times and deadlines arrive online over time at their release dates. The task is to determine a feasible preemptive schedule on m machines that maximizes the number of jobs that complete before their deadline. Due to strong impossibility results for competitive analysis, it is commonly required that jobs contain some *slack* $\varepsilon > 0$, which means that the feasible time window for scheduling a job is at least $1 + \varepsilon$ times its processing time. In this paper, we answer the question on how to handle commitment requirements which enforce that a scheduler has to guarantee at a certain point in time the completion of admitted jobs. This is very relevant, e.g., in providing cloud-computing services and disallows last-minute rejections of critical tasks. We present the first online algorithm for handling commitment on parallel machines for arbitrary slack ε. When the scheduler must commit upon starting a job, the algorithm is $\Theta(\frac{1}{\varepsilon})$-competitive. Somewhat surprisingly, this is the same optimal performance bound (up to constants) as for scheduling without commitment on a single machine. If commitment decisions must be made before a job's slack becomes less than a δ-fraction of its size, we prove a competitive ratio of $\mathcal{O}(\frac{1}{\varepsilon-\delta})$ for $0 < \delta < \varepsilon$. This result nicely interpolates between commitment upon starting a job and commitment upon arrival. For the latter commitment model, it is known that no (randomized) online algorithms admits any bounded competitive ratio.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases Deadline scheduling, throughput, online algorithms, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.41

Funding *Nicole Megow*: Partially supported by the German Science Foundation (DFG) under contract ME 3825/1.
Kevin Schewior: Partially supported by the DAAD within the PRIME program using funds of BMBF and the EU Marie Curie Actions.

1 Introduction

We consider the following fundamental online scheduling model: jobs from an unknown job set arrive online over time at their *release dates* r_j. Each job has a *processing time* $p_j \geq 0$ and a *deadline* d_j. There are m identical machines to process these jobs or a subset of them. A job is said to *complete* if it receives p_j units of processing time within the interval $[r_j, d_j)$. We allow *preemption*, i.e., the processing of a job can be interrupted at any time. We distinguish schedules *with* and *without migration*. If we allow migration, then a preempted job can resume processing on any machine whereas it must run completely on the same machine otherwise. In the three-field notation this problem is $P \,|\, \text{online } r_j, \text{pmtn} \,|\, \sum(1 - U_j)$ [16].

In a feasible schedule, two jobs are never processing at the same time on the same machine. The number of completed jobs in a feasible schedule is called *throughput*. The task is to find a feasible schedule with maximum throughput.

As jobs arrive online, we cannot hope to find an optimal schedule [12]. To assess the performance of online algorithms, we resort to standard *competitive analysis*. This means, we compare the throughput of an online algorithm with the throughput achievable by an optimal offline algorithm that knows the job set in advance.

It is well-known that "tight" jobs with $d_j - r_j \approx p_j$ prohibit competitive online decision making as jobs must start immediately and do not leave a chance for observing online arrivals [6]. Thus, it is commonly required that jobs contain some *slack* $\varepsilon > 0$, i.e., every job j satisfies $d_j - r_j \geq (1+\varepsilon)p_j$. The competitive ratio of our online algorithm will be a function of ε; the greater the slack, the better should the performance of our algorithm be. This slackness parameter has been considered in previous work, e.g., in [2, 4, 8, 14, 15, 24, 26]. Other results for scheduling with deadlines use speed scaling, which can be viewed as another way to add slack to the schedule, e.g., [1, 3, 17, 18, 25].

In this paper, we focus on the question how to handle *commitment* requirements in online throughput maximization. Modeling commitment addresses the issue that a good-throughput schedule may abort jobs close to their deadlines in favor of many shorter and more urgent tasks [13], which may not be acceptable for the job owner. Consider a company that starts outsourcing mission-critical processes to external clouds and that needs a guarantee that jobs complete before a certain time point when they cannot be moved to another computing cluster anymore. In other situations, a commitment to complete jobs might be required even earlier just before starting the job, e.g., for a faultless copy of a database [8].

Different commitment models have been formalized [2, 8, 24]. The requirement to commit at a job's release date has been ruled out for online throughput maximization by strong impossibility results (even for randomized algorithms) [8]. We distinguish *(i) commitment upon job admission* and *(ii) δ-commitment*. In the first model, an algorithm may discard a job any time before its start, we say its admission. This reflects a situation such as the faultless copy of a database. In the second model, δ-commitment, an online algorithm must commit to complete a job when its remaining slack is not less than a δ-fraction of the job size, for $0 < \delta < \varepsilon$. The latest time for committing to job j is then $d_j - (1+\delta)p_j$. This models an early enough commitment (parameterized by δ) for mission-critical jobs. Recently, a first unified approach has been presented for these models in [8]. Gaps in the performance bounds remained and it was left open if scheduling with commitment is even "harder" than without commitment.

In this work, we give tight results for online throughput maximization on parallel machines and answer the "hardness" question to the negative. We give an algorithm that achieves the provably best competitive ratio (up to constants) for the aforementioned commitment models. Somewhat surprisingly, we show that the same competitive ratio of $\mathcal{O}(\frac{1}{\varepsilon})$ can be achieved for both, scheduling *without* commitment and *with* commitment upon admission. Further, our algorithm does not require job migration. For parallel machines, our algorithm is the first online algorithm with bounded competitive ratio for arbitrary slack parameter ε.

Previous results

Preemptive online scheduling with hard deadlines and models for admission control have been studied rigorously, see, e.g., [5, 14, 15] and references therein. Already in the 90s several impossibility results were shown for jobs without slack [6, 7, 21–23]. The only positive result independent of slack for online throughput maximization without commitment seems to be a randomized $\mathcal{O}(1)$-competitive single-machine algorithm [20]. The best possible deterministic algorithm in this setting is $\Theta(1/\varepsilon)$-competitive for instances with ε-slack [8].

Throughput maximization with commitment has attracted researchers more recently [2, 8, 24]. We summarize the state-of-the art for the particular problem of online throughput maximization with commitment. For a single machine, Chen et al. [8] presented a universal algorithmic framework, which achieved bounded competitive ratios for several commitment models and even the tight result for scheduling without commitment. More precisely, their algorithm is $\mathcal{O}(1/\varepsilon^2)$-competitive for commitment upon admission and $\mathcal{O}(\varepsilon/((\varepsilon-\delta)\delta^2))$-competitive, for $0 < \delta < \varepsilon$, in the δ-commitment model. This improved on an earlier algorithm by Azar et al. [2] for the *δ-commitment model* (in the context of truthful mechanisms for a weighted setting) that is $\mathcal{O}(1/\varepsilon^2)$-competitive if the slack ε is sufficiently large. Chen et al. showed a lower bound of $\Omega(1/\varepsilon)$ for deterministic scheduling algorithms without commitment, which is tight in that model and also holds for the more restrictive commitment models. A significant gap between lower and upper bounds remained. On parallel machines, there is a competitive algorithm for online throughput maximization with commitment if the slack ε is sufficiently large [2].

In a natural generalization of our problem, jobs have associated individual weights and we aim for a schedule with maximum weighted throughput. The special case with each job j satisfying $w_j = p_j$ (aka machine utilization) is well understood. A simple greedy algorithm achieves the best possible competitive ratio $\Theta(1/\varepsilon)$ [11, 14] on a single machine in both commitment models, even for commitment upon arrival. For scheduling with commitment on m parallel identical machines there is an $\mathcal{O}(\sqrt[m]{1/\varepsilon})$-competitive algorithm and an almost matching lower bound [26]. It is worth mentioning that machine utilization without commitment even allows for constant competitive ratios independent of slack [6, 21, 22, 27]. General weighted (and even unweighted) throughput maximization is much less tractable. For general weights, there is no bounded competitive ratio possible in any of the aforementioned commitment models [2, 8, 24]. For weighted throughput maximization without commitment requirements there is an $\mathcal{O}(1/\varepsilon^2)$-competitive online algorithm [24].

The power of migration has been investigated in several contexts: While it is typically large in online scheduling, cf. machine-utilization [26] and resource-minimization [9, 10], we show that, in our online setting, the guarantees that can be achieved by migratory and non-migratory algorithms are within a constant factor, similar to the offline problem [19].

Our results and techniques

Our main result is one algorithm that is best possible (up to constant factors) for online throughput maximization with and without commitment on parallel identical machines. Our algorithm does not migrate jobs and still achieves a competitive ratio that matches the general lower bound for migratory algorithms.

For scheduling with commitment upon admission, we give an (up to constant factors) optimal online algorithm with competitive ratio $\Theta(1/\varepsilon)$. For scheduling with δ-commitment, our result interpolates between the models commitment upon starting a job and commitment upon arrival. If $\delta \leq \varepsilon/2$, the competitive ratio is $\Theta(1/\varepsilon)$ which is best possible [8]. For $\delta \to \varepsilon$, the commitment requirements essentially implies commitment upon job arrival which has unbounded competitive ratio [8]. Note that this is the first online algorithm with bounded competitive ratio for arbitrary slackness parameter ε.

▶ **Theorem 1.** *Consider throughput maximization on parallel identical machines with or without migration. There is an $\mathcal{O}(\frac{1}{\varepsilon-\delta'})$-competitive online algorithm with commitment where $\delta' = \varepsilon/2$ in the commitment upon admission model and $\delta' = \max\{\delta, \varepsilon/2\}$ in the δ-commitment model.*

Clearly, scheduling with commitment is more restrictive than without commitment. Hence, our algorithm is also (up to constants) optimal for the problem $P \,|\, \text{online } r_j, \text{pmtn} \,|\, \sum(1-U_j)$ without any commitment requirements as its competitive ratio matches the lower bound [8].

▶ **Theorem 2.** *There is a $\Theta(1/\varepsilon)$-competitive algorithm for online throughput maximization on parallel identical machines without commitment requirements, with and without migration.*

The challenge in online scheduling with commitment is that, once we committed to complete a job, the remaining slack of this job has to be spent very carefully. The key is a job admission scheme which is implemented by different parameters. The high-level objectives are:

- **(i)** never start a job for the first time if its remaining slack is too small (parameter δ),
- **(ii)** during the processing of a job, admit only significantly shorter jobs (parameter γ), and
- **(iii)** for each admitted shorter job, block some time period (parameter β) during which no other jobs of similar size are accepted.

While the first two goals are quite natural and have been used before [8, 24], the third goal is crucial for our new tight result. The intuition is the following: suppose we committed to complete a job with processing time 1 and have only a slack of $\mathcal{O}(\varepsilon)$ left before the deadline of this job. Suppose that c substantially smaller jobs of size $1/c$ arrive where c is the competitive ratio we aim for. On the one hand, if we do not accept any of them, we cannot hope to achieve c-competitiveness. On the other hand, accepting too many of them fills up the slack and, thus, leaves no room for even smaller jobs. The idea is to keep the flexibility for future small jobs by grouping jobs of similar size (within a factor two) into classes. This gives the fine-grained classification of jobs which is crucial for our new tight result. We distinguish two time periods with different class structures that guide acceptance. During the *scheduling interval* of a job j, we have a more restrictive acceptance scheme that ensures the completion of j whereas in the *blocking period* we guarantee the completion of previously accepted jobs. In contrast, the previous algorithm in [8] uses one long region with a uniform acceptance threshold and is then too conservative in accepting jobs.

As a key contribution on the technical side, we give a strong bound on the processing volume of any feasible non-migratory schedule in terms of the accepted volume of a certain class of online algorithms. It is crucial for our analysis and might be of independent interest.

2 The blocking algorithm

In this section, we describe the *blocking algorithm* which handles scheduling with commitment. We assume that the slackness constant $\varepsilon > 0$ and, in the δ-commitment model, $0 < \delta < \varepsilon$ are given. If δ is not part of the input or if $\delta \leq \varepsilon/2$, we set $\delta = \frac{\varepsilon}{2}$.

The algorithm never migrates jobs between machines, i.e., a job is only processed by the machine that initially started to process it, we say the job has been *admitted* to this machine. Moreover, our algorithm commits to completing a job upon admission. Hence, its remaining slack has to be spent very carefully on admitting other jobs while being competitive. As our algorithm does not migrate jobs, it transfers the admission decision to the shortest admitted and not yet completed job on each machine. Then, a job only admits significantly shorter jobs and prevents the admission of too many jobs of similar size. To this end, the algorithm maintains two types of intervals for each admitted job, a *scheduling interval* and a *blocking period*. A job can only be processed in its scheduling interval. Thus, it has to complete in this interval while admitting other jobs. Job j only admits jobs that are smaller by a factor of $\gamma = \frac{\delta}{16} < 1$. For an admitted job k, job j creates a blocking period of length at most βp_k, where $\beta = \frac{16}{\delta}$, which blocks the admission of similar-length jobs (cf. Figure 1).

For **scheduling**, the algorithm follows the *Shortest Processing Time (SPT)* order for the set of uncompleted jobs assigned to a machine, which is independent of the admission scheme. SPT ensures that, in the blocking periods of any job k admitted by j, j has highest priority.

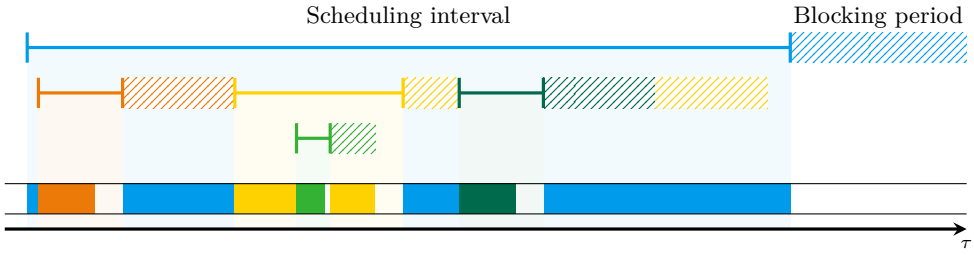


Figure 1 Scheduling interval, blocking period, and processing intervals.

For **admitting** jobs, the algorithm keeps track of *available* jobs at any time point τ. A job j with $r_j \leq \tau$ is called available if it has not yet been admitted to a machine by the algorithm and its deadline is not too close, i.e., $d_j - \tau \geq (1+\delta)p_j$.

Whenever a job j is available at a time τ when there is a machine i such that this time is not contained in the scheduling interval of any other job, the shortest such job j is immediately admitted to i, creating the scheduling interval $S(j) = [\tau, \tau + (1+\delta)p_j) := [a_j, e_j)$ and an empty blocking period $B(j) = \emptyset$. In general, however, the blocking period is a finite union of time intervals associated with job j, and its size is the sum of lengths of the intervals, denoted by $|B(j)|$. Four types of events trigger a decision of the algorithm at time τ: the release of a job, the end of a blocking period, the end of a scheduling interval, and the admission of a job. In any of these four cases, the algorithm calls the class admission routine. This subroutine iterates over all machines i and checks if j, the shortest job on i whose scheduling interval contains τ, can admit the currently shortest available job $j^\star$.

To this end, any admitted job j classifies available jobs k with $r_k \in S(j)$ and $p_k < \gamma p_j$ depending on their processing time. More precisely, job j maintains a *class structure* $(\mathcal{C}_c(j))_{c\in\mathbb{N}_0}$ where $\mathcal{C}_c(j)$ contains all jobs k that are available at some time during $S(j)$ and satisfy $\frac{\gamma}{2^{c+1}}p_j \leq p_k < \frac{\gamma}{2^c}p_j$. Only jobs $k \in \mathcal{C}_c(j)$ for $c \in \mathbb{N}_0$ qualify for admission by j. Upon admission by j, job $j^\star$ obtains two disjoint consecutive intervals, the *scheduling interval* $S(j^\star) = [a_{j^\star}, e_{j^\star})$ and the *blocking period* $B(j^\star)$ of size at most $\beta p_{j^\star}$. At the admission of job $j^\star$, the blocking period $B(j^\star)$ is planned to start at $e_{j^\star}$, the end of $j^\star$'s scheduling interval. During $B(j^\star)$ of job $j^\star \in \mathcal{C}_c(j)$, j only admits jobs k of *higher* classes, i.e., $k \in \mathcal{C}_{c'}(j)$ for $c' > c$. Particularly, j only admits job $j^\star \in \mathcal{C}_c(j)$ if the blocking period of the last job in $\mathcal{C}_c(j)$ admitted to the same machine has completed.

Hence, when job j decides if it admits the currently shortest available job $j^\star$ at time τ, it makes sure that $j^\star$ indeed belongs to a class $\mathcal{C}_c(j)$ and that no higher class $c' \geq c$ is blocking τ, i.e., it checks that $\tau \notin B(k)$ for all jobs $k \in \mathcal{C}_{c'}(j)$ admitted to the same machine. In this case, we say that $j^\star$ is a *child* of j and that j is the *parent* of $j^\star$, denoted by $\pi(j^\star) = j$. If job $j^\star$ is admitted at time τ by job j, the algorithm sets $a_{j^\star} = \tau$ and $e_{j^\star} = a_{j^\star} + (1+\delta)p_{j^\star}$ and assigns the scheduling interval $S(j^\star) = [a_{j^\star}, e_{j^\star})$ to $j^\star$.

If $e_{j^\star} \leq e_j$, the routine sets $f_{j^\star} = \min\{e_j, e_{j^\star} + \beta p_{j^\star}\}$ which determines $B(j^\star) = [e_{j^\star}, f_{j^\star})$. As the scheduling and blocking periods of children k of j are supposed to be disjoint, we have to **update the blocking periods**. First consider the job $k \in \mathcal{C}_{c'}(j)$ for $c' < c$ admitted to the same machine whose blocking period contains τ (if it exists), and let $[e'_k, f'_k)$ be the maximal interval of $B(k)$ containing τ. We set $f''_k = \min\{e_j, f'_k + (1+\delta+\beta)p_{j^\star}\}$ and replace the interval $[e'_k, f'_k)$ by $[e'_k, \tau) \cup [\tau + (1+\delta+\beta)p_{j^\star}, f''_k)$. For all other jobs $k \in \mathcal{C}_{c'}(j)$ with $B(k) \cap [\tau, \infty) \neq \emptyset$ admitted to the same machine, we replace the remaining part of their

blocking period $[e'_k, f'_k)$ by $[e'_k + (1+\delta+\beta)p_{j^\star}, f''_k)$ where $f''_k := \min\{e_j, f'_k + (1+\delta+\beta)p_{j^\star}\}$. In this update we follow the convention $[e, f) = \emptyset$ if $f \leq e$. Observe that the length of the blocking period might decrease due to such updates.

Note that $e_{j^\star} > e_j$ is also possible as j does not take the end of its own scheduling interval e_j into account when admitting jobs. Thus, the scheduling interval of $j^\star$ would end outside j's scheduling interval and inside j's blocking period. During $B(j)$, $\pi(j)$, the parent of j, did not allocate the interval $[e_j, e_{j^\star})$ for completing jobs admitted by j but for ensuring its own completion. Hence, the completion of both $j^\star$ and $\pi(j)$ is not necessarily guaranteed anymore. To prevent this, we **modify all scheduling intervals** $S(k)$ (including $S(j)$) of jobs admitted to the same machine that contain time τ and the corresponding blocking periods $B(k)$. For each job k admitted to the same machine with $\tau \in S(k)$ (i.e., including j) and $e_{j^\star} > e_k$ we set $e_k = e_{j^\star}$. We also update their blocking periods (in fact, single intervals) to reflect their new starting points. If the parent $\pi(k)$ of k does not exist, $B(k)$ remains empty; otherwise we set $B(k) := [e_k, f_k)$ where $f_k = \min\{e_{\pi(k)}, e_k + \beta p_k\}$. Note that, after this update, the blocking intervals of any but the largest such job will be empty. Moreover, the just admitted job $j^\star$ does not get a blocking period in this special case.

During the analysis of the algorithm, we show that any admitted job j still completes before $a_j + (1+\delta)p_j$ and that $e_j \leq a_j + (1+2\delta)p_j$ holds in retrospective for all admitted jobs j. Thus, any job j that admits another job $j^\star$ tentatively assigns this job a scheduling interval of length $(1+\delta)p_{j^\star}$ but, for ensuring j's completion, it is prepared for losing $(1+2\delta)p_{j^\star}$ time units of its scheduling interval $S(j)$. We summarize the blocking algorithm in Algorithm 1.

Algorithm 1 Blocking algorithm.

```
Scheduling routine: At all time τ and on all machines i, run the job with shortest
    processing time that has been admitted to i and has not yet completed
Event: Upon release of a new job at time τ:
    Call admission routine.
Event: Upon ending of a blocking period or scheduling interval at time τ:
    Call admission routine.
Admission routine:
j* ← a shortest available job at τ, i.e.,
    j* ∈ arg min{p_j | r_j ≤ τ and d_j − τ ≥ (1 + δ)p_j}
i ← 1
while j* is not admitted and i ≤ m do
    K ← the set of jobs on machine i whose scheduling intervals contain τ
    if K = ∅
        1. admit job j* to machine i, a_{j*} ← τ, e_{j*} ← a_{j*} + (1 + δ)p_{j*}, and f_{j*} ← e_{j*}
        2. call admission routine
    else
        j ← arg min{p_k | k ∈ K}
        if j* ∈ C_c(j) and there exists no c' ≥ c with t ∈ B(j') for j' ∈ C_{c'}(j), then
            1. admit job j* to machine i, a_{j*} ← τ and e_{j*} ← a_{j*} + (1 + δ)p_{j*}
               if e_{j*} ≤ e_j, then
                   f_{j*} ← min{e_j, e_{j*} + βp_{j*}}
                   set S(j*) ← [a_{j*}, e_{j*}) and B(j*) ← [e_{j*}, f_{j*})
               else
                   set e_j ← e_{j*} and f_{j*} ← e_{j*}
                   modify S(k) and B(k) for k ∈ K
            2. update B(k) for k ∈ C_{c'}(j) admitted to machine i with c' < c and
                 B(k) ∩ [τ, ∞) ≠ ∅
            3. call admission routine
        else
            i ← i + 1
```

Roadmap for the analysis

During the analysis, it is sufficient to concentrate on instances with small slack, as also noted in [8]. For $\varepsilon > 1$ we run the blocking algorithm with $\varepsilon = 1$, which only tightens the commitment requirement, and obtain constant competitive ratios. Thus, we assume $0 < \varepsilon \leq 1$. Moreover, in the δ-commitment model, committing to the completion of a job j at an earlier point in time clearly satisfies committing at a remaining slack of δp_j. Therefore, we may assume $\delta \in [\frac{\varepsilon}{2}, \varepsilon)$.

The blocking algorithm does not migrate any job. In the analysis, we first compare the throughput of our algorithm to the solution of an optimal non-migratory schedule. We then use a well-known result by Kalyanasundaram and Pruhs to compare this to an optimal solution that may exploit migration. Here, ω_m is the maximal ratio of the throughput of an optimal migratory schedule to the throughput of an optimal non-migratory schedule [19].

▶ **Theorem 3** (Theorem 1.1 in [19]). *$\omega_m \leq (6m - 5)/m$.*

The proof of our results consists of two parts. In the first part, Section 3, we show that the blocking algorithm completes all admitted jobs on time. The second part, Section 4, is to show that the blocking algorithm admits sufficiently many jobs to be competitive.

3 Completing all admitted jobs on time

We show that the blocking algorithm finishes every admitted job on time in Theorem 5. Our choice of parameters guarantees that Inequality (1) is satisfied.

As the blocking algorithm does not migrate jobs, it suffices to consider each machine individually in this section. The proof relies on the following observations: (i) the sizes of admitted jobs belonging to different classes of job j are geometrically decreasing, (ii) the scheduling intervals of jobs are completely contained in the scheduling intervals of their parents, and (iii) scheduling in Shortest Processing Time order guarantees that job j has highest priority in the blocking periods of its children. We start by proving the following technical lemma about the length of the final scheduling interval of an admitted job j. In the proof we use that $\pi(k) = j$ for two jobs j and k implies that $p_k < \gamma p_j$.

▶ **Lemma 4.** *Let $0 < \delta < \varepsilon$ be fixed. If $\gamma > 0$ satisfies $(1 + 2\delta)\gamma \leq \delta$, then the length of the scheduling interval $S(j)$ of an admitted job j is upper bounded by $(1 + 2\delta)p_j$. Moreover, $S(j)$ contains the scheduling intervals of all descendants of j.*

Proof. By definition of the blocking algorithm, the end point e_j of the scheduling interval of job j is only modified when j or one of j's descendants admits another job. Let us consider such a case: If job j admits a job $j^\star$ whose scheduling interval does not fit the scheduling interval of j, we set $e_j = e_{j^\star} = a_{j^\star} + (1+\delta)p_{j^\star}$ to accommodate the scheduling interval $S(j^\star)$ within $S(j)$. The same modification is applied to any ancestor k of j with $e_k < e_{j^\star}$. This implies that, after such a modification of the scheduling interval, neither j nor any affected ancestors k of j are the smallest jobs in their scheduling intervals anymore. In particular, no job whose scheduling interval was modified in such a case at time τ is able to admit jobs after τ. Hence, any job j can only admit other jobs within the interval $[a_j, a_j + (1 + \delta)p_j)$. In particular, $a_{j^\star} \leq a_j + (1 + \delta)p_j$ for any job $j^\star$ with $\pi(j^\star) = j$.

Thus, by induction, it is sufficient to show that $a_{j^\star} + (1 + 2\delta)p_{j^\star} \leq a_j + (1 + 2\delta)p_j$ for admitted jobs $j^\star$ and j with $\pi(j^\star) = j$ in order to prove the lemma. Note that $\pi(j^\star) = j$ implies $p_{j^\star} < \gamma p_j$. Thus,

$$a_{j^\star} + (1 + 2\delta)p_{j^\star} \leq (a_j + (1 + \delta)p_j) + (1 + 2\delta)\gamma p_j \leq a_j + (1 + 2\delta)p_j,$$

where the last inequality follows from the assumption $(1 + 2\delta)\gamma \leq \delta$. ◀

▶ **Theorem 5.** *Let $0 < \delta < \varepsilon$ be fixed. If $0 < \gamma < 1$ and $\beta \geq 1$ satisfy*

$$\frac{\beta/2}{\beta/2 + (1+2\delta)}\left(1 + \delta - 2(1+2\delta)\gamma\right) \geq 1, \tag{1}$$

then the blocking algorithm completes a job j admitted at $a_j \leq d_j - (1+\delta)p_j$ on time.

Our choice of parameters guarantees that Inequality (1) is satisfied.

Proof. Let j be a job admitted by the blocking algorithm with $a_j \leq d_j - (1+\delta)p_j$. Hence, showing that a job j completes before time $d'_j := a_j + (1+\delta)p_j$ proves the theorem. Due to scheduling in SPT order, each job j has highest priority in its own scheduling interval if the time point does not belong to the scheduling interval of a descendant of j. Thus, it suffices to show that at most δp_j units of time in $[a_j, d'_j)$ belong to scheduling intervals $S(k)$ of descendants of j. By Lemma 4, the scheduling intervals of any descendant k' of a child k of j is contained in $S(k)$. Hence, it is sufficient to only consider K, the set of children of j. In order to bound the contribution of each child $k \in K$, we partition K into two sets. The first set K_1 contains all children of j that where admitted as the first jobs in their class $\mathcal{C}_c(j)$. The set K_2 contains the remaining jobs.

We start with K_2. Consider a job $k \in \mathcal{C}_c(j)$ admitted by j. By Lemma 4, we know that $|S(k)| = (1+\mu\delta)p_k$ where $1 \leq \mu \leq 2$. Let $k' \in \mathcal{C}_c(j)$ be the previous job admitted by j in class c. Then, $B(k') \subseteq [e_{k'}, e_k)$. Since scheduling and blocking periods of children of j are always disjoint, j had highest scheduling priority in $B(k')$. Hence, during $B(k') \cup S(k)$ job j was processed for at least $|B(k')|$ units of time. In other words, j was processed for at least a $\frac{|B(k')|}{|B(k') \cup S(k)|}$-fraction of $B(k') \cup S(k)$. We can rewrite this ratio by

$$\frac{|B(k')|}{|B(k') \cup S(k)|} = \frac{\beta p_{k'}}{\beta p_{k'} + (1+\mu\delta)p_k} = \frac{\nu\beta}{\nu\beta + (1+\mu\delta)},$$

where $\nu := \frac{p_{k'}}{p_k} \in (\frac{1}{2}, 2]$. By differentiating with respect to ν and μ, we observe that the last term is increasing in ν and decreasing in μ. Thus, we can lower bound this expression by

$$\frac{|B(k')|}{|B(k') \cup S(k)|} \geq \frac{\beta/2}{\beta/2 + (1+2\delta)}.$$

Therefore, j was processed for at least a $\frac{\beta/2}{\beta/2+(1+2\delta)}$-fraction in $\bigcup_{k\in K} B(k) \cup \bigcup_{k\in K_2} S(k)$.

We now consider the set K_1. The total processing volume of these jobs is bounded by $\sum_{c=0}^{\infty} \frac{\gamma}{2^c} p_j = 2\gamma p_j$. By Lemma 4, we know that $|S(k)| \leq (1+2\delta)p_k$. Combining these two observations, we obtain $\left|\bigcup_{k\in K_1} S(k)\right| \leq 2(1+2\delta)\gamma p_j$. Combining the latter with the bound for K_2, we conclude that j is scheduled for at least

$$\left|[a_j, d'_j) \setminus \bigcup_{k\in K} S(k)\right| \geq \frac{\beta/2}{\beta/2 + (1+2\delta)}\big((1+\delta) - 2(1+2\delta)\gamma\big)p_j \geq p_j$$

units of time, where the last inequality follows from Equation (1). Thus, j completes before $d'_j = a_j + (1+\delta)p_j \leq d_j$. ◀

4 Admitting sufficiently many jobs

After proving that the blocking algorithm completes all admitted jobs on time, we show that the blocking algorithm admits enough jobs to achieve the competitive ratio of Theorem 1.

4.1 Key lemma on the size of non-admitted jobs

For the proof of the main result in this section, we rely on the following strong, structural lemma about the volume processed by a feasible non-migratory schedule in some time interval and the size of jobs admitted by a certain class of online algorithms in the same time interval. Let σ be a feasible non-migratory schedule. Let ALG be a non-migratory online algorithm satisfying the following two properties: (i) ALG never admits a job j later than $d_j - (1+\delta)p_j$ for $0 < \delta < \varepsilon$ and (ii) retrospectively, for each time τ, there is a threshold $u_\tau \in (0, \infty]$ such that any available job j with $d_j - \tau \geq (1+\delta)p_j$ that was not admitted by ALG at τ satisfies $p_j \geq u_\tau$. We will show that our blocking algorithm satisfies (i) and (ii) for a non-trivial u_τ that allows us to bound the volume of any feasible schedule.

Without loss of generality, we assume that σ completes all jobs on time that it started. Let X^σ be the jobs completed by σ and not admitted by ALG. For $1 \leq i \leq m$, let X_i^σ be the jobs in X^σ processed by machine i. Let C_x be the completion time of job $x \in X^\sigma$ in σ.

▶ **Lemma 6.** *Let σ and ALG be defined as above. Let $0 \leq \zeta_1 \leq \zeta_2$ and fix $x \in X_i^\sigma$ as well as $Y \subset X_i^\sigma \setminus \{x\}$. If*

(R) $r_x \geq \zeta_1$ as well as $r_y \geq \zeta_1$ for all $y \in Y$,
(C) $C_x \geq C_y$ for all $y \in Y$, and
(P) $\sum_{y \in Y} p_y \geq \frac{\varepsilon}{\varepsilon - \delta}(\zeta_2 - \zeta_1)$

hold, then $p_x \geq u_{\zeta_2}$ where u_{ζ_2} is the upper bound imposed by ALG at time ζ_2. In particular, if $u_{\zeta_2} = \infty$, then no such job x exists.

Proof. We show the lemma by contradiction. More precisely, we show that, if $p_x < u_{\zeta_2}$, the schedule σ cannot complete x on time and, hence, is not feasible.

Remember that $x \in X_i^\sigma$ implies that ALG did not admit job x at any point τ. At time ζ_2, there are two possible reasons why x was not admitted: $p_x \geq u_{\zeta_2}$ or $d_x - \zeta_2 < (1+\delta)p_x$. In case of the former, the statement of the lemma holds. Thus, let us assume $p_x < u_{\zeta_2}$ and, therefore, $d_x - \zeta_2 < (1+\delta)p_x$ has to hold. As job x arrived with a slack of at least εp_x at its release date r_x and $r_x \geq \zeta_1$ by assumption, we have

$$\zeta_2 - \zeta_1 \geq \zeta_2 - d_x + d_x - r_x > -(1+\delta)p_x + (1+\varepsilon)p_x = (\varepsilon - \delta)p_x. \tag{2}$$

As all jobs in Y complete earlier than x by Assumption (C) and are only released after ζ_1 by (R), the volume processed by σ in $[\zeta_1, C_x)$ on machine i is greater than $\frac{\varepsilon}{\varepsilon - \delta}(\zeta_2 - \zeta_1) + p_x$ by (P). Moreover, σ can process at most a volume of $(\zeta_2 - \zeta_1)$ on machine i in $[\zeta_1, \zeta_2)$. These two bounds imply that σ has to process job parts with a processing volume of at least

$$\frac{\varepsilon}{\varepsilon - \delta}(\zeta_2 - \zeta_1) + p_x - (\zeta_2 - \zeta_1) > \frac{\delta}{\varepsilon - \delta}(\varepsilon - \delta)p_x + p_x = (1+\delta)p_x$$

in $[\zeta_2, C_x)$, where the inequality follows using Inequality (2). Thus, $C_x > \zeta_2 + (1+\delta)p_x > d_x$ which contradicts the feasibility of σ.

Observe that the online algorithm ALG admits the shortest available job that satisfies $p_j \leq u_\tau$. In particular, if $u_\tau = \infty$ for some time point τ, ALG admits the shortest job if there is one available. Hence, for $0 \leq \zeta_1 \leq \zeta_2$ with $u_{\zeta_2} = \infty$, there does not exist a job $x \in X_i^\sigma$ and a set $Y \subset X_i^\sigma \setminus \{x\}$ satisfying (R), (C), and (P) for any machine i. ◀

4.2 Admitting sufficiently many jobs

▶ **Theorem 7.** *An optimal non-migratory (offline) algorithm can complete at most a factor $\alpha + 4$ more jobs on time than admitted by the blocking algorithm where $\alpha := \frac{\varepsilon}{\varepsilon - \delta}\left(2\beta + \frac{1+2\delta}{\gamma}\right)$.*

For proving the theorem, we fix an instance and an optimal offline algorithm Opt. Let X be the jobs that Opt scheduled and the blocking algorithm did not admit. We assume without loss of generality that Opt completes all jobs in X on time. Let J be the jobs that the blocking algorithm scheduled. Then, $X \cup J$ clearly is a superset of the jobs that Opt scheduled. Hence, to show the theorem it is sufficient to prove that $|X| \leq (\alpha+3)|J|$. Let $\overline{X} \subseteq X$ be the jobs scheduled on the machine with the *highest* throughput and let $\underline{J} \subseteq J$ be the jobs scheduled on the machine with the *lowest* throughput. In Lemma 11 we develop a charging scheme of $\overline{X}$ to jobs in $\underline{J}$ such that no job gets charged more than $\alpha+3$ jobs.

Without loss of generality, we assume that the union of all scheduling intervals of jobs in J, i.e., $\bigcup_{j\in J} S(j)$, forms one interval. If this assumption does not hold, we consider each maximal interval in $\bigcup_{j\in J} S(j)$ separately. Instead of directly charging the jobs in $\overline{X}$ to jobs in $\underline{J}$ we take a detour and charge jobs in $\overline{X}$ to intervals that cover $\bigcup_{j\in J} S(j)$. The idea behind our charging scheme is that Opt is not able to schedule arbitrarily many jobs during a scheduling interval or a blocking period created by the blocking algorithm. Intuitively, jobs that were released during a scheduling interval or a blocking period and not admitted by the algorithm have to satisfy certain lower bounds on their processing times. Thus, the charging scheme relies on the release date r_x and the size p_x of a job $x \in \overline{X}$ as well as on the precise structure of the intervals created by the blocking algorithm. The number of jobs we charge to one interval will depend on the relative length of the interval.

We retrospectively consider the interval structure created by the algorithm on the machine that schedules $\underline{J}$; let this w.l.o.g. be the first machine. Let T be the set of all time points corresponding to the admission of a new job, the end of a scheduling interval, and the start as well as the end of a blocking period of jobs in $\underline{J}$. Index the elements in T by their actual value, i.e., $\tau_1 < \tau_2 < \ldots < \tau_{|T|}$. Let $\mathcal{I}$ be the set of intervals of the form $I_t := [\tau_t, \tau_{t+1})$ for $1 \leq t < |T|$. The next lemma holds as the admission of a job adds at most three time points.

▶ **Fact 8.** *The set $\mathcal{I}$ contains at most $3|\underline{J}|$ intervals.*

For analyzing the competitive ratio of our algorithm, we first charge jobs $x \in \overline{X}$ to intervals $I_t \in \mathcal{I}$ and then assign this subset to the job that was "responsible" for not admitting other jobs during I_t because of its scheduling interval or because of its blocking period. In Lemma 11, we show that a job j is assigned at most $\frac{\varepsilon}{\varepsilon-\delta}(2\beta + \frac{1+2\delta}{\gamma}) = \alpha$ jobs and that each interval I_t gets at most one job. Fact 8 bounds the number of intervals in $\mathcal{I}$. Combining these observations then proves that $|\overline{X}| \leq (\alpha+3)|\underline{J}|$ and, thus, $|X| \leq (\alpha+3)|J|$.

Consider a time point $\tau \in \bigcup_{j\in J} S(j)$. Let $j \in J$ be the shortest job on machine i such that $\tau \in S(j) \cup B(j)$. The blocking algorithm only admits an available job k to machine i in two cases: (i) $\tau \in S(j)$ and $p_k < \gamma p_j$ or (ii) $\tau \in B(j)$ and k belongs to a smaller class of the parent of j. Condition (ii) clearly is satisfied if $p_k < p_j/2$. This implies that at any time τ the blocking algorithm maintains a threshold $u_{\tau,i}$ for each machine i so that only available jobs smaller than this threshold qualify for admission to machine i. Note that the admission of a job k at time τ to machine i decreases the threshold $u_{\tau,i}$. If τ does not belong to a scheduling interval of a job on machine i, we set $u_{\tau,i} = \infty$. By taking the maximum of these upper bounds, we obtain a time-dependent threshold u_τ that guides the admission decisions of the blocking algorithm. Hence, the conditions of Lemma 6 are met by the our algorithm.

Note that these upper bounds only change when a scheduling interval starts or ends, or when an interval belonging to a blocking period starts or ends. For an interval $I_t \in \mathcal{I}$ we define $\underline{u_t}$ as the threshold on the machine with the lowest throughput, i.e., $\underline{u_t} := u_{\tau_{t,1}} \in (0, \infty]$. If $\underline{u_t} = \infty$, then the interval $I_t \in \mathcal{I}$ does not belong to the scheduling interval of a job in J and $u_\tau = \infty$ for all $\tau \in I_t$. Then the next lemma holds.

▶ **Fact 9.** *In every interval $I_t = [\tau_t, \tau_{t+1}) \in \mathcal{I}$ the upper bound u_τ created by the blocking algorithm is lower bounded by $u_{\tau_t,1}$, i.e., $u_\tau \geq \underline{u_t}$ for all $\tau \in I_t$.*

The charging scheme developed in Lemma 11 is based on a careful modification of the following partition $(F_t)_{1\leq t<|T|}$ of the set $\overline{X}$. Fix an interval $I_t \in \mathcal{I}$ and define the set $F_t \subset \overline{X}$ as the set that contains all jobs $x \in \overline{X}$ released during I_t, i.e., $F_t := \{x \in \overline{X} : r_x \in I_t\}$. As, upon release, each job is available, the next corollary directly follows from Fact 9.

▶ **Fact 10.** *For all jobs $x \in F_t$ it holds $p_x \geq \underline{u_t}$. In particular, if $\underline{u_t} = \infty$, then $F_t = \emptyset$.*

In fact, the charging scheme maintains this property and only assigns jobs in $\overline{X}$ to intervals I_t if $p_x \geq \underline{u_t}$. In particular, no job will be assigned to an interval with $\underline{u_t} = \infty$.

We now formalize how many jobs in $\overline{X}$ we will assign to a specific interval I_t. Let $\varphi_t := \left\lfloor \frac{\varepsilon}{\varepsilon-\delta} \frac{\tau_{t+1}-\tau_t}{\underline{u_t}} \right\rfloor + 1$ be the *target number* of I_t if $\underline{u_t} < \infty$ and $\varphi_t = 0$ if $\underline{u_t} = \infty$.

If $\underline{u_t} < \infty$, let $j_t \in \underline{J}$ be the *smallest* job with $\tau_t \in S(j) \cup B(j)$. Except for one job per interval $I_t \in \mathcal{I}$ which remains assigned to I_t, the jobs assigned to I_t will be accounted for by j_t. Suppose that each of the sets F_t satisfies $|F_t| \leq \varphi_t$. Then, at most $\frac{\varepsilon}{\varepsilon-\delta} \frac{\tau_{t+1}-\tau_t}{\underline{u_t}}$ will be charged to job j_t because of interval I_t. By definition of $\underline{u_t}$, we have $\underline{u_t} \geq \gamma p_{j_t}$ if $I_t \subseteq S(j_t)$ and, if $I_t \subseteq B(j_t)$, we have $\underline{u_t} \geq p_{j_t}/2$. The total length of intervals I_t for which $j = j_t$ holds sums up to at most $(1+2\delta)p_j$ for $I_t \subseteq S(j)$ and to at most $2\beta p_j$ for $I_t \subseteq B(j)$. Hence, in total, the charging scheme assigns at most $\frac{\varepsilon}{\varepsilon-\delta}(2\beta + \frac{1+2\delta}{\gamma}) = \alpha$ jobs in $\overline{X}$ to job $j \in \underline{J}$. In combination with Fact 8, that bounds the number of intervals in $\mathcal{I}$, this would imply Theorem 7. In general, $|F_t| \leq \varphi_t$ does not have to be true as Opt may preempt jobs and process the parts during several intervals I_t. In the remainder of this section, we show that there exists another partition $(G_t)_{1\leq t<|T|}$ of the jobs in $\overline{X}$ such that $|G_t| \leq \varphi_t$ holds.

▶ **Lemma 11.** $|\overline{X}| \leq \alpha|\underline{J}| + |\mathcal{I}|$.

Proof. As observed before it suffices to show that there is a partition $\mathcal{G} = \left(G_t\right)_{1\leq t<|T|}$ such that $|G_t| \leq \varphi_t$ and $\bigcup_{1\leq t<|T|} G_t = \overline{X}$ in order to prove the lemma. The high-level idea of this proof is the following: Consider an interval $I_t = [\tau_t, \tau_{t+1})$. If F_t does not contain too many jobs, i.e., $|F_t| \leq \varphi_t$, we would like to set $G_t = F_t$. Otherwise, we find a later interval $I_{t'}$ with $|F_{t'}| < \varphi_{t'}$ such that we can assign the excess jobs in F_t to $I_{t'}$.

In order to repeatedly apply Lemma 6, we only assign such excess jobs $x \in F_t$ to $G_{t'}$ if their processing time is at least the threshold of $I_{t'}$, i.e., $p_x \geq \underline{u_{t'}}$. Then, by our choice of parameters, a set $G_{t'}$ with $\varphi_{t'}$ many jobs of size at least $\underline{u_{t'}}$ "covers" the interval $I_{t'} = [\tau_{t'}, \tau_{t'+1})$ as often as required by (P) in Lemma 6, i.e.,

$$\sum_{x\in G_{t'}} p_x \geq \varphi_{t'} \cdot \underline{u_{t'}} = \left(\left\lfloor \frac{\varepsilon}{\varepsilon-\delta} \frac{\tau_{t'+1}-\tau_{t'}}{\underline{u_{t'}}} \right\rfloor + 1 \right) \cdot \underline{u_{t'}} \geq \frac{\varepsilon}{\varepsilon-\delta}(\tau_{t'+1}-\tau_{t'}). \tag{3}$$

The proof consists of two parts: the first one is to inductively (on t) construct the partition $\mathcal{G} = \left(G_t\right)_{1\leq t<|T|}$ of $\overline{X}$ with $|G_t| \leq \varphi_t$. The second one is the proof that a job $x \in G_t$ satisfies $p_x \geq \underline{u_t}$. During the construction of $\mathcal{G}$ we define temporary sets $A_t \subset \overline{X}$ for intervals I_t. The set G_t is chosen as a subset of $F_t \cup A_t$ of appropriate size. In order to apply Lemma 6 to each job in A_t individually, alongside A_t, we construct a set $Y_{x,t}$ and a time $\tau_{x,t} \leq r_x$ for each job $x \in \overline{X}$ that is added to A_t. Let C^*_x be the completion time of some job $x \in \overline{X}$ in the optimal schedule Opt. The second part of the proof is to show that x, $\tau_{x,t}$, and $Y_{x,t}$ satisfy

(R) $r_y \geq \tau_{x,t}$ for all $y \in Y_{x,t}$,
(C) $C^*_x \geq C^*_y$ for all $y \in Y_{x,t}$, and
(P) $\sum_{y\in Y_{x,t}} p_y \geq \frac{\varepsilon}{\varepsilon-\delta}(\tau_t - \tau_{x,t})$.

Then, x, $Y = Y_{x,t}$, $\zeta_1 = \tau_{x,t}$, and $\zeta_2 = \tau_t$ satisfy the conditions of Lemma 6 and we can deduce that the processing time of x is at least the threshold at time τ_t, i.e., $p_x \geq u_{\tau_t} \geq \underline{u_t}$.

Constructing $\mathcal{G} = (G_t)_{1 \leq t \leq |T|}$. We inductively construct the sets G_t in the order defined by their indices. For simplicity, we add a singleton as last interval, i.e., $I_{|T|} = \{\tau_{|T|}\}$ with $\varphi_{|T|} = 0$. We start by setting $A_t = \emptyset$ for all intervals $1 \leq t \leq |T|$. We define $Y_{x,t} = \emptyset$ for each job $x \in \overline{X}$ and each interval I_t. The preliminary value of the time $\tau_{x,t}$ is the minimum of the start point τ_t of the interval I_t and the release date r_x of x, i.e., $\tau_{x,t} := \min\{\tau_t, r_x\}$. We refer by *step* t to the step in the construction where G_t was defined.

Starting with $t = 1$, let I_t be the next interval to consider during the construction. Depending on the cardinality of $F_t \cup A_t$, we have to distinguish two cases. If $|F_t \cup A_t| \leq \varphi_t$, we set $G_t = F_t \cup A_t$.

If $|F_t \cup A_t| > \varphi_t$, we order the jobs in $F_t \cup A_t$ in increasing order of completion times in Opt. The first φ_t jobs are assigned to G_t while the remaining $|F_t \cup A_t| - \varphi_t$ jobs are added to A_{t+1}. In this case, we might have to redefine the times $\tau_{x,t+1}$ and the sets $Y_{x,t+1}$ for the jobs x that were newly added to A_{t+1}. Fix such a job x. If there is no job z in the just defined set G_t that has a smaller release date than $\tau_{x,t}$, we set $\tau_{x,t+1} = \tau_{x,t}$ and $Y_{x,t+1} = Y_{x,t} \cup G_t$. Otherwise let $z \in G_t$ be a job with $r_z < \tau_{x,t}$ that has the smallest time $\tau_{z,t}$. We set $\tau_{x,t+1} = \tau_{z,t}$ and $Y_{x,t+1} = Y_{z,t} \cup G_t$.

Finally, we also construct $G_{|T|}$ this way. As we will show that $p_x \geq \underline{u}_{|T|}$ for all $x \in G_{|T|}$, we will get that $G_{|T|} = \emptyset$ (since $\underline{u}_{|T|} = \infty$) and therefore $G_{|T|} \leq \varphi_{|T|} = 0$.

Bounding the size of the jobs in G_t. We consider the intervals again in increasing order of their indices and show by induction that any job x in G_t satisfies $p_x \geq \underline{u_t}$ which implies $G_t = \emptyset$ if $\underline{u_t} = \infty$. Clearly, if $x \in F_t \cap G_t$, Fact 10 guarantees $p_x \geq \underline{u_t}$. Hence, in order to show the lower bound on the processing time of $x \in G_t$, it is sufficient to consider jobs in $G_t \setminus F_t \subset A_t$. To this end, we show that for such jobs (R), (C), and (P) are satisfied. Then, Lemma 6 guarantees that $p_x \geq u_{\tau_t} \geq \underline{u_t}$. Therefore, $A_t = \emptyset$ if $\underline{u_t} = \infty$ as the global bound is also unbounded, i.e., $u_{\tau_t} \geq \underline{u_t} = \infty$, by Fact 9.

By construction, $A_1 = \emptyset$. Hence, (R), (C), and (P) are satisfied for each job $x \in A_1$.

Assume that the conditions (R), (C), and (P) are satisfied for all $x \in A_t$ for all $1 \leq t < s$. Hence, for $t < s$, the set G_t only contains jobs x with $p_x \geq \underline{u_t}$. Let $t \geq s$ be the first index with $A_t \neq \emptyset$ and fix $x \in A_t$. We want to show that $p_x \geq \underline{u_t}$. By induction and by Fact 10, $p_y \geq \underline{u_{t-1}}$ holds for all $y \in G_{t-1}$. Because x did not fit in G_{t-1}, $|G_{t-1}| = \varphi_{t-1}$.

We distinguish two cases based on the jobs in G_{t-1}. If there is no $z \in G_{t-1}$ with $r_z < \tau_{x,t-1}$, then $\tau_{x,t} = \tau_{x,t-1}$, and (R) and (C) are satisfied by construction and by induction. For (P), consider

$$\begin{aligned}\sum_{y \in Y_{x,t}} p_y &= \sum_{y \in Y_{x,t-1}} p_y + \sum_{y \in G_{t-1}} p_y \geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{x,t-1}) + \underline{u_{t-1}} \cdot \varphi_{t-1} \\ &> \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{x,t-1}) + \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{t-1}) = \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{x,t})\,,\end{aligned}$$

where the first inequality holds by induction.

If there is a job $z \in G_{t-1}$ with $r_z < \tau_{x,t-1} \leq \tau_{t-1}$, then $z \in A_{t-1}$. In step t, we chose z with minimal $\tau_{z,t-1}$. Thus, $r_y \geq \tau_{y,t-1} \geq \tau_{z,t-1}$ for all $y \in G_{t-1}$ and $r_x \geq \tau_{x,t-1} > r_z \geq \tau_{z,t-1}$. Moreover, by induction, $r_y \geq \tau_{z,t-1}$ holds for all $y \in Y_{z,t-1}$. Thus, $\tau_{x,t}$ and $Y_{x,t}$ satisfy (R). For (C), consider that $C^*_x \geq C^*_y$ for all $y \in G_{t-1}$ by construction and, thus, $C^*_x \geq C^*_z \geq C^*_y$ also holds for all $y \in Y_{z,t-1}$. For (P), observe that

$$\sum_{y\in Y_{x,t}} p_y = \sum_{y\in Y_{z,t-1}} p_y + \sum_{y\in G_{t-1}} p_y \geq \frac{\varepsilon}{\varepsilon-\delta}(\tau_{t-1}-\tau_{z,t-1}) + \underline{u_{t-1}}\cdot\varphi_{t-1}$$
$$\geq \frac{\varepsilon}{\varepsilon-\delta}(\tau_{t-1}-\tau_{z,t-1}) + \frac{\varepsilon}{\varepsilon-\delta}(\tau_t-\tau_{t-1}) \geq \frac{\varepsilon}{\varepsilon-\delta}(\tau_t-\tau_{x,t}).$$

Here, the first inequality follows by induction and the second by definition of $\underline{u_{t-1}}$ and φ_{t-1}. Hence, Lemma 6 implies $p_x \geq u_{\tau_t} \geq \underline{u_t}$.

Showing $|\overline{X}| \leq \alpha|\underline{J}| + |\mathcal{I}|$. By construction, we know that $\bigcup_{t=1}^{|T|} G_t = \overline{X}$. We start with considering intervals I_t with $\underline{u_t} = \infty$. Then, I_t is not covered by a scheduling interval on machine i. Thus, $u_\tau = \infty$ for all $\tau \in I_t$ and $F_t = \emptyset$ by Fact 10. In the previous part we have seen that the conditions for Lemma 6 are satisfied. Hence, $G_t = \emptyset$ if $\underline{u_t} = \infty$. For t with $\underline{u_t} < \infty$, we have $|G_t| = \varphi_t = \left\lfloor \frac{\varepsilon}{\varepsilon-\delta}\frac{\tau_{t+1}-\tau_t}{\underline{u_t}} \right\rfloor + 1$. As explained before, we assign these jobs (except the additive one) to j_t, the shortest job in $\underline{J}$ with $I_t \subseteq S(j) \cup B(j)$, without assigning more than α jobs in $\overline{X}$ to a particular job in $\underline{J}$. Hence, the number of jobs in $\overline{X}$ is indeed bounded by $\alpha|\underline{J}| + |\mathcal{I}|$. ◀

Proof of Theorem 7. As discussed before, the union $X \cup J$ of X, the jobs only scheduled by Opt, and J, the jobs admitted by the blocking algorithm, is a superset of the jobs that Opt completed. Lemma 11 shows that $|\overline{X}| \leq \alpha|\underline{J}| + |\mathcal{I}|$.Combining this with the bound on $\mathcal{I}$ given in Fact 8, we conclude

$$\textsc{Opt} \leq m\cdot|\overline{X}| + |J| \leq m\,(\alpha+3)\,|\underline{J}| + |J| \leq (\alpha+4)\,|J|.$$ ◀

Proof of Theorem 1. In Theorem 5 we show that the blocking algorithm completes all admitted jobs J on time. This implies that the blocking algorithm is feasible for the model commitment upon admission. As no job $j \in J$ is admitted later than $d_j - (1+\delta)p_j$, this shows that the blocking algorithm also solves scheduling with δ-commitment. Theorem 1.1 in [19] (Theorem 3) gives a bound on the optimal migratory schedule in terms of an optimal non-migratory solution. In Theorem 7, we bound an optimal non-migratory solution Opt by $|J|$, the throughput of the blocking algorithm. Combining these theorems shows that the blocking algorithm achieves a competitive ratio of $c = 6(\alpha+4) = 6\left(\frac{\varepsilon}{\varepsilon-\delta}\left(2\beta + \frac{1+2\delta}{\gamma}\right) + 4\right)$. Our choice of parameters $\beta = \frac{16}{\delta}$ and $\gamma = \frac{\delta}{16}$ implies $c \in \mathcal{O}\left(\frac{\varepsilon}{(\varepsilon-\delta)\delta}\right)$. In the case where $\delta \leq \varepsilon/2$, we run the algorithm with parameter $\delta' = \varepsilon/2$. Hence, $c \in \mathcal{O}\left(\frac{1}{\varepsilon-\delta'}\right) = \mathcal{O}\left(\frac{1}{\varepsilon}\right)$. If $\delta > \varepsilon/2$, then we set $\delta' = \delta$ in our algorithm. Thus, $\frac{\varepsilon}{\delta'} \in \mathcal{O}(1)$ and, again, $c \in \mathcal{O}\left(\frac{1}{\varepsilon-\delta'}\right)$. ◀

Conclusion

We close the major questions regarding online throughput maximization with and without commitment requirements and give an optimal online algorithm on identical parallel machines for the problem $P\,|\,\text{online } r_j, \text{pmtn}\,|\,\sum(1-U_j)$. It remains open whether the problem, where m is not part of the input, admits an online algorithm with a better competitive ratio as is the case for $Pm\,|\,\text{online } r_j, \text{pmtn}\,|\,\sum p_j(1-U_j)$ [26].

Another interesting question asks whether randomization allows for improved results. On a single machine, there is indeed an $\mathcal{O}(1)$-competitive randomized algorithm for scheduling without commitment, even without any slack assumption [20]. We are not aware of any lower bound that rules out a similar result on multiple machines. Further research directions include generalizations such as weighted throughput maximization. While strong lower bounds exist for handling weighted throughput with commitment [8], there remains a gap for the problem without commitment.

References

1 Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallelizable jobs online to maximize throughput. In *Proceedings of the Latin American Theoretical Informatics Symposium (LATIN)*, pages 755–776, 2018. doi:10.1007/978-3-319-77404-6_55.

2 Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Truthful online scheduling with commitments. In *Proceedings of the ACM Symposium on Economics and Computations (EC)*, pages 715–732, 2015. doi:10.1145/2764468.2764535.

3 Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Competitive algorithms for due date scheduling. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 28–39, 2007. doi:10.1007/978-3-540-73420-8_5.

4 Sanjoy K. Baruah and Jayant R. Haritsa. Scheduling for overload in real-time systems. *IEEE Trans. Computers*, 46(9):1034–1039, 1997. doi:10.1109/12.620484.

5 Sanjoy K. Baruah, Jayant R. Haritsa, and Nitin Sharma. On-line scheduling to maximize task completions. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 228–236, 1994. doi:10.1109/REAL.1994.342713.

6 Sanjoy K. Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis E. Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992. doi:10.1007/BF00365406.

7 Ran Canetti and Sandy Irani. Bounding the power of preemption in randomized scheduling. *SIAM J. Comput.*, 27(4):993–1015, 1998. doi:10.1137/S0097539795283292.

8 Lin Chen, Franziska Eberle, Nicole Megow, Kevin Schewior, and Cliff Stein. A general framework for handling commitment in online throughput maximization. *Math. Prog.*, 2020. doi:10.1007/s10107-020-01469-2.

9 Lin Chen, Nicole Megow, and Kevin Schewior. The power of migration in online machine minimization. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 175–184, 2016. doi:10.1145/2935764.2935786.

10 Lin Chen, Nicole Megow, and Kevin Schewior. An $\mathcal{O}(\log m)$-competitive algorithm for online machine minimization. *SIAM J. Comput.*, 47(6):2057–2077, 2018. doi:10.1137/17M116032X.

11 Bhaskar DasGupta and Michael A. Palis. Online real-time preemptive scheduling of jobs with deadlines. In *Proceedings of the International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 96–107, 2000. doi:10.1007/3-540-44436-X_11.

12 Michael L. Dertouzos and Aloysius K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. Software Eng.*, 15(12):1497–1506, 1989. doi:10.1109/32.58762.

13 Andrew D. Ferguson, Peter Bodík, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, pages 99–112, 2012. doi:10.1145/2168836.2168847.

14 Juan A. Garay, Joseph Naor, Bülent Yener, and Peng Zhao. On-line admission control and packet scheduling with interleaving. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 94–103, 2002. doi:10.1109/INFCOM.2002.1019250.

15 Michael H. Goldwasser. Patience is a virtue: The effect of slack on competitiveness for admission control. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 396–405, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.314592.

16 Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979. doi:10.1016/S0167-5060(08)70356-X.

17 Sungjin Im and Benjamin Moseley. General profit scheduling and the power of migration on heterogeneous machines. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 165–173, 2016. doi:10.1145/2935764.2935771.

18 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000. doi:10.1145/347476.347479.

19 Bala Kalyanasundaram and Kirk Pruhs. Eliminating migration in multi-processor scheduling. *J. Algorithms*, 38(1):2–24, 2001. doi:10.1006/jagm.2000.1128.

20 Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. *J. Algorithms*, 49(1):63–85, 2003. doi:10.1016/S0196-6774(03)00074-9.

21 Gilad Koren and Dennis E. Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theor. Comput. Sci.*, 128(1–2):75–97, 1994. doi:10.1016/0304-3975(94)90165-1.

22 Gilad Koren and Dennis E. Shasha. D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM J. Comput.*, 24(2):318–339, 1995. doi:10.1137/S0097539792236882.

23 Richard J. Lipton and Andrew Tomkins. Online interval scheduling. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 302–311, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314506.

24 Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs: extended abstract. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 305–314, 2013. doi:10.1145/2486159.2486187.

25 Kirk Pruhs and Clifford Stein. How to schedule when you have to buy your energy. In *Proceedings of the International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 352–365, 2010. doi:10.1007/978-3-642-15369-3_27.

26 Chris Schwiegelshohn and Uwe Schwiegelshohn. The power of migration for online slack scheduling. In *Proceedings of the European Symposium of Algorithms (ESA)*, volume 57, pages 75:1–75:17, 2016. doi:10.4230/LIPIcs.ESA.2016.75.

27 Gerhard J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theor. Comput. Sci.*, 130(1):5–16, 1994. doi:10.1016/0304-3975(94)90150-3.

A Polynomial Kernel for Line Graph Deletion

Eduard Eiben
Department of Computer Science, Royal Holloway University of London, Egham, UK
eduard.eiben@rhul.ac.uk

William Lochet
Department of Informatics, University of Bergen, Bergen, Norway
william.lochet@uib.no

Abstract

The line graph of a graph G is the graph $L(G)$ whose vertex set is the edge set of G and there is an edge between $e, f \in E(G)$ if e and f share an endpoint in G. A graph is called line graph if it is a line graph of some graph. We study the Line-Graph-Edge Deletion problem, which asks whether we can delete at most k edges from the input graph G such that the resulting graph is a line graph. More precisely, we give a polynomial kernel for Line-Graph-Edge Deletion with $\mathcal{O}\left(k^5\right)$ vertices. This answers an open question posed by Falk Hüffner at Workshop on Kernels (WorKer) in 2013.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Kernelization, line graphs, H-free editing, graph modification problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.42

Funding *William Lochet*: Supported by The Bergen Research Foundation (BFS).

1 Introduction

For a family $\mathcal{G}$ of graphs, the general $\mathcal{G}$-Graph Modification problem asks whether we can modify a graph G into a graph in $\mathcal{G}$ by performing at most k simple operations. Typical examples of simple operations well-studied in the literature include vertex deletion, edge deletion, edge addition, or a combination of edge deletion and addition. We call these problems $\mathcal{G}$-Vertex Deletion, $\mathcal{G}$-Edge Deletion, $\mathcal{G}$-Edge Addition, and $\mathcal{G}$-Edge Editing, respectively. By a classical result by Lewis and Yannakakis [20], $\mathcal{G}$-Vertex Deletion is NP-complete for all non-trivial hereditary graph classes. The situation is quite different for the edge modification problems. Earlier efforts for edge deletion problems [13, 24], though having produced fruitful concrete results, shed little light on a systematic answer, and it was noted that such a generalization is difficult to obtain.

$\mathcal{G}$-Graph Modification problems have been extensively investigated for graph classes $\mathcal{G}$ that can be characterized by a finite set of forbidden induced subgraphs. We say that a graph is $\mathcal{H}$-free if it contains none of the graphs in $\mathcal{H}$ as an induced subgraph. For this special case, the $\mathcal{H}$-free Vertex Deletion is well understood. If $\mathcal{H}$ contains a graph on at least two vertices, then all of these problems are NP-complete, but admit a $c^k n^{\mathcal{O}(1)}$ algorithm [4], where c is the size of the largest graph in $\mathcal{H}$ (the algorithms with running time $f(k)n^{\mathcal{O}(1)}$ are called fixed-parameter tractable (FPT) algorithms [7, 11]). On the other hand, the NP-hardness proof of Lewis and Yannakakis [20] excludes algorithms with running time $2^{o(k)}n^{\mathcal{O}(1)}$ under the Exponential Time Hypothesis (ETH) [18]. Finally, as observed by Flum and Grohe [15] a simple application of sunflower lemma [14] gives a *kernel* with $\mathcal{O}\left(k^c\right)$ vertices, where c is again the size of the largest graph in $\mathcal{H}$. A kernel is a polynomial time preprocessing algorithm which outputs an equivalent instance of the same problem such that the size of the reduced instance is bounded by some function $f(k)$ that depends only on

k. We call the function $f(k)$ the size of the kernel. It is well-known that any problem that admits an FPT algorithm admits a kernel. Therefore, for problems with FPT algorithms one is interested in polynomial kernels, i.e., kernels whose size is a polynomial function.

For the edge modification problems, the situation is more complicated. While all of these problems also admit $c^k n^{\mathcal{O}(1)}$ time algorithm, where c is the maximum number of edges in a graph in $\mathcal{H}$ [4], the P vs NP dichotomy is still not known. Only recently Aravind et al. [1] gave the dichotomy for the special case when $\mathcal{H}$ contains precisely one graph H. From the kernelization point of view, the situation is also more difficult. The reason is that deleting or adding an edge to a graph can introduce a new copy of H and this might further propagate. Hence, we cannot use the sunflower lemma to reduce the size of the instance. Cai asked the question whether H-free Edge Deletion admits a polynomial kernel for all graphs H [3]. Kratsch and Wahlström [19] showed that this is probably not the case and gave a graph H on 7 vertices such that H-free Edge Deletion and H-free Edge Editing does not admit a polynomial kernel unless coNP $\subseteq$ NP/poly. Consequently, it was shown that this is not an exception, but rather a rule [5, 16]. Indeed the result by Cai and Cai [5] shows that H-free Edge Deletion, H-free Edge Addition, and H-free-Edge Editing do not admit a polynomial kernel whenever H or its complement is a path or a cycle with at least 4 edges or a 3-connected graph with at least 2 edges missing. Very recently, Marx and Sandeep [21] gave a list of nine graphs, all on 5 vertices such that if H-free-Edge Editing does not admit a kernel for any of these nine graphs under standard complexity assumptions, then H-free-Edge Editing admits a polynomial kernel for $|H| \geq 5$ if and only if H is either empty or complete graph. They also provided a similar characterization for H-free Edge Deletion and H-free Edge Editing. This suggests that actually the H-free edge modification problems with a polynomial kernels are rather rare and only for small graphs H. Recently, Eiben, Lochet, and Saurabh [12] announced a polynomial kernel for the case when H is a paw, which leaves only one last graph on 4 vertices for which the kernelization of H-free edge modification problems remains open, namely $K_{1,3}$ known also as the claw.

The class of claw-free graphs is a very well studied class of graphs with some interesting algorithmic properties. The most prominent example is probably the algorithm of Sbihi [22] for computing the maximum independent set in polynomial time. It also has been extensively studied from a structural point of view, and Chudnosky and Seymour proposed, after a series of papers, a complete characterization of claw-free graphs [6]. Because of such a characterization, it seems reasonable to believe that a polynomial kernel for Claw-free Edge Deletion exists. However, the characterization of Chudnosky and Seymour is quite complex, which makes it hard to use. For this reason, as noted by Cygan et al. [8], trying to show the existence of a polynomial kernel in the cases of sub-classes of claw-free graphs seems like a good first step to try to understand this problem. In this paper, we prove the result for the most famous such class, line graphs.

▶ **Theorem 1.** *Line-Graph Edge Deletion admits a kernel with $\mathcal{O}\left(k^5\right)$ vertices.*

Overview of the Algorithm

As the first step of the kernelization algorithm, we use the characterization of line graphs by forbidden induced subgraphs to find a set S of at most $6k$ vertices such that for every vertex $v \in S$, $G - (S \setminus \{v\})$ is a line graph. This is simply done by a greedy edge-disjoint packing of forbidden induced subgraphs. Having the set S, we use the algorithm by Degiorgi and Simon [9] to find a partition of edges of $G - S$ into cliques such that each vertex is in precisely 2 cliques. Let $\mathcal{C} = \{C_1, \ldots, C_q\}$ be the cliques in the partition. Since $G - (S \setminus \{v\})$

is also a line graph, it is a rather simple consequence of Whitney's isomorphism theorem that the neighborhood of v can be covered by constantly many cliques of $\mathcal{C}$. Furthermore, we will show that if a clique C in $\mathcal{C}$ has more than $k+7$ vertices then the optimal solution does not contain an edge in C. Hence, we can partition the cliques in $\mathcal{C}$ into two groups "large" and "small". Note that if the optimal solution contains an edge in some small clique C, then for this change to be necessary, it has to be propagated from S by modifying small cliques on some clique-path from S to C using only small cliques. We will therefore define the distance of a clique to S, without going into too many details in here, to be basically the length of a shortest clique-path from the clique to S using only small cliques. Since there are only $\mathcal{O}(|S|)$ cliques in immediate neighborhood of S and the number of cliques in the neighborhood of a small clique is bounded by its size, we obtain that there are at most $\mathcal{O}\left(k^d\right)$ cliques at distance at most d. Our main contribution and most technical part of our proof is to show that we can remove the edges covered by cliques at distance at least 5 from G. This is covered in Section 4. Afterwards we end up with an instance with all cliques in $\mathcal{C}$ at distance at least 5 from S being singletons. As discussed above there are only $\mathcal{O}\left(k^4\right)$ cliques at distance at most 4 and because large cliques stay intact in any optimal solution, it suffices to keep $k+7$ vertices in each large clique, which leads to the desired kernel of size $\mathcal{O}\left(k^5\right)$.

2 Preliminaries

We assume familiarity with the basic notations and terminologies in graph theory. We refer the reader to the standard book by Diestel [10] for more information. Given a graph G and a set of edges $F \subseteq E(G)$, we denote by $G - F$ the graph whose set of vertices is $V(G)$ and set of edges is the set $E(G) \setminus F$. Given two vertices $u, v \in V(G)$, we let the *distance* between u and v in G, denoted $\mathrm{dist}_G(u,v)$), be the number of edges on a shortest path from u to v. Furthermore, for $S \subseteq V(G)$ and $u \in V(G)$ we let $\mathrm{dist}_G(u,S) = \min_{v \in S} \mathrm{dist}_G(u,v)$). We omit the subscript G, if the graph is clear from the context.

Parameterized Algorithms and Kernelization. For a detailed illustration of the following facts the reader is referred to [7, 11]. A *parameterized problem* is a language $\Pi \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet; the second component k of instances $(I,k) \in \Sigma^* \times \mathbb{N}$ is called the *parameter*. A parameterized problem Π is *fixed-parameter tractable* if it admits a *fixed-parameter algorithm*, which decides instances (I,k) of Π in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for some computable function f.

A *kernelization* for a parameterized problem Π is a polynomial-time algorithm that given any instance (I,k) returns an instance (I',k') such that $(I,k) \in \Pi$ if and only if $(I',k') \in \Pi$ and such that $|I'| + k' \leq f(k)$ for some computable function f. The function f is called the *size* of the kernelization, and we have a polynomial kernelization if $f(k)$ is polynomially bounded in k. It is known that a parameterized problem is fixed-parameter tractable if and only if it is decidable and has a kernelization. However, the kernels implied by this fact are usually of superpolynomial size.

A *reduction rule* is an algorithm that takes as input an instance (I,k) of a parameterized problem Π and outputs an instance (I',k') of the same problem. We say that the reduction rule is *safe* if (I,k) is a *yes*-instance if and only if (I',k') is a *yes*-instance. In order to describe our kernelization algorithm, we present a series of reduction rules.

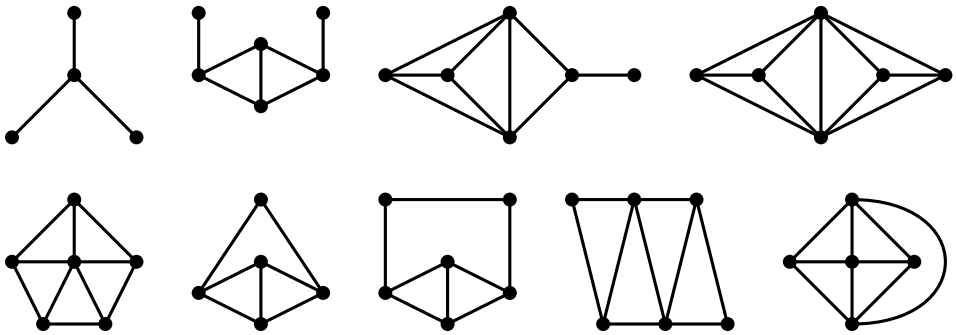


Figure 1 The nine minimal non-line graphs, from characterization of line graphs by forbidden induced subgraphs of Beineke [2]. Note that all of these graphs have at most 6 vertices.

Line graphs. Given a graph G, its *line graph* $L(G)$ is a graph such that each vertex of $L(G)$ represents an edge of G and two vertices of $L(G)$ are adjacent if and only if their corresponding edges share a common endpoint (are incident) in G. It is well known that if the line graphs of two connected graphs G_1 and G_2 are isomorphic then either G_1 and G_2 are K_3 and $K_{1,3}$, respectively, or G_1 and G_2 are isomorphic as well (Whitney's isomorphism theorem [23], see also Theorem 8.3 in [17]). We say that a graph H is a line graph, if there exists a graph G such that $H = L(G)$. Note that in this paper we only consider simple graphs, *i.e.*, the graphs without loops or multiple edges and in particular we also only consider line graphs of simple graphs. Formally, we then study the following parameterized problem:

Line-Graph-Edge Deletion	
Input:	A graph $G = (V, E)$ and $k \in \mathbb{N}$.
Parameter:	k.
Question:	Is there a set of edges $F \subseteq E(G)$ such that $G - F$ is a line graph and $\lvert F\rvert \leq k$.

We call a set of edges $F \subseteq V(G)$ such that $G - F$ is a line graph a *solution* for G. A solution F is *optimal*, if there does not exists a solution F' such that $|F'| < |F|$. To obtain our kernel, we will make use of several equivalent characterizations of line graphs.

▶ **Theorem 2** (see, e.g., Theorem 8.4 in [17]). *The following statements are equivalent:*

(1) *G is a line graph.*

(2) *The edges of G can be partitioned into complete subgraphs in such a way that no vertex lies in more than two of the subgraphs.*

(3) *G does not have $K_{1,3}$ as an induced subgraph, and if two odd triangles (triangles with the property that there exists another vertex adjacent to an odd number of triangle vertices) share a common edge, then the subgraph induced by their vertices is K_4.*

(4) *None of nine graphs of Figure 1 is an induced subgraph of G.*

3 Structure of Line Graphs

To obtain our kernel, we heavily rely on different characterizations of line graphs given by Theorem 2. The two main characterizations used throughout the paper are given in points (2) and (4) To ease the presentation of our techniques, we will define a notion of a *clique partition witness* for G, whose existence is implied by the point (2) of Theorem 2. Let G be a line graph, a *clique partition witness* for G is a set $\mathcal{C} = \{C_1, \ldots, C_q\}$ be such that:

- $C_i \subseteq V(G)$ for all $i \in [q]$,
- $G[C_i]$ is a complete graph for all $i \in [q]$, that is every C_i is a clique in G,
- $|C_i \cap C_j| \leq 1$ for all $i \neq j \in [q]$,
- every $v \in V(G)$ is in exactly two sets in $\mathcal{C}$, and
- for every edge $uv \in E(G)$ there exists exactly one set $C_i \in \mathcal{C}$ such that $\{u, v\} \subseteq C_i$.

Note that by Theorem 2, G is a line graph if and only if there exists a clique partition witness for G. The following three observations follow directly from the definition of clique partition witness and will be useful throughout the paper.

▶ **Observation 3.** *If $\mathcal{C}$ is clique partition witness for G then every clique in $\mathcal{C}$ is either a singleton, K_2, or a maximal clique in G.*

▶ **Observation 4.** *If $\mathcal{C}$ is clique partition witness for G, then every maximal clique in G of size at least 4 is in $\mathcal{C}$.*

▶ **Observation 5.** *If $\mathcal{C}$ is clique partition witness for G, then any clique of G which is not a sub-clique of some element of $\mathcal{C}$ is a triangle.*

We would like to point out that given a line graph G one can find a clique partition witness for G for example by using an algorithm of Degiorgi and Simon [9] for recognition of line graphs in polynomial time. In the following lemma, we sketch the main procedure of their algorithm together with necessary modifications to actually output a clique partition witness instead of the underlying graph H such that $G = L(H)$, for completeness.

▶ **Lemma 6.** *Given a graph G, there is an algorithm that in time $\mathcal{O}(|E(G)| + |V(G)|)$ decides whether G is a line graph and if so, constructs a clique partition witness for G.*

Proof. The algorithm by Degiorgi and Simon construct the input graph G by adding vertices one at a time, at each step it chooses a vertex to add that is already adjacent to at least one previously-added vertex. That is it construct graphs G_1, G_2, …, $G_n = G$ such that G_i is a connected subgraph of G on i vertices. At each step it maintains a graph H_i such that G_i is a line graph of H_i. In here, we can actually keep a clique partition witness $\mathcal{C}_i$ for G_i such that there is a bijection φ_i between vertices of H_i and clique in $\mathcal{C}_i$ such that $uv \in E(H_i)$ if and only if $|\varphi_i(u) \cap \varphi_i(v)| = 1$.

The algorithm heavily relies on the Whitney's isomorphism theorem that implies that if the underlying graph of G_i has at least 4 vertices, then the underlying graph H_i is unique up to isomorphism. When adding a vertex v to a graph G_i for $i \leq 4$, the algorithm simply brute-forces the possibilities for H_i and $\mathcal{C}_i$.

When adding a vertex v to G_i when $i > 4$, let S be the subgraph of H_i formed by the edges that correspond to the neighbors of v in G_i. Check that S has a vertex cover consisting of one vertex or two non-adjacent vertices, *i.e.*, there are cliques C_1 and C_2 in $\mathcal{C}_i$ with $C_i \cap C_2 = \emptyset$ and $S \subseteq C_1 \cap C_2$. If there are two vertices in the cover, add an edge (corresponding to v) that connects these two vertices in H_i and add v to both C_1 and C_2. If there is only one vertex u in the cover, then add a new vertex to H_i, adjacent to this vertex, add v to the clique $\varphi_i(u)$ in $\mathcal{C}_i$ and add a new clique $\{v\}$ to $\mathcal{C}_i$ to create $\mathcal{C}_{i+1}$. ◀

3.1 Level Structure of Instances

For the rest of the paper, let G be the input graph and let S be a set of at most $6k$ vertices such that for every $v \in S$ the graph $G - (S \setminus \{v\})$ is a line graph. We let $\mathcal{C} = \{C_1, \ldots, C_q\}$ be a clique partition witness for $G - S$. The goal of this subsection is to split the cliques in

$\mathcal{C}$ to levels such that 1) each level contains only bounded number of cliques (that are not singletons) and 2) if we do not remove any edge at level i, then we do not need to remove any edge at level $j > i$. We will later show that we do not need to remove any edges in cliques in level 5. The following lemma is useful to define/bound the number of cliques at the first level, i.e., cliques that interact with S.

▶ **Lemma 7.** *For every vertex $v \in S$ there are at most two cliques $C_1, C_2 \in \mathcal{C}$ such that v is adjacent to all vertices in $C_1 \cup C_2$ and to at most 6 vertices in $V(G) \setminus (S \cup C_1 \cup C_2)$.*

Proof. By the choice of the set S, it follows that $G - (S \setminus \{v\})$ is a line graph. Let $\mathcal{C}'$ be clique partition witness for $G - (S \setminus \{v\})$. By definition, there are at most two cliques C'_1 and C'_2 in $\mathcal{C}'$ that contains v and all its neighbors. If $|C'_i| \geq 5$, for some $i \in \{1, 2\}$, then by Observation 4, $C'_i \setminus \{v\}$ is a clique in $\mathcal{C}$ and we can set C_i to be $C'_i \setminus \{v\}$. Else $|C'_i \setminus v| \leq 3$ and C'_i contributes to at most 3 neighbors of v in $G - S$. ◀

The following lemma shows that cliques of size at least $k+7$ can serve as kind of separators that will never be changed by a solution of size at most k. Hence, we can remove all cliques separated from S by large cliques. Moreover, it allows us to define the $(i+1)$-st level by only considering the cliques of size at most $k+6$ at level i.

▶ **Lemma 8.** *Let $C \in \mathcal{C}$ such that $|C| \geq k+7$ and let $A \subset E(G)$ be an optimal solution for G. Then $A \cap E(G[C]) = \emptyset$. Moreover, the clique partition witness $\mathcal{C}'$ for $G - A$ contains a clique C' such that $C' \setminus S = C$.*

Proof. Let $\{u, v\} \in A$ such that $\{u, v\} \subseteq C$. Clearly there are at most $k-1$ vertices w in C such that either $\{u, w\} \in A$ or $\{w, v\} \in A$. Let $x \in C$ be such that xv, xu are edges in $G - A$. Similarly, there are at most $k-1$ non-edges to u, v, x in $G - A$, so let $y \in C$ be a vertex such that yu, yv, yx are edges in $G - A$. Repeating the same argument once again, there is $z \in C$ such that zu, zv, zx, zy are edges in $G - A$. However, the subgraph of $G - A$ induced on u, v, x, y, z is K_5 minus an edge, which is one of the forbidden induced subgraphs in the characterization of line graphs.

The moreover part follows from the following argument. Since $|C| \geq k + 7 \geq 4$ and, by Observation 4 it follows that the clique partition witness $\mathcal{C}'$ contains a maximal clique $C' \supseteq C$. It remains to show that no vertex in $V(G) \setminus (S \cup C)$ is in C'. Every vertex in $V(G) \setminus S$ is in two cliques C_1, C_2 in $\mathcal{C}$ that cover all its incident edges in $G - S$. If none of these two cliques is C, then C intersect each of these two cliques in at most 1 vertex. It follows that, because $|C| \geq 3$, there is no vertex in $V(G) \setminus (S \cup C)$ adjacent to all vertices of C. ◀

Let us now partition the cliques in $\mathcal{C}$ into two parts $\mathcal{C}_{<k+7}$ and $\mathcal{C}_{\geq k+7}$ such that $\mathcal{C}_{<k+7}$ contains precisely all the cliques in $\mathcal{C}$ with less than $k+7$ vertices and $\mathcal{C}_{\geq k+7}$ contains the remaining cliques. We will refer to the cliques in $\mathcal{C}_{<k+7}$ as *small* cliques and the cliques in $\mathcal{C}_{\geq k+7}$ as *large* cliques. Intuitively, if we are forced to delete some edge in $G - S$, then this change had to be propagated from S only by changes in small cliques.

We are now ready to define the level structure on the cliques in $\mathcal{C}$. We divide the cliques in $\mathcal{C}$ into levels $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_p$, for some $p \in \mathbb{N}$, that intuitively reflects on how far from S the clique $C \in \mathcal{C}$ is if we consider a shortest path using only small cliques. We will define the levels recursively as follows. By Lemma 7 for every vertex $v \in S$ there exists at most two cliques $C_1, C_2 \in \mathcal{C}$ such that v is adjacent to all vertices in $C_1 \cup C_2$ and to at most 6 vertices in $V(G) \setminus (S \cup C_1 \cup C_2)$. Now, for a vertex $v \in S$, let $\mathcal{N}^v$ denote the set of cliques that contains C_1, C_2 and all the cliques in $\mathcal{C}$ that contain at least one of the neighbors of v in $V(G) \setminus (S \cup C_1 \cup C_2)$. We let $\mathcal{L}_1$ be precisely the set $\bigcup_{v \in S} \mathcal{N}^v$. Note that vertices in

$C_1 \cup C_2$ can each appear in one other clique that is not in $\mathcal{N}^v$ and in particular there are cliques that contain a vertex adjacent to a vertex in S and are not in $\mathcal{L}_1$. For $i > 1$, we then let $\mathcal{L}_i$ be the set of cliques C in $\mathcal{C} \setminus (\bigcup_{j\in\{1..i-1\}} \mathcal{L}_j)$ such that there is a small clique C' in the previous level (*i.e.*, $C' \in \mathcal{L}_{i-1} \cap \mathcal{C}_{<k+7}$) such that $C \cap C'$ is not empty.

▶ **Observation 9.** *Let $C \in \mathcal{C}$ and w a vertex in C. If w has a neighbor in S, then either $C \in \mathcal{L}_1 \cup \mathcal{L}_2$ or w is in a large clique.*

Proof. Let $v \in S$ be a neighbor of w. Then $\mathcal{N}^v \subseteq \mathcal{L}_1$ contains a clique C' with $w \in C'$. Clearly C' intersects C in w. Hence either C' is a large clique or by the definition of $\mathcal{L}_2$ the clique C is in $\mathcal{L}_1 \cup \mathcal{L}_2$. ◀

Let $p \in \mathbb{N}$ be such that $\mathcal{L}_p \neq \emptyset$ and $\mathcal{L}_{p+1} = \emptyset$. While the following Reduction Rule is not completely necessary and would be subsumed by Reduction Rule 2, we include it to showcase some of the ideas needed for the proof in a simplified setting.

▶ **Reduction Rule 1.** *Remove all vertices in $V(G) \setminus S$ that are not in a clique in $\bigcup_{i\in[p]} \mathcal{L}_i$.*

Proof of safeness. Let H be the resulting graph and let $\mathcal{C}_H$ be a set of cliques of H obtained from $\mathcal{C}$, by taking all cliques in $\bigcup_{i\in[p]} \mathcal{L}_i$ and for every clique in $C \in (\mathcal{C} \setminus \bigcup_{i\in[q]} \mathcal{L}_i)$, $\mathcal{C}_H$ contains $C \cap V(H)$, if it is nonempty. Since H is an induced subgraph of G and line graphs can be characterized by a set for forbidden induced subgraphs, it follows that for every $A \in E(G)$, if $G - A$ is a line graph, then $H - A$ is a line graph. It remains to show that if there is a set of edges $A \in E(H)$ such that $|A| \leq k$ and $H - A$ is a line graph, then $G - A$ is also a line graph. Let A be such a set of edges of minimum size and let $\mathcal{C}_A$ be a clique partition witness for $H - A$. It suffices to show that for every clique in $C \in (\mathcal{C}_H \setminus \bigcup_{i\in[p]} \mathcal{L}_i)$, it holds that $C \in \mathcal{C}_A$. If this is the case, we get a clique partition witness for $G - A$ by replacing the cliques of $\mathcal{C}_H \setminus \bigcup_{i\in[p]} \mathcal{L}_i$ in $\mathcal{C}_A$ by $\mathcal{C} \setminus \bigcup_{i\in[p]} \mathcal{L}_i$.

Now, $C \in (\mathcal{C}_H \setminus \bigcup_{i\in[p]} \mathcal{L}_i)$ means that all cliques intersecting C are large. Moreover, because all vertices in H are in some clique on some level, by Lemma 8, for each clique $C_1 \in \mathcal{C}_H$ that intersect C there is a clique in $C'_1 \in \mathcal{C}_A$ that is the union of C_1 and some vertices in S. Hence, all vertices in C are already in at least one clique in $\mathcal{C}_A \setminus C$ and all the edges incident to exactly one vertex in C are already covered by these cliques. And hence every clique that contains a vertex in C and intersects every other clique in $\mathcal{C}_A$ in at most one vertex has to be a subset of C. Moreover, the cliques in $\mathcal{C}_A$ that are subsets of C have to be vertex disjoint, since every vertex is in at most 2 cliques in $\mathcal{C}_A$. Hence, if C is not in $\mathcal{C}_A$, then some of the edges in C have to be in A, but replacing all the subsets of C in $\mathcal{C}_A$ by C gives a clique partition witness for $H - A'$ for some $A' \subsetneq A$ which contradicts the fact that A is of minimum size. ◀

We will also say that $C \in \mathcal{C}$ is at $\mathcal{L}$-distance d from S, denoted by $\text{dist}^{\mathcal{L}}(C)$, if C is in $\mathcal{L}_d$. We note that $\mathcal{C}$ still contains some cliques that are not in any of $\mathcal{L}_i$'s. We will let $\text{dist}^{\mathcal{L}}(C) = \infty$ for such a clique C. We can now upper bound the number of cliques at $\mathcal{L}$-distance d from S.

▶ **Lemma 10.** *There are at most $14|S|(k+6)^{d-1}$ cliques in $\mathcal{C}$ at level d,* i.e., *in $\mathcal{L}_d$.*

Proof. By the definition of $\mathcal{L}_1 = \bigcup_{v\in S} \mathcal{N}^v$, where $\mathcal{N}^v$ denote the set of cliques that contains C_1, C_2 and all the cliques in $\mathcal{C}$ that contain at least one of the neighbors of v in $V(G) \setminus (S \cup C_1 \cup C_2)$. By Lemma 7 for every vertex $v \in S$ there exists at most two cliques $C_1, C_2 \in \mathcal{C}$ such that v is adjacent to all vertices in $C_1 \cup C_2$ and to at most 6 vertices in $V(G) \setminus (S \cup C_1 \cup C_2)$. Since every vertex appears in two cliques of $\mathcal{C}$, it follows that $|\mathcal{N}^v| \leq 14$ and consecutively

$\mathcal{L}_1$ contains at most $14|S|$ cliques. Now by the definition of $\mathcal{L}_d$ we know that for any $d \geq 2$ a clique is at level d if and only if it shares a vertex with a small clique at level $d-1$. Since no three cliques in $\mathcal{C}$ can share a vertex the number of cliques at level d is at most the number of vertices in the small cliques at level $d-1$ and the lemma follows by a simple induction on d. ◀

The remainder of the algorithm consists of two steps. First, in Section 4, we show that we can remove all edges from cliques that are at $\mathcal{L}$-distance at least 5 from S. Afterwards, due to Lemma 10, we are left with only $\mathcal{O}\left(k^4\right)$ non-singleton cliques in $\mathcal{C}$. To finish the algorithm in Section 5, for each clique $C \in \mathcal{C}$ that is not a singleton, we mark an arbitrary subset of $k+7$ vertices in C and remove all unmarked vertices from G. It is then rather straightforward consequence of Lemma 8 that this rule is safe and we get an equivalent instance with $\mathcal{O}\left(k^5\right)$ vertices.

4 Bounding the Distance from S

The purpose of this section is to show that it is only necessary to keep the cliques in $\mathcal{C}$ that are at $\mathcal{L}$-distance at most 4 from S (and adding a singleton for vertices covered by exactly one clique at $\mathcal{L}$-distance at most 4). To do so, we need to show that there is always a solution that does not change the cliques at $\mathcal{L}$-distance 5 at all. For this purpose, we first need to understand the interaction of cliques at $\mathcal{L}$-distance 4 from S with the solution. The first step will be to show that there is an optimal solution A with clique partition witness $\mathcal{C}_A$ such that all cliques in $\mathcal{C}_A$ that share an edge with a clique in $\mathcal{C}$ at $\mathcal{L}$-distance at least 4 from S are actually subcliques of a clique in $\mathcal{C}$ (when restricted to $G-S$). It is a simple consequence of Lemma 8 that this is true for any clique that intersect a large clique in an edge. Hence, we can only care about cliques in $\mathcal{C}_A$ that intersect a small clique C in an edge. By Observation 9, no vertex in C has a neighbor in S. It then follows by Observation 5 that any clique in $\mathcal{C}_A$ that intersects C in an edge and is not a subclique of a clique in $\mathcal{C}$ is indeed a triangle. This leads us to the following definition.

▶ **Definition 11** (bad triangle). *Let $A \subseteq E(G)$ be such that $G-A$ is a line graph and let $\mathcal{C}_A$ be a clique partition witness of $G-A$. A triangle $xyz \in \mathcal{C}_A$ is said to be* bad *if it is not a sub-clique of a clique in $\mathcal{C}$, and one of the edges of the triangle, say xy, is an edge contained in a clique of $\mathcal{L}$-distance at least 4 from S.*

▶ **Lemma 12.** *There exists an optimal solution without any bad triangle.*

Proof. Let A be an optimal solution and $\mathcal{C}_A$ the clique partition witness of $G-A$. Suppose xyz is a bad triangle and let C_1, C_2 and C_3 be the elements of $\mathcal{C}$ containing the edges xy, yz and zx respectively. See also Figure 2 for an illustration. Since xyz is a bad triangle, no clique in $\mathcal{C}_A$ is a superset of C_i, $i \in \{1,2,3\}$ and it is a simple consequence of Lemma 8 that C_i is a small clique. By definition of bad triangle, at least one of C_1, C_2, and C_3 is at $\mathcal{L}$-distance at least 4 from S and hence all of these cliques are at $\mathcal{L}$-distance at least 3 from S. Let X (resp. Y, Z) denote the other clique of $\mathcal{C}_A$ containing x (reps. y, z). Let us define $X_1 = X \cap C_1, X_3 = X \cap C_3, Y_1 = Y \cap C_1, Y_2 = Y \cap C_2, Z_3 = Z \cap C_3$ and $Z_2 = Z \cap C_2$.

Let $C_1' = X_1 \cup Y_1$, $C_2' = Y_2 \cup Z_2$ and $C_3' = Z_3 \cup X_3$. Note that C_i' is a sub-clique of C_i for $i \in [3]$. Now for every $i \in [3]$ we will update C_i' as follows. As long as there exists an edge e in C_i' such that e belongs to $K_i \in \mathcal{C}_A$, K_i is a sub-clique of C_i and $K_i \not\subseteq C_i'$, we set $C_i' := C_i' \cup K_i$ (see also Figure 2b). When this process stops, C_i' corresponds to the union of a set of elements of $\mathcal{C}_A : K_1^i, \ldots, K_{l_i}^i$ which are sub-cliques of C_i, and C_i'. Moreover, for

any edge e of C'_i which is strictly contained in another clique of $\mathcal{C}_A$ (meaning this clique is not e), then this clique has to be a triangle by Observation 5, as the clique of $\mathcal{C}$ containing e is C_i. Let $e^i_1, \dots, e^i_{s_i}$ denote the set of such edges and let $C^i_1, \dots, C^i_{s_i}$ be the triangles of $\mathcal{C}_A$ containing these edges. Note that $|A \cap C'_1| \geq s_1$, as for any edge e^1_j, either x or y has to be non adjacent to each extremity in $G - A$ or the edge would be in two cliques of $\mathcal{C}_A$ (the same statement is also correct for $|A \cap C'_2|$ and $|A \cap C'_3|$). Let A' be the set obtained from A by

- Removing all the edges of $A \cap C'_1$, $A \cap C'_2$ and $A \cap C'_3$.
- Adding one of the two edges of C^i_j different from e^i_j for every $i \in [3]$ and $j \in [s_i]$ (see Figure 2c illustrating the replacement of C^j_i in $\mathcal{C}_A$ by its proper subclique in $\mathcal{C}_{A'}$ implied by this addition of an edge in A'.).

▷ Claim 13. A' is a set of edges not larger than A and such that $G - A'$ is a line graph with fewer bad triangles than $G - A$.

Proof. The fact that $|A'| \leq |A|$ follows from the fact that $|A \cap C'_i| \geq s_i$ for all $i \in [3]$. To see that $G - A'$ is a line graph, let us show that $\mathcal{C}_{A'}$ defined as follows is a clique partition witness for $G - A'$. Let $\mathcal{C}_{A'}$ be the set defined from $\mathcal{C}_A$ by

- Removing C_A, X, Y, Z, every C^i_j for $i \in [3]$, $j \in [s_i]$, every K^i_j for every $i \in [3]$ and $j \in [l_i]$ and every edge which are contained in one of the C'_i.
- Adding C'_i for $i \in [3]$ and for every $i \in [3]$ and $j \in [s_i]$ the edge of C^i_j which has not been removed from A, as well as singletons for vertices belonging to only one clique.

First it is clear that any set added to $\mathcal{C}'_A$ is a clique as A' does not contain any edge in $A \cap C'_1$, $A \cap C'_2$ and $A \cap C'_3$ and these sets are cliques of G.

Now take B and C two cliques of $\mathcal{C}'_A$. If B and C belong to $\mathcal{C}_A$, then clearly their intersection has size at most 1. If one belongs to $\mathcal{C}_A$ and the other is the remaining edge of C^i_j for $i \in [3]$ and $j \in [s_i]$, then it is also clear as it is true for C^i_j. For $i, j \in [3]^2$, C'_i and C'_j also intersect on one vertex, because C_i and C_j do and moreover, the cliques of $\mathcal{C}_A$ intersecting C'_i on two vertices are exactly the C^i_j, so if $B = C'_i$ and $A \in \mathcal{C}_A$, the intersection has also size at most 1, and we covered all the cases for $|C \cap B|$.

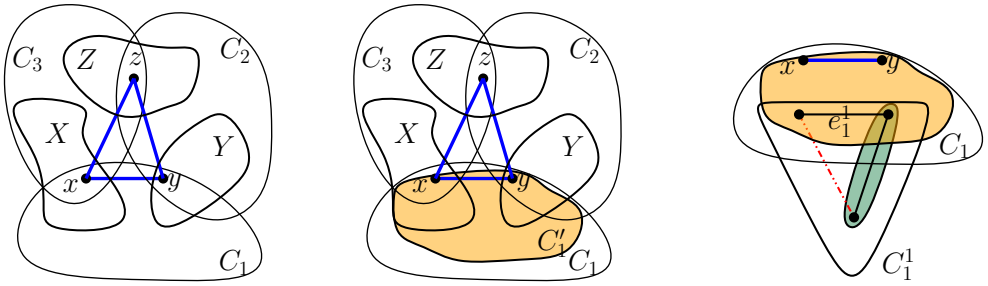
Now for every vertex $x \in V(G)$, if x does not belong to C'_1, C'_2 and C'_3, then it belongs to the same cliques as in $\mathcal{C}_A$ (where the C^i_j have been reduced to an edge and a singleton). For the vertices of C'_1, C'_2 and C'_3 different from x, y, z, we replaced one sub-clique of C_i by another. Finally x belongs to C'_1 and C'_3, y to C'_1 and C'_2 and z to C'_2 and C'_3.

Suppose uv is an edge of $E(G - A')$. If uv belongs to one of the C'_i, then by definition of the C^i_j and because we removed all these triangles, uv only belongs to one clique. For the other edges of $E(G - A')$, the fact that uv belongs to exactly one clique of $\mathcal{C}'_A$ follows from the fact that A' differs on those edges from A only because we added some edges of the C^i_j, and $\mathcal{C}_A$ differs on these vertices only because we changed C^i_j into the remaining edge outside C'_i.

Overall $\mathcal{C}_{A'}$ is indeed a clique partition for $G - A'$. Moreover, to obtain it, we removed at least one bad triangle from $\mathcal{C}_A$ (C_A) without adding one. This ends the proof of the claim. ◁

Finally, we can repeat the process until $\mathcal{C}_{A'}$ is without any bad triangles, which ends the proof of the lemma. ◀

Before we show that indeed all cliques at $\mathcal{L}$-distance at least 5 from S are intact in some optimal solution, we show another auxiliary lemma that is rather simple consequence of Lemma 12, namely that there is a clique partition witness for some optimal solution A such that no two cliques $\mathcal{C}_A$ that intersect the same clique $C \in \mathcal{C}$ at $\mathcal{L}$-distance at least 4 from S in an edge can intersect. This is important later to show that indeed no vertex in a clique $C \in \mathcal{C}$ at $\mathcal{L}$-distance 5 from S will be in two cliques in $\mathcal{C}_A$ that are not subsets of C.



(a) A bad triangle xyz in $\mathcal{C}_A$. C_1, C_2, C_3 are three cliques in $\mathcal{C}$ containing xy, yz, and xz respectively. $X, Y, Z \in \mathcal{C}_A$ are the cliques containing x, y, z other than xyz.

(b) C'_1 is the inclusion minimal clique such that $(X \cup Y) \cap C_1 \subseteq C'_1 \subseteq C_1$ and for all $K \in \mathcal{C}_A$ if $K \subseteq C_1$ and $|K \cap C'_1| \geq 2$, then $K \subseteq C'_1$. C'_2 and C'_3 are defined analogously.

(c) C^1_1 intersects C'_1 in an edge e^1_1. C^1_1 is replaced by an edge other than e^1_1. This forces to include one edge in C^1_1 to a solution A'. However, this can be seen as replacing an edge between $\{x, y\}$ and endpoints of e^1_1 that is in A.

Figure 2 The treatment of bad triangles. Let $A \subseteq E(G)$ be an optimal solution, $\mathcal{C}_A$ a clique partition witness for A. A bad triangle xyz together with cliques X, Y, Z, as defined in Subfigure 2a are replaced by cliques C'_1, C'_2, and C'_3 defined in Subfigure 2b. Subfigure 2c shows the treatment of cliques in $\mathcal{C}_A$ that intersect C'_i in an edge. By definition of C'_i, such clique is not a subclique of C_i and hence a triangle.

▶ **Lemma 14.** *There exists an optimal solution $A \subseteq E(G)$ without any bad triangles and clique partition witness $\mathcal{C}_A$ for $G - A$ such that for every $C \in \mathcal{C}$ of $\mathcal{L}$-distance at least 4 and every $w \in C$, if C^w_1 and C^w_2 are the two cliques in $\mathcal{C}_A$ containing w, then either $C^w_1 \cap C = \{w\}$ or $C^w_2 \cap C = \{w\}$.*

Proof. Let $A \subseteq E(G)$ be an optimal solution for G without any bad triangles and clique partition witness $\mathcal{C}_A$ for $G - A$ minimizing the number of pairs (C, w) for which C is at $\mathcal{L}$-distance at least 4, $w \in C$ and the two cliques, denoted C^w_1 and C^w_2, in $\mathcal{C}_A$ containing w intersect C in two vertices. Furthermore, it follows from Lemma 8 that C is a small clique, as the clique containing C as a subclique in $\mathcal{C}_A$ would intersect C^w_1 in two vertices. Since there are no bad triangles and C is at $\mathcal{L}$-distance at least 4, it follows that $C^w_1 \subseteq C$ and $C^w_2 \subseteq C$ and in particular $C^w_1 \cup C^w_2$ is a clique in G. Indeed, our goal is to replace C^w_1 and C^w_2 by a clique D such that $C^w_1 \cup C^w_2 \subseteq D \subseteq C$. We start by setting $D = C^w_1 \cup C^w_2$. We will also keep a track of cliques we will remove from $\mathcal{C}_A$. This set will be $\mathcal{D}$ and initialize it as $\mathcal{D} = \{C_1, C_2\}$.

As in the proof of Lemma 12, the only reason why we cannot replace C_1 and C_2 by D and obtain a solution that removes a subset of edges of A is because there exist two vertices $v_1, v_2 \in D$ and a clique $C_{12} \in \mathcal{C}_A$ with $\{v_1, v_2\} \subseteq C_{12}$. Observe that by our assumption there is no bad triangle and $C_{12} \subseteq C$. We let $D = D \cup C_{12}$ and $\mathcal{D} = \mathcal{D} \cup C_{12}$ and repeat until there is no such pair of vertices. Note that every vertex in G is in at most two cliques of $\mathcal{C}_A$. Therefore, this process has to stop after at most $2|C|$ steps.

When there are no two vertices in D that appear together in a different clique, we remove $\mathcal{D}$ from $\mathcal{C}_A$ and replace it by D and $\{v\}$. For every vertex that appears in D, we removed one clique that it appeared in. Hence, every vertex appears in at most 2 cliques and we can always add a singleton to clique partition witness for vertices that are only in one clique. Moreover, no two cliques intersect in two vertices, since D is the only clique we added, and we removed/changed all the cliques that intersected D in at least two vertices. Finally, all edges in $G - A$ remain covered, we only potentially covered some additional edges in D.

Note that this procedure does not introduce any bad triangles or new pair (C', w') for which C' is at $\mathcal{L}$-distance at least 4, $w' \in C'$ and the two cliques in $\mathcal{C}_A$ containing w' intersect C' in two vertices. As it also removes one such pair, we obtain a contradiction with the choice of A. We can therefore deduce that A does not contain such pair (C, w) and the lemma follows. ◀

Finally, we can state the main lemma of this section.

▶ **Lemma 15.** *There exists an optimal solution A for G and a clique partition witness $\mathcal{C}_A$ for $G - A$ such that for every clique $C \in \mathcal{C}$ at $\mathcal{L}$-distance at least 5 it holds that $C \in \mathcal{C}_A$.*

Proof. Let A be an optimal solution without any bad triangles and clique partition witness $\mathcal{C}_A$ for $G - A$ such that for every $C \in \mathcal{C}$ of $\mathcal{L}$-distance at least 4 and every $w \in C$, if C_1^w and C_2^w are the two cliques in $\mathcal{C}_A$ containing w, then either $C_1^w \cap C = \{w\}$ or $C_1^w \cap C = \{w\}$. Note that existence of such a solution is guaranteed by Lemma 14. Moreover let $(A, \mathcal{C}_A)$ be such an optimal solution satisfying properties in Lemma 14 that minimizes the number of cliques $C \in \mathcal{C}$ of $\mathcal{L}$-distance at least 5 such that $C \notin \mathcal{C}_A$. We claim that A satisfies the properties of the lemma.

For a contradiction let $C \in \mathcal{C}$ be a clique at $\mathcal{L}$-distance at least 5 and let $C_1, \ldots, C_p$ be the cliques in $\mathcal{C}_A$ that intersects C in at least 2 vertices. Since there is no bad triangle, it follows that $C_i \subseteq C$ for all $i \in [p]$ and by optimality of A, $p = 1$ (else $\bigcup_{i\in[p]} C_i$ is missing at least one edge). We claim that $C = C_1$. Else let $v \in C \setminus C_1$. Note that C is a small clique and hence by Observation 9 v does not have a neighbor in S. In particular all neighbors of v are covered by two cliques in $\mathcal{C}$, one of those cliques is C and let the other clique be C^v. Moreover, Let C_1^v and C_2^v be the two cliques in $\mathcal{C}_A$ containing v. Since $v \in C \setminus C_1$ both C_1^v and C_2^v are subsets of C^v. However, C^v is either a large clique and $\mathcal{C}_A$ contains C^v and the cliques C_1^v and C_2^v are C^v and $\{v\}$ respectively, or C^v is a small clique, in which case C^v is at $\mathcal{L}$-distance at least 4 from S, because it shares a vertex with the clique C at $\mathcal{L}$-distance at least 5 from S. It follows by the choice of A that either $C^v \cap C_1^v = \{v\}$ or $C^v \cap C_2^v = \{v\}$, but then again either C_1^v or C_2^v is the singleton $\{v\}$. However then the clique partition witness $(\mathcal{C}_A \setminus \{C_1, \{v\}\}) \cup \{C_1 \cup \{v\}\}$ defines a better solution. It follows that indeed $C \in \mathcal{C}_A$ for all cliques in $\mathcal{C}$ at $\mathcal{L}$-distance at least 5 in G. ◀

We are now ready to present our main reduction rule. Note that it would seem that we could remove just the vertices that do no appear in a clique at distance at most 4. However, because of the large cliques in at the first four levels, we would be potentially left with many cliques at $\mathcal{L}$-distance infinity that we cannot remove because all of their vertices are in a large clique at $\mathcal{L}$-distance at most 4 from S. While this case could have been dealt with separately, we can actually show a stronger claim, *i.e.,* that we can remove all edges from G that are covered by a clique at $\mathcal{L}$-distance at least 5 from S. Note that in this case we cannot easily claim that if (G, k) is YES-instance then so is the reduced instance and we crucially need the fact that cliques at $\mathcal{L}$-distance at least 5 are kept in clique partition witness of some optimal solution.

▶ **Reduction Rule 2.** *Remove all edges $uv \in E(G)$ such that $\{u, v\} \subseteq C$ for some clique C with $\text{dist}^{\mathcal{L}}(C) \geq 5$. Afterwards remove all isolated vertices from G.*

Let $\mathcal{D}$ be the set of cliques at $\mathcal{L}$-distance at least 5 from S, V_5 the set of vertices that appear in a clique in $\mathcal{D}$ and in a clique in $\mathcal{C} \setminus \mathcal{D}$ and G' be the graph obtained after applying the reduction rule and let $\mathcal{C}' = (\mathcal{C} \setminus \mathcal{D}) \cup \bigcup_{v\in V_5} \{v\}$. Note that $\mathcal{C}'$ is a clique partition witness for $G' - S$ and that $\{v\}$, for $v \in V_5$, is a clique at $\mathcal{L}$-distance at least 5.

Proof of safeness. Let $\mathcal{D}$, V_5, G', $\mathcal{C}'$ be as described above and let A be an optimal solution for G', that is $G' - A$ is a line graph, and let $\mathcal{C}_A$ be clique partition witness for $G' - A$. By Lemma 15, we can assume that $\bigcup_{v \in V_5}\{v\} \subseteq \mathcal{C}_A$. We will show that $(\mathcal{C}_A \setminus \bigcup_{v \in V_5}\{v\}) \cup \mathcal{D}$ is a clique partition witness for $G - A$. Clearly each edge in $G - A$ is either covered by $(\mathcal{C}_A \setminus \bigcup_{v \in V_5}\{v\})$ or by $\mathcal{D}$. It is also easy to see that every vertex is in precisely two cliques. Moreover, two cliques in $\mathcal{D}$ intersect in at most 1 vertex, because $\mathcal{D} \subseteq \mathcal{C}$ and similarly two cliques in $\mathcal{C}_A$ intersect in at most one vertex. Finally, let $D \in \mathcal{D}$ and $C \in (\mathcal{C}_A \setminus \bigcup_{v \in V_5}\{v\})$. Clearly, $D \cap C \subseteq V_5$. Moreover, for $\{u, v\} \subseteq D$, the edge uv is not in G' and hence $\{u, v\} \not\subseteq C$. Hence, $|D \cap C| \leq 1$.

On the other hand, let A be an optimal solution for G and a clique partition witness $\mathcal{C}_A$ for $G - A$ such that for every clique $C \in \mathcal{C}$ at $\mathcal{L}$-distance at least 5 it holds that $C \in \mathcal{C}_A$. Note that the existence of $(A, \mathcal{C}_A)$ is guaranteed by Lemma 15. We claim that $G' - A$ is a line graph. By the choice of $(A, \mathcal{C}_A)$, it follows that $\mathcal{D} \subseteq \mathcal{C}_A$. Moreover, for every edge e that is covered by a clique in $\mathcal{D}$ it holds that $e \notin E(G')$. It follows rather straightforwardly that $\mathcal{C}_A \setminus \mathcal{D} \cup \bigcup_{v \in V_5}\{v\}$ is indeed a clique partition witness for $G' - A$. ◀

5 Finishing the Proof

Suppose now that G, S, and $\mathcal{C}$ correspond to the instance after applying Reduction Rules 1 and 2. Clearly all cliques in $\mathcal{C}$ are either at $\mathcal{L}$-distance at most 4 from S or there are singletons at distance 5 or infinity, depending on whether the singleton intersects a small or a large clique, respectively. It follows from Lemma 10 that there are at most $\mathcal{O}\left(k^4\right)$ cliques at distance at most 4. We let M be any minimal w.r.t. inclusion set of vertices such that for every clique C in $\mathcal{C}$ at $\mathcal{L}$-distance at most 4 it holds that $|M \cap C| \geq \min\{|C|, k + 7\}$. Such a set M can be easily obtained by including arbitrary $\min\{|C|, k + 7\}$ vertices from every clique C at distance at most 4 and then removing the vertices v such that $|(M \setminus \{v\}) \cap C| \geq \min\{|C|, k + 7\}$ for all $C \in \mathcal{C}$ at $\mathcal{L}$-distance at most 4. From this construction it is easy to see that $|M| = \mathcal{O}\left(k^5\right)$.

▶ **Reduction Rule 3.** *Remove all vertices in $V(G) \setminus (S \cup M)$ from G.*

Proof of safeness. Let the clique partition witness $\mathcal{C}'$ for $G - (S \cup M)$ be $\{C \cap M \mid C \in \mathcal{C}, C \cap M \neq \emptyset\}$. Since line graphs are characterized by a finite set of forbidden induced subgraphs, it is easy to see that if $G - A$ is a line graph, for some $A \subseteq E(G)$, then $G[S \cup M] - A = (G - A)[S \cup M]$ is also a line graph. For the other direction, let $A \subseteq E(G)$ be such that $G[S \cup M] - A$ is line graph. We will show that $G - A$ is a line graph. Let $\mathcal{C}_A$ be a clique partition witness for $G[S \cup M] - A$. Now let $\mathcal{C}'_A$ be the set we obtain from $\mathcal{C}_A$ by adding to it all the singleton cliques in $\mathcal{C}$ that do not contain a marked vertex and for every clique $C \in \mathcal{C}_A$ for which there exists $C' \in \mathcal{C}$ with $C \setminus S \subseteq C'$, we replace C by $C' \cup (C \cap S)$.

First let us verify that every vertex in $V(G)$ is in precisely two cliques in $\mathcal{C}'_A$. It is easy to see that this holds for $v \in S \cup M$, because $\mathcal{C}_A$ is a clique partition witness for $G[S \cup M] - A$ and we only added new cliques containing vertices in $V(G) \setminus (M \cup S)$ or extended existing cliques in $\mathcal{C}_A$ by vertices in $V(G) \setminus (M \cup S)$. Now let $v \in V(G) \setminus M$ and let $C_1, C_2 \in \mathcal{C}$ be two cliques that contain v. Because all cliques in $\mathcal{C}$ at $\mathcal{L}$-distance at least 5 are singletons and we keep all vertices of the cliques at $\mathcal{L}$-distance at most 4 of size less than $k + 7$, it follows that C_1 and C_2 either both contain at least $k + 7$ vertices or one of them, say C_2, is a singleton and the other, C_1, contains at least $k + 7$ vertices. If C_2 is a singleton, then $C_2 \in \mathcal{C}'_A$. Else for C_i, $i \in \{1, 2\}$, with $|C_i| \geq k + 7$ there is $C'_i \in \mathcal{C}'$ with $|C'_i| \geq k + 7$ and $C'_i \subseteq C_i$. By Lemma 8, $\mathcal{C}_A$ contains a clique C_i^A such that $C_i^A \setminus S = C'_i \setminus C_i$. By the construction of $\mathcal{C}'_A$ it now follows

that $\mathcal{C}'_A$ contains $C_i^A \cup C_i$. From Lemma 7 it follows that if $u \in S$ is adjacent to at least 7 vertices in a clique in $\mathcal{C}$, then it is adjacent to the whole clique. Hence $C_i^A \cup C_i$ indeed induces a complete subgraph of $G - A$. It follows that v is indeed in precisely two cliques in $\mathcal{C}'_A$. Note that above also shows that the sets in $\mathcal{C}'_A$ induce cliques in $G - A$. Furthermore every edge in $G - A$ either has both endpoints in $S \cup M$ and are covered by a clique C in $\mathcal{C}_A$ such that $\mathcal{C}'_A$ contains a superset of C, or they are in the same clique of size at least $k + 7$ in $\mathcal{C}$ that is a subset of a clique in $\mathcal{C}'_A$ as well.

It remains to show that $|C_1 \cap C_2| \leq 1$ for all cliques in $\mathcal{C}'_A$. If $|C_1 \cap C_2| \geq 2$, then at least one of the vertices in $C_1 \cap C_2$ has to be outside $S \cup M$. But then from the above discussion follows that $C_1 \setminus S$ and $C_2 \setminus S$ are in $\mathcal{C}$, $|C_1 \setminus S| \geq k + 7$, $|C_2 \setminus S| \geq k + 7$ and at least $k + 7$ vertices from each of $C_1 \setminus S$ and $C_2 \setminus S$ are in $G[S \cup M]$. Clearly, $C_1 \setminus S$ and $C_2 \setminus S$ intersect in at most one vertex, let us denote it u, and the other vertices in the intersection of C_1 and C_2 are in S. Let v be arbitrary vertex in $C_1 \cap C_2 \cap S$. Note that v is adjacent to at least 7 vertices in both $C_1 \setminus S$ and $C_2 \setminus S$ and by Lemma 7 it is adjacent to all vertices in $(C_1 \cup C_2) \setminus S$. Since $G - (S \setminus \{v\})$ is a line graph, it follows that $G[(C_1 \cup C_2) \setminus (S \setminus \{v\})]$ is a line graph. Every vertex in $C_1 \setminus (S \cup \{u\})$ is in exactly one other clique in $\mathcal{C}$. This clique intersects $C_2 \setminus (S \cup \{u\})$ in at most one vertex. Therefore, there is a pair of vertices $w_1 \in C_1 \setminus (S \cup \{u\})$, $w_2 \in C_2 \setminus (S \cup \{u\})$ such that $w_1 w_2 \notin E(G)$. Now uvw_1 and uvw_2 are two odd triangles (any vertex in $C_i \setminus (S \cup \{u, w_i\})$ is adjacent to three vertices of the triangle uvw_i) that share a common edge, however uvw_1w_2 is not a K_4. Hence, $G[(C_1 \cup C_2) \setminus (S \setminus \{v\})]$ is not a line graph, a contradiction. It follows that if two cliques in $\mathcal{C}$ of size at least $k + 7$ intersect in a vertex in $G - S$, then no vertex in S is adjacent to both cliques and consequently no two cliques in $\mathcal{C}'_A$ intersect in at least two vertices.

It follows that $\mathcal{C}'_A$ is indeed a clique partition witness for $G - A$ and by point (2) in Theorem 2, $G - A$ is indeed a line graph. ◀

We are now ready to prove Theorem 1.

▶ **Theorem 1.** *Line-Graph Edge Deletion admits a kernel with $\mathcal{O}(k^5)$ vertices.*

Proof. We start the algorithm by finding the set S of at most $6k$ vertices such that for every $v \in S$ the graph $G - (S \setminus \{v\})$ is a line graph. This is simply done by greedily finding maximal set of pairwise edge-disjoint forbidden induced subgraphs. Afterwards, we construct a clique partition witness $\mathcal{C}$ for $G - S$ by using the algorithm of Lemma 6. Finally, we apply Reduction Rules 1, 2, and 3 in this order. By the discussion above Reduction Rule 3, after applying all the reduction rules, the resulting instance has $\mathcal{O}(k^5)$ vertices. The correctness of the kernelization algorithm follows from the safeness proofs of the reduction rules. ◀

6 Concluding Remarks

In this paper, we positively answered the open question from WorKer 2013 about kernelization of Line-Graph-Edge Deletion by giving a kernel for the problem with $\mathcal{O}(k^5)$ vertices. Our techniques crucially depend on the structural characterization of the line graphs. We believe that similar techniques could lead also to polynomial kernels for Line-Graph-Edge Addition and Line-Graph-Edge Editing. In particular, a result similar to Lemma 8 still holds when we allow addition of the edges. However, we were not able to bound the distance from S. Main difficulty seems to be the possibility of merging of some small cliques into one in a clique partition witness. It is also worth noting that the line graphs of multigraphs (*i.e.,* graphs that allow multiple edges between the same pair of vertices) have a similar structural characterization with the main difference being that the cliques in a clique partition witness

can intersect in more than just one vertex. The kernelization of the edge deletion (as well as addition or editing) to a line graph of a multigraph remains open as well. Finally, the kernelization of Claw-free Edge Deletion as well as of the edge deletion to some of the other natural subclasses of claw-free graphs remain wide open.

References

1 N. R. Aravind, R. B. Sandeep, and Naveen Sivadasan. Dichotomy results on the hardness of H-free edge modification problems. *SIAM J. Discrete Math.*, 31(1):542–561, 2017. doi:10.1137/16M1055797.

2 Lowell W. Beineke. Characterizations of derived graphs. *Journal of Combinatorial Theory*, 9(2):129–135, 1970. doi:10.1016/S0021-9800(70)80019-9.

3 Hans L Bodlaender, Leizhen Cai, Jianer Chen, Michael R Fellows, Jan Arne Telle, and Dániel Marx. Open problems in parameterized and exact computation-iwpec 2006. *UU-CS*, 2006, 2006.

4 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

5 Leizhen Cai and Yufei Cai. Incompressibility of H-free edge modification problems. *Algorithmica*, 71(3):731–757, 2015. doi:10.1007/s00453-014-9937-x.

6 Maria Chudnovsky and Paul Seymour. Claw-free graphs. IV. decomposition theorem. *Journal of Combinatorial Theory, Series B*, 98(5):839–938, 2008. doi:10.1016/j.jctb.2007.06.007.

7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

8 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, Erik Jan van Leeuwen, and Marcin Wrochna. Polynomial kernelization for removing induced claws and diamonds. *Theory Comput. Syst.*, 60(4):615–636, 2017. doi:10.1007/s00224-016-9689-x.

9 Daniele Giorgio Degiorgi and Klaus Simon. A dynamic algorithm for line graph recognition. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science, 21st International Workshop, WG '95, Aachen, Germany, June 20-22, 1995, Proceedings*, volume 1017 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 1995. doi:10.1007/3-540-60618-1_64.

10 R. Diestel. *Graph Theory, 4th Edition*. Springer, 2012.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

12 Eduard Eiben, William Lochet, and Saket Saurabh. A polynomial kernel for paw-free editing. *CoRR*, abs/1911.03683, 2019. arXiv:1911.03683.

13 Ehab S. El-Mallah and Charles J. Colbourn. The complexity of some edge deletion problems. *IEEE Transactions on Circuits and Systems*, 35(3):354–362, 1988. doi:10.1109/31.1748.

14 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

16 Sylvain Guillemot, Frédéric Havet, Christophe Paul, and Anthony Perez. On the (non-)existence of polynomial kernels for P_ℓ-free edge modification problems. *Algorithmica*, 65(4):900–926, 2013. doi:10.1007/s00453-012-9619-5.

17 Frank Harary. *Graph theory*. Addison-Wesley series in mathematics. Addison-Wesley Pub. Co., 1969.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

19 Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. *Discrete Optimization*, 10(3):193–199, 2013. doi:10.1016/j.disopt.2013.02.001.

20 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

21 Dániel Marx and R. B. Sandeep. Incompressibility of H-free edge modification problems: Towards a dichotomy. *CoRR*, abs/2004.11761, 2020. arXiv:2004.11761.

22 Najiba Sbihi. Algorithme de recherche d'un stable de cardinalite maximum dans un graphe sans etoile. *Discrete Mathematics*, 29(1):53–76, 1980. doi:10.1016/0012-365X(90)90287-R.

23 Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932. URL: http://www.jstor.org/stable/2371086.

24 Mihalis Yannakakis. Edge-deletion problems. *SIAM J. Comput.*, 10(2):297–309, 1981. doi:10.1137/0210021.

Approximate CVP_p in Time $2^{0.802\,n}$

Friedrich Eisenbrand
Ecole Polytechnique Fédérale de Lausanne, Switzerland
friedrich.eisenbrand@epfl.ch

Moritz Venzin
Ecole Polytechnique Fédérale de Lausanne, Switzerland
moritz.venzin@epfl.ch

Abstract

We show that a constant factor approximation of the shortest and closest lattice vector problem w.r.t. any ℓ_p-norm can be computed in time $2^{(0.802+\varepsilon)\,n}$. This matches the currently fastest constant factor approximation algorithm for the shortest vector problem w.r.t. ℓ_2. To obtain our result, we combine the latter algorithm w.r.t. ℓ_2 with geometric insights related to coverings.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Shortest and closest vector problem, approximation algorithm, sieving, covering convex bodies

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.43

Funding The authors acknowledge support from the Swiss National Science Foundation within the project Lattice Algorithms and Integer Programming (Nr. 200021-185030).

Acknowledgements The authors would like to thank the reviewers for their careful reviews and suggestions. The second author would like to thank Christoph Hunkenschröder, Noah Stephens-Davidowitz and Márton Naszódi for inspiring discussions.

1 Introduction

The *shortest vector problem* (SVP) and the *closest vector problem* (CVP) are important algorithmic problems in the geometry of numbers. Given a rational lattice

$$\mathcal{L}(B) = \{B\mathbf{x} \,:\, \mathbf{x} \in \mathbb{Z}^n\}$$

with $B \in \mathbb{Q}^{n\times n}$ and a target vector $\mathbf{t} \in \mathbb{Q}^n$ the closest vector problem asks for lattice vector $\mathbf{v} \in \mathcal{L}(B)$ minimizing $\|\mathbf{t} - \mathbf{v}\|$. The *shortest vector problem* asks for a *nonzero* lattice vector $\mathbf{v} \in \mathcal{L}(B)$ of minimal norm. When using the ℓ_p norms for $1 \le p \le \infty$, we denote the problems by SVP_p resp. CVP_p.

Much attention has been devoted to the hardness of approximating SVP and CVP. In a long sequence of papers, including [42, 7, 32, 10, 18, 28, 23] it has been shown that SVP and CVP are hard to approximate to within almost polynomial factors under reasonable complexity assumptions. The best polynomial-time approximation algorithms have exponential approximation factors [29, 41, 8].

The first algorithm to solve CVP for any norm that has exponential running time in the dimension only was given by Lenstra [30]. The running time of his procedure is $2^{O(n^2)}$ times a polynomial in the encoding length. In fact, Lenstra's algorithm solves the more general *integer programming* problem. Kannan [27] improved this to $n^{O(n)}$ time and polynomial space. It took almost 15 years until Ajtai, Kumar and Sivakumar presented a randomized algorithm for SVP_2 with time and space $2^{O(n)}$ and a $2^{O(1+1/\varepsilon)n}$ time and space algorithm for $(1+\varepsilon)$-CVP_2 [8, 9]. Here $(1+\varepsilon)$-CVP_2 is the problem of finding a lattice vector, whose

distance to the target is at most $1+\varepsilon$ times the minimal distance. Blömer and Naewe [14] extended the randomized sieving algorithm of Ajtai et al. to solve SVP_p and obtain a $2^{O(n)}$ time and space exact algorithm for SVP_p and an $O(1+1/\varepsilon)^{2n}$ time algorithm to compute a $(1+\varepsilon)$ approximation for CVP_p. For CVP_∞, one has a faster approximation algorithm. Eisenbrand et al. [20] showed how to boost any constant approximation algorithm for CVP_∞ to a $(1+\varepsilon)$-approximation algorithm in time $O(\log(1+1/\varepsilon))^n$. Recently, this idea was adapted in [36] to all ℓ_p norms, showing that $(1+\varepsilon)$ approximate CVP_p can be solved in time $O(1+1/\varepsilon)^{n/\min(2,p)}$ by boosting the deterministic CVP algorithm for general (even asymmetric) norms with a running time of $(1+1/\varepsilon)^n$ that was developed by Dadush and Kun [16].

The first deterministic singly-exponential time and space algorithm for exact CVP_2 (and SVP_2) was developed by [33]. The fastest exact algorithms for SVP_2 and CVP_2 run in time and space $2^{n+o(n)}$ [3, 1, 6]. Single exponential time and space algorithms for exact CVP are only known for ℓ_2. Whether CVP and the more general integer programming problem can be solved in time $2^{O(n)}$ is a prominent mystery in algorithms.

Recently there has been exciting progress in understanding the *fined grained complexity* of exact and constant approximation algorithms for CVP [2, 12, 5]. Under the assumption of the *strong exponential time hypothesis (SETH)* and for $p \neq 0 \pmod 2$, exact CVP_p cannot be solved in time $2^{(1-\varepsilon)d}$. Here d is the *ambient dimension* of the lattice, which is the number of vectors in a basis of the lattice. Under the assumption of a *gap-version* of the strong exponential time hypothesis *(gap-SETH)* these lower bounds also hold for the approximate versions of CVP_p. More precisely, for each $\varepsilon > 0$ there exists a constant $\gamma_\varepsilon > 1$ such that there exits no $2^{(1-\varepsilon)d}$ algorithm that computes a γ_ε-approximation of CVP_p.

Unfortunately, the currently fastest algorithms for CVP_p resp. SVP_p do not match these lower bounds, even for large approximation factors. These algorithms are based on randomized sieving, [8, 9]. Many lattice vectors are generated that are then, during many stages, subtracted from each other to obtain shorter and shorter vectors w.r.t. ℓ_p (resp. any norm) until a short vector is found. However, the algorithm needs to start out with sufficiently many lattice vectors just to guarantee that two of them are close. This issue directly relates to the *kissing number* (w.r.t. some norm) which is the maximum number of unit norm balls that can be arranged so that they touch another given unit norm ball. In the setting of sieving, this is the number of vectors of length r that are needed to guarantee that the difference of two of them is strictly smaller than r. Among all known upper bounds on the kissing numbers, the best (i.e. smallest) upper bound is known for ℓ_2 and equals $2^{0.401n}$, [26]. For ℓ_2 the fastest such approximation algorithms require time $2^{0.802n}$ - the square of the kissing number w.r.t. ℓ_2. For ℓ_∞ the kissing number equals 3^n-1 which is also an upper bound on the kissing number for any norm. The current best constant factor approximation algorithms for SVP_∞ and CVP_∞ require time 3^n, their counterparts w.r.t. ℓ_p require even more time, see [4, 35]. This then suggests the question, originally raised by Aggarwal et al. in [2] for ℓ_∞, whether the kissing number w.r.t. ℓ_p is a natural lower bound on the running time of SVP_p resp. CVP_p.

Our results indicate otherwise. For constant approximation factors, we are able to reduce these problems w.r.t. ℓ_p to another lattice problem but w.r.t. ℓ_2. This directly improves the running time of the algorithms for ℓ_p norms that hinge on the kissing number. Furthermore, given that the development of algorithms for ℓ_2 has been much more dynamic than for arbitrary ℓ_p norms and the difficulty of establishing hardness results for ℓ_2, there is hope to find still faster algorithms for SVP_2 that may not even rely on the kissing number w.r.t. ℓ_2. It is likely that this would then improve the situation for ℓ_p norms as well.

Our *main results* are resumed in the following theorem.

▶ **Theorem.** *For each $\varepsilon > 0$, there exists a constant γ_ε such that a γ_ε approximate solution to CVP_p, as well as to SVP_p for $p \in [1, \infty]$ can be found in time $2^{(0.802+\varepsilon)n}$.*

Our main idea is to use coverings in order to obtain a constant factor approximation to the shortest resp. closest vector w.r.t. ℓ_p by using a (approximate) shortest vector algorithm w.r.t. ℓ_2. We need to distinguish between the cases $p \in [2, \infty]$ and $p \in [1, 2)$. For $p \in [2, \infty]$, we show that exponentially many short vectors w.r.t. ℓ_2 cannot all have large pairwise distance w.r.t. ℓ_p. This follows from a bound on the number of ℓ_p norm balls scaled by some constant that are required to cover the ℓ_2 norm ball of radius $n^{1/2-1/p}$. The final procedure is then to sieve w.r.t. ℓ_2 and to pick the smallest non zero pairwise difference w.r.t. ℓ_p of the (exponentially many) generated lattice vectors. This yields a constant factor approximation to the shortest resp. closest vector w.r.t. ℓ_p, $p \in [2, \infty]$. For $p \in [1, 2)$, we use a more direct covering idea. There is a collection of at most $2^{\varepsilon n}$ balls w.r.t. ℓ_2, whose union contains the ℓ_p norm ball but whose union is contained in the ℓ_p norm ball scaled by some constant. This leads to a simple algorithm for ℓ_p norms ($p \in [1, 2)$) by using the approximate closest vector algorithm w.r.t. ℓ_2 from this paper.

This paper is organized as follows. In Section 2 we present the main idea for $p = \infty$ that also applies to the case $p \geq 2$. In Section 3 we first reintroduce the list-sieve method originally due to [34] but with a slightly more general viewpoint, we resume this in Theorem 4. We then present in detail our approximate CVP_∞ resp. SVP_∞ algorithm and extend this idea resp. algorithm to ℓ_p, $p \geq 2$. This is Theorem 5. Finally, in Section 4, using the covering technique from Section 2 and our approximate CVP_2 algorithm from Section 3.1, we show how to solve approximate CVP_p for $p \in [1, 2)$. This is Theorem 8.

2 Covering balls with boxes

We now outline our main idea in the setting of an approximate SVP_∞ algorithm. Let us assume that the shortest vector of $\mathcal{L}$ w.r.t. ℓ_∞ is $\mathbf{s} \in \mathcal{L} \setminus \{0\}$. We can assume that the lattice is scaled such that $\|\mathbf{s}\|_\infty = 1$ holds. The euclidean norm of $\mathbf{s}$ is then bounded by $\sqrt{n}$. Suppose now that there is a procedure that, for some constant $\gamma > 1$ independent of n, generates distinct lattice vectors $\mathbf{v}_1, \ldots, \mathbf{v}_N \in \mathcal{L}$ of length at most $\|\mathbf{v}_i\|_2 \leq \gamma\sqrt{n}$.

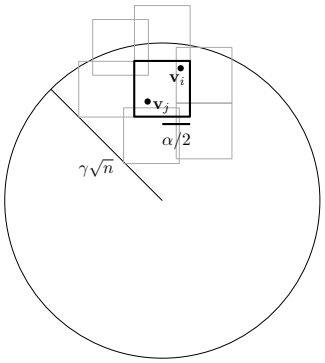


Figure 1 The difference $\mathbf{v}_i - \mathbf{v}_j$ is an α-approximate shortest vector w.r.t. ℓ_∞.

How large does the number of vectors N have to be such that we can guarantee that there exist two indices $i \neq j$ with

$$\|\mathbf{v}_i - \mathbf{v}_j\|_\infty \leq \alpha, \tag{1}$$

where $\alpha \geq 1$ is the approximation guarantee for SVP_∞ that we want to achieve? Suppose that N is larger than the minimal number of copies of the box $(\alpha/2)B_\infty^n$ that are required to cover the ball $\sqrt{n}B_2^n$. Here $B_p^n = \{x \in \mathbb{R}^n \,:\, \|x\|_p \leq 1\}$ denotes the unit ball w.r.t. the ℓ_p-norm. Then, by the pigeon-hole principle, two different vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ must be in the same box. Their difference satisfies (1) and thus is an α-approximate shortest vector w.r.t. ℓ_∞, see Figure 1.

Thus we are interested in the *translative covering number* $N(\sqrt{n}B_2^n, aB_\infty^n)$, which is the number of translated copies of the box aB_∞^n that are needed to cover the ℓ_2-ball of radius $\sqrt{n}$. In the setting above, a is the constant $\alpha/(2\gamma)$. For this procedure to be efficient, we need $N(\sqrt{n}B_2^n, aB_\infty^n)$ to be relatively small for a large enough - this is equivalent to decreasing the number of vectors N we need to generate by worsening (increasing) the approximation guarantee α. Since $2^{O(n)}\operatorname{vol}(B_\infty^n) = \operatorname{vol}(\sqrt{n}B_2^n)$ and by a simple covering argument, we have that $N(\sqrt{n}B_2^n, B_\infty^n) \leq 2^{Cn}$. This gives hope that by taking a large enough (but independent of n), we can decrease $N(\sqrt{n}B_2^n, aB_\infty^n)$ to, say, $2^{0.401n}$ or $2^{\varepsilon n}$ for $\varepsilon > 0$.

Covering problems like these have received considerable attention in the field of convex geometry, see [11, 37]. These techniques rely on the classical *set-cover problem* and the logarithmic integrality gap of its standard LP-relaxation, see, e.g. [43, 15]. To keep this paper self-contained, we briefly explain how this can be applied to our setting.

If we cover the finite set $(1/n)\mathbb{Z}^n \cap \sqrt{n}B_2^n$ with cubes whose centers are on the grid $(1/n)\mathbb{Z}^n$, then by increasing the side-length of those cubes by an additive $1/n$, one obtains a full covering of $\sqrt{n}B_2^n$. Thus we can focus on the corresponding *set-covering problem* with ground set $U = (1/n)\mathbb{Z}^n \cap \sqrt{n}B_2^n$ and sets

$$S_t = U \cap aB_\infty^n + t,\, t \in (1/n)\,\mathbb{Z}^n,$$

ignoring empty sets. An element of the ground set is contained in exactly $|(1/n)\mathbb{Z}^n \cap aB_\infty^n|$ many sets. Therefore, by assigning each element of the ground set the fractional value $1/|(1/n)\mathbb{Z}^n \cap aB_\infty^n|$, one obtains a feasible fractional covering. The weight of this fractional covering is

$$\frac{T}{|(1/n)\mathbb{Z}^n \cap aB_\infty^n|}$$

where T is the number of sets. Clearly, if a cube intersects $\sqrt{n}B_2^n$, then its center is contained in the *Minkowski sum* $\sqrt{n}B_2^n + aB_\infty^n$ and thus the weight of the fractional covering is

$$\frac{|(\sqrt{n}B_2^n + aB_\infty^n) \cap \frac{1}{n}\mathbb{Z}^n|}{|\frac{1}{n}\mathbb{Z}^n \cap aB_\infty^n|} = O\left(\frac{\operatorname{vol}(\sqrt{n}B_2^n + aB_\infty^n)}{\operatorname{vol}(aB_\infty^n)}\right)$$

Since the size of the ground-set is bounded by $n^{O(n)}$ and since the integrality gap of the set-cover LP is at most the logarithm of this size, one obtains

$$N(\sqrt{n}B_2^n, aB_\infty^n) \leq \operatorname{poly}(n)\,\frac{\operatorname{vol}(\sqrt{n}B_2^n + aB_\infty^n)}{\operatorname{vol}(aB_\infty^n)} \tag{2}$$

By *Steiner's formula*, see [22, 40, 24], the volume of $K + tB_2^n$ is a polynomial in t, with coefficients $V_j(K)$ only depending on the convex body K:

$$\operatorname{vol}(K + tB_2^n) = \sum_{j=0}^{n} V_j(K)\operatorname{vol}(B_2^{n-j})t^{n-j}$$

For $K = aB_\infty^n$, $V_j(K) = (2a)^j\binom{n}{j}$. Setting $t = \sqrt{n}$, the resulting expression has been evaluated in [25, Theorem 7.1].

▶ **Theorem 1** ([25]). *Denote by* H *the binary entropy function and let* $\phi \in (0,1)$ *the unique solution to*

$$\frac{1-\phi^2}{\phi^3} = \frac{2a^2}{\pi} \tag{3}$$

Then

$$\mathrm{vol}(aB_\infty^n + \sqrt{n}B_2^n) = O(2^{n[\mathrm{H}(\phi)+(1-\phi)\log(2a)+\frac{\phi}{2}\log(\frac{2\pi e}{\phi})]})$$

Using this bound in inequality (2) and simplifying, we find

$$N(\sqrt{n}B_2^n, aB_\infty^n) \leq \mathrm{poly}(n)\, 2^{n[\mathrm{H}(\phi)+\frac{\phi}{2}\log(\frac{2\pi e}{\phi})]}$$

Both $\mathrm{H}(\phi)$ and $\frac{\phi}{2}\log(\frac{2\pi e}{\phi})$ decrease to 0 as ϕ decreases to 0. Since ϕ, the unique solution to (3), satisfies $\phi \leq \sqrt[3]{(\pi/2)}a^{-\frac{2}{3}}$, we obtain the following bound.

▶ **Lemma 2.** *For each* $\varepsilon > 0$*, there exists* $a_\varepsilon \in \mathbb{R}_{>0}$ *independent of* n*, such that*

$$N(\sqrt{n}B_2^n, a_\varepsilon B_\infty^n) \leq 2^{\varepsilon n}.$$

Going back to the idea for an approximate SVP_∞ algorithm, we will use Lemma 2 with $\varepsilon = 0.401$. If we generate $2^{0.401n}$ distinct lattice vectors of euclidean length at most $\gamma\sqrt{n}$, then there must exist a pair of lattice vectors with pairwise distance w.r.t. ℓ_∞ shorter than $2\gamma a_{0.401}$. We find it by trying out all possible pairwise combinations, this takes time $2^{0.802n}$.

The main idea for approximate SVP_p is similar. Set $\tilde{\mathbf{s}}$ the shortest vector in $\mathcal{L}$ w.r.t. ℓ_p and scale the lattice so that $\|\tilde{\mathbf{s}}\|_p = 1$. The euclidean norm of $\tilde{\mathbf{s}}$ is bounded by $n^{1/2-1/p}$. Again, we can consider the question of how many different lattice vectors there have to be within a ball of radius $\gamma n^{1/2-1/p}$ so that we can guarantee that there exist two lattice vectors with constant pairwise distance w.r.t. ℓ_p. This leads us to consider the translative covering number $N(n^{1/2-1/p}B_2^n, aB_p^n)$. Since $n^{-1/p}B_\infty^n \subseteq B_p^n$, the following is immediate from Lemma 2.

▶ **Lemma 3.** *For each* $\varepsilon > 0$*, there exists* $a_\varepsilon \in \mathbb{R}_{>0}$ *independent of* n*, such that*

$$N(n^{1/2-1/p}B_2^n, a_\varepsilon B_p^n) \leq 2^{\varepsilon n}.$$

3 Approximate CVP_p for $p \geq 2$

We now describe our main contribution. As we mentioned already, SVP_2 can be approximated up to a constant factor in time $2^{(0.802+\varepsilon)n}$ for each $\varepsilon > 0$. This follows from a careful analysis of the *list sieve* algorithm of Micciancio and Voulgaris [34], see [31, 38]. The running time and space of this algorithm is directly related to the *kissing number* of the ℓ_2-norm. The running time is the square of the best known upper bound by Kabatiansky and Levenshtein [26].

The main insight of our paper is that the current list-sieve variants can be used to approximate SVP_p and CVP_p by testing all pairwise differences of the generated lattice vectors.

3.1 List sieve

We begin by describing the list-sieve method [34] to a level of detail that is necessary to understand our main result. Our exposition follows closely the one given in [38]. Let $\mathcal{L}(B)$ be a given lattice and $\mathbf{s} \in \mathcal{L}$ be an unknown lattice vector. This unknown lattice vector $\mathbf{s}$ is typically the shortest, respectively closest vector in $\mathcal{L}(B)$.

The list-sieve algorithm has two stages. The input to the *first stage* of the algorithm is an LLL-reduced lattice basis B of $\mathcal{L}(B)$, a constant $\varepsilon > 0$ and a guess μ on the length of $\mathbf{s}$ that satisfies

$$\|\mathbf{s}\|_2 \leq \mu \leq (1 + 1/n)\|\mathbf{s}\|_2. \tag{4}$$

The first stage then constructs a list of lattice vectors $L \subseteq \mathcal{L}(B)$ that is random. This list of lattice vectors is then passed on to the second stage of the algorithm.

The *second stage* of the algorithm proceeds by sampling points $\mathbf{y}_1, \ldots, \mathbf{y}_N$ uniformly and independently at random from the ball

$$(\xi_\varepsilon \cdot \mu)B_2^n,$$

where ξ_ε is an explicit constant depending on ε only. It then transforms these points via a deterministic algorithm $\texttt{ListRed}_L$ into lattice points

$$\texttt{ListRed}_L(\mathbf{y}_1), \ldots, \texttt{ListRed}_L(\mathbf{y}_N) \in \mathcal{L}(B).$$

The deterministic algorithm $\texttt{ListRed}_L$ uses the list $L \subseteq \mathcal{L}(B)$ from the first stage.

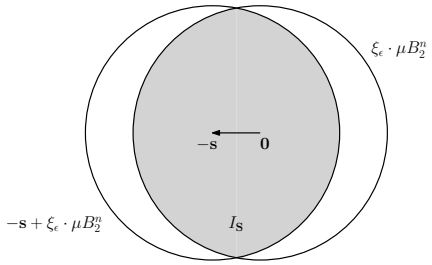


Figure 2 The lens $I_\mathbf{s}$.

As we mentioned above, the list $L \subseteq \mathcal{L}(B)$ that is used by the deterministic algorithm $\texttt{ListRed}_L$ is random. We will show the following theorem in the next section. The novelty compared to the literature is the reasoning about *pairwise differences* lying in *centrally symmetric* sets. In this theorem, $\varepsilon > 0$ is an arbitrary constant, ξ_ε as well as c_ε are explicit constants and K is some centrally symmetric set. Furthermore, we assume that μ satisfies (4).

The theorem reasons about an area $I_\mathbf{s}$ that is often referred as the *lens*, see Figure 2. The lens was introduced by Regev as a conceptual modification to facilitate the proof of the original AKS algorithm [39].

$$I_\mathbf{s} = (\xi_\varepsilon \cdot \mu)B_2^n \cap (-\mathbf{s} + (\xi_\varepsilon \cdot \mu)B_2^n) \tag{5}$$

▶ **Theorem 4.** *With probability at least* $1/2$*, the list* L *that was generated in the first stage satisfies the following. If* $\mathbf{y}_1, \cdots, \mathbf{y}_N$ *are chosen independently and uniformly at random within* $B_2^n(0, \xi_\varepsilon\mu)$ *then*

i) *The probability of the event that two different samples* $\mathbf{y}_i, \mathbf{y}_j$ *satisfy*

$$\mathbf{y}_i, \mathbf{y}_j \in I_\mathbf{s} \text{ and } \texttt{ListRed}_L(\mathbf{y}_i) - \texttt{ListRed}_L(\mathbf{y}_j) \in K$$

is at most twice the probability of the event that two different samples $\mathbf{y}_i, \mathbf{y}_j$ *satisfy*

$$\texttt{ListRed}_L(\mathbf{y}_i) - \texttt{ListRed}_L(\mathbf{y}_j) \in K + \mathbf{s}$$

ii) *For each sample* $\mathbf{y}_i$ *the probability of the event*

$$\|\texttt{ListRed}_L(\mathbf{y}_i)\|_2 \leq c_\varepsilon \|\mathbf{s}\|_2 \text{ and } \mathbf{y}_i \in I_\mathbf{s}$$

is at least $2^{-\varepsilon n}$.

The complete procedure, i.e. the construction of the list L in stage one and applying $\texttt{ListRed}_L$ to the N samples $\mathbf{y}_1, \ldots, \mathbf{y}_n$ in stage two takes time $N2^{(0.401+\varepsilon)n} + 2^{(0.802+\varepsilon)n}$ and space $N + 2^{(0.401+\varepsilon)n}$.

The proof of Theorem 4 follows verbatim from Pujol and Stehlé [38], see also [31]. In [38], $\mathbf{s}$ is a shortest vector w.r.t. ℓ_2. But this fact is never used in the proof and in the analysis. Part ii) follows from Lemma 5 and Lemma 6 in [38]. Their probability of a sample being in the lens $I_\mathbf{s} \subseteq \xi \|\mathbf{s}\|_2 B_2^n$ depends only on ξ (corresponding to our ξ_ε). By choosing ξ large enough, this happens with probability at least $2^{-\varepsilon n}$. Their Lemma 6 then guarantees that the list L, with probability $1/2$, when $\mathbf{y}_i \sim I_\mathbf{s}$ is sampled uniformly, returns a lattice vector of length at most $r_0 \|\mathbf{s}\|_2$ (r_0 corresponds to our c_ε). This corresponds to part ii) in our setting. The size of their list (denoted by N_T) is bounded above by $2^{(0.401+\delta)n}$ where $\delta > 0$ decreases to 0 as the ratio r_0/ξ increases, this is their Lemma 4.

Finally, part i) also follows from Pujol and Stehlé [38]. It is in their proof of correctness, Lemma 7, involving the lens $I_\mathbf{s}$. We briefly comment on our general viewpoint. Given $\mathbf{y} \sim (\xi \cdot \mu)B_2^n$, the algorithm computes the linear combination w.r.t. to the lattice basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$

$$\mathbf{y} = \sum_{i=1}^n \lambda_i \mathbf{b}_i$$

and then the *remainder*

$$\mathbf{y} \pmod{\mathcal{L}} = \sum_{i=1}^n \lfloor \lambda_i \rfloor \mathbf{b}_i.$$

The important observation is that this remainder is the same for all vectors $\mathbf{y} + \mathbf{v}$, $\mathbf{v} \in \mathcal{L}$. Next, it keeps reducing (minus) the remainder w.r.t. the list, as long as the length decreases. This results in a vector of the form

$$-\mathbf{y} \pmod{\mathcal{L}} - \mathbf{v}_1 - \cdots - \mathbf{v}_k, \text{ for some } \mathbf{v}_i \in L.$$

The output $\texttt{ListRed}_L(\mathbf{y})$ is then

$$-\mathbf{y} \pmod{\mathcal{L}} - \mathbf{v}_1 - \cdots - \mathbf{v}_k + \mathbf{y} \in \mathcal{L}.$$

The algorithm bases its decisions on $\mathbf{y} \pmod{\mathcal{L}}$ and not on $\mathbf{y}$ directly. This is why one can imagine that, after $\mathbf{y} \pmod{\mathcal{L}}$ has been created, one applies a bijection τ of the ball $\tau(\cdot) : \xi\mu B_2^n \to \xi\mu B_2^n$ on $\mathbf{y}$ with probability $1/2$. For $\mathbf{y} \in I_\mathbf{s}$ one has $\tau(\mathbf{y}) = \mathbf{y} + \mathbf{s}$. We refer to [38] for the definition of τ. Since τ is a bijection and preserves the measure, the result of applying $\tau(\mathbf{y})$ with probability $1/2$ is distributed uniformly. This means that for $\mathbf{y} \in I_s$ this modified but equivalent procedure outputs $\texttt{ListRed}_L(\mathbf{y})$ or $\texttt{ListRed}_L(\mathbf{y}) + \mathbf{s}$, both with probability $1/2$. If $\texttt{ListRed}_L(\mathbf{y}_i) - \texttt{ListRed}_L(\mathbf{y}_j) \in K$, we toss a coin for i and j each. With probability $1/2$, their difference is in $\pm K + \mathbf{s}$.

3.2 Approximation to CVP_p and SVP_p for $p \in [2, \infty]$

We now combine Theorem 4 with the covering ideas presented in Section 2.

▶ **Theorem 5.** *For $p \geq 2$, there is a randomized algorithm that computes with constant probability a constant factor (depending on ε) approximation to CVP_p and SVP_p respectively. The algorithm runs in time $2^{(0.802+\varepsilon)n}$ and it requires space $2^{(0.401+\varepsilon)n}$.*

In short, the algorithm is the standard list-sieve algorithm with a slight twist: *Check all pairwise differences.*
We first present in detail the case $p = \infty$. Even though there is an approximation preserving reduction from SVP to CVP, [21], we present separately the case SVP and CVP to highlight the ideas from Section 2 and Theorem 4. The case $p \geq 2$ then follows from this, we briefly comment on it.

Proof for $p = \infty$. We assume that the list L that was computed in the first stage satisfies the properties described in Theorem 4. Recall that this is the case with probability at least $1/2$.
We first consider SVP_∞. By Lemma 2, there is $a > 0$ such that $N(\sqrt{n}B_2^n, aB_\infty^n) \leq 2^{0.401n}$. Let $\mathbf{s}$ be a shortest vector w.r.t. ℓ_∞ and let $\mu > 0$ such that $\|\mathbf{s}\|_2 \leq \mu < (1 + \frac{1}{n}) \|\mathbf{s}\|_2$ as above. Since $\|\mathbf{s}\|_2 \leq \sqrt{n} \|\mathbf{s}\|_\infty$ we have $N(c_\varepsilon \|\mathbf{s}\|_2 B_2^n, c_\varepsilon a \|\mathbf{s}\|_\infty B_\infty^n) \leq 2^{0.401n}$. This means that, if $\lceil 2^{0.401n} \rceil + 1$ lattice vectors are contained in the ball $c_\varepsilon \|\mathbf{s}\|_2 B_2^n$ at least two of them have ℓ_∞-distance bounded by $2c_\varepsilon a$ which is a constant.

Set $N = 2 \cdot \lceil 2^{(\varepsilon+0.401)n} + 1 \rceil$ and $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\} \overset{iid}{\sim} B_2^n(0, \xi_\varepsilon \mu)$ uniformly and independently at random. By Theorem 4 ii) and by the Chebychev inequality, see [38], the following event has probability at least $1/2$.

(Event A): There is a subset $S \subseteq \{1, \ldots, N\}$ with $|S| = \lceil 2^{0.401n} \rceil + 1$ such that for each $i \in S$

$$\mathbf{y}_i \in I_{\mathbf{s}} \text{ and } \|\,\mathtt{ListRed}_L(\mathbf{y}_i)\|_2 \leq c_\varepsilon \|\mathbf{s}\|_2. \tag{6}$$

This event is the disjoint union of the event $A \cap B$ and $A \cap \overline{B}$, where B denotes the event where the vectors $\mathtt{ListRed}_L(\mathbf{y}_i)$, $\mathbf{y}_i \in I_{\mathbf{s}}$ are all distinct. Thus

$$\Pr(A) = \Pr(A \cap B) + \Pr(A \cap \overline{B}).$$

The probability of at least one of the events $A \cap B$ and $A \cap \overline{B}$ is bounded below by $1/4$. In the event $A \cap B$, there exists $i \neq j$ such that

$$\|\,\mathtt{ListRed}_L(\mathbf{v}_i) - \mathtt{ListRed}_L(\mathbf{v}_j)\|_\infty \leq 2c_\varepsilon a.$$

By Theorem 4 i) with $K = \{0\}$ one has

$$\Pr(A \cap \overline{B}) \leq 2 \Pr\left(\exists i \neq j \;:\; \mathtt{ListRed}_L(\mathbf{v}_i) - \mathtt{ListRed}_L(\mathbf{v}_j) = \mathbf{s}\right).$$

Therefore, with constant probability, there exist $i, j \in \{1, \ldots, N\}$ with

$$0 < \|\,\mathtt{ListRed}_L(\mathbf{y}_i) - \mathtt{ListRed}_L(\mathbf{y}_j)\|_\infty \leq 2c_\varepsilon a.$$

We try out all the pairs of N elements, which amounts to $N^2 = 2^{(0.802+\varepsilon')n}$ additional time.

We next describe how list-sieve yields a constant approximation for CVP_∞. Let $\mathbf{w} \in \mathcal{L}(B)$ be the closest lattice vector w.r.t. ℓ_∞ to $\mathbf{t} \in \mathbb{R}^n$ and let $\mu > 0$ such that $\|\mathbf{t} - \mathbf{w}\|_2 \leq \mu < (1 + \frac{1}{n}) \|\mathbf{t} - \mathbf{w}\|_2$. We use Kannan's embedding technique [27] and define a new lattice $\mathcal{L}'$ with basis

$$\widetilde{B} = \begin{pmatrix} B & \mathbf{t} \\ 0 & \frac{1}{n}\mu \end{pmatrix} \in \mathbb{Q}^{(n+1)\times(n+1)},$$

Finding the closest vector to $\mathbf{t}$ w.r.t. ℓ_∞ in $\mathcal{L}(B)$ amounts to finding the shortest vector w.r.t. ℓ_∞ in $\mathcal{L}'(\widetilde{B}) \cap \{\mathbf{x} \in \mathbb{R}^{n+1} \ : \ x_{n+1} = \frac{1}{n}\mu\}$. The vector $\mathbf{s} = (\mathbf{t} - \mathbf{w}, \frac{1}{n}\mu)$ is such a vector and its euclidean length is smaller than $(1 + \frac{1}{n})\mu$. Let $a > 0$ be such that

$$N(\sqrt{n}B_2^n, aB_\infty^n) \leq 2^{0.401n}.$$

This means that there is a covering of the n-dimensional ball $(c_\varepsilon \|\mathbf{s}\|_2)B_2^{n+1} \cap \{\mathbf{x} \in \mathbb{R}^{n+1} \ : \ x_{n+1} = 0\}$ by $2^{0.401n}$ translated copies of K, where

$$K = (c_\varepsilon \cdot a(1 + 1/n)\|\mathbf{s}\|_\infty)B_\infty^{n+1} \cap \{\mathbf{x} \in \mathbb{R}^{n+1} \ : \ x_{n+1} = 0\}. \tag{7}$$

(The factor $(1 + 1/n)$ is a reminiscent of the embedding trick, $\mathbf{s}$ is $n + 1$ dimensional.) Similarly, we may cover $(c_\varepsilon \|\mathbf{s}\|_2)B_2^{n+1} \cap \{\mathbf{x} \in \mathbb{R}^{n+1} \ : \ x_{n+1} = k \cdot \frac{\mu}{n}\}$ for all $k \in \mathbb{Z}$ (such that the intersection is not empty) by translates of K. There are only $2c_\varepsilon(n + 1) + 1$ such layers to consider and so $(2c_\varepsilon(n + 1) + 1)2^{0.401n}$ translates of K suffice. The last component of a lattice vector of $\mathcal{L}'$ is of the form $k \cdot \frac{\mu}{n}$ and it follows that these translates of K cover all lattice vectors of euclidean norm smaller than $c_\varepsilon \|\mathbf{s}\|_2$, see Figure 3.

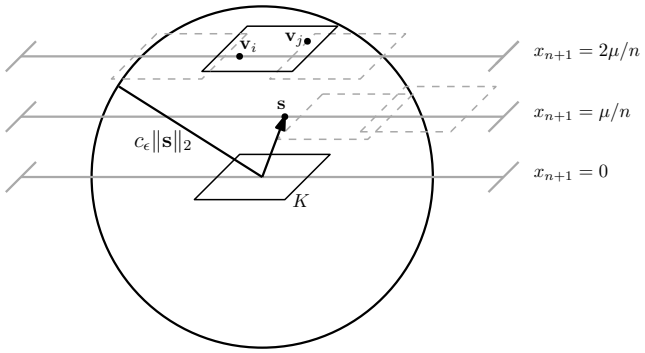


Figure 3 Covering the lattice points with translates of K.

.

Set $N = \lceil (2c_\varepsilon(n+1)+2)2^{(\varepsilon+0.401)n} \rceil$ and sample again $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\} \overset{iid}{\sim} B_2^n(0, \xi_\varepsilon \mu)$ uniformly and independently at random. By Theorem 4 ii) and by the Chebychev inequality, see [38], the following event has a probability at least $1/2$.

(Event A'): There is a subset $S \subseteq \{1, \ldots, N\}$ with $|S| = (2c_\varepsilon(n + 1) + 1)2^{0.401n} + 1$ such that for each $i \in S$

$$\mathbf{y}_i \in I_\mathbf{s} \text{ and } \|\,\mathtt{ListRed}_L(\mathbf{y}_i)\|_2 \leq c_\varepsilon \|\mathbf{s}\|_2. \tag{8}$$

In this case, there exists a translate of K that holds at least two vectors $\mathtt{ListRed}_L(\mathbf{y}_i)$ and $\mathtt{ListRed}_L(\mathbf{y}_j)$ for different samples $\mathbf{y}_i$ and $\mathbf{y}_j$, see Figure 3 with $\mathbf{v}_i, \mathbf{v}_j \in \mathcal{L}'$ instead. Thus, with probability at least $1/2$, there are $i, j \in [N]$ with $\mathbf{y}_i, \mathbf{y}_j \in I_\mathbf{s}$ such that

$$\mathtt{ListRed}_L(\mathbf{y}_i) - \mathtt{ListRed}_L(\mathbf{y}_j) \in 2K$$

Theorem 4 i) implies that, with probability at least $1/4$, there exist different samples $\mathbf{y}_i$ and $\mathbf{y}_j$ such that

$$\mathtt{ListRed}_L(\mathbf{y}_i) - \mathtt{ListRed}_L(\mathbf{y}_j) \in 2K + \mathbf{s}$$

In this case, the first n coordinates of $\mathtt{ListRed}_L(\mathbf{y}_i) - \mathtt{ListRed}_L(\mathbf{y}_j)$ can be written of the form $\mathbf{t} - \mathbf{v}$ for $\mathbf{v} \in \mathcal{L}$ and the first n coordinates on the right hand side are of the of the form $(\mathbf{t} - \mathbf{w}) + \mathbf{z}$, where $\mathbf{z} \in \mathcal{L}'$ and $\|\mathbf{z}\|_\infty \leq 2c_\varepsilon(1 + 1/n)a \|\mathbf{s}\|_\infty = 2c_\varepsilon(1 + 1/n)a \|\mathbf{t} - \mathbf{w}\|_\infty$. In particular, the lattice vector $\mathbf{v} \in \mathcal{L}$ is a $2ac_\varepsilon(1 + 1/n) + 1$ approximation to the closest vector to $\mathbf{t}$. Since we need to try out all pairs of the N elements, this takes time $N^2 = 2^{(0.802+\varepsilon')n}$ and space N. ◀

▶ Remark 6. For clarity we have not optimized the approximation factor. There are various ways to do so. We remark that for SVP_∞ we actually get a smaller approximation factor than the one that we describe. Let $\tilde{a}$ be such that $N(\sqrt{n}B_2^n, \tilde{a}B_\infty^n) \leq 2^{0.802n}$, the algorithm described above yields a $2c_\varepsilon\tilde{a}$ approximation instead of a $2c_\varepsilon a$ approximation to the shortest vector. This follows by applying the *birthday paradox* in the way that it was used by Pujol and Stehlé [38]. The same argument also applies to CVP_∞. Finally, we remark that in the case of SVP we have not really used property i) of Theorem 4. We only use this property to ensure that the generated vectors are different. It is plausible that this can be done more efficiently or with a better approximation factor.

Proof continued, $p \geq 2$. For SVP_p, $p \geq 2$, we define $\mathbf{s}$ to be shortest vector w.r.t. ℓ_p instead. Since $\|\mathbf{s}\|_2 \leq n^{1/2-1/p} \|\mathbf{s}\|_p$, we simply use Lemma 3 instead of Lemma 2 to conclude that there is some $a > 0$ such that if we have a set of $2^{0.401n}$ different lattice vectors of (euclidean) length smaller than $c_\varepsilon \|\mathbf{s}\|_2$, then two of them must have pairwise distance smaller than $2c_\varepsilon a$ w.r.t. ℓ_p.
For CVP_p, we define $\mathbf{w}$ to be the closest lattice vector to $\mathbf{t}$ w.r.t. ℓ_p. Both $\mathbf{s}$ and $\mathcal{L}'$ are defined analogously. We will need to replace the convex body K in (7) by

$$K = (c_\varepsilon \cdot a(1 + 1/n)\|\mathbf{s}\|_p)B_p^{n+1} \cap \{\mathbf{x} \in \mathbb{R}^{n+1} \ : \ x_{n+1} = 0\}.$$

The respective algorithms for SVP_p and CVP_p and the proof of correctness now follow from the case $p = \infty$. In particular, we can use the same parameters c_ε and a.

For the important case $p = 2$ we note that we can chose $a = 1$. This yields a approximation to the closest vector with the approximation guarantee c_ε matching that of the fastest approximate shortest vector problem w.r.t. ℓ_2, see [31]. ◀

4 Approximate CVP_p for $p \in [1, 2)$

In the previous section, we have extended the approximate SVP_2 solver to yield constant factor approximations to SVP_p and CVP_p for $p \in [2, \infty]$ in time $2^{(0.802+\varepsilon)n}$. From simple volumetric considerations, the technique from the previous section cannot be adapted to solve SVP_p and CVP_p for $p \in [1, 2)$ (in single exponential time). Instead, we can use a simple covering technique similar to the one considered by Eisenbrand et al. in [20]. We first show that for any constant $\varepsilon > 0$, there is a constant $a_\varepsilon > 0$, so that the crosspolytope B_1^n can be covered by $2^{\varepsilon n}$ balls (w.r.t. ℓ_2) with radius $(a_\varepsilon/\sqrt{n})$ and whose union is contained inside the crosspolytope scaled by a_ε. A similar covering also exists for B_p^n. Using the centers of these balls as targets, we can use the approximate CVP_2 algorithm to solve approximate CVP_1 resp. CVP_p. This is also similar to the technique of Dadush et al. in [17] resp. [16] where they cover general norm balls with M-ellipsoids to solve SVP and CVP w.r.t. to this norm by using the CVP_2 algorithm due to [33]. Unfortunately, there is only an upper bound of 2^{Cn} for some (large) constant $C > 0$ on the number of required M-ellipsoids, for our purpose we need a finer estimate. To achieve this, we rely on the set-covering idea and volume computations as outlined in Section 2. The following analogue to Lemma 2 is shown in the appendix.

▶ **Lemma 7.** *For each $\varepsilon > 0$, there exists $a_\varepsilon \in \mathbb{R}_{>0}$ independent of n such that*

$$\frac{\mathrm{vol}(B_1^n + (a_\varepsilon/\sqrt{n})B_2^n)}{\mathrm{vol}((a_\varepsilon/\sqrt{n})B_2^n)} \leq 2^{\varepsilon n}.$$

We now sketch the covering procedure for CVP_1 and SVP_1. Up to scaling the lattice and a guess on the distance of the closest (resp. shortest) lattice vector $\mathbf{v}$ to the target $\mathbf{t}$, we may assume that $1 - 1/n \leq \|\mathbf{v} - \mathbf{t}\|_1 \leq 1$ (resp. $1 - 1/n \leq \|\mathbf{v}\|_1 \leq 1$). We uniformly sample a point $\mathbf{x}$, [19], within $\mathbf{t} + B_1^n + (a_\varepsilon/\sqrt{n})B_2^n$ (set $\mathbf{t} = 0$ for SVP_1) and place a ball of radius $a_\varepsilon/\sqrt{n}$ around $\mathbf{x}$ (or $\mathbf{x}'$, the closest point to $\mathbf{x}$ in B_1^n, see Fig. 4).

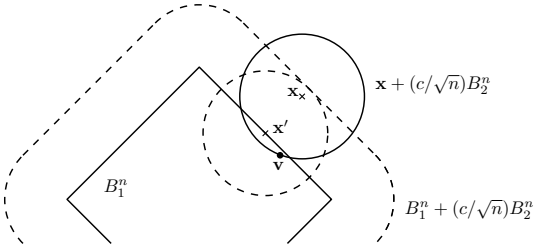


Figure 4 Generating a covering of B_1^n by $(c/\sqrt{n})B_2^n$.

.

By Lemma 7, with probability at least $2^{-\varepsilon n}$, $\mathbf{v}$ is covered by $\mathbf{x} + (a_\varepsilon/\sqrt{n})B_2^n$. Running the c-approximate (randomized) CVP_2 algorithm with target $\mathbf{x}$ (provided $\|\mathbf{v} - \mathbf{x}\|_2 \leq (a_\varepsilon/\sqrt{n})$), a lattice vector $\mathbf{w} \in \mathbf{x} + (c \cdot a_\varepsilon/\sqrt{n})B_2^n \subseteq \mathbf{t} + c \cdot (a_\varepsilon + 1)B_1^n$ is returned. The lattice vector $\mathbf{w}$ is thus a $c \cdot (a_\varepsilon + 1)$ approximation to the closest (resp. shortest) vector. In general, we run the c-approximate CVP_2 algorithm $O(\mathrm{poly}(n)2^{\varepsilon n})$ times with targets uniformly chosen within $\mathbf{t} + B_1^n + (a_\varepsilon/\sqrt{n})B_2^n$ and only output the closest of the resulting lattice vectors if it is within $c \cdot (a_\varepsilon + 1)B_1^n$. This ensures that, if there is lattice vector $\mathbf{v}$ in $\mathbf{t} + B_1^n$, a constant factor approximation to $\|\mathbf{t} - \mathbf{v}\|_1$ is found with high probability.
The same covering technique can be applied to B_p^n, $p \in (1, 2)$. By Hölder's inequality,

$$B_p^n \subseteq n^{1-1/p}B_1^n \text{ and } n^{1/2-1/p}B_2^n \subseteq B_p^n.$$

The first of these inclusions implies that for any $\varepsilon > 0$, we can pick the same constant a_ε as in Lemma 7 and cover B_p^n by at most $2^{\varepsilon n}$ translates of $a_\varepsilon n^{1/2-1/p}B_2^n$.

$$\frac{\mathrm{vol}(B_p^n + cn^{1/2-1/p}B_2^n)}{\mathrm{vol}(cn^{1/2-1/p}B_2^n)} \leq \frac{\mathrm{vol}(n^{1-1/p}B_1^n + cn^{1/2-1/p}B_2^n)}{\mathrm{vol}(cn^{1/2-1/p}B_2^n)} = \frac{\mathrm{vol}(B_1^n + (c/\sqrt{n})B_2^n)}{\mathrm{vol}((c/\sqrt{n})B_2^n)}$$

The second inclusion implies that these translates do not overlap B_p^n by more then a constant factor. It is then straightforward to adapt the boosting procedure described for CVP_1 to CVP_p. Using the approximate CVP_2 algorithm from the previous section then implies the following algorithm.

▶ **Theorem 8.** *There is a randomized algorithm that computes with constant probability a constant (depending on ε) factor approximation to CVP_p, $p \in [1, 2)$. The algorithm runs in time $2^{(0.802+\varepsilon)n}$ and requires space $2^{(0.401+\varepsilon)n}$.*

5 Proof of Lemma 7

Recall that the volume of $K + tB_2^n$ is a polynomial in t, with coefficients $V_j(K)$ that only depend on the convex body K:

$$\mathrm{vol}(K + tB_2^n) = \sum_{j=0}^{n} V_j(K)\,\mathrm{vol}(B_2^{n-j})t^{n-j}$$

The coefficients $V_j(K)$ are known as the intrinsic volumes of K. The intrinsic volumes of the crosspolytope B_1^n were computed by Betke and Henk in [13], and are given by the following formulae:

$$V_n(B_1^n) = \frac{2^n}{n!}$$

and for $0 \le j \le n-1$

$$V_j(B_1^n) = 2^n \binom{n}{j+1} \frac{\sqrt{j+1}}{j!\sqrt{\pi}^{n-j}} \cdot \int_0^\infty e^{-x^2} \left(\int_0^{x/\sqrt{j+1}} e^{-y^2}\,dy \right)^{n-j-1} dx$$

Given that the upper bound of Lemma 7 is exponential in n, we do not care about polynomial factors in n. For the sake of brevity, we will hide these polynomial factors by "$\lesssim$", i.e. $\mathrm{poly}(n) \lesssim 1$. This already simplifies the intrinsic volumes and, for $1 \le j \le n$:

$$V_j(B_1^n) \lesssim \frac{2^j}{j!}\binom{n}{j}$$

The volume of the $k-$dimensional ball B_2^k is given by

$$\mathrm{vol}(B_2^k) = \frac{\pi^{k/2}}{\Gamma(k/2+1)}$$

$\Gamma(\cdot)$ is the Gamma function. For $n \in \mathbb{N}$, we have $\Gamma(n+1) = n!$. By Stirling's formula we have the following estimate on $\Gamma(\cdot)$.

$$\left(\frac{z}{e}\right)^z \lesssim \Gamma(z+1) \lesssim \left(\frac{z}{e}\right)^z$$

With these estimates at hand, we can now prove Lemma 7.

$$\begin{aligned}
\frac{\mathrm{vol}(B_1^n + (c/\sqrt{n})B_2^n)}{\mathrm{vol}((c/\sqrt{n})B_2^n)} &= \frac{\sum_{j=0}^n V_j(B_1^n)\,\mathrm{vol}(B_2^{n-j})(c/\sqrt{n}))^{n-j}}{(c/\sqrt{n})^n\,\mathrm{vol}(B_2^n)} \\
&\lesssim \sum_{j=0}^{n} \frac{2^j\,n!}{j!(n-j)!j!} \frac{n^{j/2}}{c^j} \frac{\mathrm{vol}(B_2^{n-j})}{\mathrm{vol}(B_2^n)} \\
&\lesssim \sum_{j=0}^{n} \frac{(2e)^j\,n^n}{j^j\,(n-j)^{n-j}j^j} \frac{n^{j/2}}{c^j} \frac{n^{n/2}}{(n-j)^{(n-j)/2}(2\pi e)^{j/2}} \\
&\lesssim \sum_{j=0}^{n} \frac{n^{3n/2}n^{j/2}}{j^{2j}(n-j)^{3(n-j)/2}} \left(\frac{2e}{\pi c^2}\right)^{j/2} \\
&\overset{(j=\phi\,n)}{\lesssim} \max_{\phi\in[0,1]} \frac{e^{(3/2)\ln(n)n+\ln(n)n\phi/2}}{e^{2\ln(\phi n)\phi n+(3/2)\ln((1-\phi)n)(1-\phi)n}} \left(\frac{2e}{\pi c^2}\right)^{\phi n/2} \\
&\lesssim \max_{\phi\in[0,1]} e^{-2\ln(\phi)\phi n-2\ln(1-\phi)(1-\phi)n} \left(\frac{2e}{\pi c^2}\right)^{\phi n/2} \\
&= \max_{\phi\in[0,1]} 2^{2\,\mathrm{H}(\phi)n} \left(\frac{2e}{\pi c^2}\right)^{\phi n/2}
\end{aligned}$$

In passing to the second last line, we have added the factor $e^{-(1/2)\ln(1-\phi)(1-\phi)n}$ which is always greater than 1 for $\phi \in [0,1]$. $\mathrm{H}(\cdot)$ is the binary entropy function, i.e. $H(\phi) = -\ln(\phi)\phi - \ln(1-\phi)(1-\phi)$. $\mathrm{H}(\phi) \leq 1$ for $\phi \in [0,1]$ and $\mathrm{H}(\phi) = \mathrm{H}(1-\phi) \to 0$ monotonically as $\phi \to 0$. Thus, for some fixed c, the above expression reaches a maximum for some $\phi \in (0,1)$. If we increase c, we see that the ϕ^* realizing the maximum will decrease which then implies the lemma. This can be shown formally by fixing some c and taking a derivative w.r.t. ϕ. This will then show that the maximum is reached when $\phi^* = \Theta(\frac{1}{\sqrt{c}})$.
Thus, for any $\varepsilon > 0$, we can chose c large enough so that Lemma 7 holds.

References

1. D. Aggarwal, D. Dadush, and N. Stephens-Davidowitz. Solving the closest vector problem in 2^n time – the discrete gaussian strikes again! In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 563–582, October 2015. doi:10.1109/FOCS.2015.41.
2. Divesh Aggarwal, Huck Bennett, Alexander Golovnev, and Noah Stephens-Davidowitz. Fine-grained hardness of cvp (p) – everything that we can prove (and nothing else). *arXiv preprint*, 2019. arXiv:1911.02440.
3. Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2n time using discrete gaussian sampling. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 733–742, 2015.
4. Divesh Aggarwal and Priyanka Mukhopadhyay. Faster algorithms for SVP and CVP in the infinity norm. *CoRR*, abs/1801.02358, 2018. arXiv:1801.02358.
5. Divesh Aggarwal and Noah Stephens-Davidowitz. (gap/s)eth hardness of svp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 228–238, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188840.
6. Divesh Aggarwal and Noah Stephens-Davidowitz. Just take the average! An embarrassingly simple 2ˆn-time algorithm for SVP (and CVP). In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 12:1–12:19, 2018. doi:10.4230/OASIcs.SOSA.2018.12.
7. Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 10–19, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/276698.276705.
8. Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 601–610, 2001. doi:10.1145/380752.380857.
9. Miklós Ajtai, Ravi Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Québec, Canada, May 21-24, 2002*, pages 53–57, 2002. doi:10.1109/CCC.2002.1004339.
10. Sanjeev Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 1995. UMI Order No. GAX95-30468.
11. Shiri Artstein-Avidan and Boaz A Slomka. On weighted covering numbers and the levi-hadwiger conjecture. *Israel Journal of Mathematics*, 209(1):125–155, 2015.
12. H. Bennett, A. Golovnev, and N. Stephens-Davidowitz. On the quantitative hardness of cvp. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 13–24, October 2017. doi:10.1109/FOCS.2017.11.
13. Ulrich Betke and Martin Henk. Intrinsic volumes and lattice points of crosspolytopes. *Monatshefte für Mathematik*, 115(1):27–33, 1993. doi:10.1007/BF01311208.

14 Johannes Blömer and Stefanie Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. *Theor. Comput. Sci.*, 410(18):1648–1665, 2009. doi:10.1016/j.tcs.2008.12.045.

15 V. Chvatal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235, August 1979. doi:10.1287/moor.4.3.233.

16 Daniel Dadush and Gábor Kun. Lattice sparsification and the approximate closest vector problem. *Theory of Computing*, 12(1):1–34, 2016. doi:10.4086/toc.2016.v012a002.

17 Daniel Dadush, Chris Peikert, and Santosh S. Vempala. Enumerative lattice algorithms in any norm via m-ellipsoid coverings. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 580–589. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.31.

18 Irit Dinur, Guy Kindler, Ran Raz, and Shmuel Safra. Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica*, 23(2):205–243, 2003. doi:10.1007/s00493-003-0019-y.

19 Martin E. Dyer, Alan M. Frieze, and Ravi Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991. doi:10.1145/102782.102783.

20 Friedrich Eisenbrand, Nicolai Hähnle, and Martin Niemeier. Covering cubes and the closest vector problem. In *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 417–423, 2011. doi:10.1145/1998196.1998264.

21 O. Goldreich, D. Micciancio, S. Safra, and Jean-Pierre Seifert. Approximating shortest lattice vectors is not harder than approximating closet lattice vectors. *Inf. Process. Lett.*, 71(2):55–61, July 1999. doi:10.1016/S0020-0190(99)00083-6.

22 Peter Gruber. *Convex and Discrete Geometry*. Encyclopedia of Mathematics and its Applications. Springer, 2007.

23 Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 469–477, 2007.

24 Martin Henk, Jürgen Richter-Gebert, and Günter M Ziegler. Basic properties of convex polytopes. In *Handbook of discrete and computational geometry*, pages 243–270. CRC Press, 1997.

25 Varun Jog and Venkat Anantharam. A geometric analysis of the awgn channel with a (σ, ρ)-power constraint. *IEEE Transactions on Information Theory*, April 2015. doi:10.1109/TIT.2016.2580545.

26 Grigorii Anatol'evich Kabatiansky and Vladimir Iosifovich Levenshtein. On bounds for packings on a sphere and in space. *Problemy Peredachi Informatsii*, 14(1):3–25, 1978.

27 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

28 Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *J. ACM*, 52(5):789–808, September 2005. doi:10.1145/1089023.1089027.

29 A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. doi:10.1007/BF01457454.

30 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

31 Mingjie Liu, Xiaoyun Wang, Guangwu Xu, and Xuexin Zheng. Shortest lattice vectors in the presence of gaps. *IACR Cryptology ePrint Archive*, 2011:139, 2011.

32 Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM journal on Computing*, 30(6):2008–2035, 2001.

33 Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 351–358, 2010. doi:10.1145/1806689.1806739.

34 Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, page 1468–1480, USA, 2010. Society for Industrial and Applied Mathematics.

35 Priyanka Mukhopadhyay. Faster provable sieving algorithms for the shortest vector problem and the closest vector problem on lattices in ℓ_p norm. *CoRR*, abs/1907.04406, 2019. `arXiv:1907.04406`.

36 Márton Naszódi and Moritz Venzin. Covering convex bodies and the closest vector problem. *arXiv preprint*, 2019. `arXiv:1908.08384`.

37 Márton Naszódi. On some covering problems in geometry. *Proceedings of the American Mathematical Society*, 144, April 2014. `doi:10.1090/proc/12992`.

38 Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time 2 2.465n. *IACR Cryptology ePrint Archive*, 2009:605, January 2009.

39 Oded Regev. Lattices in computer science, lecture 8: $2^{O(n)}$ algorithm for svp, 2004.

40 Rolf Schneider. *Convex Bodies: The Brunn–Minkowski Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2 edition, 2013. `doi:10.1017/CBO9781139003858`.

41 Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53(2-3):201–224, 1987.

42 P. van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. *Technical Report 81-04, Mathematische Instituut, University of Amsterdam*, 1981.

43 Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

A $(1 - e^{-1} - \varepsilon)$-Approximation for the Monotone Submodular Multiple Knapsack Problem

Yaron Fairstein
Computer Science Department, Technion, Haifa, Israel
yyfairstein@gmail.com

Ariel Kulik
Computer Science Department, Technion, Haifa, Israel
kulik@cs.technion.ac.il

Joseph (Seffi) Naor
Computer Science Department, Technion, Haifa, Israel
naor@cs.technion.ac.il

Danny Raz
Computer Science Department, Technion, Haifa, Israel
danny@cs.technion.ac.il

Hadas Shachnai
Computer Science Department, Technion, Haifa, Israel
hadas@cs.technion.ac.il

Abstract

We study the problem of maximizing a monotone submodular function subject to a Multiple Knapsack constraint (SMKP) . The input is a set I of items, each associated with a non-negative weight, and a set of bins having arbitrary capacities. Also, we are given a submodular, monotone and non-negative function f over subsets of the items. The objective is to find a subset of items $A \subseteq I$ and a packing of these items in the bins, such that $f(A)$ is maximized.

SMKP is a natural extension of both Multiple Knapsack and the problem of monotone submodular maximization subject to a knapsack constraint. Our main result is a nearly optimal polynomial time $(1 - e^{-1} - \varepsilon)$-approximation algorithm for the problem, for any $\varepsilon > 0$. Our algorithm relies on a refined analysis of techniques for constrained submodular optimization combined with sophisticated application of tools used in the development of approximation schemes for packing problems.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases Sumodular Optimization, Multiple Knapsack, Randomized Rounding

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.44

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.12224`.

Funding *Joseph (Seffi) Naor*: This research was supported in part by US-Israel BSF grant 2018352 and by ISF grant 2233/19 (2027511).

1 Introduction

Submodular optimization has recently attracted much attention as it provides a unifying framework capturing many fundamental problems in combinatorial optimization, economics, algorithmic game theory, networking, and other areas. Furthermore, submodularity also captures many real world practical applications where economy of scale is prevalent. Classic examples of submodular functions are coverage functions [9], matroid rank functions [3] and graph cut functions [10]. A recent survey on submodular functions can be found in [1].

Submodular functions are defined over sets. Given a ground set I, a function $f : 2^I \to \mathbb{R}_{\geq 0}$ is called *submodular* if for every $A \subseteq B \subseteq I$ and $u \in I \setminus B$, $f(A+u)-f(A) \geq f(B+u)-f(B)$.[1] This reflects the diminishing returns property: the marginal value from adding $u \in I$ to a solution diminishes as the solution set becomes larger. A set function $f : 2^I \to \mathbb{R}$ is *monotone* if for any $A \subseteq B \subseteq I$ it holds that $f(A) \leq f(B)$. While in many cases, such as coverage and matroid rank function, the submodular function is monotone, this is not always the case (cut functions are a classic example).

The focus of this work is optimization of monotone submodular functions. In [19] Nemhauser and Wolsey presented a greedy based $(1 - e^{-1})$-approximation for maximizing a monotone submodular function subject to a cardinality constraint, along with a matching lower bound in the oracle model. A $(1 - e^{-1})$ hardness of approximation bound is also known for the problem under P $\neq$ NP, due to the hardness of max-k-cover [9] which is a special case. The greedy algorithm of [19] was later generalized to monotone submodular optimization with a knapsack constraint [16, 21].

A major breakthrough in the field was the continuous greedy algorithm presented in [22]. Initially used to derive a $(1 - e^{-1})$-approximation for maximizing a monotone submodular function subject to a matroid constraint, the algorithm has become a primary tool in the development of monotone submodular maximization algorithms subject to various other constraints. These include d-dimensional knapsack constraints [17], and combinations of d-dimensional knapsack and matroid constraints [7]. A variant of the continuous greedy algorithm for non-monotone functions is given in [11].

In the *multiple knapsack problem* (MKP) we are given a set of items, where each item has a weight and a profit, and a set of bins of arbitrary capacities. The objective is to find a packing of a subset of the items that respects the bin capacities and yields a maximum profit. The problem is one of the most natural extensions of the classic Knapsack problem that arises in the context of Virtual Machine (VM) allocation in cloud computing. The practical task is to assign VMs to physical machines such that capacity constraints are satisfied, while maximizing the profit of the cloud provider. A submodular cost function allows cloud providers to offer complex cost models to high-volume customers, where the price customers pay for each VM can depend on the overall number of machines used by the customer.

A *polynomial time approximation scheme* for MKP was first presented by Chekuri and Khanna [5]. The authors also ruled out the existence of a *fully polynomial time approximation scheme* for the problem. An *efficient* polynomial time approximation scheme was later developed by Jansen [14, 15].

1.1 Our Results

In this paper we consider the *submodular multiple knapsack problem* (SMKP). The input consists of a set of n items I and m bins B. Each item $i \in I$ is associated with a weight $w_i \geq 0$, and each bin $b \in B$ has a capacity $W_b \geq 0$. We are also given an oracle to a non-negative monotone submodular function $f : 2^I \to \mathbb{R}_{\geq 0}$. A feasible solution to the problem is a tuple of m subsets $(A_b)_{b \in B}$ such that for every $b \in B$ it holds that $\sum_{i \in A_b} w_i \leq W_b$. The value of a solution $(A_b)_{b \in B}$ is $f\left(\bigcup_{b \in B} A_b\right)$. The goal is to find a feasible solution of maximum value.[2]

[1] Equivalently, for every $A, B \subseteq I$: $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$.

[2] We note that the set of bins B is part of the input for SMKP, thus the number of bins is non-constant. This is one difference between SMKP and the problem of maximizing a submodular set function subject to d knapsack constraints (or, a d-dimensional knapsack constraint) where d is fixed (for more details see, e.g., [17]).

The problem is a natural generalization of both Multiple Knapsack [5] (where f is modular or linear), and the problem of monotone submodular maximization subject to a knapsack constraint [21] (where $m = 1$). Our result is stated in the next theorem.

▶ **Theorem 1.** *For any $\varepsilon > 0$, there is a randomized $(1 - e^{-1} - \varepsilon)$-approximation algorithm for SMKP.*

As mentioned above, a $(1 - e^{-1})$ hardness of approximation bound is known for the problem under P $\neq$ NP, due to the hardness of max-k-cover [9] which is a special case of SMKP. This is a vast improvement over previous results. Feldman presented in [12] a $\left(\frac{e-1}{3e-1} - o(1)\right) \approx 0.24$-approximation for the special case of identical bin capacities, along with a $\frac{1}{9}$-approximation for general capacities. To the best of our knowledge, this is the best known approximation ratio for the problem.[3]

In very recent work, carried out independently of our work, Sun et al. [20] present a *deterministic* greedy based $(1 - e^{-1} - \varepsilon)$-approximation for the special case of identical bins. They also derive a randomized $(1 - e^{-1} - \varepsilon)$-approximation for the special case where the ratio between the capacity of the largest and smallest bins is a constant. In this paper we give a randomized algorithm for the most general case, based on a different approach (as described below).

1.2 Tools and Techniques

Our algorithm relies on a refined analysis of techniques for submodular optimization subject to d-dimensional knapsack constraints [17, 4, 7], combined with sophisticated application of tools used in the development of approximation schemes for packing problems [8].

At the heart of our algorithm lies the observation that SMKP for a large number of identical bins (i.e., $\forall b \in B,\ W_b = W$ for some $W \geq 0$) can be easily approximated via a reduction to the problem of maximizing a submodular function subject to a 2-dimensional knapsack constraint (see, e.g., [17]). Given such an SMKP instance and $\varepsilon > 0$, we partition the items to *small* and *large*, where an item $i \in I$ is small if $w_i \leq \varepsilon W$ and large otherwise. We further define a *configuration* to be a subset of large items which fits into a single bin, and let $\mathcal{C}$ be the set of all configurations. It follows that for fixed $\varepsilon > 0$, the number of configurations is polynomial.

Using the above we define a new submodular optimization problem, to which we refer as the *block-constraint* problem. We define a new universe E which consists of all configurations $\mathcal{C}$ and all small items, $E = \mathcal{C} \cup \{\{i\} |\ i \text{ is small}\}$. We also define a new submodular function $g : 2^E \to \mathbb{R}_{\geq 0}$ by $g(T) = f\left(\bigcup_{A \in T} A\right)$. Now, we seek a subset of elements $T \subseteq E$ such that T has at most $m = |B|$ configurations, i.e., $|T \cap \mathcal{C}| \leq m$, and the total weight of sets selected is at most $m \cdot W$; namely, $\sum_{A \in T} w(A) \leq m \cdot W$, where $w(A) = \sum_{i \in A} w_i$.

It is easy to see that the optimal value of the block-constraint problem is at least the value of the optimum for the original instance. Moreover, a solution T for the block-constraint problem can be used to generate a solution for the SMKP instance with only a small loss in value. As there are no more than m configurations, and all other items are small, the items in T can be easily packed into $(1+\varepsilon)m+1$ bins of capacity W using First Fit. Then, it is possible

[3] Sun et al. [20] indicate that a $\left(1 - e^{1-e^{-1}} - o(1)\right) \approx 0.468$-approximation for the problem can be derived using the techniques of [4]. We note that this derivation is non-trivial (no details were given in [4]).

to remove $\varepsilon m + 1$ of the bins while maintaining at least $\frac{m}{\varepsilon m+1} \geq \frac{1}{1+2\varepsilon}$ of the solution value, for $m \geq \frac{1}{\varepsilon}$. Once these $\varepsilon m + 1$ bins are removed, we have a feasible solution for the SMKP instance. The block-constraint problem can be viewed as monotone submodular optimization subject to a 2-dimensional knapsack constraint. Thus, a $(1 - e^{-1} - \varepsilon)$-approximate solution can be found efficiently [17].

Our approximation algorithm for SMKP is based on a generalization of the above. We refer to a set of bins of identical capacity as a *block*, and show how to reduce an SMKP instance into a submodular optimization problem with a d-dimensional knapsack constraint, in which d is twice the number of blocks plus a constant. While, generally, this problem cannot be solved for non-constant d, we use a refined analysis of known algorithms [17, 7] to show that the problem can be efficiently solved if the blocks admit a certain structure, to which we refer as *leveled*.

We utilize a grouping technique, inspired by the work of Fernandez de la Vega and Lueker [8], to convert a general SMKP instance to a leveled instance. We sort the bins in decreasing order by capacity and then partition them into levels, where level t, $t \geq 0$, has N^{2+t} bins, divided into N^2 consecutive blocks, each containing N^t bins. We decrease the capacity of each bin to the smallest capacity of a bin in the same block. While the decrease in capacity generates the leveled structure required for our algorithm to work, it only slightly decreases the optimal solution value. The main idea is that given an optimal solution, each block of decreased capacity can now be used to store the items assigned to the subsequent block on the same level. Also, the items assigned to N blocks from each level can be evicted, while only causing a reduction of $\frac{1}{N}$ to the profit (as only N of the N^2 blocks of the level are evicted). These evicted blocks are then used for the items assigned to the first block in the next level.

2 Preliminaries

Our analysis utilizes several basic properties of submodular functions. Given a monotone submodular function $f : 2^I \to \mathbb{R}_{\geq 0}$ and a set $S \subseteq I$, we define $f_S : 2^I \to \mathbb{R}_{\geq 0}$ by $f_S(A) = f(S \cup A) - f(S)$. It follows that f_S is a monotone, non-negative submodular function. The proof of the next claim is given in Appendix A.

▷ Claim 2. Let $f : 2^I \to \mathbb{R}_{\geq 0}$ be a non-negative, monotone and submodular function, and let $E \subseteq 2^I \times X$ for some set X (each element of E is a pair (S, h) with $S \subseteq I$ and $h \in X$). Then function $g : 2^E \to \mathbb{R}_{\geq 0}$ defined by $g(A) = f\left(\cup_{(S,h)\in A} S\right)$ is non-negative, monotone and submodular.

While Claim 2 is essential for our algorithm, it is important to emphasize it does not hold for non-monotone submodular functions.

Many modern submodular optimization algorithms rely on the submodular *Multilinear Extension* ([3, 17, 18, 23, 11, 2]). Given a function $f : 2^I \to \mathbb{R}_{\geq 0}$, its multilinear extension is $F : [0, 1]^I \to \mathbb{R}_{\geq 0}$ defined as:

$$F(\bar{x}) = \sum_{S \subseteq I} f(S) \prod_{i \in S} \bar{x}_i \prod_{i \in I \setminus S} (1 - \bar{x}_i).$$

The multilinear extension can be interpreted as an expectation of a random variable. Given $\bar{x} \in [0, 1]^I$ we say that a random set X is distributed according to $\bar{x}$, $X \sim \bar{x}$, if $\Pr(i \in X) = \bar{x}_i$ and the events $(i \in X)_{i \in I}$ are independent. It follows that $F(\bar{x}) = \mathbb{E}_{X \sim \bar{x}}[f(X)]$.

The continuous greedy of [3] can be used to find approximate solution for maximization problems of the form $\max F(\bar{x})$ s.t. $\bar{x} \in P$, where F is the multilinear extension of a monotone submodular function f, and P is a down-monotone polytope. The algorithm uses two oracles, one for f and another which given $\bar{\lambda} \in \mathbb{R}^I$ returns a vector $\bar{x} \in P$ such that $\bar{x} \cdot \bar{\lambda}$ is maximal. The algorithm returns $\bar{x} \in P$ such that $F(\bar{x}) \geq (1 - e^{-1}) \max_{\bar{y} \in P} F(\bar{y})$.

We use $\mathcal{I} = (I, w, B, W, f)$ to denote an SMKP instance consisting of a set of items I with weights w_i for $i \in I$, a set of bins B with capacities W_b for $b \in B$, and objective function f. Given a set $A \subseteq I$, let $w(A) = \sum_{i \in A} w_i$. We denote by $\mathrm{OPT}(\mathcal{I})$ the optimal solution value for the instance $\mathcal{I}$.

3 The Approximation Algorithm

In this section we present our approximation algorithm for SMKP. Given an instance $\mathcal{I}$ of the problem, let $A^* = \cup_{b \in B} A^*_b$ be an optimal solution of value $OPT(\mathcal{I})$. We first observe that there exists a constant size subset $A = \cup_{b \in B} A_b$, where $A_b \subseteq A^*_b$, satisfying the following property: the value gained from any item in $i \in A^* \setminus A$ is small relative to $OPT(\mathcal{I})$. Thus, our algorithm initially enumerates over all possible partial assignments of constant size. Each assignment is then extended to an approximate solution for $\mathcal{I}$. Among all possible partial assignments and the respective extensions the algorithm returns the best solution. Thus, from now on we restrict our attention to finding a solution for the *residual* problem, obtained by fixing the initial partial assignment.

Formally, given an SMKP instance, $\mathcal{I} = (I, w, B, W, f)$, a feasible partial solution $(A_b)_{b \in B}$ and $\xi \in \mathbb{N}$, we define the *residual* instance $\mathcal{I}' = (I', w, B, W', f')$ with respect to $(A_b)_{b \in B}$ and ξ as follows. Let $A = \cup_{b \in B} A_b$ and set $I' = \left\{ i \in I \setminus A \;\middle|\; f_A(\{i\}) \leq \frac{f(A)}{\xi} \right\}$. The weights of the items remain the same and so is the set of bins. For every $b \in B$ we set $W'_b = W_b - w(A_b)$. Finally, the objective function of the residual instance is $f' = f_A$.

▶ **Lemma 3.** *Let $\mathcal{I}$ be an SMKP instance, $\xi \in \mathbb{N}$, and $(A^*_b)_{b \in B}$ an optimal solution for $\mathcal{I}$ such that $A^*_{b_1} \cap A^*_{b_2} = \emptyset$ for any $b_1, b_2 \in B$, $b_1 \neq b_2$. If $\sum_{b \in B} |A^*_b| \geq \xi$ there is a feasible solution $(A_b)_{b \in B}$ for $\mathcal{I}$ such that $A_b \subseteq A^*_b$ for any $b \in B$, $\sum_{b \in B} |A_b| = \xi$, and $(A^*_b \setminus A_b)_{b \in B}$ is a feasible solution for the residual instance of $\mathcal{I}'$ w.r.t $(A_b)_{b \in B}$ and ξ.*

Proof. Let $(A^*_b)_{b \in B}$ be an optimal solution to the SMKP instance. Define $A^* = \cup_{b \in B} A^*_b$ and order the items of A^* by their *marginal values*. That is $A^* = \{a_1, \ldots, a_r\}$ where $f_{T_{\ell-1}}(\{a_\ell\}) = \max_{a \in A^* \setminus T_{\ell-1}} f_{T_{\ell-1}}(\{a\})$ with $T_\ell = \{a_1, \ldots, a_\ell\}$ for every $1 \leq \ell \leq r$ (also, $T_0 = \emptyset$). Define $(A_b)_{b \in B}$ by $A_b = A^*_b \cap \{a_1, \ldots, a_\xi\}$ for every $b \in B$ and $A = \cup_{b \in B} A_b$. We therefore have $A = \{a_1, \ldots, a_\xi\}$.

For any $b \in B$ it holds that $w(A_b) \leq w(A^*_b) \leq W_b$, and thus $(A_b)_{b \in B}$ is a feasible solution for $\mathcal{I}$. Furthermore, for any $b \in B$ it holds that $A_b \subseteq A^*_b$ by definition. As the sets $(A^*_b)_{b \in B}$ are disjoint it follows that $\sum_{b \in B} |A_b| = \xi$.

Let $\mathcal{I}' = (I', w, B, W', f')$ be the residual instance of $\mathcal{I}$ w.r.t $(A_b)_{b \in B}$ and ξ. We are left to show that $(A^*_b \setminus A_b)_{b \in B}$ is a feasible solution for $\mathcal{I}'$. For every $\xi < i \leq r$ and $1 \leq \ell \leq \xi$ it holds that $f_A(\{a_i\}) \leq f_{T_{\ell-1}}(\{a_i\}) \leq f_{T_{\ell-1}}(\{a_\ell\})$ where the first inequality follows from the submodularity of f and the second by the definition of a_ℓ. Combining the last inequality with $f' = f_A$ we obtain,

$$\xi \cdot f'(\{a_i\}) = \xi \cdot f_A(\{a_i\}) \leq \sum_{\ell=1}^{\xi} f_{T_{\ell-1}}(\{a_\ell\}) = f(A) - f(\emptyset) \leq f(A).$$

Thus, $a_i \in I'$, implying that $A^*_b \setminus A_b \subseteq I'$ for any $b \in B$. Furthermore, for any $b \in B$,

$$w(A^*_b \setminus A_b) = w(A^*_b) - w(A_b) \leq W_b - w(A_b) = W'_b.$$

It follows that $(A^*_b \setminus A_b)_{b \in B}$ is a solution to the residual instance. ◀

Next, we observe that instances of SMKP are easier to solve when the number of distinct bin capacities is small (e.g., uniform bin capacities), leading us to consider *bin blocks*:

▶ **Definition 4.** *For a given instance of SMKP we say that a subset of bins $\tilde{B} \subseteq B$ is a* block *if all the bins in $\tilde{B}$ have the same capacity, i.e., for bins b_1 and b_2 belonging to the same block it holds that $W_{b_1} = W_{b_2}$.*

Following an enumeration over partial assignments, our algorithm reduces the number of blocks by altering the bin capacities. To this end, we use a specific structure that we call *leveled*, defined as follows.

▶ **Definition 5.** *For any $N \in \mathbb{N}$, we say that a partition $(B_j)_{j=0}^k$ of a set B of bins with capacities $(W_b)_{b \in B}$ is* N-leveled *if B_j is a block, and $|B_j| = N^{\lfloor \frac{j}{N^2} \rfloor}$ for all $0 \leq j \leq k$.*

By the above definition, we can view each set of consecutive blocks of the same size as a *level*. For $0 \leq j \leq k$, block j belongs to level $\ell = \lfloor \frac{j}{N^2} \rfloor$. Thus, for $\ell \geq 0$, the number of bins in each block increases by factor of N when moving from level ℓ to level $\ell + 1$.

In Section 3.1 we give Algorithm 2 which generates an N-leveled partition of the bins, $\tilde{B} = \cup_{j=0}^k \tilde{B}_j$ with the capacities of the bins $(W_b)_{b \in B}$ modified to $(\tilde{W}_b)_{b \in \tilde{B}}$. We show that solving the problem with these new bin capacities may cause only a small harm to the optimal solution value. In particular, we prove (in Section 3.1) the following.

▶ **Lemma 6.** *Given a set of bins B, capacities $(W_b)_{b \in B}$ and a parameter N, Algorithm 2 returns in polynomial time a subset of bins $\tilde{B} \subseteq B$, capacities $(\tilde{W}_b)_{b \in \tilde{B}}$ and an N-leveled partition $(\tilde{B}_j)_{j=0}^k$ of $\tilde{B}$, such that*

1. *The bin capacities satisfy $\tilde{W}_b \leq W_b$, for all $b \in \tilde{B}$.*
2. $\mathrm{OPT}(\tilde{\mathcal{I}}) \geq \left(1 - \frac{1}{N}\right) \mathrm{OPT}(\mathcal{I})$, *for any SMKP instance $\mathcal{I} = (I, w, B, W, f)$ and $\tilde{\mathcal{I}} = (I, w, \tilde{B}, \tilde{W}, f)$.*

Once the instance is N-leveled, we proceed to solve the problem (fractionally) and apply randomized rounding to obtain an integral solution (see Section 3.2). Algorithm 4 utilizes efficiently the leveled structure of the instance. Instead of having a separate constraint for each bin in a block − to bound the total size of the items packed in this bin − we use only two constraints for each block. The first constraint is a knapsack constraint referring to the total capacity of a block, and the second constraint restricts the number of *configurations* assigned to the block.[4] Thus, the number of constraints significantly decreases if the blocks are large. Since leveled instances also have a constant number of blocks consisting of a *single* bin, those are handled separately via the notion of δ-restricted SMKP.

An input for *δ-restricted SMKP* includes the same parameters as an input for SMKP, and also a subset $B^r \subseteq B$ of *restricted* bins. A solution for δ-restricted SMKP is a feasible assignment $(A_b)_{b \in B}$ satisfying also the property that $\forall b \in B^r$ the items assigned to b are relatively small; namely, for any $i \in A_b$ $w_i \leq \delta W_b$.

[4] We defined a configuration in Section 1.2.

Given the N-leveled instance of our problem, we turn the blocks of a single bin (that is blocks $\tilde{B}_j$ such that $|\tilde{B}_j| = 1$) to be *restricted*. We note that while items of weight greater than δW_b may be assigned to these blocks in some optimal solution, the overall number of such items is bounded by a constant. Indeed, our initial enumeration guarantees that evicting these items from an optimal solution may cause only small harm to the optimal solution value, allowing us to consider the instance as δ-restricted.

In Section 3.2 we show the following bound on the performance guarantee of Algorithm 4, which uses randomized rounding. The algorithm is parameterized by $\mu \in (0, 0.1)$ (to be determined). Suppose we are given a δ-restricted SMKP instance $\mathcal{I}$, such that the unrestricted bins are partitioned into blocks, i.e., $B \setminus B^r = B_1 \cup \ldots \cup B_k$, and $\upsilon = \max_{i \in I} f(\{i\}) - f(\emptyset)$.

▶ **Lemma 7.** *For $\mu \in (0, 0.1)$, Algorithm 4 returns a feasible solution $(S_b)_{b \in B}$ such that $\mathbb{E}\left[f(\cup_{b \in B} S_b)\right] \geq (1 - e^{-1})\frac{(1-\mu)^2}{1+\mu}(1 - \gamma)\mathrm{OPT}(\mathcal{I})$, where*

$$\gamma = \exp\left(-\frac{\mu^3}{16} \cdot \frac{\mathrm{OPT}(\mathcal{I})}{\upsilon}\right) + |B^r| \exp\left(-\frac{\mu^2}{12} \cdot \frac{1}{\delta}\right) + 2 \cdot \sum_{j=1}^{k} \exp\left(-\frac{\mu^2}{12}|B_j|\right).$$

Algorithm 1 gives the pseudocode of our approximation algorithm for general SMKP instances. The algorithm uses several configuration parameters that will be set in the proof of Lemma 8.

Algorithm 1 Algorithm for SMKP.

Input : An SMKP instance $\mathcal{I} = (I, w, B, W, f)$ and the parameters N, ξ, δ and μ.

1 **forall** *feasible assignments $A = (A_b)_{b \in B}$ such that $\sum_{b \in B} |A_b| \leq \xi$* **do**

2 Let $\mathcal{I}' = (I', w, B, W', f')$ be the residual instance of $\mathcal{I}$ w.r.t $(A_b)_{b \in B}$ and ξ.

3 Run Algorithm 2 with the bins B and capacities $(W'_b)_{b \in B}$. Let $\tilde{B}$ and $(\tilde{W}_b)_{b \in \tilde{B}}$ be the output, and $\tilde{B} = \cup_{j=0}^{k} \tilde{B}_j$ the partition of $\tilde{B}$ to leveled blocks. Let $\tilde{\mathcal{I}} = (I', w, \tilde{B}, \tilde{W}, f')$ be the resulting instance.

4 Let $\tilde{\mathcal{I}}_R$ be the δ-restricted SMKP instance of $\tilde{\mathcal{I}}$ with the restricted bins $\tilde{B}^r = \cup_{j=0}^{\min\{N^2-1,k\}} \tilde{B}_j$.

5 Solve $\tilde{\mathcal{I}}_R$ using Algorithm 4 with parameter μ, and the partition $\tilde{B} \setminus \tilde{B}^r = \cup_{j=1}^{k} \tilde{B}_j$. Denote the returned assignment by $(\tilde{S}_b)_{b \in \tilde{B}}$, and let $S_b = \tilde{S}_b$ for $b \in \tilde{B}$ and $S_b = \emptyset$ for $b \in B \setminus \tilde{B}$.

6 If $f(\cup_{b \in B}(A_b \cup S_b))$ is higher than the value of the current best solution, set $(A_b \cup S_b)_{b \in B}$ as the current best solution.

7 **end**

8 Return the best solution found.

▶ **Lemma 8.** *For any $\varepsilon > 0$, there are parameters N, ξ, δ, μ such that, for any SMKP instance $\mathcal{I}$, Algorithm 1 returns a solution of expected value at least $(1 - e^{-1} - \varepsilon)\mathrm{OPT}(\mathcal{I})$.*

Proof. We start by setting the parameter values. The reason for selecting these values will become clear later. Given a fixed $\varepsilon \in (0, 0.1)$, there is $\mu \in (0, 0.1)$ such that $\frac{(1-\mu)^2}{1+\mu} \geq (1 - \varepsilon^2)$. By the Monotone Convergence Theorem,

$$\lim_{N \to \infty} 2N^2 \cdot \sum_{t=1}^{\infty} \exp\left(-\frac{\mu^2 \cdot N^t}{12}\right) = \sum_{t=1}^{\infty} \lim_{N \to \infty} 2N^2 \exp\left(-\frac{\mu^2 \cdot N^t}{12}\right) = 0.$$

It follows that there are $N > \frac{1}{\varepsilon^2}$ and $\delta > 0$ such that

$$N^2 \exp\left(-\frac{\mu^2}{12} \cdot \frac{1}{\delta}\right) + 2N^2 \cdot \sum_{t=1}^{\infty} \exp\left(-\frac{\mu^2}{12} N^t\right) < \frac{\varepsilon^2}{2}. \tag{1}$$

Finally, we select ξ such that $\xi \geq \frac{N^2}{\varepsilon^2 \delta}$ and $\exp\left(-\frac{\mu^3}{16}\frac{\xi}{5}\right) \leq \frac{\varepsilon^2}{2}$.

Let $\mathcal{I} = (I, w, B, W, f)$ be an SMKP instance, and let $(A_b^*)_{b \in B}$ be an optimal solution for $\mathcal{I}$. Assume w.l.o.g that $A_{b_1}^* \cap A_{b_2}^* = \emptyset$ for any $b_1, b_2 \in B$, $b_1 \neq b_2$. Define $A^* = \cup_{b \in B} A_b^*$. If $|A^*| \leq \xi$, there is an iteration of Line 1 in which $A_b^* = A_b$ for all $b \in B$. Therefore, in this iteration we have at Line 6 $f(\cup_{b \in B}(A_b \cup S_b)) \geq f(A^*)$, and the algorithm returns a solution of value at least $f(A^*)$. Otherwise, by Lemma 3, there is a feasible solution $(A_b)_{b \in B}$ such that $A_b \subseteq A_b^*$, $\sum_{b \in B} |A_b^*| = \xi$ and $(A_b^* \setminus A_b)_{b \in B}$ is a feasible solution to $\mathcal{I}'$, the residual instance of $\mathcal{I}$ w.r.t $(A_b)_{b \in B}$ and ξ. It follows that there is an iteration of Line 1 which considers this solution $(A_b)_{b \in B}$. We focus on this iteration for the rest of the analysis.

Let $A = \cup_{b \in B} A_b$. If $f(A) \geq (1 - e^{-1}) f(A^*)$ then when the algorithm reaches Line 6 it holds that $f(\cup_{b \in B}(A_b \cup S_b)) \geq f(A) \geq (1 - e^{-1}) f(A^*)$; therefore, the algorithm returns a $(1 - e^{-1})$-approximation in this case. Henceforth, we assume that $f(A) \leq (1 - e^{-1}) f(A^*)$. Then,

$$f'(\cup_{b \in B}(A_b^* \setminus A_b)) = f'(A^* \setminus A) = f(A^*) - f(A) \geq \frac{f(A)}{1 - e^{-1}} - f(A) = \frac{1}{e - 1} f(A). \tag{2}$$

As $(A_b^* \setminus A_b)_{b \in B}$ is a feasible solution for $\mathcal{I}'$, it holds that $\mathrm{OPT}(\mathcal{I}') \geq f'(A^* \setminus A)$. Therefore, by Lemma 6 and the choice of N, it holds that

$$\mathrm{OPT}(\tilde{\mathcal{I}}) \geq \left(1 - \frac{1}{N}\right) f'(A^* \setminus A) \geq (1 - \varepsilon^2) f'(A^* \setminus A), \tag{3}$$

where $\mathcal{I}'$ is the instance output by Algorithm 2. Let $(D_b)_{b \in \tilde{B}}$ by an optimal solution for $\tilde{\mathcal{I}}$. Consider $(D_b^r)_{b \in \tilde{B}}$ where $D_b^r = D_b \setminus \{i \in D_b | w_i > \delta \cdot \tilde{W}_b\}$ for $b \in \tilde{B}^r$ (the set $\tilde{B}^r$ is defined in Line 4) and $D_b^r = D_b$ for $b \in \tilde{B} \setminus \tilde{B}^r$. It follows that D_b^r is a solution for the δ-restricted SMKP instance $\tilde{\mathcal{I}}_R$. As for any $b \in \tilde{B}^r$, $|\{i \in D_b | w_i > \delta \cdot \tilde{W}_b\}| \leq \frac{1}{\delta}$, we have that

$$\mathrm{OPT}\left(\tilde{\mathcal{I}}_R\right) \geq f'\left(\cup_{b \in \tilde{B}} D_b^r\right) \geq \mathrm{OPT}\left(\tilde{\mathcal{I}}\right) - \frac{N^2}{\delta \cdot \xi} f(A) \geq (1 - \varepsilon^2) f'\left(A^* \setminus A\right) - \varepsilon^2 \cdot f(A). \tag{4}$$

The second inequality follows from the definition of residual instance, and the third inequality from (3) and the choice of ξ. Since $f'(A^* \setminus A) \geq \frac{1}{e-1} f(A)$ and $\varepsilon \in (0, 0.1)$, it follows that $\mathrm{OPT}(\tilde{\mathcal{I}}_R) \geq \frac{f(A)}{5}$.

By Lemma 7, we have that

$$\mathbb{E}\left[f'(\cup_{b \in \tilde{B}} \tilde{S}_b)\right] \geq (1 - e^{-1}) \frac{(1 - \mu)^2}{1 + \mu} (1 - \gamma) \mathrm{OPT}(\tilde{\mathcal{I}}_R) \geq (1 - e^{-1})(1 - \varepsilon^2)(1 - \gamma) \mathrm{OPT}(\tilde{\mathcal{I}}_R), \tag{5}$$

where

$$\begin{aligned} \gamma &= \exp\left(-\frac{\mu^3}{16} \cdot \frac{\mathrm{OPT}(\tilde{\mathcal{I}}_R)}{\xi^{-1} f(A)}\right) + |\tilde{B}^r| \exp\left(-\frac{\mu^2}{12} \cdot \frac{1}{\delta}\right) + 2 \cdot \textstyle\sum_{j=N^2}^{k} \exp\left(-\frac{\mu^2}{12} |\tilde{B}_j|\right) \\ &\leq \exp\left(-\frac{\mu^3}{16} \cdot \frac{\xi}{5}\right) + N^2 \exp\left(-\frac{\mu^2}{12} \cdot \frac{1}{\delta}\right) + 2 \cdot N^2 \cdot \textstyle\sum_{t=1}^{\infty} \exp\left(-\frac{\mu^2}{12} N^t\right) \leq \varepsilon^2. \end{aligned} \tag{6}$$

The first equality uses $f'(\{i\}) \leq \xi^{-1} f(A)$ (by the definition of $\mathcal{I}'$). The first inequality holds since $\mathrm{OPT}(\tilde{\mathcal{I}}_R) \geq \frac{f(A)}{5}$, $|\tilde{B}^r| \leq N^2$ and there are at most N^2 blocks $\tilde{B}_j$ of size N^t. The second inequality uses (1) and the choice of ξ. Combining (6) with (5) and (4), we obtain

$$\mathbb{E}\left[f(\cup_{b\in B}(A_b \cup S_b))\right] \geq f(A) + \mathbb{E}\left[f'(\cup_{b\in\tilde{B}})\tilde{S}_b)\right] \geq f(A) + (1-e^{-1})(1-\varepsilon^2)^2\mathrm{OPT}(\tilde{\mathcal{I}}_R)$$
$$\geq f(A) + (1-e^{-1})(1-\varepsilon^2)^3 f'(A^* \setminus A) - \varepsilon^2 f(A) \geq (1-e^{-1}-\varepsilon)f(A^*).$$

Hence, in this iteration the solution considered in Line 6 has expected value at least $(1-e^{-1}-\varepsilon)f(A^*)$. This completes the proof of the lemma. ◀

▶ **Lemma 9.** *For any constant parameters N, ξ, δ and μ, Algorithm 1 returns a feasible solution for the input instance in polynomial time.*

Proof. We first note that for any fixed parameter values the algorithm has a polynomial running time. The number of assignments considered in Line 1 can be trivially bounded by $(n \cdot m)^{\xi}$. As Algorithms 2 and 4 are polynomial in their input size, the operations in each iteration are also done in polynomial time.

For each iteration of Line 1, by Lemma 7, $(\tilde{S}_b)_{b\in\tilde{B}}$ is a feasible solution to $\tilde{\mathcal{I}}_R$. Therefore, for any $b \in B$ either $w(S_b) = w(\emptyset) \leq W'_b$ or $w(S_b) = w(\tilde{S}_b) \leq \tilde{W}_b \leq W'_b$, where the last equality follows from Lemma 6. Therefore, $w(A_b \cup S_b) \leq w(A_b) + W'_b \leq W_b$. Hence, the solution considered in each iteration is feasible for the input instance. ◀

Theorem 1 follows from Lemmas 8 and 9.

3.1 Structuring the Instance

In this section we present Algorithm 2 and prove Lemma 6. Our technique for generating an N-leveled partition can be viewed as a variant of the linear grouping technique of [8] which requires the use of *non-uniform* group sizes (each group of bins then becomes a *block*). Given a set of bins B with capacities $(W_b)_{b\in B}$, we sort the bins in non-increasing order by capacities. We now use the numbering $B = \{1, \ldots, m\}$, where $W_1 \geq W_2 \geq \ldots \geq W_m$. To generate an N-leveled partition of the bins and the modified capacities, we define groups (or blocks) of bins, where each group j consists of $N^{\lfloor \frac{j}{N^2} \rfloor}$ consecutive bins, for $j \geq 0$. Starting from the first bin, we keep generating such groups as long as there are enough bins to form a group of the desired size. We omit the remaining bins and decrease the capacity of each bin to the minimal capacity of a bin in its group. We formalize this procedure in Algorithm 2.

Algorithm 2 Structure in Blocks.

Input : A set of bins B, capacities $(W_b)_{b\in B}$ and N.

1 Let $B = \{1, \ldots, m\}$ where $W_1 \geq W_2 \geq \ldots \geq W_m$.

2 Let $k = \max\left\{\ell \in \mathbb{N} \;\middle|\; \sum_{r=0}^{\ell} N^{\lfloor \frac{r}{N^2} \rfloor} \leq m\right\}$.

3 Define $\tilde{B}_j = \left\{b \;\middle|\; \sum_{r=0}^{j-1} N^{\lfloor \frac{r}{N^2} \rfloor} < b \leq \sum_{r=0}^{j} N^{\lfloor \frac{r}{N^2} \rfloor}\right\}$ for $0 \leq j \leq k$.

4 Let $\tilde{B} = \cup_{j=0}^{k} \tilde{B}_j$, and $\tilde{W}_b = \min_{b'\in\tilde{B}_j} W_{b'}$ for all $0 \leq j \leq k$ and $b \in \tilde{B}_j$.

5 Return $\tilde{B}$, $(\tilde{W}_b)_{b\in\tilde{B}}$ and the partition $(\tilde{B}_j)_{j=0}^{k}$.

The following standard result for submodular functions is used in the proof of Lemma 6.

▶ **Lemma 10.** *Let $h : 2^{\Omega} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function, and let $S_{i,1}, \ldots, S_{i,N} \subseteq \Omega$ for $1 \leq i \leq M$. Then for every $1 \leq i \leq M$ there is $1 \leq j_i^* \leq N$ such that*

$$h\left(\bigcup_{i=1}^{M} \bigcup_{1\leq j\leq N,\ j\neq j_i^*} S_{i,j}\right) \geq \left(1 - \frac{1}{N}\right) h\left(\bigcup_{i=1}^{M}\bigcup_{j=1}^{N} S_{i,j}\right).$$

The proof for the Lemma is given in Appendix A.

Proof of Lemma 6. By construction, we have that $(\tilde{B}_j)_{j=0}^{k}$ is an N-leveled partition of $(\tilde{W}_b)_{b\in\tilde{B}}$ and $\tilde{W}_b \leq W_b$ for any $b \in \tilde{B}$. Also, Algorithm 2 has a polynomial running time.

To complete the proof we need to show Property 2 in the lemma. Let $\mathcal{I} = (I, w, B, W, f)$ be an SMKP instance, and let $\tilde{\mathcal{I}} = (I, w, \tilde{B}, \tilde{W}, f)$ be the instance with bins and capacities generated (as output) by Algorithm 2. We need to show that $\mathrm{OPT}(\tilde{\mathcal{I}}) \geq \left(1 - \frac{1}{N}\right)\mathrm{OPT}(\mathcal{I})$.

Let $A_1^*, \ldots, A_m^*$ be an optimal solution for $\mathcal{I}$, and $A^* = \cup_{b\in B} A_b^*$. We modify this solution using a sequence of steps, eventually obtaining a feasible solution for $\tilde{\mathcal{I}}$. The latter is used to lower bound $\mathrm{OPT}(\tilde{\mathcal{I}})$. Define $\tilde{B}_{k+1} = B \setminus \tilde{B}$. We note that $\tilde{B}_{k+1}$ may be empty. We partition $\{\tilde{B}_j \mid 0 \leq j \leq k+1\}$ into *levels* and *super-blocks*. We consider each N^2 consecutive blocks to be a *level*, and each N consecutive blocks within a level to be a *super-block*. Formally, level t is

$$\mathcal{L}_t = \left\{j \mid t \cdot N^2 \leq j < \min\{(t+1)N^2,\ k+2\}\right\}$$

for $0 \leq t \leq \ell$ with $\ell = \left\lfloor \frac{k+1}{N^2} \right\rfloor$. Also, super-block r of level t is

$$\mathcal{S}_{t,r} = \left\{j \mid t \cdot N^2 + r \cdot N \leq j < t \cdot N^2 + (r+1) \cdot N\right\}$$

for $0 \leq r < N$ and level $0 \leq t < \ell$ (we do not partition the last level to super-blocks). It follows that $B = \cup_{t=0}^{\ell} \cup_{j\in\mathcal{L}_t} \tilde{B}_j$ and $\mathcal{L}_t = \cup_{r=0}^{N-1} \mathcal{S}_{t,r}$ for $0 \leq t < \ell$. Furthermore, for any $j \in \mathcal{L}_t$, $j \neq k+1$ it holds that $|\tilde{B}_j| = N^t$ and $|\tilde{B}_{k+1}| < N^\ell$. Essentially, all the blocks of level t are of the same size.

We modify $A_1^*, \ldots, A_m^*$ using the following steps. First, in each level (except the last one) we evict all the bins from a singe super-block. Since there are N super-blocks in each of these levels, this may decrease the value of the assignment at most by factor $\frac{1}{N}$. Then, we slightly shuffle the content of the bins in all levels (except the last one). In each level, we place the content of the bins of the last super-block in bins of the evicted super-block in the same level. As the bins are ordered by capacity, this will keep the assignment feasible with respect to the original capacities. In the last step, for each level (except level 0) we move the content of the bins from the first block to the bins of the last super-block in the previous level (the two sets of bins have the same cardinality), and content of bins from other blocks (except level 0) to the previous block from the same level. This yields a feasible assignment for the leveled instance. We formally describe these steps in the following.

Eviction: We first evict a super-block of bins from each level (except the last one). Let $R = \cup_{b\in\mathcal{L}_\ell} A_b^*$ be the subset of items assigned to the last level. By Lemma 10, for any $0 \leq t < \ell$ there is r_t^* such that

$$\begin{aligned} f_R\left(\bigcup_{t=0}^{\ell-1} \bigcup_{0\leq r<N,\ r\neq r_t^*} \bigcup_{j\in\mathcal{S}_{t,r}} \bigcup_{b\in\tilde{B}_j} A_b^*\right) &\geq \left(1-\frac{1}{N}\right) f_R\left(\bigcup_{t=0}^{\ell-1}\bigcup_{r=0}^{N-1} \bigcup_{j\in\mathcal{S}_{t,r}} \bigcup_{b\in\tilde{B}_j} A_b^*\right) \\ &= \left(1-\frac{1}{N}\right) f_R(A^*). \end{aligned}$$

Define $T_1, \ldots, T_m$ by $T_b = \emptyset$ for any $b \in \bigcup_{t=0}^{\ell-1} \bigcup_{j\in\mathcal{S}_{t,r_t^*}} \tilde{B}_j$, and $T_b = A_b^*$ for any $b \in B \setminus \left(\bigcup_{t=0}^{\ell-1} \bigcup_{j\in\mathcal{S}_{t,r_t^*}} \tilde{B}_j\right)$. Then,

$$f\left(\bigcup_{b\in B} T_b\right) = f(R) + f_R\left(\bigcup_{t=0}^{\ell-1} \bigcup_{0\leq r<N-1,\ r\neq r_t^*} \bigcup_{j\in\mathcal{S}_{t,r}} \bigcup_{b\in\tilde{B}_j} A_b^*\right) \geq \left(1-\frac{1}{N}\right) f(A^*).$$

It also holds that $T_1, \ldots, T_m$ is a feasible solution for $\mathcal{I}$.

Shuffling: We generate a new assignment $\tilde{T}_1, \ldots, \tilde{T}_m$ such that $\cup_{b\in B}\tilde{T}_b = \cup_{b\in B}T_b$ and the last super-blocks in each level (except the last one) is empty. This property is obtained by moving the content of the bins in super-block $N-1$ to the bins of super-block r_t^* for every $0 \le t < \ell$.

We define $(\tilde{T}_b)_{b\in B}$ as follows. For any $0 \le t < \ell$, $j \in \mathcal{S}_{t,r_t^*}$ let $\varphi_t : \bigcup_{j\in\mathcal{S}_{t,r_t^*}} \tilde{B}_j \to \bigcup_{j\in\mathcal{S}_{t,N-1}} \tilde{B}_j$ be a bijection (since $\left|\bigcup_{j\in\mathcal{S}_{t,N-1}} \tilde{B}_j\right| = \left|\bigcup_{j\in\mathcal{S}_{t,r_t^*}} \tilde{B}_j\right| = N \cdot N^t$, such a function exists). For any $b \in \bigcup_{j\in\mathcal{S}_{t,r_t^*}} \tilde{B}_j$ set $\tilde{T}_b = T_{\varphi_t(b)}$. By definition we have $\varphi_t(b) \ge b$; therefore, in this case

$$w(\tilde{T}_b) = w(T_{\varphi_t(b)}) \le W_{\varphi_t(b)} \le W_b.$$

For any $0 \le t < \ell$, $j \in \mathcal{S}_{t,N-1}$ and $b \in \tilde{B}_j$ set $\tilde{T}_b = \emptyset$. For any other bin $b \in B$ set $\tilde{T}_b = T_b$.

It follows that $\cup_{b\in B}\tilde{T}_b = \cup_{b\in B}T_b$, since $T_b = \emptyset$ for every $0 \le t < \ell$, $j \in \mathcal{S}_{t,r_t^*}$ and $b \in B_j$. Also, $(\tilde{T}_b)_{b\in B}$ is a feasible solution for $\mathcal{I}$.

Shifting: In this step we generate the assignment $(A_b)_{b\in\tilde{B}}$ which satisfies the properties in the lemma. As the bins of the last super-block in each level (except the last one) are vacant in $\tilde{T}_1, \ldots, \tilde{T}_m$, we use them for the content assigned to the first block of the next level. This can be done since N blocks of level t contain the same number of bins as a single block of level $t+1$. We also use blocks in levels greater than 0 which are not in the last super-block to store the content of the next block in the same level.

For any $0 < t \le \ell$ and $j \in \mathcal{L}_t$, consider a block $\tilde{B}_j$. Suppose that $j \ne t \cdot N^2$ and $j \notin \mathcal{S}_{t,N-1}$ where $t \ne \ell$; that is, $\tilde{B}_j$ is not the first block in the level, and is not in the last super-block of a level other than the last one. Then, let $\psi_j : \tilde{B}_j \to \tilde{B}_{j-1}$ be a bijection, and define $A_{\psi_t(b)} = \tilde{T}_b$ for any $b \in \tilde{B}_j$. If $j = t \cdot N^2$ (that is, $\tilde{B}_j$ is the first block in a level), let $\psi_j : \tilde{B}_j \to \bigcup_{j'\in\mathcal{S}_{t-1,N-1}} \tilde{B}_{j'}$ be a bijection, and define $A_{\psi_t(b)} = \tilde{T}_b$ for any $b \in \tilde{B}_j$. Finally, for any $0 \le j < N^2 - N$, let $\psi_j : \tilde{B}_j \to \tilde{B}_j$ be the identity function and define $A_{\psi_t(b)} = A_b = \tilde{T}_b$ for the single bin $b \in \tilde{B}_j$. For any bin $b \in \tilde{B}$ not handled in this process, set $A_b = \emptyset$.

We note that the definition is sound as the ranges of the functions ψ_j above do not intersect. Let $b \in \tilde{B}$. If $A_b = \emptyset$ then $w(A_b) \le \tilde{W}_b$. Otherwise, if $b \in \tilde{B}_j$ for $0 \le j < N^2 - N$ then $w(A_b) = w(\tilde{T}_b) < W_b \le \tilde{W}_b$. Finally, the only option left is that $b = \psi_{j'}(b')$ for some $0 < t \le \ell$, $j' \in \mathcal{L}_t$ and $b' \in \tilde{B}_{j'}$, such that $j' \ne t \cdot N^2$ and $j' \notin S_{t,N-1}$ if $t \ne \ell$. By definition, it holds that $b' \in \tilde{B}_j$ for some $j < j'$. Since the bins were ordered by capacity, we have

$$w(A_b) = w(A_{\phi_{j'}(b')}) = w(\tilde{T}_{b'}) \le W_{b'} \le \min_{b''\in\tilde{B}_j} W_{b''} = \tilde{W}_b.$$

Thus, the assignment is feasible with respect to the bins $\tilde{B}$ with capacities $\left(\tilde{W}_b\right)_{b\in\tilde{B}}$.

It also holds that for any $b \in B$ such that $\tilde{T}_b \ne 0$ there is $b' \in \tilde{B}$ such that $A_{b'} = \tilde{T}_b$. Therefore, $\bigcup_{b\in\tilde{B}} A_b = \bigcup_{b\in B} \tilde{T}_b = \bigcup_{b\in B} T_b$. Hence, $f(\cup_{b\in\tilde{B}} A_b) = f(\cup_{b\in B} T_b) \ge \left(1 - \frac{1}{N}\right) f(A^*)$. We conclude that $\mathrm{OPT}(\tilde{\mathcal{I}}) \ge \left(1 - \frac{1}{N}\right) \mathrm{OPT}(\mathcal{I})$. ◀

3.2 Solving a Continuous Relaxation and Rounding

In this section we give Algorithm 4 which outputs a solution satisfying Lemma 7. The input for the algorithm is a δ-restricted SMKP instance along with a partition $B \setminus B^r = B_1 \cup \ldots \cup B_k$ of the bins, where B_j is a block for all $1 \le j \le k$. Algorithm 4 uses Algorithm 3 as a subroutine which converts a solution for an auxiliary *block-constraint* problem into a solution for δ-restricted SMKP.

3.2.1 The Block-Constraint Problem

We now define the *block-constraint problem*, to be solved for a given instance $\mathcal{I}$ of δ-restricted SMKP, using the partition $B \setminus B^r = B_1 \cup \ldots \cup B_k$ and the parameter μ. The input for the block-constraint problem is a universe of elements E, a monotone submodular function $g : 2^E \to \mathbb{R}_{\geq 0}$ and a polytope $P \subseteq [0,1]^E$ (see below).

For simplicity, let $\{B_{k+1}, \ldots, B_\ell\} = \{\{b\} |\ b \in B^r\}$ be the set of blocks, each consisting of a single bin. Thus, $B = \cup_{j=1}^{\ell} B_j$. Denote the (uniform) capacity of the bins in block B_j by W_j^*, for $1 \leq j \leq \ell$. That is, for any $b \in B_j$ it holds that $W_j^* = W_b$. For $1 \leq j \leq k$, we say that an item $i \in I$ is *j-small* if $w_i \leq \mu \cdot W_j^*$, otherwise i is *j-large*. Let $I_j = \{\{i\} |\ i \text{ is } j\text{-small}\}$ for $1 \leq j \leq k$. For $k < j \leq \ell$ define $I_j = \{\{i\} \mid w_i \leq \delta W_j^*\}$.

A *j-configuration* is a subset of j-large items which can be packed into a single bin in B_j. That is, $C \subseteq I$ is a j-configuration if every item $i \in C$ is j-large and $w(C) \leq W_j^*$. Let $\mathcal{C}_j$ be the set of all j-configurations for $1 \leq j \leq k$ and $\mathcal{C}_j = \emptyset$ for $k < j \leq \ell$. As any j-configuration has at most μ^{-1} items, it follows that $|\mathcal{C}_j| \leq |I|^{\mu^{-1}}$, i.e., the number of configurations is polynomial in the size of $\mathcal{I}$. Furthermore, for $A \subseteq I$ such that $w(A) \leq W_j^*$, $1 \leq j \leq k$, there are $C \in \mathcal{C}_j$ and $S \subseteq I$ such that all the items in S are j-small and $A = C \cup S$. Our algorithm exploits this property.

Towards solving the block-constraint problem we define a submodular function g over a new universe of elements. Let $E = \{(S,j) |\ S \in \mathcal{C}_j \cup I_j,\ 1 \leq j \leq \ell\}$. Informally, the element $(S,j) \in E$ represents an assignment of all the items in S to a single bin $b \in B_j$. We now define $g : 2^E \to \mathbb{R}_{\geq 0}$ by $g(T) = f\left(\bigcup_{(S,j)\in T} S\right)$. By Claim 2, g is a submodular, monotone and non-negative function‘.

We define a polytope P for the instance $\mathcal{I}$ as follows.

$$P = \left\{ \bar{x} \in [0,1]^E \;\middle|\; \begin{array}{ll} \sum_{C \in \mathcal{C}_j} \bar{x}_{(C,j)} \leq |B_j| & \forall 1 \leq j \leq k \\ \sum_{S \in \mathcal{C}_j \cup I_j} w(S) \cdot \bar{x}_{(S,j)} \leq |B_j| \cdot W_j^* & \forall 1 \leq j \leq \ell \end{array} \right\} \tag{7}$$

The polytope represents a relaxed version of the capacity constraints over the bins. For each block B_j, $1 \leq j \leq k$, we only require that the *total weight* of items assigned to bins in B_j does not exceed the total capacity of the bins in this block. We also require that the number of j-configurations selected for B_j is no greater than the number of bins in this block.

Given an instance $\mathcal{I}$ of δ-restricted SMKP, along with the partition $B \setminus B^r = B_1 \cup \ldots \cup B_k$ and the parameter μ, we use for the *block-constraint problem* the universe E, the function g and the polytope $P \subseteq [0,1]^E$ as defined above.

We start by establishing a connection between the solution for the block-constraint problem for the given instance $\mathcal{I}$ and the optimal solution for δ-restricted SMKP on this instance. For a set $T \subseteq E$, we use $\bar{x}^T$ to denote the vector $\bar{x}^T \in \{0,1\}^E$ defined by $\bar{x}_e^T = 1$ for $e \in T$, and $\bar{x}_e^T = 0$ for $e \in E \setminus T$. Also, given a polytope Q and $\eta \geq 0$ we use the notation $\eta \cdot Q = \{\eta \bar{x} \mid \bar{x} \in Q\}$.

Our algorithm for solving δ-restricted SMKP on $\mathcal{I}$ solves first the block-constraint problem on this instance and then transforms the solution into a feasible solution for δ-restricted SMKP. We give the pseudocode for the transformation in Algorithm 3.

Algorithm 3 Employ a Block-Constraint Solution for SMKP.

Input : A δ-restricted SMKP instance $\mathcal{I} = (I, w, B, W, f)$, the partition of bins to block $\cup_{j=1}^{\ell} B_j$ and $T \subseteq E$.

1 Set $A_b = \emptyset$ for every $b \in B$.
2 Sort the elements (S, j) in T in decreasing order by the $w(S)$ values.
3 **for** *each* $(S, j) \in T$ *in the sorted order* **do**
4 | Set $A_b \leftarrow A_b \cup S$ where $b = \arg\min_{b \in B_j} w(A_b)$.
5 **end**
6 Return $(A_b)_{b \in B}$.

▶ **Lemma 11.** *Given an instance $\mathcal{I}$ of δ-restricted SMKP, consider the universe E and the polytope P as defined above. Then the following hold:*

1. *There is $T \subseteq E$, $\bar{x}^T \in P$ such that $g(T) \geq \mathrm{OPT}(\mathcal{I})$, where $\mathrm{OPT}(\mathcal{I})$ is the optimal solution value for δ-restricted SMKP on $\mathcal{I}$.*
2. *Given $T \subseteq E$ such that $\bar{x}^T \in (1-\mu) \cdot P$, Algorithm 3 returns in polynomial time a feasible solution $(A_b)_{b \in B}$ for δ-restricted SMKP instance $\mathcal{I}$ satisfying $f(\cup_{b \in B} A_b) = g(T)$.*

Proof. We start by proving part 1. Let $(A_b^*)_{b \in B}$ be an optimal solution for the δ-restricted SMKP instance, and let L_j be the set of all j-large items for $1 \leq j \leq k$ and $L_j = \emptyset$ for $k < j \leq \ell$. Define

$$T = \left(\bigcup_{j=1}^{k} \{(A_b^* \cap L_j, j) \mid b \in B_j\} \right) \cup \left(\bigcup_{j=1}^{\ell} \bigcup_{b \in B_j} \{(\{i\}, j) \mid i \in A_b^* \setminus L_j\} \right).$$

It can be easily shown that $g(T) = f(\cup_{b \in B} A_b^*)$. Furthermore, as $(A_b^*)_{b \in B}$ is a feasible solution, it holds that $\bar{x}^T \in P$.

We now prove part 2. Let $(A_b)_{b \in B}$ be the output of Algorithm 3 for the given input. We first note that $\cup_{b \in B} A_b = \cup_{(S,j) \in T} S$, and thus $g(T) = f(\cup_{b \in B} A_b)$.

For any $b \in B^r$, there is $k < j \leq \ell$ such that $B_j = \{b\}$. Therefore $A_b = \{i | (\{i\}, j) \in T\}$, and since $\bar{x}^T \in (1-\mu)P$ it follows that $w(A_b) \leq W_j^* = W_b$.

Let $1 \leq j \leq k$ and $b \in B_j$. Assume by negation that $w(A_b) > W_b = W_j^*$. Let $(S, j) \in T$ be the last element in T such that $S \neq \emptyset$ and S was added to A_b in Line 4. We conclude that $w(A_b \setminus S) > 0$, as otherwise $w(A_b) = w(S) \leq W_b$, by the definition of E. Therefore there are at least $|B_j|$ elements $(S', j) \in T$ such that $w(S') \geq w(S)$ (else, on the iteration of (S, j) there must be $b \in B_j$ with $A_b = \emptyset$). If $S \in \mathcal{C}_j$ then $w(S) > \mu \cdot W_j^*$ and thus

$$|\{S' \neq \emptyset |\ (S', j) \in T,\ S' \in \mathcal{C}_j\}| \ \geq\ |\{S' |\ (S', j) \in T,\ w(S') \geq w(S)\}| \ >\ |B_j|,$$

contradicting $\bar{x}^T \in (1-\mu)P$.

Therefore $S \notin \mathcal{C}_j$, and we can conclude that $S = \{i\}$ with $w_i \leq \mu \cdot W_j^*$. Thus, $w(A_b \setminus S) > (1-\mu) \cdot W_j^*$. Here, S has been allocated to A_b (which is itself a set of minimum weight). Then, for any $b' \in B_j$, we have $w(A_{b'}) \geq w(A_b) > (1-\mu) \cdot W_j^*$. Thus,

$$\sum_{(S', j) \in T} w(S') \geq \sum_{b' \in B_j} w(A_{b'}) > |B_j|(1-\mu) \cdot W_j^*,$$

contradicting $\bar{x}^T \in (1-\mu)P$. We conclude that $w(A_b) \leq W_b$.

Also, by definition, we have that for any $b \in B^r$ and $i \in A_b$ it holds that $w_i \leq \delta W_b$. Hence, $(A_b)_{b \in B}$ is a solution to the restricted SMKP instance. ◀

3.2.2 An Algorithm for δ-restricted SMKP

We are now ready to present our algorithm for δ-restricted SMKP. We note that in Line 3 of Algorithm 4 we use sampling by a solution vector $\bar{x}^*$, as defined in Section 2.

Algorithm 4 Solve and Round.

Input : A δ-restricted SMKP instance $\mathcal{I}$, a partition to blocks $B \setminus B^r = \cup_{j=k}^{j} B_j$, and a parameter $\mu > 0$.

1 Define E, g and P for the block-constraint problem on $\mathcal{I}$ and $\cup_{j=1}^{k} B_j$.
2 Let $G : [0,1]^E \to \mathbb{R}_{\geq 0}$ be the multilinear extension of g. Find a solution $\bar{x}^*$ for $\max_{\bar{x} \in \frac{1-\mu}{1+\mu} P} G(\bar{x})$ using the continuous greedy of [4].
3 Sample a set $T \sim \bar{x}^*$.
4 **if** $T \in (1-\mu)P$ **then**
5 Use Algorithm 3 to convert T into a solution $(A_b)_{b \in B}$ for δ-restricted SMKP on $\mathcal{I}$ and return $(A_b)_{b \in B}$.
6 **else**
7 Return $(A_b)_{b \in B}$ with $A_b = \emptyset$ for every $b \in B$.
8 **end**

For the analysis, consider first the running time. We note that, for any $\bar{\lambda} \in \mathbb{R}^E$, a vector $\bar{x} \in \frac{1-\mu}{1+\mu} P$ which maximizes $\bar{x} \cdot \bar{\lambda}$ can be found in polynomial time. Therefore, the continuous greedy in Line 2 runs in polynomial time. Thus, Algorithm 4 has a polynomial running time.

It remains to show that the algorithm returns a solution of expected value as stated in Lemma 7. The approach we use to prove the statement of the lemma is similar to the approach taken in [6]. In fact, it is possible to prove a variant of this claim using an approach of [17]. While eliminating the dependency on v, this will result in a more involved proof.

Proof of Lemma 7. For any $e \in E$ define X_e to be a random variable such that $X_e = 1$ if $e \in T$ and $X_e = 0$ otherwise. It follows that $(X_e)_{e \in E}$ are independent Bernoulli random variables, $\mathbb{E}[X_e] = \bar{x}^*_e$ and $T = \{e \in E | X_e = 1\}$.

We first consider blocks $k < j \leq \ell$. Let $k < j \leq \ell$ and $B_j = \{b\}$. Since $\bar{x}^* \in \frac{1-\mu}{1+\mu} P$, it follows that $\mathbb{E}\left[\sum_{(S,j) \in E} w(S) \cdot X_{(S,j)}\right] \leq \frac{1-\mu}{1+\mu} \cdot W_b$. Also, $w_{(S,j)} \leq \delta \cdot W_b$ for every $(S,j) \in E$. Using Chernoff's bound (Theorem 3.1 in [13], see also Lemma 15), we have

$$\Pr\left(\sum_{(S,j) \in T} w(S) > (1-\mu) W_b\right) \leq \exp\left(-\frac{\mu^2}{3} \cdot \frac{1-\mu}{1+\mu} \cdot \frac{1}{\delta}\right) \leq \exp\left(-\frac{\mu^2}{12} \cdot \frac{1}{\delta}\right), \tag{8}$$

with the last inequality following from $\mu \in (0, 0.1)$.

Now, let $1 \leq j \leq k$. For every $(S,j) \in E$ it holds that $w(S) \leq W^*_j$. Also, since $\bar{x}^* \in \frac{1-\mu}{1+\mu} P$, $\mathbb{E}\left[\sum_{(S,j) \in E} w(S) \cdot X_{(S,j)}\right] \leq \frac{1-\mu}{1+\mu} \cdot |B_j| W^*_j$, and $\mathbb{E}\left[\sum_{(S,j) \in E:\ S \in \mathcal{C}_j} 1 \cdot X_{(S,j)}\right] \leq \frac{1-\mu}{1+\mu} \cdot |B_j|$. Therefore, by Chernoff's bound (Theorem 3.1 in [13] and Lemma 15), we have

$$\Pr\left(\sum_{(S,j) \in T} w(S) > (1-\mu)|B_j| W^*_j\right) \leq \exp\left(-\frac{\mu^2}{3} \cdot \frac{1-\mu}{1+\mu} \cdot |B_j|\right) \leq \exp\left(-\frac{\mu^2}{12} \cdot |B_j|\right) \tag{9}$$

$$\Pr\left(\sum_{(S,j) \in T:\ S \in \mathcal{C}_j} 1 > (1-\mu)|B_j|\right) \leq \exp\left(-\frac{\mu^2}{3} \cdot \frac{1-\mu}{1+\mu} \cdot |B_j|\right) \leq \exp\left(-\frac{\mu^2}{12} \cdot |B_j|\right). \tag{10}$$

By Lemma 11, $\max_{\bar{z}\in P} G(\bar{z}) \geq \mathrm{OPT}(\mathcal{I})$. Since the second derivatives of G are non-positive (see [4]) it follows that $\max_{\bar{z}\in \frac{1-\mu}{1+\mu}P} G(\bar{z}) \geq \frac{1-\mu}{1+\mu}\mathrm{OPT}(\mathcal{I})$. As the continuous greedy of [4] yields a $(1-e^{-1})$-approximation for the problem of maximizing the multilinear extension subject to a polytope constraint, it follows that

$$G(\bar{x}^*) \geq (1-e^{-1})\frac{1-\mu}{1+\mu}\mathrm{OPT}(\mathcal{I}). \tag{11}$$

For any $(S,j) \in E$ we have $|S| \leq \mu^{-1}$, and from the submodularity of f, $g(\{(S,j)\}) - g(\emptyset) \leq \mu^{-1}\upsilon$ (recall that υ is defined in Lemma 7). Therefore, by the concentration bound of [6] (see Lemma 16), we have

$$\begin{aligned}&\Pr\left(g(T) \leq (1-e^{-1})\frac{(1-\mu)^2}{1+\mu}\mathrm{OPT}(\mathcal{I})\right) \leq \Pr\left(g(\{e\in E|X_e=1\}) \leq (1-\mu)G(\bar{x}^*)\right)\\ &\leq \exp\left(-\frac{\mu^3\cdot G(\bar{x}^*)}{2\upsilon}\right) \leq \exp\left(-\frac{\mu^3(1-e^{-1})}{2\upsilon}\frac{1-\mu}{1+\mu}\mathrm{OPT}(\mathcal{I})\right) \leq \exp\left(-\frac{\mu^3\cdot\mathrm{OPT}(\mathcal{I})}{16\cdot\upsilon}\right)\end{aligned} \tag{12}$$

The first and third inequality are due to (11).

Let ω be the event $\bar{x}^T \in (1-\mu)P$ and $g(T) \geq \frac{(1-\mu)^2}{1+\mu}(1-e^{-1})\mathrm{OPT}(\mathcal{I})$. By applying the union bound over (8), (9), (10) and (12), we have

$$\Pr(\omega) \geq 1-\left(|B^r|\exp\left(-\frac{\mu^2}{12}\frac{1}{\delta}\right) - 2\sum_{j=1}^{k}\exp\left(-\frac{\mu^2}{12}|B_j|\right) - \exp\left(-\frac{\mu^3}{16}\frac{\mathrm{OPT}(\mathcal{I})}{\upsilon}\right)\right) = 1-\gamma.$$

In case the event ω occurs, the algorithm executes Line 5, and by Lemma 11, $f(\cup_{b\in B}A_b) = f(T)$. Hence,

$$\mathbb{E}\left[f(\cup_{b\in B}A_b)\right] = \Pr(\omega)\cdot E\left[f(\cup_{b\in B}A_b)|\omega\right] \geq (1-\gamma)\frac{(1-\mu)^2}{1+\mu}(1-e^{-1})\mathrm{OPT}(\mathcal{I}).$$

Also, the algorithm either returns an empty solution when Line 7 executes, or Line 5 executes. In the latter case the solution is feasible by Lemma 11. Therefore the algorithm always returns a feasible solution. ◀

4 Discussion

In this paper we presented a randomized $(1-e^{-1}-\varepsilon)$-approximation for the monotone submodular multiple knapsack problem. Our algorithm relies on three main building blocks. The structuring technique (Section 3.1) which converts a general instance to a leveled instance, the reduction to the block-constraint problem (Section 3.2.1) and a refined analysis of known algorithms for submodular optimization with a d-dimensional knapsack constraint (Section 3.2.2). While the structuring technique and the refined analysis seem to be fairly robust, the reduction to the block-constraint problem proved to be limiting when generalizations of the problem were considered.

A notable example is the *non-monotone* submodular multiple knapsack problem, in which the set function f is non-monotone. Unfortunately, when f is non-monotone the function g used for solving the block-constraint problem is not submodular. A variant of the block-constraint problem which does not alter the set function may be used to overcome this hurdle. However, this variant limits the knapsacks utilization and degrades the approximation ratio. Our preliminary results for the non-monotone case guarantee an approximation ratio of $\frac{1}{2}\cdot e^{-\frac{1}{2}} - \varepsilon \approx 0.303 - \varepsilon$ using this approach.

Another natural generalization of SMKP is monotone submodular optimization subject to a multiple knapsack and a *matroid* constraints, in which the solution $(A_b)_{b\in B}$ must also satisfy $\cup_{b\in B} A_b \in \mathcal{M}$ for a matoid $\mathcal{M}$. However, the matroid properties are not preserved throughout the reduction to the block-constraint problem, rendering existing techniques for submodular optimization with matroid and d-dimensional knapsack constraints [6] ineffective.

On the positive side, we believe that the techniques described in this paper can be extended to handle the problem for maximizing a monotone submodular function subject to a multiple knapsack constraint and an additional d-dimensional knapsack constraint, for a fixed d. We defer the details to the full version of the paper.

References

1 Niv Buchbinder and Moran Feldman. Submodular functions maximization problems. *Handbook of Approximation Algorithms and Metaheuristics*, 1:753–788, 2017.

2 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1433–1452. SIAM, 2014.

3 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 182–196. Springer, 2007.

4 Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

5 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

6 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding for matroid polytopes and applications. *arXiv preprint*, 2009. `arXiv:0909.4348`.

7 Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 575–584. IEEE, 2010.

8 W Fernandez De La Vega and George S. Lueker. Bin packing can be solved within 1+ ε in linear time. *Combinatorica*, 1(4):349–355, 1981.

9 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

10 Uriel Feige and Michel Goemans. Approximating the value of two power proof systems, with applications to max 2sat and max dicut. In *Proceedings Third Israel Symposium on the Theory of Computing and Systems*, pages 182–189. IEEE, 1995.

11 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 570–579. IEEE, 2011.

12 Moran Feldman and Seffi Naor. *Maximization problems with submodular objective functions*. PhD thesis, Computer Science Department, Technion, 2013.

13 Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM (JACM)*, 53(3):324–360, 2006.

14 Klaus Jansen. Parameterized approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 39(4):1392–1412, 2010.

15 Klaus Jansen. A fast approximation scheme for the multiple knapsack problem. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 313–324. Springer, 2012.

16 Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999.

17 Ariel Kulik, Hadas Shachnai, and Tami Tamir. Approximations for monotone and nonmonotone submodular maximization with knapsack constraints. *Mathematics of Operations Research*, 38(4):729–739, 2013.

18 Jon Lee, Vahab S Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM Journal on Discrete Mathematics*, 23(4):2053–2078, 2010.

19 George L Nemhauser and Laurence A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3):177–188, 1978.

20 Xiaoming Sun, Jialin Zhang, and Zhijie Zhang. Tight algorithms for the submodular multiple knapsack problem. *arXiv preprint*, 2020. `arXiv:2003.11450`.

21 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

22 Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 67–74, 2008.

23 Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.

A Basic Properties of Submodular Functions

▷ Claim 12. Let $f : 2^I \to \mathbb{R}_{\geq 0}$ be monotone and submodular function, then for any $A \subseteq B \subseteq I$ and $S \subseteq I$ it holds that $f(A \cup S) - f(A) \geq f(B \cup S) - f(B)$.

Proof. By the submodularity of f, we have

$$f(A \cup S) + f(B) \geq f(A \cup S \cup B) + f((A \cup S) \cap B) \geq f(B \cup S) + f(A)$$

where the second inequality follows from $A \subseteq (A \cup S) \cap B$ and the monotonicity of f. By rearranging the terms in the above we get

$$f(A \cup S) - f(A) \geq f(B \cup S) - f(B)$$

as required. ◁

▷ Claim 13. Let $f : 2^I \to \mathbb{R}_{\geq 0}$ be be a non-negative, monotone and submodular function, and let $S \subseteq I$. Then f_S is a submodular, monotone and non-negative function.

Proof. Let $A \subseteq I$. As f is monotone, we have

$$f_S(A) = f(S \cup A) - f(S) \geq 0.$$

That is, f is non-negative.

By claim 12, for any $A \subseteq B \subseteq I$ and $x \in I \setminus B$ it holds that

$$\begin{aligned} f_S(A \cup \{x\}) - f_S(A) &= f(S \cup A \cup \{x\}) - f(S \cup A) \\ &\geq f(S \cup B \cup \{x\}) - f(S \cup B) = f_S(B \cup \{x\}) - f_S(B). \end{aligned}$$

Therefore, f_S is submodular.

Finally, for $A \subseteq B \subseteq I$, as f is monotone we have that

$$f_S(A) = f(S \cup A) - f(S) \geq f(S \cup B) - f(S) = f_S(B).$$

Thus, f_S is also monotone. ◁

Proof of Claim 2. It is easy to see that g is non-negative, as f is non negative. In addition, for any two subsets $A \subseteq B \subseteq E$ we have $\cup_{(S,h)\in A} s \subseteq \cup_{(S,h)\in B} s$. Thus, since f is monotone, g is monotone as well.

All that is left to prove is that g is submodular. Consider subsets $A \subseteq B \subseteq E$ and $(S,h) \in E \setminus B$.

$$\begin{aligned} g(A \cup \{(S,h)\}) - g(A) &= f(\cup_{(S',h')\in A} S' \cup S) - f(\cup_{(S',h')\in A} s) \\ &\leq f(\cup_{(S',h')\in B} S' \cup S) - f(\cup_{(S',h')\in B} S') \\ &= g(B \cup \{(S,h)\}) - g(B). \end{aligned}$$

The inequality follows from Claim 12 and $\cup_{(S',h')\in A} S' \subseteq \cup_{(S',h')\in B} S'$. ◁

To prove Lemma 10 we first prove a special case of the lemma.

▶ **Lemma 14.** *Let $h : 2^{\Omega} \to \mathbb{R}_{\geq 0}$ be a submodular monotone and non-negative function, and let $S_1, \ldots, S_N \subseteq \Omega$. Then there is $1 \leq j^* \leq N$ such that*

$$h\left(\bigcup_{1\leq j\leq N,\ j\neq j^*} S_j\right) \geq \left(1 - \frac{1}{N}\right) h(S_1 \cup \ldots \cup S_N).$$

Proof. As h is submodular and monotone, using Claim 2, we have

$$\begin{aligned} h(S_1 \cup \ldots \cup S_N) - h(\emptyset) &= \sum_{j=1}^{N} \left(h(S_1 \cup \ldots \cup S_j) - h(S_1 \cup \ldots \cup S_{j-1})\right) \\ &\geq \sum_{j=1}^{N} \left(h\left(\bigcup_{j'=1}^{N} S_{j'}\right) - h\left(\bigcup_{1\leq j'\leq N, j'\neq j} S_{j'}\right)\right) \end{aligned}$$

Therefore there is $1 \leq j^* \leq N$ such that

$$h\left(\bigcup_{j=1}^{N} S_j\right) - h\left(\bigcup_{1\leq j\leq N, j\neq j^*} S_j\right) \leq \frac{1}{N}\left(h(S_1 \cup \ldots \cup S_N) - h(\emptyset)\right).$$

By rearranging the terms and using $h(\emptyset) \geq 0$ we obtain

$$h\left(\bigcup_{1\leq j\leq N, j\neq j^*} S_j\right) \geq \left(1 - \frac{1}{N}\right) h(S_1 \cup \ldots \cup S_N),$$

as required. ◀

Proof of Lemma 10. Let $h : 2^{\Omega} \to \mathbb{R}_+$ be a submodular, non-negative and monotone function, and $S_{i,1}, \ldots, S_{i,N} \subseteq \Omega$ for every $1 \leq i \leq M$.

Define $T_i = \bigcup_{j=1}^{N} S_{i,j}$. Now,

$$h\left(\bigcup_{i=1}^{M}\bigcup_{j=1}^{N} S_{i,j}\right) - h(\emptyset) = \sum_{i=1}^{M} h_{\left(\bigcup_{i'=1}^{i-1} T_{i'}\right)}(T_i) = \sum_{i=1}^{M} h_{\left(\bigcup_{i'=1}^{i-1} T_{i'}\right)}\left(\bigcup_{j=1}^{N} S_{i,j}\right).$$

By Lemma 14 for every $1 \leq i \leq M$ there is $1 \leq j_i^* \leq N$ such that

$$h_{\left(\bigcup_{i'=1}^{i-1} T_{i'}\right)}\left(\bigcup_{1\leq j\leq N,\ j\neq j_i^*} S_{i,j}\right) \geq \left(1 - \frac{1}{N}\right) h_{\left(\bigcup_{i'=1}^{i-1} T_{i'}\right)}(T_i).$$

Therefore,

$$\begin{aligned} h\left(\bigcup_{i=1}^{M}\bigcup_{1\le j\le M,\ j\neq j_i^*} S_{i,j}\right) - h(\emptyset) &= \sum_{i=1}^{M} h_{\left(\bigcup_{i'=1}^{i-1}\bigcup_{1\le j\le M,\ j\neq j_{i'}^*} S_{i',j}\right)}\left(\bigcup_{1\le j\le M,\ j\neq j_i^*} S_{i,j}\right) \\ &\ge \sum_{i=1}^{M} h_{\left(\bigcup_{i'=1}^{i-1} T_{i'}\right)}\left(\bigcup_{1\le j\le M,\ j\neq j_i^*} S_{i,j}\right) \ge \left(1-\frac{1}{N}\right)\sum_{i=1}^{M} h_{\left(\bigcup_{i'=1}^{i-1} T_{i'}\right)}(T_i) \\ &= \left(1-\frac{1}{N}\right)\left(h\left(\bigcup_{i=1}^{M}\bigcup_{j=1}^{N} S_{i,j}\right) - h(\emptyset)\right). \end{aligned}$$

The first inequality follows from $h_{T_1}(A) \ge h_{T_2}(A)$ for any $T_1 \subseteq T_2 \subseteq \Omega$ and $A \subseteq \Omega$ due to Claim 2. As h is non-negative, we conclude that

$$h\left(\bigcup_{i=1}^{M}\bigcup_{1\le j\le M,\ j\neq j_i^*} S_{i,j}\right) \ge \left(1-\frac{1}{N}\right)\cdot h\left(\bigcup_{i=1}^{M}\bigcup_{j=1}^{N} S_{i,j}\right). \qquad \blacktriangleleft$$

B Chernoff Bounds

In the analysis of the algorithm we use the following Chernoff-like bounds.

▶ **Lemma 15** (Theorem 3.1 in [13]). *Let $X = \sum_{i=1}^{n} X_i \cdot \lambda_i$ where $(X_i)_{i=1}^{n}$ is a sequence of independent Bernoulli random variable and $\lambda_i \in [0,1]$ for $1 \le i \le n$. Then for any $\varepsilon \in (0,1)$ and $\eta \ge \mathbb{E}[X]$ it holds that*

$$\Pr\left(X > (1+\varepsilon)\eta\right) < \exp\left(-\frac{\varepsilon^2}{3}\eta\right)$$

▶ **Lemma 16** (Theorem 1.3 in [6]). *Let $I = \{1,\ldots,n\}$, $\upsilon > 0$ and $f : 2^I \to \mathbb{R}_+$ be a monotone submodular function such that $f(\{i\}) - f(\emptyset) \le \upsilon$ for any $i \in I$. Let $X_1,\ldots,X_n$ be independent random variables and $\eta = \mathbb{E}[f(\{i \in I | X_i = 1\})]$. Then for any $\varepsilon > 0$ it holds that*

$$\mathbb{E}[f(\{i \in I | X_i = 1\}) \le (1-\varepsilon)\eta] \le \exp\left(-\frac{\eta\cdot\varepsilon^2}{2\upsilon}\right)$$

Linear Expected Complexity for Directional and Multiplicative Voronoi Diagrams

Chenglin Fan
Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA
cxf160130@utdallas.edu

Benjamin Raichel
Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA
benjamin.raichel@utdallas.edu

Abstract

While the standard unweighted Voronoi diagram in the plane has linear worst-case complexity, many of its natural generalizations do not. This paper considers two such previously studied generalizations, namely multiplicative and semi Voronoi diagrams. These diagrams both have quadratic worst-case complexity, though here we show that their expected complexity is linear for certain natural randomized inputs. Specifically, we argue that the expected complexity is linear for: (1) semi Voronoi diagrams when the visible direction is randomly sampled, and (2) for multiplicative diagrams when either weights are sampled from a constant-sized set, or the more challenging case when weights are arbitrary but locations are sampled from a square.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Voronoi Diagrams, Expected Complexity, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.45

Related Version `https://arxiv.org/abs/2004.09385`

Funding Partially supported by NSF CRII Award 1566137 and CAREER Award 1750780.

Acknowledgements The authors want to thank Sariel Har-Peled for useful discussions concerning the case of sampled site locations. Also, thank you to the reviewers for their helpful comments.

1 Introduction

Given a set of point sites in the plane, the Voronoi diagram is the corresponding partition of the plane into cells, where each cell is the locus of points in the plane sharing the same closest site. This fundamental structure has a wide variety of applications. When coupled with a point location data structure, it can be used to quickly answer nearest neighbor queries. Other applications include robot motion planning, modeling natural processes in areas such as biology, chemistry, and physics, and moreover, the dual of the Voronoi diagram is the well known Delaunay triangulation. See the book [5] for an extensive coverage of the topic.

Many of the varied applications of Voronoi diagrams require generalizing its definition in one way or another, such as adding weights or otherwise altering the distance function, moving to higher dimensions, or considering alternative types of sites. While some of these generalizations retain the highly desirable linear worst-case complexity of the standard Voronoi diagram, many others unfortunately have quadratic worst-case complexity or more.

Within the field of Computational Geometry, particularly in recent years, there have been a number of works analyzing the expected complexity of various geometric structures when the input is assumed to have some form of randomness. Here we continue this line of work, by studying the expected complexity of two previously considered Voronoi diagram variants.

Directional and weighted Voronoi diagrams. A natural generalization to consider is when each site is only visible to some subset of the plane. Here we are interested in the so called visual restriction Voronoi diagram (VRVD) [6, 13], where a given site p is only visible to the subset of points contained in some cone with base point p and angle α_p. These diagrams model scenarios where the site has a restricted field of view, such as may be the case with various optical sensors or human vision. For example, in a football game, each player has their own field of view at any given time, and the location of the ball in the VRVD tells us which player is the closest to the ball among those who can see it. When the visible region for each site is a half-plane whose boundary passes through the site (i.e., a VRVD where $\alpha_p = \pi/2$ for each site p), such diagrams are called semi Voronoi diagrams [10]. Just like general VRVD's, semi Voronoi diagrams have $\Theta(n^2)$ worst-case complexity. Our expected analysis is shown for the semi Voronoi diagram case, however, we remark a similar analysis implies the same bounds hold more generally for VRVD's.

The other generalization we consider is when sites have weights. There are many natural ways to incorporate weights into the distance function of each site. Three of the most common are additive [14, 18, 19], power [3], and multiplicative [4] Voronoi diagrams (see also [5]). (For brevity, throughout we use the prefix *multiplicative*, rather than the more common *multiplicatively-weighted*.) For additive Voronoi diagrams the distance from a point x in the plane to a given site p is $d(x,p) = ||x-p|| + \alpha_p$, for some constant α_p, which can vary for each site. For power diagrams the distance is given by $d(x,p) = ||x-p||^2 - \alpha_p^2$. For multiplicative diagrams the distance is given by $d(x,p) = \alpha_p \cdot ||x-p||$. The worst-case complexity for additive and power diagrams is only linear [3, 19]. Here our focus is on multiplicative diagrams, whose worst-case complexity is known to be $\Theta(n^2)$ [4]. Multiplicative diagrams are used to model, for example, crystal growth where crystals grow together from a set of sites at different rates.

Previous expected complexity bounds. There are many previous results on the expected complexity of various geometric structures under one type of randomness assumption or another. Here we focus on previous results relating to Voronoi diagrams. For point sites in $\mathbb{R}^d$, the worst-case complexity of the Voronoi diagram is known to be $\Theta(n^{\lceil d/2 \rceil})$, however, if the sites are sampled uniformly at random from a d-ball or hypercube, for constant d, then the expected complexity is $O(n)$ [7, 12]. For a set of n point sites on a terrain consisting of m triangles, Aronov *et al.* [2] showed that, if the terrain satisfies certain realistic input assumptions then the worst-case complexity of the geodesic Voronoi diagram is $\Theta(m + n\sqrt{m})$. Under a relaxed set of assumption, Driemel *et al.* [11] showed that the expected complexity is only $O(n+m)$, when the sites are sampled uniformly at random from the terrain domain.

Relevant to the current paper, Har-Peled and Raichel [17] showed that, for any set of point site locations in the plane and any set of weights, if the weights are assigned to the points according to a random permutation, then the expected complexity of the multiplicative Voronoi diagram is $O(n \log^2 n)$. The motivation for this work came from Agarwal *et al.*[1], who showed that if one randomly fattens a set of segments in the plane, by taking the Minkowski sum of each segment with a ball of random radius, then the expected complexity of the union is near linear, despite having quadratic worst-case complexity. In a follow-up work to [17], Chang *et al.*[9] defined the candidate diagram, a generalization of weighted diagrams to multi-criteria objective functions, and showed that under similar randomized input assumptions the expected complexity of such diagrams is $O(n \operatorname{polylog} n)$.

Our results and significance. Our first result concerns semi Voronoi diagrams, where the visible region of each site is a half-plane whose bounding line passes through the site, and where the worst-case complexity is quadratic. For any set of site locations and bounding

lines, we show that if the visible side of each site's bounding line is sampled uniformly at random, then the expected complexity of the semi Voronoi diagram is linear, and $O(n \log^3 n)$ with high probability. To achieve this we argue that our randomness assumption implies that any point in the plane is likely to be visible by one of its k nearest neighbors, for large enough k. Thus within each cell of the order-k Voronoi diagram, one can argue the complexity of semi Voronoi diagram is $O(k^2)$, and so summing over all $O(nk)$ cells gives an $O(nk^3)$ bound. This is a variant of the candidate diagram approach introduced in [17]. Unfortunately, in general this approach requires k to be more than a constant, which will not yield the desired linear bound. Thus here, in order to get a linear bound, we give a new refined version of this approach, by carefully allowing k to vary as needed, in the end producing a diagram which is the union of order-k cells, for various values of k.

The main focus of the paper is the second part concerning multiplicative Voronoi diagrams, where the distance to each site is the Euclidean distance multiplied by some site-dependent weight, and where the worst-case complexity is quadratic. We first argue that if the weights are sampled from a set with constant size, then interestingly a similar refined candidate diagram approach yields a linear bound. Our main result, however, considers the more challenging case when no restrictions are made on the weights, but instead the site locations are sampled uniformly at random from the unit square. For this case we show that in any grid cell of side length $1/\sqrt{n}$ in the unit square the expected complexity of the multiplicative diagram is $O(1)$. This implies an $O(n)$ expected bound for the multiplicative diagram over the whole unit square. This improves over the $O(n \log^2 n)$ bound of [17] for this case, by using an interesting new approach which "stretches" sites about a given cell based on their weights, thus approximately transforming the weighted diagram into an unweighted one with respect to a given cell. As this main result is technically very challenging, it is placed last.

It is important to note the significance of our bounds being linear. Given a randomness assumption, one wishes to prove an optimal expected complexity bound, and for the diagrams we consider linear bounds are immediately optimal. However, even when a linear bound seems natural, many standard approaches to bounding the expected complexity yield additional log factors (e.g. [1, 9, 17]). That is, not only are linear bounds desirable, but also they are nontrivial to obtain. However, when they are obtained, they reveal the true expected complexity of the structure, free from artifacts of the analysis.

▶ Remark 1. Our paper focuses on diagram complexity not running time. However, Corollary 4 below shows our approach implies that the semi Voronoi diagram can be computed in $O(n \log^3 n)$ expected time in our randomized model, by using the previous quadratic time algorithm of Fan *et al.* [13] on appropriate subsets of sites. For the multiplicate Voronoi diagram, [17] remarked that their approach implied an $O(n \log^3 n)$ expected time algorithm in their model, which also covers all multiplicative models considered in the current paper.

2 Preliminaries

The standard Voronoi diagram: Let $S = \{s_1, s_2, ..., s_n\} \subset \mathbb{R}^2$ be a set of n point *sites* in the plane. Let $\|x - y\|$ denote the Euclidean distance from x to y, and for two sites $s_i, s_j \in S$ let $\beta(s_i, s_j)$ denote their bisector, that is the set of points x in plane such that $\|s_i - x\| = \|s_j - x\|$. Any site $s_i \in S$ induces a distance function $f_i(x) = \|s_i - x\|$ defined for any point x in the plane. For any subset $T \subseteq S$, the *Voronoi cell* of $s_i \in T$ with respect to T, $\mathcal{V}_{\text{cell}}(s_i, T) = \{x \in \mathbb{R}^2 \mid \forall s_j \in T \quad f_i(x) \leq f_j(x)\}$, is the locus of points in the plane having s_i as their closest site from T. We define the *Voronoi diagram* of T, denoted $\mathcal{V}(T)$, as the partition of the plane into Voronoi cells induced by the minimization diagram (see [15]) of the set of distance functions $\{f_i \mid s_i \in T\}$, that is the projection onto the plane of the lower envelope of the surfaces defined by these bivariate functions.

One can view the union, U, of the boundaries of the cells in the Voronoi diagram as a planar graph. Specifically, define a *Voronoi vertex* as any point in U which is equidistant to three sites in S (happening at the intersection of bisectors). For simplicity, we make the general position assumption that no point is equidistant to four or more sites. Furthermore, define a *Voronoi edge* as any maximal connected subset of U which does not contain a Voronoi vertex. (For each edge to have two endpoints we include the "point" at infinity, i.e., the graph is defined on the stereographic projection of the plane onto the sphere.) The complexity of the Voronoi diagram is then defined as the total number of Voronoi edges, vertices, and cells. As the cells are simply connected sets, which are faces of a straight-line planar graph, the overall complexity is $\Theta(n)$.

Order-k Voronoi diagram: Let S be a set of n point sites in the plane. The *order-k Voronoi diagram* of S is the partition of the plane into cells, where each cell is the locus of points having the same set of k nearest sites of S (the ordering of these k sites by distance can vary within the cell). It is not hard to see that this again defines a straight-line partition of the plane into cells where the edges on the boundary of a cell are composed of bisector pieces. The worst-case complexity of this diagram is $\Theta(k(n-k))$ (see [5, Section 6.5]).

We also consider the diagram where every point in a cell not only has the same k nearest sites, but also the same ordering of distances to these sites. We refer to this as the order-k *sequence* Voronoi diagram, known to have $O(nk^3)$ worst-case complexity (see Appendix B).

Semi Voronoi diagram: Let $S = \{s_1, s_2, ..., s_n\}$ be a set of n point sites in the plane, where for each s_i there is an associated closed half-plane $\mathsf{H}(s_i)$, whose boundary passes through s_i. We use $L(s_i)$ to denote the bounding line of $\mathsf{H}(s_i)$. For any point x in the plane and any site $s_i \in S$, we say that x and s_i are visible to each other when $x \in \mathsf{H}(s_i)$. Given a point x, let $S(x) = \{s_i \in S \mid x \in \mathsf{H}(s_i)\}$ denote the set of sites which are visible to x.

For any subset $T \subseteq S$, define the *semi Voronoi cell* of $s_i \in T$ with respect to T as, $\mathcal{SV}_{\text{cell}}(s_i, T) = \{x \in \mathsf{H}(s_i) \mid \forall s_j \in T \cap S(x) \quad \|x - s_i\| \leq \|x - s_j\|\}$. It is possible that there are points in the plane which are not visible by any site in S. If desired, this technicality can be avoided by adding a pair of sites far away from S which combined can see the entire plane. As before, semi Voronoi cells define a straight-line partition of the plane, where now the edges on the boundary of a cell are either portions of a bisector or of a half-plane boundary. In the worst case, the semi Voronoi diagram can have quadratic complexity [13].

Random semi Voronoi diagram: We consider semi Voronoi diagrams where the set of sites $S = \{s_1, \ldots, s_n\}$ is allowed to be any fixed set of n points in general position. For each site s_i, the line bounding the half-plane of s_i, $L(s_i)$, is allowed to be any fixed line in $\mathbb{R}^2$ passing through s_i. Such a line defines two possible visible closed half-spaces. We assume that independently for each site s_i, one of these two spaces is sampled uniformly at random.

An alternative natural assumption is that the normal of the half-plane for each site is sampled uniformly at random from $[0, 2\pi)$. Note our model is strictly stronger, that is any bound we prove will imply the same bound for this alternative formulation. This is because one can think of sampling normals from $[0, 2\pi)$, as instead first sampling directions for the bounding lines from $[0, \pi)$, and then sampling one of the two sides of each line for the normal.

Multiplicative Voronoi diagram: Let $S = \{s_1, s_2, ..., s_n\}$ be a set of n point sites in the plane, where for each site s_i there is an associated weight $w_i > 0$. Any site $s_i \in S$ induces a distance function $f_i(x) = w_i \cdot \|s_i - x\|$ defined for any point x in the plane. For any subset $T \subseteq S$, the

Voronoi cell of $s_i \in T$ with respect to T, $\mathcal{V}_{\text{cell}}(s_i, T) = \{x \in \mathbb{R}^2 \mid \forall s_j \in T \quad f_i(x) \leq f_j(x)\}$, is the locus of points in the plane having s_i as their closest site from T. The *multiplicative Voronoi diagram* of T, denoted $\mathcal{WV}(T)$, is the partition of the plane into Voronoi cells induced by the minimization diagram of the distance functions $\{f_i \mid s_i \in T\}$.

Note that unlike the standard Voronoi diagram, the bisector of two sites is in general an Apollonius disk, potentially leading to disconnected cells. Ultimately, the diagram still defines a planar arrangement and so its complexity, denoted by $|\mathcal{WV}(S)|$, can still be defined as the number of edges, faces, and vertices of this arrangement. Note the edges are circular arcs and straight line segments and thus are still constant complexity curves. In the worst case, the multiplicative Voronoi diagram has $\Theta(n^2)$ complexity [4].

3 The Expected Complexity of Random Semi Voronoi Diagrams

3.1 The probability of covering the plane

As it is used in our later calculations, we first bound the probability that for a given subset X of k sites of S that there exists a point in the plane not visible to any site in X.

▶ **Lemma 2.** *For any set $X = \{x_1, \ldots, x_k\}$ of k sites* $\mathbf{Pr}\left[\left(\bigcup_{x_i \in X} \mathcal{SV}_{\text{cell}}(x_i, X)\right) \neq \mathbb{R}^2\right] \leq (k(k+1)+2)/2^{k+1} = O(k^2/2^k)$.

Proof. Consider the arrangement of the k bounding lines $L(x_1), \ldots, L(x_k)$. Let $\mathcal{F}$ denote the set of faces in this arrangement (i.e., the connected components of the complement of the union of lines), and note that $|\mathcal{F}| \leq k(k+1)/2+1 = O(k^2)$. Observe that for any face $f \in \mathcal{F}$ and any fixed site $x_i \in X$, either every point in f is visible by x_i or no point in f is visible by x_i. Moreover, the probability that face f is not visible by site x_i is $\mathbf{Pr}[\mathsf{H}(x_i) \cap f = \emptyset] \leq 1/2$. Hence the probability that a face f is not visible by any of the k sites is

$$\mathbf{Pr}\left[\bigcup_{x_i \in X} \mathsf{H}(x_i) \cap f = \emptyset\right] \leq 1/2^k.$$

Hence the probability that at least one face in $\mathcal{F}$ is is not visible by any sites in X is

$$\mathbf{Pr}\left[\left(\bigcup_{x_i \in X} \mathcal{SV}_{\text{cell}}(x_i, X)\right) \neq \mathbb{R}^2\right] \leq \sum_{f \in \mathcal{F}} \mathbf{Pr}\left[\bigcup_{x_i \in X} \mathsf{H}(x_i) \cap f = \emptyset\right] \leq \frac{k(k+1)+2}{2^{k+1}} = O\left(\frac{k^2}{2^k}\right). \quad \blacktriangleleft$$

3.2 A simple near linear bound

Ultimately we can show that the expected complexity of a random semi Voronoi diagram is linear, however, here we first show that Lemma 2 implies a simple near linear bound which also holds with high probability. Specifically, we say that a quantity is bounded by $O(f(n))$ with high probability, if for any constant α there exists a constant β, depending on α, such that the quantity is at most $\beta \cdot f(n)$ with probability at least $1 - 1/n^\alpha$.

▶ **Lemma 3.** *Let $S = \{s_1, \ldots, s_n\} \subset \mathbb{R}^2$ be a set of n sites, where each site has a corresponding line $L(s_i)$ passing through s_i. For each s_i, sample a half-plane $\mathsf{H}(s_i)$ uniformly at random from the two half-planes whose boundary is $L(s_i)$. Then the expected complexity of the semi Voronoi diagram on S is $O(n \log^3 n)$, and moreover this bound holds with high probability.*

Proof. Let $k = c \log n$, for some constant c. Consider the order-k Voronoi diagram of S. First triangulate this diagram so the boundary of each cell has constant complexity. (Note triangulating does not asymptotically change the number of cells.) Fix any cell Δ in this triangulation, which in turn fixes some (unordered) set X of k-nearest sites. By Lemma 2,

$$\mathbf{Pr}\left[\left(\bigcup_{x_i \in X} \mathcal{SV}_{\text{cell}}(x_i, X)\right) \neq \mathbb{R}^2\right] = O(k^2/2^k) = O((c \log n)^2/2^{c \log n}) = O(1/n^{c-\varepsilon'}),$$

for any arbitrarily small value $\varepsilon' > 0$. Thus with polynomially high probability for every point in Δ its closest visible site will be in X. Now let T be the set of all $O(k(n-k)) = O(n \log n)$ triangles in the triangulation of the order-k diagram. Observe that the above high probability bound applies to any triangle $\Delta \in T$. Thus taking the union bound we have that with probability at least $1 - 1/n^{c-(1+\varepsilon)}$ (where $\varepsilon > \varepsilon' > 0$ is an arbitrarily small value), simultaneously for every triangle Δ, every point in Δ will be visible by one of its k closest sites. Let e denote this event happening (and $\overline{e}$ denote it not happening). Conditioning on e happening, there are only $O(k) = O(\log n)$ relevant sites which contribute to the semi Voronoi diagram of any cell Δ. Thus the total complexity of the semi Voronoi diagram restricted to any cell Δ is at most $O(\log^2 n)$, as the semi Voronoi diagram has worst-case quadratic complexity [13]. (Note that as Δ is a triangle, we can ignore the added complexity due to clipping the semi Voronoi diagram of these sites to Δ.) On the other hand, if $\overline{e}$ happens, then the worst-case complexity of the entire semi Voronoi diagram is still $O(n^2)$.

Now the complexity of the semi Voronoi diagram is bounded by the sum over the cells in the triangulation of the complexity of the diagram restricted to each cell. Thus the above already implies that with high probability the complexity of the semi Voronoi diagram is $O\big(\sum_{\Delta \in T} \log^2 n\big) = O(n \log^3 n)$. As for the expected value, by choosing c sufficiently large,

$$\begin{aligned}
&= \mathbf{E}\big[|\mathcal{SV}(S)| \;\big|\; e\big]\,\mathbf{Pr}[e] + \mathbf{E}\big[|\mathcal{SV}(S)| \;\big|\; \overline{e}\big]\,\mathbf{Pr}[\overline{e}] = O\left(\sum_{\Delta \in T} \log^2 n\right)\mathbf{Pr}[e] + O(n^2)\,\mathbf{Pr}[\overline{e}] \\
&= O(n \log^3 n)\,\mathbf{Pr}[e] + O(n^2)\,\mathbf{Pr}[\overline{e}] = O(n \log^3 n)\,\mathbf{Pr}[e] + O(n^2) \cdot (1/n^{c-(1+\varepsilon)}) \\
&= O(n \log^3 n) + O(1/n^{c-(3+\varepsilon)}) = O(n \log^3 n).
\end{aligned}$$

◀

The analysis of Lemma 3 immediately implies an algorithm with the same time bounds, by using the quadratic time algorithm for semi Voronoi diagrams of Fan *et al.* [13] for the subset of sites of each order-k cell.

▶ **Corollary 4.** *Let the input be as in Lemma 3. Then the semi Voronoi diagram can be computed in $O(n \log^3 n)$ expected time, and moreover this bound holds with high probability.*

Proof. Compute the order-k Voronoi diagram, for $k = c \log n$, where c is the constant determined in the proof of Lemma 3. Triangulate the diagram. Check if every point of every triangle is visible by one of its $k = \Theta(\log n)$ closest sites. If so, then compute the semi Voronoi diagram of the k nearest sites in each triangle, and clip it to the triangle. If not, ignore the triangles, and compute the semi Voronoi diagram of all n sites.

For $k = \Theta(\log n)$, the order-k diagram can be computed in $O(n \log^3 n)$ time [8]. Triangulating the diagram takes linear time in the complexity of the diagram. By Lemma 3, with high probability every point of every triangle is visible by one of its k closest sites. Thus with high probability the running time is $O(n \log^3 n)$, as computing the semi Voronoi diagram of the $k = \Theta(\log n)$ nearest sites in each triangle takes $O(\log^2 n)$ time per triangle, and thus $O(n \log^3 n)$ over all the $O(n \log n)$ triangles. If some triangle is not fully visible, then computing the semi Voronoi diagram of all n sites takes $O(n^2)$ time. Thus by the same expected analysis at the end of the proof of Lemma 3, the expected time is $O(n \log^3 n)$. ◀

3.3 An optimal linear bound

The previous subsection partitioned the plane based on the order-k Voronoi diagram, for $k = c \log n$, and then argued that simultaneously for all cells one of the k nearest sites will be visible. This argument is rather coarse, but instead of using a fixed large value for k, if one is more careful and allows k to vary, then one can argue the expected complexity is linear. Note that in the following, rather than using the standard unordered order-k Voronoi diagram, we use the more refined order-k *sequence* Voronoi diagram as defined above.

▶ **Theorem 5.** *Let $S = \{s_1, \ldots, s_n\} \subset \mathbb{R}^2$ be a set of n sites, where each site has a corresponding line $L(s_i)$ passing through s_i. For each s_i, sample a half-plane $\mathsf{H}(s_i)$ uniformly at random from the set of two half-planes whose boundary is $L(s_i)$ (i.e., each has $1/2$ probability). Then the expected complexity of the semi Voronoi diagram on S is $\Theta(n)$.*

Proof. Consider the partition of the plane by the first order Voronoi diagram of S, i.e., the standard Voronoi diagram. We iteratively refine the cells of this partition into higher order sequence Voronoi diagram cells. At each iteration we have a collection of order-i sequence Voronoi diagram cells, and for each cell we either mark it final and stop processing, or further refine the cell into its constituent order-$(i+1)$ sequence cells. Specifically, a cell Δ is marked final in the ith iteration if every point in Δ is visible by one of its i nearest sites, or when $i = n$ and the cell cannot be refined further. The process stops when all cells are marked final. Below we use F^k to denote the set of cells which were marked final in the kth iteration.

Let C^k be the set of all cells of the order-k sequence Voronoi diagram of S. Note that the order-k sequence Voronoi diagram can be constructed by iteratively refining lower order cells, and hence any cell seen at any point in the above process is a cell of the order-k sequence diagram for some value of k. So consider any cell $\Delta_j \in C^k$ of the order-k sequence diagram of S. Let $\#(\Delta_j)$ be the number of order-$(k+1)$ sequence diagram cells inside Δ_j, and let X_j be the indicator variable for the event that Δ_j is refined into its constituent order-$(k+1)$ sequence cells in the kth round of the above iterative process. Note that $\mathbf{Pr}[X_j = 1]$ is upper bounded by the probability that Δ_j has not been marked final by the end of the kth round, which in turn is bounded by Lemma 2. Thus letting Z^{k+1} be the random variable denoting the total number of order-$(k+1)$ sequence cells created in the above process, we have

$$\mathbf{E}\left[Z^{k+1}\right] = \mathbf{E}\left[\sum_{\Delta_j \in C^k} \#(\Delta_j) \cdot X_j\right] = \sum_{\Delta_j \in C^k} \#(\Delta_j) \cdot \mathbf{E}[X_j] = O\left(\frac{k^2}{2^k}\right) \cdot \sum_{\Delta_j \in C^k} \#(\Delta_j) = O\left(\frac{nk^5}{2^k}\right),$$

as $\sum \#(\Delta_j)$ is at most the total number of cells in the order-k sequence Voronoi diagram, which is bounded by $O(nk^3)$ (see Appendix B).

Using the same argument as in Lemma 3, the complexity of the random semi Voronoi diagram is bounded by $\sum_{k=1}^{n} |F^k| \cdot O(k^2)$. (If $k = n$ then a cell may not be fully visible, though the bound still applies as the worst-case complexity is $O(n^2)$.) Thus by the above, the expected complexity is

$$\mathbf{E}[|\mathcal{SV}(S)|] \leq \sum_{k=1}^{n} \mathbf{E}\left[|F^k|\right] \cdot O(k^2) \leq \sum_{k=1}^{n} \mathbf{E}\left[Z^k\right] \cdot O(k^2) \leq O\left(\sum_{k=1}^{n} \frac{nk^7}{2^k}\right) = O(n). \qquad \blacktriangleleft$$

▶ Remark 6. For simplicity the results above were presented for semi Voronoi diagrams, though they extend to the more general VRVD case, where the visibility region of s_i is determined by a cone with base point s_i and angle α_i, where the orientation of the cone is sampled uniformly at random. Specifically, the plane can be covered by a set of $(2\pi)/(\alpha_i/2) = 4\pi/\alpha_i$ cones around s_i each with angle $\alpha_i/2$. Any one of these smaller cones is completely contained

in the randomly selected α_i cone with probability $\alpha_i/(4\pi)$. If the α_i are lower bounded by a constant β, one can then prove a variant of Lemma 2 (and hence Lemma 3 and Theorem 5), as the arrangement of all these smaller cones still has $O(k^2)$ complexity, and the probability a face is not visible to any site is still exponential in k but with base proportional to $1-\beta/(4\pi)$.

4 The Expected Complexity of Multiplicative Voronoi Diagrams

In this section we consider the expected complexity of multiplicative Voronoi diagrams under different randomness assumptions. First, by using the approach from the previous section, we show that for any set of site locations, if each site samples its weight from a set of constant size c, then the expected complexity is $O(nc^6)$. Next, we consider the case when the sites can have arbitrary weights, but the site locations are sampled uniformly at random from the unit square. As making no assumptions on the weights makes the problem considerably more challenging, as a warm-up, we first assume the weights are in an interval $[1, c]$. In this case, we consider a $1/\sqrt{n}$ side length grid, and argue that locally in each grid cell the expected complexity of the multiplicative diagram is $O(c^4)$ (inspired by the approach in [11]), and thus over the entire unit square the complexity is $O(nc^4)$. Finally, we remove the bounded weight assumption, and argue that for any set of weights, the expected complexity is linear when site locations are sampled uniformly at random from the unit square, by introducing the notion of "stretched" sites.

4.1 Sampling from a small set of weights

▶ **Lemma 7.** *Let $W = \{w_1, w_2, ..., w_c\}$ be a set of non-negative real weights and let $S = \{s_1, s_2, s_3, \ldots, s_n\}$ be a set of point sites in the plane, where each site in S is assigned a weight independently and uniformly at random from W. Then the expected complexity of the multiplicative Voronoi diagram of S is $O(n \cdot c^6)$.*

Proof. Consider the partition of the plane determined by the unweighted first order Voronoi diagram of S, i.e., the standard Voronoi diagram. Following the strategy of the proof of Theorem 5, we iteratively refine the cells of this partition into higher order sequence Voronoi diagram cells, except now a cell Δ is marked final in the ith iteration if at least one of its i-nearest sites has weight $w_m = \min\{w_1, w_2, w_3, ..., w_c\}$. Analogous to the semi Voronoi diagram case, with this new definition if a cell Δ is marked final in the ith iteration then only its i unweighted nearest sites can contribute to the diagram within Δ. This is because one of these i nearest sites is both closer and has weight less than or equal to any site outside of the i nearest sites. Note also that the probability that k sites are all assigned weight larger than w_m is $(1-1/c)^k$. Using the same notation as in the proof of Theorem 5, we have

$$\begin{aligned}\mathbf{E}\left[Z^{k+1}\right] &= \mathbf{E}\left[\sum_{\Delta_j \in C^k} \#(\Delta_j) \cdot X_j\right] = \sum_{\Delta_j \in C^k} \#(\Delta_j) \cdot \mathbf{E}[X_j] \\ &= O\left((1-1/c)^k\right) \cdot \sum_{\Delta_j \in C^k} \#(\Delta_j) = O\left(nk^3(1-1/c)^k\right).\end{aligned}$$

As the worst-case complexity of the multiplicative Voronoi diagram of k sites is $O(k^2)$, overall the complexity of the multiplicative Voronoi diagram is bounded by $\sum_{k=1}^{n} |F^k| \cdot O(k^2)$. Thus by the above, the expected complexity is

$$\mathbf{E}[|\mathcal{WV}(S)|] \leq \sum_{k=1}^{n} \mathbf{E}\left[|F^k|\right] \cdot O(k^2) \leq \sum_{k=1}^{n} \mathbf{E}\left[Z^k\right] \cdot O(k^2) \leq O\left(\sum_{k=1}^{n} nk^5(1-1/c)^k\right) = O(n \cdot c^6),$$

where the last step is obtained by viewing the sum as a power series in $x = (1 - 1/c)$,

$$\sum_{k>0} k^5(1-1/c)^k = \sum_{k>0} k^5 x^k = \left(x \cdot \frac{d}{dx}\right) \sum_{k>0} k^4 x^k = \left(x \cdot \frac{d}{dx}\right)^5 \sum_{k>0} x^k = \left(x \cdot \frac{d}{dx}\right)^5 \frac{x}{1-x}$$
$$= \frac{(x^5 + 26x^4 + 66x^3 + 26x^2 + x)}{(1-x)^6} = 120c^6 - 360c^5 + 390c^4 - 180c^3 + 31c^2 - c = O(c^6). \quad \blacktriangleleft$$

4.2 Sampling site locations with bounded weights

In this section we argue that the expected complexity of the multiplicative Voronoi diagram is linear when the site locations are uniformly sampled and the weights are in a constant spread interval. In the next section we remove the bounded weight assumption. Thus the current section can be viewed as a warm-up, and serves to illustrate the extent to which assuming bounded weights simplifies the problem. However, as the results of the next section subsume those here, this section can be safely skipped if desired.

The following fact is used both in the proof of the lemma below and the next section.

▶ **Fact 8.** *Consider doing m independent experiments, where the probability of success for each experiment is α. Let X be the total number of times the experiments succeed. Then $\mathbf{E}[X^2] \leq \alpha m + \alpha^2 m^2 = \mathbf{E}[X] + \mathbf{E}[X]^2$.*

Note that the above fact holds since X is a binomial random variable, $Bin(\alpha, m)$, and so $\mathbf{E}[X^2] = \mathbf{Var}[X] + \mathbf{E}[X]^2 = m\alpha(1-\alpha) + (m\alpha)^2$.

▶ **Lemma 9.** *Let $S = \{s_1, s_2, s_3, ..., s_n\}$ be a set of point sites in the plane, where for some value $c \geq 1$, each site in S is assigned a weight $w_i \in [1, c]$. Suppose that the location of each site in S is sampled uniformly at random from the unit square U. Then the expected complexity of the multiplicative Voronoi diagram of S within U is $O(n \cdot c^4)$.*

Proof. Place a regular grid over U, where grid cells have side length $1/\sqrt{n}$. Fix any grid cell $\square = \square_{x,y}$, where $(x, y) \in \sqrt{n} \times \sqrt{n}$. We now argue that the expected complexity of the multiplicative Voronoi diagram within $\square$ is $O(c^4)$, and thus by linearity of expectation, the expected complexity over all n grid cells in U is $O(nc^4)$.

Let ρ be the random variable denoting the unweighted distance of the closest site in S to the center of the grid cell $\square$, and let X_ρ be the random variable denoting the number of points which contribute to the multiplicative Voronoi diagram in $\square$ conditioned on the value ρ. (For now ignore the contribution of the point at distance exactly ρ.) Observe that any point in S which contributes to the multiplicative Voronoi diagram within $\square$ must lie within the annulus centered at the center of $\square$, with inner radius ρ and outer radius $c(\rho + \sqrt{2/n})$. Conditioned on the value ρ, let α_ρ be the probability for a point to fall in this annulus, and let X_ρ be the binomial random variable $Bin(\alpha_\rho, n)$ representing the number of points which fall into this annulus. For now assume $\rho \leq 1/4$, in which case the region outside the disk centered at the center of $\square$ and with radius ρ has area at least $3/4$. We then have that

$$\mathbf{E}[X_\rho] \leq n\alpha_\rho \leq n\frac{(\pi(c(\rho + \sqrt{2/n}))^2 - \pi\rho^2)}{3/4}$$
$$= O(nc^2(\rho^2(1 - 1/c^2) + \rho/\sqrt{n} + 1/n)) = O(c^2 + n(c\rho)^2).$$

Let Y_ρ be the random variable denoting the complexity of the multiplicative Voronoi diagram within $\square$ when conditioned on the value ρ. As the worst-case complexity of the multiplicative Voronoi diagram is quadratic, we have $Y_\rho = O((1+X_\rho)^2) = O(1+X_\rho+X_\rho^2)$, where the plus 1 counts the point at distance exactly ρ. Thus using Fact 8, and again assuming $\rho \leq 1/4$,

$$\mathbf{E}[Y_\rho] = O(1+\mathbf{E}[X_\rho]+\mathbf{E}\left[X_\rho^2\right]) = O(c^2+n(c\rho)^2+(c^2+n(c\rho)^2)^2) = O(c^4(1+n\rho^2+n^2\rho^4)).$$

Now consider the event that $\rho \in [i/\sqrt{n},(i+1)/\sqrt{n}]$ for some integer i. For this to happen, the open disk with radius $i/\sqrt{n}$ centered at the center of $\square$ must be empty, and one of the n points must lie in the annulus with inner radius $i/\sqrt{n}$ and outer radius $(i+1)/\sqrt{n}$. Therefore,

$$\begin{aligned}\mathbf{Pr}\left[\rho \in [i/\sqrt{n},(i+1)/\sqrt{n}]\right] &\leq n\cdot(\pi((i+1)/\sqrt{n})^2-\pi(i/\sqrt{n})^2)\cdot(1-\pi(i/\sqrt{n})^2)^{n-1}\\ &= \pi(2i+1)\cdot(1-\pi i^2/n)^{n-1} \leq \pi(2i+1)e^{-\pi i^2(n-1)/n} \leq \pi(2i+1)e^{-i^2}.\end{aligned}$$

Furthermore, by the above, when $\rho \in [i/\sqrt{n},(i+1)/\sqrt{n}]$ and $\rho \leq 1/4$, we have

$$\mathbf{E}[Y_\rho] = O(c^4(1+n\rho^2+n^2\rho^4)) = O(c^4(1+(i+1)^2+(i+1)^4)),$$

and for $\rho \geq 1/4$ we have the trivial bound $\mathbf{E}[Y_\rho] = O(n^2)$.

Finally, let Y be the random variable denoting the complexity of the multiplicative Voronoi diagram within $\square$. By the law of total expectation we have,

$$\begin{aligned}\mathbf{E}[Y] &\leq \left(\sum_{i=0}^{\sqrt{n}/4-1}\mathbf{Pr}\left[\rho\in\left[\frac{i}{\sqrt{n}},\frac{(i+1)}{\sqrt{n}}\right]\right]\cdot O(c^4(1+(i+1)^2+(i+1)^4))\right)\\ &\quad+\left(\sum_{i=\sqrt{n}/4}^{\sqrt{n}}\mathbf{Pr}\left[\rho\in\left[\frac{i}{\sqrt{n}},\frac{(i+1)}{\sqrt{n}}\right]\right]\cdot O(n^2)\right)\\ &\leq \left(\sum_{i=0}^{\sqrt{n}/4-1}\frac{\pi(2i+1)}{e^{i^2}}\cdot O(c^4(1+(i+1)^2+(i+1)^4))\right)+\left(\sum_{i=\sqrt{n}/4}^{\sqrt{n}}\frac{\pi(2i+1)}{e^{i^2}}\cdot O(n^2)\right)\\ &= O(1)+\sum_{i=0}^{\sqrt{n}/4-1}\frac{\pi(2i+1)}{e^{i^2}}\cdot O(c^4(1+(i+1)^2+(i+1)^4))\\ &= O(1)+\sum_{i=0}^{\sqrt{n}/4}\frac{2i+1}{e^{i^2}}\cdot O(c^4(i+1)^4) = O(c^4)+O\left(\sum_{i=1}^{\sqrt{n}/4}\frac{c^4i^5}{e^{i^2}}\right) = O(c^4).\end{aligned}$$

◀

4.3 Sampling sites locations in general

In this section we argue that the expected complexity of the multiplicative Voronoi diagram is linear when the site locations are uniformly sampled. Here the weights can be any arbitrary set of positive values. Without loss of generality we can assume the smallest weight is exactly 1, as dividing all weights by the same positive constant does not change the diagram. Throughout, m denotes the maximum site weight, hence all weights are in the interval $[1,m]$.

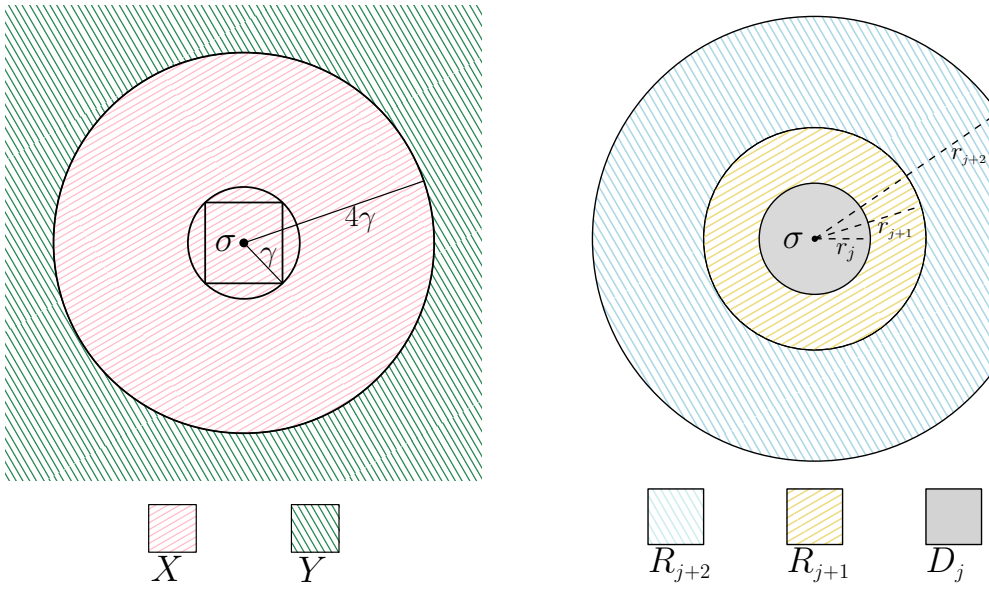
Let σ be an arbitrary point in the unit square. The high level idea is that we want to apply a transformation to the sites such that the weighted Voronoi diagram around σ can be interpreted as an unweighted Voronoi diagram. Specifically, for a site s with weight w and distance $d = ||s-\sigma||$, let the *stretched site* of s with respect to σ, denoted by t, be the point at Euclidean distance $w\cdot d$ from σ which lies on the ray from σ through s. That is, the weighted distance from s to σ is the same as the unweighted distance from t to σ.

So let σ be an arbitrary fixed point in the unit square, $S = \{s_1, \ldots, s_n\}$ be a set of sites with weights $\{w_1, \ldots, w_n\} \subset [1, m]$ whose positions have been uniformly sampled from the unit square, and let $T = \{t_1, \ldots, t_n\}$ be the corresponding set of stretched sites.

Let $\gamma = \sqrt{1/(2n)}$. Our goal is to argue that for any arbitrary choice of σ in U the expected complexity of the multiplicative Voronoi diagram in the ball $B(\sigma, \gamma)$ is constant, i.e., $\mathbf{E}[|\mathcal{WV}(S) \cap B(\sigma, \gamma)|] = O(1)$. Then by the grid argument in the previous section this immediately implies a linear bound on the expected complexity of the overall diagram in U. Namely, place a uniform grid over U, where the grid cell side length is $1/\sqrt{n}$. Then as each cell is contained in a ball $B(\sigma, \gamma)$ for some σ, and there are n cells overall, by the linearity of expectation the expected complexity of the overall diagram is $O(n)$.

At a high level, the idea is simple. We wish to argue that sites whose stretched location is far from σ will be blocked from contributing by sites whose stretched location is closer to σ. However, putting this basic plan into action is tricky and requires handling various edge cases. In particular, later it will become clear why we need to consider the following cases for where sites lie relative to σ.

▶ **Definition 10.** *For a fixed point σ in U, let X be the subset of S which falls in $B(\sigma, 4\gamma)$, i.e., $X = S \cap B(\sigma, 4\gamma)$. Let Y be the complement set, i.e., $Y = S \setminus X$. See Figure 4.1.*



(a) Sets X and Y from Definition 10, and the grid cell and γ radius ball centered at σ.

(b) Exponentially increasing radii, and rings R_{j+1}, R_{j+2} around disk D_j, all centered at σ.

Figure 4.1 Pictorial representations of some of the defined quantities from Section 4.3.

▶ Remark 11. In the following, we typically condition on the set of sites which fall in Y as being fixed, but not the actual precise locations of those sites. We refer to this as "Fixing Y". Ultimately the statements below hold regardless of what sites actually fall in Y.

Fix Y. Let r_1 be the radius such that the expected number of stretched sites from Y that are contained in $B(\sigma, r_1)$ is $n \cdot \pi(16\gamma)^2$. Observe that a site can only be moved further from σ after stretching, thus $r_1 \geq 16\gamma$. (Potentially r_1 is significantly larger.) Also, note that as Y is fixed, the value of r_1 is fixed.

For any integer $j > 0$, let D_j be the disk with radius $r_j = r_1 \times 2^{j-1}$, centered at σ. Moreover, define the rings $R_{j+1} = D_{j+1} \setminus D_j$, for any $j > 0$. See Figure 4.1.

▶ **Lemma 12.** *Consider two sites $s_j \in Y$, and s_i such that either $s_i \in Y$ or $s_i \in X$ with $w_i \leq w_j$. For any $k' \geq k \geq 1$, if $t_i \in D_k$ and $t_j \in R_{k'+2}$, then s_j cannot contribute to the multiplicative Voronoi diagram in $B(\sigma, \gamma)$.*

Proof. For site s_i, let $d_i = ||s_i - \sigma||$ and $d'_i = w_i \cdot d_i$. Similarly define d_j and d'_j for site s_j. Note that the furthest weighted distance of a point in $B(\sigma, \gamma)$ to s_i is $w_i \cdot (d_i + \gamma)$, and the closest weighted distance of a point in $B(\sigma, \gamma)$ to s_j is $w_j \cdot (d_j - \gamma)$. Thus it suffices to argue $w_i \cdot (d_i + \gamma) < w_j \cdot (d_j - \gamma)$, or equivalently $(w_i + w_j)\gamma < d'_j - d'_i$. To that end, observe $d'_j - d'_i \geq d'_j - d'_j/2 = d'_j/2$. Thus we only need to argue $(w_i + w_j)\gamma < d'_j/2$.

Case 1, $w_i \leq w_j$: In this case $(w_i + w_j)\gamma \leq w_j \cdot 2\gamma < w_j \cdot d_j/2 = d'_j/2$.

Case 2, $w_i > w_j$: In this case $(w_i + w_j)\gamma \leq w_i \cdot 2\gamma < w_i \cdot d_i/2 = d'_i/2 \leq d'_j/2$. ◀

Note that a site with weight 1 does not move after stretching. Thus as S always has a site with weight 1, there must be at least one stretched site in U. Therefore, if we define $Z - 1$ to be the smallest value of j such that $U \subseteq D_j$, then D_{Z-1} must contain a stretched site. Thus the above lemma implies any site which contributes to the multiplicative Voronoi diagram within $B(\sigma, \gamma)$ must lie within D_Z after being stretched, and so going forward we only consider stretched sites in D_Z.

▶ **Definition 13.** *Fix Y, and consider any $1 \leq j \leq Z - 2$. For $s_i \in Y$, let $p_{i,j}$ denote the probability that t_i is located in D_j. For $s_i \in Y$, let $q_{i,j}$ denote the probability that s_i is located in $U \setminus B(\sigma, 16\gamma)$ and t_i is located in $R_{j+2} \cup R_{j+1}$, conditioned on the event that no stretched sites from Y are located in D_j.*[1]

▶ **Lemma 14.** *Fix Y. For any $1 \leq j \leq Z - 2$ and $s_i \in Y$ we have $q_{i,j} \leq 32 \cdot p_{i,j}$.*

Proof. Note in the following we always take as given that $s_i \in Y$. Observe that

$$p_{i,j} = \mathbf{Pr}[t_i \in D_j] = \mathbf{Pr}[s_i \in U \cap B(\sigma, r_j/w_i)] = \frac{area((U \cap B(\sigma, r_j/w_i)) \setminus B(\sigma, 4\gamma))}{area(U \setminus B(\sigma, 4\gamma))}.$$

Then as the location of the sites in Y are independent, for $q_{i,j}$ we have,

$$\begin{aligned}
q_{i,j} &= \mathbf{Pr}[(s_i \in U \setminus B(\sigma, 16\gamma)) \cap (t_i \in D_{j+2} \setminus D_j) \mid \forall s_k \in Y, t_k \notin D_j] \\
&= \mathbf{Pr}[(s_i \in U \setminus B(\sigma, 16\gamma)) \cap (t_i \in D_{j+2} \setminus D_j) \mid t_i \notin D_j] \\
&= \mathbf{Pr}[(s_i \in U \setminus B(\sigma, 16\gamma)) \cap (t_i \in D_{j+2} \setminus D_j)] \,/\, \mathbf{Pr}[t_i \notin D_j] \\
&= \mathbf{Pr}[s_i \in (U \cap B(\sigma, 4r_j/w_i)) \setminus (B(\sigma, r_j/w_i) \cup B(\sigma, 16\gamma))] \,/\, \mathbf{Pr}[t_i \notin D_j] \\
&\leq \mathbf{Pr}[s_i \in (U \cap B(\sigma, 4r_j/w_i)) \setminus B(\sigma, 16\gamma)] \,/\, \mathbf{Pr}[t_i \notin D_j] \\
&= \frac{area((U \cap B(\sigma, 4r_j/w_i)) \setminus B(\sigma, 16\gamma))}{area(U \setminus B(\sigma, 4\gamma)) \cdot \mathbf{Pr}[t_i \notin D_j]} \leq \frac{16 \cdot area((U \cap B(\sigma, r_j/w_i)) \setminus B(\sigma, 4\gamma))}{area(U \setminus B(\sigma, 4\gamma)) \cdot \mathbf{Pr}[t_i \notin D_j]} \\
&= \frac{16 \cdot p_{ij}}{\mathbf{Pr}[t_i \notin D_j]} = \frac{16 \cdot p_{ij}}{1 - p_{ij}}.
\end{aligned}$$

If $p_{ij} \leq 1/2$, then the above implies $q_{ij} \leq 16 \cdot p_{ij}/(1 - p_{ij}) \leq 32 \cdot p_{ij}$. On the other hand, if $p_{ij} > 1/2$, then $p_{ij} > 1/2 \geq q_{ij}/2$. ◀

▶ **Fact 15.** *Fix Y. For any $1 \leq j \leq Z - 2$, consider the event, C_j, that j is the largest index such that there are no stretched sites from Y located in D_j. We have $\mathbf{Pr}[C_j] \leq \prod_{s_i \in Y}(1 - p_{i,j})$.*

[1] Note for $j \leq Z - 2$, $D_j \subsetneq U$, thus the condition that no stretched sites are in D_j has non-zero probability.

▶ **Lemma 16.** *Fix Y. Let ψ be the number of sites that fall outside $B(\sigma, 16\gamma)$ and contribute to the multiplicative Voronoi diagram within $B(\sigma, \gamma)$. Then we have*

$$\mathbf{E}\big[\psi^2\big] \leq O(1) + 2\sum_{j=1}^{Z-2}\left(\left(\prod_{s_i\in Y}(1-p_{i,j})\right)\cdot\left(1+3\sum_{s_i\in Y}q_{i,j}+\left(\sum_{s_i\in Y}q_{i,j}\right)^2\right)\right).$$

Proof. For $1 \leq j \leq Z-2$, let C_j be the event that j is the largest index such that D_j contains no stretched sites from Y. (Note for $i \neq j$, C_i and C_j are disjoint events.) If C_j occurs then only sites in $R_{j+1} \cup R_{j+2}$ can contribute to the weighted diagram in $B(\sigma, \gamma)$, based on Lemma 12. Let y_j be the random variable denoting the number of sites such that $s_i \notin B(\sigma, 16\gamma)$ and $t_i \in R_{j+1} \cup R_{j+2}$. If C_j occurs, then y_j is an upper bound on the number of sites which fall outside $B(\sigma, 16\gamma)$ and contribute to the weighted diagram in $B(\sigma, \gamma)$. Note we must also consider the event that C_j does not occur for any $j \geq 1$, namely there exist stretched sites from Y contained in D_1. Call this event C_0, and let y_0 be the random variable for the number of sites such that $s_i \notin B(\sigma, 16\gamma)$ and $t_i \in D_2$. By the law of total expectation,

$$\mathbf{E}\big[\psi^2\big] = \sum_{j=0}^{Z-2}(\mathbf{Pr}[C_j]\cdot\mathbf{E}\big[(y_j)^2 \mid C_j\big]),$$

where the sum stops at $Z-2$ since there always exists a stretched site with weight 1 in D_{Z-1} (by definition of Z), and so stretched sites outside of D_Z can be ignored.

So consider any $\mathbf{E}\big[(y_j)^2 \mid C_j\big]$ term, for some $j > 0$. Let A_j be the event that D_j contains no stretched sites from Y and let B_j be the event that R_{j+1} has a stretched site from Y, and observe that $C_j = A_j \cap B_j$. The following claim is intuitive, and its proof is in Appendix A.

▷ Claim 17. $\mathbf{E}\big[y_j^2 \mid C_j\big] \leq 2\,\mathbf{E}\big[(y_j+1)^2 \mid A_j\big]$.

The events, over all $s_i \in Y$, that s_i falls outside $B(\sigma, 16\gamma)$ while t_i is located in $R_{j+1} \cup R_{j+2}$, are independent. Moreover, this event for any $s_i \in Y$, when conditioned on A_j, has probability $q_{i,j}$ (see Definition 13). Hence conditioned on A_j, the random variable y_j has a Poisson Binomial distribution. Thus we have $\mathbf{E}[y_j \mid A_j] = \sum_{s_i\in Y}(q_{i,j})$, and the variance is then $\mathbf{Var}[y_j \mid A_j] = \sum_{s_i\in Y}(q_{i,j}(1-q_{i,j})) \leq \sum_{s_i\in Y}(q_{i,j}) = \mathbf{E}[y_j \mid A_j]$. Thus using the above claim,

$$\begin{aligned}\mathbf{E}\big[(y_j)^2 \mid C_j\big] &\leq 2\,\mathbf{E}\big[(y_j+1)^2 \mid A_j\big] = 2\,\mathbf{E}\big[1+2y_j+y_j^2 \mid A_j\big]\\ &= 2(1+2\,\mathbf{E}[y_j \mid A_j] + (\mathbf{Var}[y_j \mid A_j] + (\mathbf{E}[y_j \mid A_j])^2))\\ &\leq 2(1+3\,\mathbf{E}[y_j \mid A_j] + (\mathbf{E}[y_j \mid A_j])^2) \leq 2\left(1+3\sum_{s_i\in Y}q_{i,j}+\left(\sum_{s_i\in Y}q_{i,j}\right)^2\right).\end{aligned}$$

Suppose that $\mathbf{Pr}[C_0]\cdot\mathbf{E}\big[(y_0)^2|C_0\big] = O(1)$, then the lemma statement follows. Specifically, by Fact 15, $\mathbf{Pr}[C_j] \leq \prod_{s_i\in Y}(1-p_{i,j})$, for any $1 \leq j \leq Z-2$. Thus by the above,

$$\begin{aligned}\mathbf{E}\big[\psi^2\big] &= \sum_{j=0}^{Z-2}(\mathbf{Pr}[C_j]\cdot\mathbf{E}\big[(y_j)^2 \mid C_j\big])\\ &\leq O(1) + 2\sum_{j=1}^{Z-2}\left(\left(\prod_{s_i\in Y}(1-p_{i,j})\right)\cdot\left(1+3\sum_{s_i\in Y}q_{i,j}+\left(\sum_{s_i\in Y}q_{i,j}\right)^2\right)\right).\end{aligned}$$

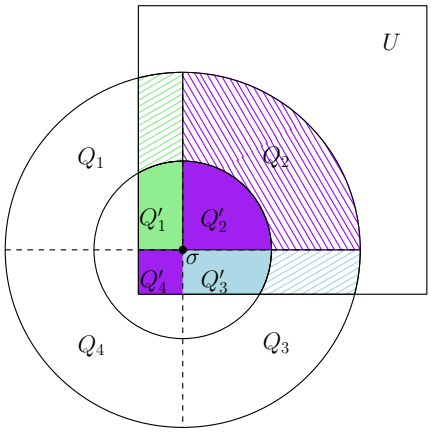


Figure 4.2 The locations of the four defined quarters for each disk, and how they intersect U.

Thus what remains is to show $\mathbf{Pr}[C_0] \cdot \mathbf{E}\left[(y_0)^2 | C_0\right] = O(1)$. First, observe that

$$\mathbf{Pr}[C_0]\,\mathbf{E}\left[(y_0)^2|C_0\right] = \mathbf{Pr}[C_0] \sum_{k=1}^{|Y|^2} k \cdot \mathbf{Pr}\left[(y_0)^2 = k | C_0\right] = \mathbf{Pr}[C_0] \sum_{k=1}^{|Y|^2} \frac{k \cdot \mathbf{Pr}\left[((y_0)^2 = k) \cap C_0\right]}{\mathbf{Pr}[C_0]}$$
$$= \sum_{k=1}^{|Y|^2} k \cdot \mathbf{Pr}\left[((y_0)^2 = k) \cap C_0\right] \leq \sum_{k=1}^{|Y|^2} k \cdot \mathbf{Pr}\left[(y_0)^2 = k\right] = \mathbf{E}\left[(y_0)^2\right].$$

Thus it suffices to argue $\mathbf{E}\left[(y_0)^2\right] = O(1)$.

For any $s_i \in Y$, let $q_{i,0}$ denote the probability that $s_i \notin B(\sigma, 16\gamma)$ while $t_i \in D_2$, then $\mathbf{E}[y_0] = \sum_{s_i \in Y} q_{i,0}$. Note that as $area(B(\sigma, r_2/w_i) \setminus B(\sigma, 16\gamma)) \leq 4 \cdot area(B(\sigma, r_1/w_i) \setminus B(\sigma, 4\gamma))$ it follows that $area((U \cap B(\sigma, r_2/w_i)) \setminus B(\sigma, 16\gamma)) \leq 4 \cdot area((U \cap B(\sigma, r_1/w_i)) \setminus B(\sigma, 4\gamma))$. This implies $q_{i,0} \leq 4 \cdot p_{i,1}$, and hence $\sum_{s_i \in Y} q_{i,0} \leq 4 \sum_{s_i \in Y} p_{i,1} = 4 \cdot n\pi(16\gamma)^2 = O(1)$, by the definition of r_1. Thus $\mathbf{E}[y_0] = O(1)$, and since y_0 has a Poisson Binomial distribution, $\mathbf{E}\left[(y_0)^2\right] = (\mathbf{E}[y_0])^2 + \mathbf{Var}[y_0] \leq (\mathbf{E}[y_0])^2 + \mathbf{E}[y_0] = O(1)$. ◀

▶ **Lemma 18.** *Let $P_j = \sum_{s_i \in Y} p_{i,j}$. Then for any $j \leq Z - 3$, we have $P_j \geq P_{j-1} + 1$.*

Proof. Note that by definition, the expected number of stretched sites from Y that are contained in $B(\sigma, r_1)$ is $n\pi(16\gamma)^2 > 2$, and thus $P_1 = (\sum_{s_i \in Y} p_{i,1}) > 2$.

We argue that for any $j \leq Z - 3$, that $area(U \cap D_j) \geq 2 \cdot area(U \cap D_{j-1})$. To this end, break D_j into 4 quarters, by cutting it with a vertical and horizontal line through σ. Let the quarters be labeled Q_1, Q_2, Q_3, and Q_4, in clockwise order, starting with the northwest quarter, see Figure 4.2. Similarly break D_{j-1} into quarters, Q'_1, Q'_2, Q'_3, and Q'_4, with the same clockwise labeling order. Recall that by definition $Z - 1$ is the smallest j such that $U \subseteq D_j$. Since $\sigma \in U$, this implies that for any $j \leq Z - 3$ at least one of the quarters of D_j is fully contained in U, and without loss of generality assume it is Q_2. Note that $area(D_j) = 4area(D_{j-1})$, thus this implies that $area(U \cap Q_2) = area(Q_2) = 2(area(Q'_2) + area(Q'_4)) \geq 2(area(U \cap Q'_2) + area(U \cap Q'_4))$. It is easy to argue that since $r_j = 2r_{j-1}$ that $area(U \cap Q_1) \geq 2area(U \cap Q'_1)$ and $area(U \cap Q_3) \geq 2area(U \cap Q'_3)$, thus summing over all quarters $area(U \cap D_j) \geq 2 \cdot area(U \cap D_{j-1})$.

This implies that $area(U \cap B(\sigma, r_j/w_i)) \geq 2 \cdot area(U \cap B(\sigma, r_{j-1}/w_i))$, which in turn implies $area((U \cap B(\sigma, r_j/w_i)) \setminus B(\sigma, 4\gamma)) \geq 2 \cdot area((U \cap B(\sigma, r_{j-1}/w_i)) \setminus B(\sigma, 4\gamma))$. Hence by Definition 13, $p_{i,j} \geq 2p_{i,j-1}$, and therefore $P_j \geq 2P_{j-1} \geq P_{j-1} + 1$. ◀

▶ **Lemma 19.** $\sum_{j=1}^{Z-2}\left(\left(\prod_{s_i\in Y}(1-p_{i,j})\right)\cdot\left(1+3\sum_{s_i\in Y}q_{i,j}+\left(\sum_{s_i\in Y}q_{i,j}\right)^2\right)\right)=O(1).$

Proof. We argue that $\sum_{j=1}^{Z-2}\left(\left(\prod_{s_i\in Y}(1-p_{i,j})\right)\cdot\left(\sum_{s_i\in Y}q_{i,j}\right)^{\alpha}\right)=O(1)$, where $\alpha=0$, 1, or 2, thus implying the lemma statement. By Lemma 14, $q_{i,j}\leq 32\cdot p_{i,j}$, and therefore

$$\sum_{j=1}^{Z-2}\left(\left(\prod_{s_i\in Y}(1-p_{i,j})\right)\cdot\left(\sum_{s_i\in Y}q_{i,j}\right)^{\alpha}\right)\leq\sum_{j=1}^{Z-2}\left(\left(\prod_{s_i\in Y}(1-p_{i,j})\right)\cdot\left(32\sum_{s_i\in Y}p_{i,j}\right)^{\alpha}\right)$$
$$\leq 32^{\alpha}\sum_{j=1}^{Z-2}\left(\left(e^{-\sum_{s_i\in Y}p_{i,j}}\right)\cdot\left(\sum_{s_i\in Y}p_{i,j}\right)^{\alpha}\right)=32^{\alpha}\sum_{j=1}^{Z-2}\left(\left(e^{-P_j}\right)\cdot\left(P_j\right)^{\alpha}\right),$$

where the last inequality follows as $1-x\leq e^{(-x)}$, and the last equality is by the definition of P_j from the Lemma 18 statement.

Note that by definition, the expected number of stretched sites from Y that are contained in $B(\sigma,r_1)$ is $n\pi(16\gamma)^2$, and thus $P_1=(\sum_{s_i\in Y}p_{i,1})>2$. Moreover, the function $x^{\alpha}e^{-x}$ is monotonically decreasing for $x>2$, and always has value less than 1, for $\alpha=0,1,$ or 2. Thus since by Lemma 18, $P_j\geq P_{j-1}+1$ for $j\leq Z-3$, and since $P_{Z-2}\geq P_{Z-3}$, we have,

$$\sum_{j=1}^{Z-2}P_j^{\alpha}\cdot e^{-P_j}\leq 1+\sum_{j=1}^{Z-3}P_j^{\alpha}\cdot e^{-P_j}\leq 1+\sum_{x=2}^{\infty}x^{\alpha}\cdot e^{-x}\leq 2+\int_2^{\infty}x^{\alpha}\cdot e^{-x}dx\leq 4,$$

for $\alpha=0,1,$ or 2. Combining the above two equalities thus yields the lemma statement. ◀

Now that we have the above lemmas for any fixed Y, we are finally ready to prove our main lemma (where Y is no longer assumed to be fixed).

▶ **Lemma 20.** *Let S be a set of n point sites in the plane, with arbitrary positive weights. Suppose that the location of each site in S is sampled uniformly at random from the unit square U. Then for any point $\sigma\in U$, $\mathbf{E}[|\mathcal{WV}(S)\cap B(\sigma,\gamma)|]=O(1)$.*

Proof. Let $\hat{\Psi}=S\cap B(\sigma,16\gamma)$ be the sites which fall in $B(\sigma,16\gamma)$ and let $\Psi=S\setminus\hat{\Psi}$ be the complement set. Let $\hat{\psi}$, ψ, be the random variables denoting the number of sites respectively from $\hat{\Psi}$, Ψ, which contribute to the multiplicative diagram in $B(\sigma,\gamma)$.

Recall that the worst-case complexity of the multiplicative diagram is quadratic in the number of sites. Thus it suffices to bound $\mathbf{E}\left[(\hat{\psi}+\psi)^2\right]\leq\mathbf{E}\left[2(\hat{\psi}^2+\psi^2)\right]=2(\mathbf{E}\left[\hat{\psi}^2\right]+\mathbf{E}\left[\psi^2\right])$. Thus we now show each of the above two expected value terms are constant. First, note that Lemma 16 and Lemma 19 combined imply that $\mathbf{E}\left[\psi^2\right]=O(1)$ (as those lemmas hold regardless of which sites fall in Y). Thus we only need to bound $\mathbf{E}\left[\hat{\psi}^2\right]$. Observe that clearly $|\hat{\Psi}|\geq\hat{\psi}$. To bound $|\hat{\Psi}|$, observe that $area(B(\sigma,16\gamma))=O(1/n)$, and thus $\mathbf{E}\left[|\hat{\Psi}|\right]=O(1)$. Moreover, the number of sites which fall into this ball is a binomial random variable. Thus by Fact 8, $\mathbf{E}\left[\hat{\psi}^2\right]\leq\mathbf{E}\left[|\hat{\Psi}|^2\right]\leq\mathbf{E}\left[|\hat{\Psi}|\right]+\mathbf{E}\left[|\hat{\Psi}|\right]^2=O(1)$. ◀

Consider placing a uniform grid with side length $1/\sqrt{n}$ over the unit square U. Then for any grid cell, if we set σ to be the center of the grid cell then $B(\sigma,\gamma)$ contains the grid cell, and hence the above lemma implies the expected complexity in the grid cell is constant. Thus using linearity of expectation over all n grid cells implies the following main theorem.

▶ **Theorem 21.** *Let S be a set of n point sites in the plane, with arbitrary positive weights. Suppose the location of each site in S is sampled uniformly at random from the unit square U. Then the expected complexity of the multiplicative Voronoi diagram of S within U is $O(n)$.*

References

1 Pankaj K. Agarwal, Sariel Har-Peled, Haim Kaplan, and Micha Sharir. Union of random Minkowski sums and network vulnerability analysis. *Discrete & Computational Geometry*, 52(3):551–582, 2014.

2 Boris Aronov, Mark de Berg, and Shripad Thite. The complexity of bisectors and Voronoi diagrams on realistic terrains. In *Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings*, pages 100–111, 2008.

3 Franz Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM J. Comput.*, 16(1):78–96, 1987. doi:10.1137/0216006.

4 Franz Aurenhammer and Herbert Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition*, 17(2):251–257, 1984.

5 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013.

6 Franz Aurenhammer, Bing Su, Yin-Feng Xu, and Binhai Zhu. A note on visibility-constrained Voronoi diagrams. *Discrete Applied Mathematics*, 174:52–56, 2014.

7 Marcin Bienkowski, Valentina Damerow, Friedhelm Meyer auf der Heide, and Christian Sohler. Average case complexity of Voronoi diagrams of n sites from the unit cube. In *(Informal) Proceedings of the 21st European Workshop on Computational Geometry (EuroCG)*, pages 167–170, 2005.

8 Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discret. Comput. Geom.*, 56(4):866–881, 2016. doi:10.1007/s00454-016-9784-4.

9 Hsien-Chih Chang, Sariel Har-Peled, and Benjamin Raichel. From proximity to utility: A Voronoi partition of Pareto optima. *Discrete & Computational Geometry*, 56(3):631–656, 2016.

10 Yongxi Cheng, Bo Li, and Yinfeng Xu. Semi Voronoi diagrams. In *Computational Geometry, Graphs and Applications - 9th International Conference, CGGA 2010, Dalian, China, November 3-6, 2010, Revised Selected Papers*, pages 19–26, 2010.

11 Anne Driemel, Sariel Har-Peled, and Benjamin Raichel. On the expected complexity of Voronoi diagrams on terrains. *ACM Trans. Algorithms*, 12(3):37:1–37:20, 2016.

12 R. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. In *Proc. 5th Annual Symposium on Computational Geometry (SOCG)*, pages 326–333, 1989. doi:10.1145/73833.73869.

13 Chenglin Fan, Jun Luo, Wencheng Wang, and Binhai Zhu. Voronoi diagram with visual restriction. *Theor. Comput. Sci.*, 532:31–39, 2014.

14 Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2:153–174, 1987. doi:10.1007/BF01840357.

15 Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry, Second Edition*. Chapman and Hall/CRC, 2004.

16 Sariel Har-Peled, Haim Kaplan, and Micha Sharir. Approximating the k-level in three-dimensional plane arrangements. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1193–1212, 2016. doi:10.1137/1.9781611974331.ch83.

17 Sariel Har-Peled and Benjamin Raichel. On the complexity of randomly weighted multiplicative Voronoi diagrams. *Discrete & Computational Geometry*, 53(3):547–568, 2015.

18 D. T. Lee and Robert L. (Scot) Drysdale III. Generalization of voronoi diagrams in the plane. *SIAM J. Comput.*, 10(1):73–87, 1981. doi:10.1137/0210006.

19 Micha Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM J. Comput.*, 14(2):448–468, 1985. doi:10.1137/0214034.

A Claim Proof

Here we prove the claim from Lemma 16. Below is the restated claim.

▷ Claim 17. $\mathbf{E}\left[y_j^2 \mid C_j\right] \leq 2\,\mathbf{E}\left[(y_j+1)^2 \mid A_j\right]$.

Proof. Let y_j and $C_j = A_j \cap B_j$ be as defined in the proof of Lemma 16. Throughout j is fixed and so we drop the j subscripts. Thus we must prove $\mathbf{E}\left[y^2|A \cap B\right] \leq 2\,\mathbf{E}\left[(y+1)^2|A\right]$.

As the conditioning on A appears in all terms, for simplicity of exposition we write that we want to show $\mathbf{E}\left[y^2|B\right] \leq 2\,\mathbf{E}\left[(y+1)^2\right]$ where the conditioning on A is implicit.

Note that $y = y' + y''$, where y' is the number of stretched sites falling in R_{j+1} and y'' the number falling in R_{j+2}. Moreover, $(y'+y'')^2 \leq 2((y')^2 + (y'')^2)$.

▶ **Lemma 22.** $\mathbf{E}\left[(y'')^2 \mid y' \neq 0\right] \leq \mathbf{E}\left[(y'')^2\right]$.

Proof. Let $\alpha = \mathbf{E}\left[(y'')^2 \mid y' = 0\right]$ and $\beta = \mathbf{E}\left[(y'')^2 \mid y' \neq 0\right]$. It is easy to verify that $\alpha = \mathbf{E}\left[(y'')^2 \mid y' = 0\right] \geq \mathbf{E}\left[(y'')^2\right]$. Now, observe that

$$\begin{aligned}\mu = \mathbf{E}\left[(y'')^2\right] &= \mathbf{E}\left[(y'')^2 \mid y' = 0\right]\mathbf{Pr}[y' = 0] + \mathbf{E}\left[y''^2 \mid y' \neq 0\right]\mathbf{Pr}[y' \neq 0] \\ &= \alpha\,\mathbf{Pr}[y' = 0] + \beta(1 - \mathbf{Pr}[y' = 0]).\end{aligned}$$

Namely, μ is a convex combination of α and β, and since $\alpha \geq \mu$, it must be that $\beta \leq \mu$, as claimed. ◀

As $B = (y' \neq 0)$, by the above lemma and linearity of expectation we have

$$\mathbf{E}\left[y^2 \mid B\right] \leq 2\,\mathbf{E}\left[(y')^2 \mid y' \neq 0\right] + 2\,\mathbf{E}\left[(y'')^2 \mid y' \neq 0\right] \leq 2\,\mathbf{E}\left[(y')^2 \mid y' \neq 0\right] + 2\,\mathbf{E}\left[(y'')^2\right].$$

Observe that $2\,\mathbf{E}\left[(y'+1)^2\right] + 2\,\mathbf{E}\left[(y'')^2\right] \leq 2\,\mathbf{E}\left[(y+1)^2\right]$. Thus if we can prove that $\mathbf{E}\left[(y')^2 \mid y' \neq 0\right] \leq \mathbf{E}\left[(y'+1)^2\right]$, then the above implies that $\mathbf{E}\left[y^2 \mid B\right] \leq 2\,\mathbf{E}\left[(y+1)^2\right]$ as claimed.

So to prove $\mathbf{E}\left[(y')^2 \mid y' \neq 0\right] \leq \mathbf{E}\left[(y'+1)^2\right]$, note that $(y' \neq 0) = \cup_i X_i$, where X_i is the event that the ith stretched site is in R_{j+1}. Thus,

$$\begin{aligned}E[(y')^2 \mid y' \neq 0] = &E[(y')^2 \mid X_1] \cdot Pr[X_1] + E[(y')^2 \mid \overline{X_1} \cap X_2] \cdot Pr[\overline{X_1} \cap X_2] + \ldots \\ &+ E[y'^2 \mid \overline{X_1} \cap \ldots \cap \overline{X_{n-1}} \cap X_n] \cdot Pr[\overline{X_1} \cap \ldots \cap \overline{X_{n-1}} \cap X_n] \\ &+ E[(y')^2 \mid \overline{X_1} \cap \ldots \cap \overline{X_n}] \cdot Pr[\overline{X_1} \cap \ldots \cap \overline{X_n}].\end{aligned}$$

Note the last term above is zero and can be ignored. Also note that

$$Pr[X_1] + Pr[\overline{X_1} \cap X_2] + \ldots + Pr[\overline{X_1} \cap \ldots \cap \overline{X_{n-1}} \cap X_n] + Pr[\overline{X_1} \cap \ldots \cap \overline{X_n}] = 1.$$

Thus the claim follows if we can argue that each expectation in the above sum is upper bounded by $E[(y'+1)^2]$. So consider any term $E[(y')^2 \mid \overline{X_1} \cap \ldots \cap \overline{X_{k-1}} \cap X_k]$. Let z be the number of sites from $\{p_{k+1} \ldots p_n\}$ falling in R_{j+1}. Then since the points were sampled independently we have

$$\begin{aligned}E[(y')^2 \mid \overline{X_1} \cap \ldots \cap \overline{X_{k-1}} \cap X_k] &\leq E[(z+1)^2 \mid \overline{X_1} \cap \ldots \cap \overline{X_{k-1}} \cap X_k] \\ &\leq E[(z+1)^2] \leq E[(y'+1)^2].\end{aligned}$$ ◁

B Complexity Sketch

Here we give a very brief description of why the order-k sequence Voronoi diagram has $O(nk^3)$ worst-case complexity. First, recall that by standard lifting, the regular order-k diagram is described by the exact kth level in the arrangement of hyperplanes tangent to the unit paraboloid. Since we care about the ordering of the k sites, we are instead concerned with the at most k level. There is a shallow cutting covering the at most k level with $O(n/k)$ vertical prisms each intersecting $O(k)$ planes [16]. Each prism projects to a triangle in the plane, and thus within this triangle only $O(k)$ sites (corresponding to the planes intersecting the prism) are relevant. Note that $O(k)$ sites can define at most $O(k^4)$ different orderings as they define $O(k^2)$ bisectors and the the arrangment of these bisectors has $O(k^4)$ complexity. Thus the plane is covered by $O(n/k)$ triangles within which the order-k sequence Voronoi diagram has $O(k^4)$ complexity, and thus in total the complexity is $O(nk^3)$.

We remark that for the purposes of this paper an $O(nk^5)$ bound would have sufficed, and is trivial to obtain. Namely, the worst case complexity of the regular order-k diagram is $O(nk)$, and by the same argument, within each cell there can be at most $O(k^4)$ orderings.

Polynomial Time Approximation Schemes for Clustering in Low Highway Dimension Graphs

Andreas Emil Feldmann
Charles University, Prague, Czech Republic
feldmann.a.e@gmail.com

David Saulpic
LIP6, Sorbonne Université, Paris, France
david.saulpic@lip6.fr

Abstract

We study clustering problems such as k-Median, k-Means, and Facility Location in graphs of low highway dimension, which is a graph parameter modeling transportation networks. It was previously shown that approximation schemes for these problems exist, which either run in quasi-polynomial time (assuming constant highway dimension) [Feldmann et al. SICOMP 2018] or run in FPT time (parameterized by the number of clusters k, the highway dimension, and the approximation factor) [Becker et al. ESA 2018, Braverman et al. 2020]. In this paper we show that a polynomial-time approximation scheme (PTAS) exists (assuming constant highway dimension). We also show that the considered problems are NP-hard on graphs of highway dimension 1.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Approximation Scheme, Clustering, Highway Dimension

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.46

Related Version A full version of the paper is available at `http://arxiv.org/abs/2006.12897`.

Funding *Andreas Emil Feldmann*: Supported by the Czech Science Foundation GAČR (grant #19-27871X), and by the Center for Foundations of Modern Computer Science (Charles Univ. project UNCE/SCI/004).

Acknowledgements We thank Vincent Cohen-Addad for helpful discussions.

1 Introduction

Clustering is a standard optimization task that seeks a "good" partition of a metric space, such that two points that are "close" should be in the same part. A good clustering of a dataset allows to retrieve and exploit data, and is therefore a common routine in data analysis. The underlying data can come from various sources and represent many different objects. In particular, it is often interesting to cluster geographic data. In that case, the metric space can be given by a transportation network, which can be modeled by graphs with low highway dimension.

In this article, we study some popular clustering objectives, namely Facility Location, k-Median, and k-Means, in graphs with constant highway dimension. The two latter problems seek to find a set S of k points called *centers* in a metric (V, dist) that minimizes $\sum_{v\in V}(\min_{f\in S}\mathrm{dist}(v,f))^p$, with $p=1$ for k-Median and $p=2$ for k-Means. The objective for Facility Location is slightly different: each point f of the metric space has an *opening cost* w_f, and the goal is to find a set S that minimizes $\sum_{f\in S} w_f + \sum_{v\in V}\min_{f\in S}\mathrm{dist}(v,f)$. These problems are APX-hard in general metric spaces [4].

To bypass the hardness of approximation known for these problems, researchers have considered low dimensional input, such as Euclidean spaces of fixed dimension, metrics with bounded doubling dimension, or with bounded genus. Many algorithmic tools were developed

for that purpose: in their seminal work, Arora et al. [3] gave the first polynomial time approximation scheme (PTAS) for k-Median in $\mathbb{R}^2$, which generalizes to a quasi-polynomial time approximation scheme (QPTAS) in $\mathbb{R}^d$ for fixed d. This result was generalized by Talwar [20], who gave a QPTAS for metrics with bounded doubling dimension, and more recently by Cohen-Addad et al. [10], who gave a near-linear time approximation scheme.

In this work we focus on transportation networks, for which it can be argued that metric spaces with bounded doubling dimension are not a suitable model: for instance, hub-and-spoke networks seen in air traffic networks do not have low doubling dimension. Therefore we study graphs with constant *highway dimension*, which formalize structural properties of such networks. The following definition is taken from Feldmann et al. [14]. Here the *ball* $\beta_v(r)$ of radius r around $v \in V$ is the set of all vertices at distance at most r from v.

▶ **Definition 1.** *The* highway dimension *of a graph G is the smallest integer h such that, for some universal constant $c > 4$, for every $r \in \mathbb{R}^+$, and every ball $\beta_v(cr)$ of radius cr, there are at most h vertices in $\beta_v(cr)$ hitting all shortest paths of length more than r that lie in $\beta_v(cr)$.*

For this class of graphs, the only known approximation algorithms for clustering that compute $(1+\varepsilon)$-approximations for any $\varepsilon > 0$ either run in quasi-polynomial time, i.e., QPTASs [14], or with runtime $f(h,k,\varepsilon) \cdot n$ for some exponential function f, i.e., parameterized approximation schemes [6, 8]. Thus an open problem is to identify polynomial-time approximation schemes (PTASs) for clustering in graphs of constant highway dimension.

1.1 Our results

Our main result is a PTAS for clustering problems on graphs of constant highway dimension. For convenience, we define slightly more general problems than those stated above. The k-Clustering^q problem is defined as follows. An instance $\mathcal{I}$ consists of a metric (V, dist), a set of *facilities* (or *centers*) $F \subseteq V$, and a *demand function* $\chi : V \to \mathbb{N}_0$. The goal is to find a set $S \subseteq F$ with $|S| \leq k$ minimizing $\sum_{v \in V} \chi(v) \cdot \min_{f \in S} \mathrm{dist}(v,f)^q$. We call all vertices $v \in V$ with $\chi(v) > 0$ the *clients* of $\mathcal{I}$. k-Median and k-Means are special cases of k-Clustering^q, where $q = 1$ and $q = 2$.

The input to the Facility Location^q problem is the same as for k-Clustering^q, but additionally each facility $f \in F$ has an *opening cost* $w_f \in \mathbb{R}^+$. The goal is to find a set $S \subseteq F$ minimizing $\sum_{f \in S} w_f + \sum_{v \in V} \chi(v) \cdot \min_{f \in S} \mathrm{dist}(v,f)^q$. Facility Location is a special case of Facility Location^q, where $q = 1$.

Our main theorem is the following, where $X = \max_{v \in V} \chi_{\mathcal{I}}(v)$ is the largest demand (note that for k-Median, k-Means, or Facility Location we typically have $X = 1$).

▶ **Theorem 2.** *For any $\varepsilon > 0$, a $(1+\varepsilon)$-approximation for k-Clustering^q and Facility Location^q can be computed in $(nX)^{(hq/\varepsilon)^{O(q)}}$ time on graphs of highway dimension h.*

In particular, this algorithm is much faster than the quasi-polynomial time approximation scheme of Feldmann et al. [14] for k-Median or Facility Location. The runtime of our algorithm also significantly improves over the exponential dependence on k in the approximation schemes of Becker et al. [6], Braverman et al. [8] for k-Median.

It has so far been open whether these clustering problems are NP-hard on graphs of constant highway dimension. We complement our main theorem by showing that they are NP-hard even for the smallest possible highway dimension. This answers an open problem given in [14]. Here the *uniform* Facility Location^q problem has unit opening costs for all facilities.

▶ **Theorem 3.** *The k-Clustering^q and uniform Facility Location^q problems are NP-hard on graphs of highway dimension* 1.

1.2 Related work

On clustering problems. The problems we focus on in this article are known to be APX-hard, even in Euclidean spaces (see e.g. [4]). In general metric spaces, the current best polynomial-time algorithm for Facility Location achieves a 1.488-approximation [19], while the best approximation factor is 2.67 for k-Median ([9]) and 6.357 for k-Means [2].

When restricting the class of graphs, a near-linear time approximation scheme for doubling metrics was developed in [10]; we will discuss the close relations between our work and this one in Section 1.3. Local search techniques also yield a PTAS in minor-free graphs or with bounded doubling dimension [11, 15], and a $\Theta(q)$-approximation for the k-Clustering^q problem in general metric spaces [17].

Another technique for dealing with clustering problems is to compute *coresets*, a compressed representation of the input. An ε-coreset is a weighted set of points such that for every set of centers, the cost for the original set of points is within a $(1+\varepsilon)$-factor of the cost for the coreset. Braverman et al. [8] recently proved that graphs with highway dimension h admit coreset of size $\widetilde{O}((k+h)^{O(1/\varepsilon)})$. This enables to compute a $(1+\varepsilon)$-approximation by enumerating all possible solutions of the coreset. However, this coreset does not have small highway dimension,[1] and thus cannot be used to boost our algorithms.

On highway dimension. The highway dimension was originally defined by Abraham et al. [1], who specifically chose balls of radius $4r$ in the Definition 1. Since the original definition in [1], several other definitions have been proposed. In particular, Feldmann et al. [14] proved that when choosing a radius cr in Definition 1 for any constant c strictly larger than 4, it is possible to exploit the structure of graphs with constant highway dimension in order to obtain a QPTAS for problems such as TSP, Facility Location, and Steiner Tree. As Abraham et al. [1] point out, the choice of the constant is somewhat arbitrary, and we use the above definition so that we may exploit the structural insights of [14] for our algorithm. These structural properties were also leveraged by Becker et al. [6] who gave a PTAS for the Bounded-Capacity Vehicle Routing problem, and a parameterized approximation scheme for the k-Center problem (which is essentially k-Clustering^q with $q=\infty$) and k-Median. In the lower bound side, Disser et al. [12] showed that Steiner Tree and TSP are weakly NP-hard even when the highway dimension is 1, i.e., each of them is NP-hard but an FPTAS exists for graphs of highway dimension 1.

It is worth mentioning that further definitions of the highway dimension exist (for a detailed discussion see [7, 14]). In particular, for a more general definition of the highway dimension than the one of Definition 1, Feldmann [13] gave a parameterized 3/2-approximation algorithm with runtime $2^{O(kh\log h)}n^{O(1)}$ for k-Center.

1.3 Our techniques

To obtain Theorem 2, we rely on the framework recently developed by Cohen-Addad et al. [10] for doubling metrics. More precisely, they show that the *split-tree decomposition* of Talwar [20] has some interesting properties, and exploit them to design their algorithm.

[1] Indeed, a subset of a metric with small highway dimension does not necessarily have small highway dimension as well: think of a star metric on which the center is removed.

Our main contribution is to provide a decomposition with similar properties in graphs with constant highway dimension. This is done relying on some structural properties of such graphs presented by Feldmann et al. [14]. We start by giving the outline of the algorithm from [10], and then explain how to carry the results over to the highway dimension setting.

On doubling metrics. The starting point of many approximation algorithms for doubling metrics is a decomposition of the metric, as presented in the following lemma.[2] A *hierarchical decomposition* $\mathcal{D}$ of a metric (V, dist) is a set of partitions $\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_\lambda$, where $\mathcal{B}_i$ refines $\mathcal{B}_{i+1}$, i.e., every part $B \in \mathcal{B}_i$ is contained in some part of $\mathcal{B}_{i+1}$. Moreover, in $\mathcal{B}_0$ every part contains a singleton vertex, while $\mathcal{B}_\lambda$ contains only one part, namely V. For a point $v \in V$ and a radius $r > 0$, we say that the ball $\beta_v(r)$ is *cut at level* i if i is the largest integer for which the ball $\beta_v(r)$ is not contained in a single part of $\mathcal{B}_i$. For any subset $W \subseteq V$ we define $\lambda(W) = \lceil \log_2 \mathrm{diam}(W) \rceil$.

▶ **Lemma 4** (Reformulation of [20, 5] as found in [10]). *For any metric (V, dist) of doubling dimension d and any $\rho > 0$, there exists a polynomial-time computable randomized hierarchical decomposition $\mathcal{D} = \{\mathcal{B}_0, \ldots, \mathcal{B}_{\lambda(V)}\}$ such that:*

1. ***Scaling probability**: for any $v \in V$, radius r, and level i, we have $\Pr[\mathcal{D} \text{ cuts } \beta_v(r) \text{ at level } i] \leq 2^{O(d)} \cdot r/2^i$.*
2. ***Portal set:** every part $B \in \mathcal{B}_i$ where $\mathcal{B}_i \in \mathcal{D}$ comes with a set of* portals $P_B \subseteq B$ *that is*
   a. ***concise:** the size of the portal set is bounded by $|P_B| \leq 1/\rho^d$, and*
   b. ***precise:** for every node $u \in B$ there is a portal $p \in P_B$ with $\mathrm{dist}(u, p) \leq \rho 2^{i+1}$.*

We sketch briefly the standard use of this decomposition. For clustering problems, one can show that there exists a *portal-respecting solution* with near-optimal cost (see Talwar [20]). In this structured solution, each client connects to a facility via a *portal-respecting path* that enters and leaves any part B of $\mathcal{D}$ only through a node of the portal set P_B. Those portals therefore act as separators of the metric. A standard dynamic program approach can then compute the best portal respecting solution.

To ensure that there is a portal-respecting solution with near-optimal cost, one uses the preciseness property of the portal set: the distortion of connecting a client c with a facility f through portals instead of directly is bounded as follows. Let i be the level at which $\mathcal{D}$ *cuts* c and f, meaning that i is the maximum integer for which c and f lie in different parts of $\mathcal{B}_i$. At every level $j \leq i$, the distortion incurred by using portals is $\rho 2^j$. Hence the total distortion is $\sum_{j \leq i} \rho 2^j = \rho 2^{i+1}$. Now, property (1) of the decomposition ensures that c and f are cut at level i with probability $O(\mathrm{dist}(c, f)/2^i)$. Hence combining those two bounds over all levels ensures that, in expectation, the distortion between c and f is $O(\mathrm{dist}(c, f) \cdot \rho\lambda(V))$. Since $\lambda(V) = O(\log n)$, choosing $\rho = \varepsilon / \log n$ gives a distortion of $O(\varepsilon \,\mathrm{dist}(c, f))$. Summing over all clients proves that there exists a near-optimal portal-respecting solution.

The issue with this approach is that the number of needed portals is $O(\log^d n)$, and the dynamic program has a runtime that is exponential in this number. Thus the time complexity is quasipolynomial. The novelty of [10] is to show how to reduce the number of portals to a constant. The idea is to reduce the number of levels on which a client can be cut from its facility.

[2] We remark that in [10] the preciseness of Lemma 4 was expressed akin to the weaker property found in Lemma 5, which however would not lead to a near-linear time approximation scheme as claimed in [10], but rather a PTAS as shown in this work. This can however easily be alleviated for [10] by using the stronger preciseness as stated here in Lemma 4.

For this, they present a processing step of the instance, that helps deal with clients cut from their facility at a high level. Roughly speaking, their algorithm computes a constant factor approximation L, and a client c is called *badly-cut* if $\mathcal{D}$ cuts it from its closest facility of L at a level larger than $\log(\text{dist}(c, L)/\varepsilon)$. Every badly-cut client is moved to its closest facility of L. Moreover, every client at distance less than $\varepsilon \,\text{dist}(c, L)$ of its closest facility of L can be moved to it as well. It is then shown that this new instance $\mathcal{I}_{\mathcal{D}}$ has *small distortion*, which essentially means that any solution to $\mathcal{I}_{\mathcal{D}}$ can be converted to a solution of the original instance $\mathcal{I}$ while only losing a $(1+\varepsilon)$-factor in quality. In this instance $\mathcal{I}_{\mathcal{D}}$, all clients are cut from their closest facility of L at some level between $\log(\varepsilon \,\text{dist}(c, L))$ and $\log(\text{dist}(c, L)/\varepsilon)$. Using this property, it can be shown that c and its closest center in the optimal solution are also cut at a level in that range. As there are only $O(\log(1/\varepsilon))$ levels in this range, by the previous argument, the number of portals is a constant. (See Section 2 for formal definitions and lemmas.)

On highway dimension. The above arguments for doubling metrics hold thanks to Lemma 4. In this work, we show how to construct a similar decomposition for low highway dimension:

▶ **Lemma 5.** *Given a shortest-path metric* (V, dist) *of a graph with highway dimension* h*, a subset* $W \subseteq V$*, and* $\rho > 0$*, there exists a polynomial-time computable randomized hierarchical decomposition* $\mathcal{D} = \{\mathcal{B}_0, \dots, \mathcal{B}_{\lambda(W)}\}$ *of* W *such that:*

1. ***Scaling probability****: for any* $v \in V$*, radius* r*, and level* i*, we have* $\Pr[\mathcal{D} \text{ cuts } \beta_v(r) \text{ at level } i] \leq \sigma \cdot r/2^i$*, where* $\sigma = (h \log(1/\rho))^{O(1)}$*.*
2. ***Interface****: for any* $B \in \mathcal{B}_i$ *on level* $i \geq 1$ *there exists an interface* $I_B \subseteq V$*, which is*
   a. ***concise****:* $|I_B| \leq (h/\rho)^{O(1)}$*, and*
   b. ***precise****: for any* $u, v \in B$ *such that* u *and* v *are cut by* $\mathcal{D}$ *at level* $i-1$*, there exists* $p \in I_B$ *with* $\text{dist}(u, p) + \text{dist}(p, v) \leq \text{dist}(u, v) + 34 \cdot \rho 2^i$*.*

Our construction relies on the *town decomposition* from [14], which has the following properties: for a graph with highway dimension h and a given $\rho > 0$, every part T of the decomposition (called a *town*) has a set X_T of *hubs* with doubling dimension $O(h \log 1/\rho)$, such that for any two vertices u and v in different child towns of T, there is a vertex $x \in X_T$ such that $\text{dist}(u, x) + \text{dist}(x, v) \leq (1 + 2\rho) \cdot \text{dist}(u, v)$ – see Theorem 8 for more details.

This hub set X_T is similar to the portal set of Lemma 4, but has some fundamental differences: the first one is that the decomposition is deterministic, and so it may happen that a client and its facility are cut at a very high level – something that happens only with tiny probability in the doubling setting thanks to the scaling probability. Another main difference is that the size of X_T might be unbounded. As a consequence, it cannot be directly used as a portal set in a dynamic program. To deal with this, we combine the town decomposition with a hierarchical decomposition of each set X_T according to Lemma 4, to build an *interface* as stated in Lemma 5.

A further notable difference to portals is that the preciseness property of the resulting interface is weaker. In particular, while there is a portal close to each vertex of a part, the hubs can be far from some vertices as long as they lie close to the shortest path to other vertices, which however can be far (due to Lemma 9). As a consequence no analog of near-optimal portal-respecting paths exist. Instead, when connecting a client c with a facility f we need to use the interface point of I_B provided by the preciseness property of Lemma 5 close to the shortest path between c and f, where B contains both c and f. This shifts the perspective from externally connecting vertices of a part to vertices outside a part, as done for portals, to internally connecting vertices of parts, as done here.

As a consequence, we develop a dynamic program, which follows more or less standard techniques as for instance given in [3, 18], but needs to handle the weaker preciseness property of the interface. The main idea is to guess the distances from interface points to facilities while recursing on the decomposition $\mathcal{D}$ of Lemma 5. Due to the shifted perspective towards internally connecting vertices of parts, the runtime of the dynamic program depends exponentially on the *total* number of levels. However, it can be shown that it suffices to compute a solution on a carefully chosen subset W of the metric for which only a logarithmic number of levels of the decomposition need to be considered, and thus the runtime is polynomial.

1.4 Outline

After defining the concepts we use, and stating various structural lemmas in Section 2, we show how to incorporate our decomposition into the framework of [10]. The proof of Lemma 5 is then presented in Section 3. The formal algorithm is deferred to Section 4. We conclude the main body of this paper with the hardness proof of Theorem 3 in Section 5.

2 Preliminaries

On doubling metrics. The *doubling dimension* of a metric is the smallest integer d such that for any $r > 0$ and $v \in V$, the ball $\beta_v(2r)$ of radius $2r$ around v can be covered by at most 2^d balls of half the radius r. A doubling metric is a metric space where the doubling dimension is bounded. In those spaces, one can show the existence of small *nets*:

▶ **Definition 6.** *A* δ-net *of a metric* (V, dist) *is a subset of nodes* $N \subseteq V$ *with the property that every node in* V *is at distance at most* δ *from a net point of* N*, and each pair of net points of* N *are at distance more than* δ*.*

▶ **Lemma 7** ([16])**.** *Let* (V, dist) *be a metric space with doubling dimension* d*. If its diameter is* D*, and* N *is a* δ*-net of* V*, then* $|N| \leq 2^{d \cdot \lceil \log_2(D/\delta) \rceil}$*. Moreover, any subset* $W \subseteq V$ *has doubling dimension at most* $2d$*.*

On highway dimension. We note that for simplicity we will set $c = 8$ in Definition 1 throughout this paper, even if all claimed results are also true for other values of c. When we refer to a metric as having highway dimension h, we mean that it is the shortest-path metric of a graph of highway dimension h. The main result we will use about highway dimension is existence the of the following decomposition:

▶ **Theorem 8** ([14])**.** *Given a shortest-path metric* (V, dist) *of highway dimension* h*, and* $\rho > 0$*, there exists a polynomial-time computable deterministic hierarchical decomposition* $\mathcal{T}$*, called the* town decomposition*, such that every part* $T \in \mathcal{T}$*, called a* town*, has a set of* hubs[3] $X_T \subseteq T$ *with the following properties:*

a. ***doubling****: the doubling dimension of* X_T *is* $d = O(\log(h \log(1/\rho)))$*, and*

b. ***precise****: for any two vertices* u *and* v *in different child parts of* T*, there is a vertex* $x \in X_T$ *such that* $\mathrm{dist}(u, x) + \mathrm{dist}(x, v) \leq (1 + 2\rho) \cdot \mathrm{dist}(u, v)$*.*

The town decomposition behaves differently from those in Lemmas 4 and 5 in several ways. The main properties we will need here are the following.

▶ **Lemma 9** ([14])**.** *For any* $T \in \mathcal{T}$ *we have* $\mathrm{diam}(T) < \mathrm{dist}(T, V \setminus T)$*. Furthermore, for any child town* T' *of* T *we have* $\mathrm{diam}(T') \leq \mathrm{diam}(T)/2$*.*

[3] called *approximate core hubs* in [14].

On how to incorporate our decomposition into the framework of [10]. Assume we are given an instance $\mathcal{I}$ of k-Clustering^q or Facility Location^q on some metric (V, dist), together with a hierarchical decomposition $\mathcal{D}$ of the metric with the properties listed in Lemma 5. We start by defining the *badly cut* clients. In the following, we fix an optimal solution OPT and an approximate solution L, and we define $\tau(\varepsilon, q, \sigma) = \log_2(\sigma(q+1)^q/\varepsilon^{q+1})$.

▶ **Definition 10** (badly cut [10]). *Let (V, dist) be a metric of an instance $\mathcal{I}$ of k-Clustering^q or Facility Location^q, $\mathcal{D}$ be a hierarchical decomposition of the metric with scaling probability factor σ, and $\varepsilon > 0$. If L_v is the distance from v to the closest facility of an approximate solution L to $\mathcal{I}$, then a client c is* badly cut w.r.t. $\mathcal{D}$ *if the ball $\beta_c(3L_c/\varepsilon)$ is cut as some level i greater than $\log_2(3L_c/\varepsilon) + \tau(\varepsilon, q, \sigma)$.*

Similarly, if OPT_v is the distance from v to the closest facility of the optimum solution OPT of $\mathcal{I}$, then a facility $f \in L$ is badly cut w.r.t. $\mathcal{D}$ *if $\beta_f(3OPT_f)$ is cut at some level i greater than $\log_2(3OPT_f) + \tau(\varepsilon, q, \sigma)$.*

Given an instance $\mathcal{I}$ of k-Clustering^q or Facility Location^q and a decomposition $\mathcal{D}$ of the metric, a new instance $\mathcal{I}_\mathcal{D}$ is computed to get rid of badly cut clients. The instance $\mathcal{I}_\mathcal{D}$ is built from $\mathcal{I}$ by moving clients that are badly cut w.r.t. $\mathcal{D}$ to their closest facility in L.[4] For any client c of $\mathcal{I}_\mathcal{D}$ we denote by $\tilde{c}$ the original position of this client in $\mathcal{I}$, i.e., if $\tilde{c}$ is a badly cut client of $\mathcal{I}$ then $c = L(\tilde{c})$ and otherwise $c = \tilde{c}$. The set F of potential centers in unchanged, and thus any solution of $\mathcal{I}$ is a solution of $\mathcal{I}_\mathcal{D}$, and vice versa. Note that $\mathcal{I}_\mathcal{D}$ does not contain any badly cut client w.r.t. $\mathcal{D}$, and that the definition of $\mathcal{I}_\mathcal{D}$ depends on the randomness of $\mathcal{D}$.

To describe the properties we obtain for the new instance, given a solution S to any instance $\mathcal{I}_0$ of k-Clustering^q or Facility Location^q, we define $\text{cost}_{\mathcal{I}_0}(S) = \sum_{v \in V} \chi_{\mathcal{I}_0}(v) \cdot \text{dist}(v, S)^q$ to be the cost incurred by only the distances to the facilities. Given some $\varepsilon > 0$ and the computed instance $\mathcal{I}_\mathcal{D}$ from $\mathcal{I}$, we define

$$\nu_{\mathcal{I}_\mathcal{D}} = \max_{\text{solution } S} \left\{ \text{cost}_\mathcal{I}(S) - (1+2\varepsilon)\,\text{cost}_{\mathcal{I}_\mathcal{D}}(S)\ ,\ (1-2\varepsilon)\,\text{cost}_{\mathcal{I}_\mathcal{D}}(S) - \text{cost}_\mathcal{I}(S) \right\}.$$

If $B_\mathcal{D}$ denotes the set of badly cut facilities (w.r.t $\mathcal{D}$) of the solution L to $\mathcal{I}$ from which instance $\mathcal{I}_\mathcal{D}$ is constructed, we say that $\mathcal{I}_\mathcal{D}$ has *small distortion w.r.t. $\mathcal{I}$* if $\nu_{\mathcal{I}_\mathcal{D}} \le \varepsilon\,\text{cost}_\mathcal{I}(L)$, and there exists a *witness solution* $\hat{S} \subseteq F$ that contains $B_\mathcal{D}$ and for which $\text{cost}_{\mathcal{I}_\mathcal{D}}(\hat{S}) \le (1 + O(\varepsilon))\,\text{cost}_\mathcal{I}(\text{OPT}) + O(\varepsilon)\,\text{cost}_\mathcal{I}(L)$. Moreover, in the case of Facility Location^q, $\hat{S} = \text{OPT} \cup B_\mathcal{D}$ and $\sum_{f \in B_\mathcal{D}} w_f \le \varepsilon \cdot \sum_{f \in L} w_f$.

Based on these definitions, we now state the main tool we use from [10], and which exploits the scaling probability of our decomposition in Lemma 5 to obtain the required structure.

▶ **Lemma 11** ([10]). *Let (V, dist) be a metric, and $\mathcal{D}$ be a randomized hierarchical decomposition of (V, dist) with scaling probability factor σ. Let $\mathcal{I}$ be an instance of k-Clustering^q or Facility Location^q on (V, dist), with optimum solution OPT and approximate solution L. For any (sufficiently small) $\varepsilon > 0$, with probability at least $1 - \varepsilon$ (over $\mathcal{D}$), the instance $\mathcal{I}_\mathcal{D}$ constructed from $\mathcal{I}$ and L as descibed above has small distortion with a witness solution $\hat{S}$. Furthermore, every client c of $\mathcal{I}_\mathcal{D}$ is cut by $\mathcal{D}$ from its closest facility in $\hat{S}$ at level at most $\log_2(3L_{\tilde{c}}/\varepsilon + 4OPT_{\tilde{c}}) + \tau(\varepsilon, q, \sigma)$, where $\tilde{c}$ is the original position of c in $\mathcal{I}$.*

As a consequence of Lemma 11, a dynamic program can compute a solution recursively on the parts of $\mathcal{D}$ in polynomial time, as sketched in Section 1.3 and detailed in Section 4.

[4] More concretely, let $\chi_\mathcal{I}$ and $\chi_{\mathcal{I}_\mathcal{D}}$ be the demand functions of $\mathcal{I}$ and $\mathcal{I}_\mathcal{D}$, respectively. Initially we let $\mathcal{I}_\mathcal{D}$ be a copy of $\mathcal{I}$, so that in particular $\chi_{\mathcal{I}_\mathcal{D}} = \chi_\mathcal{I}$. Then, for each client c of $\mathcal{I}$ that is badly cut in L w.r.t. $\mathcal{D}$, if $L(c)$ denotes the closest facility of L to c, in $\mathcal{I}_\mathcal{D}$ we set $\chi_{\mathcal{I}_\mathcal{D}}(c) = 0$ and increase $\chi_{\mathcal{I}_\mathcal{D}}(L(c))$ by the value of $\chi_\mathcal{I}(c)$ in $\mathcal{I}$.

3 Decomposing the graph

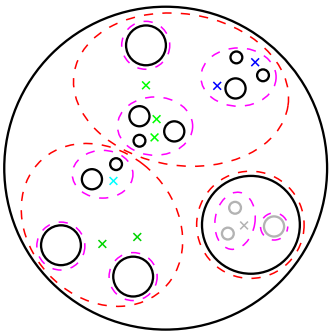


Figure 1 A town T and its child towns (black circles). The hubs (crosses) are decomposed by $\mathcal{X}_T$ (indicated by different colours). Parts $B \in \mathcal{B}_{i+1}$ (red dashed) are decomposed into parts on level i (pink dashed). Parts of $\mathcal{B}_{i-1}$ can lie in different towns (e.g., the child town of T with subtowns in grey).

This section is dedicated to the proof of Lemma 5. The general idea to construct $\mathcal{D}$ is as follows. For doubling metrics, to decompose a part at level i, it is enough to pick a random diameter $\delta \in [2^{i-2}, 2^{i-1})$ and divide the part into child parts of diameter δ. This is not doable in the highway dimension setting: if one wishes to decompose a town T, it cannot divide any of the child towns, since it is not possible to use the approximate core hubs of T to approximate paths inside one of the child towns. The big picture of our decomposition is therefore as follow. To decompose a town at level i, we group randomly (as in the doubling decomposition) the "small" child towns, and put every "big" child town in its own subpart. As we will see, this turns out to be enough.

In order to decompose a town T, we need the following definitions. For each child town T' of T we identify the *connecting hub* $x \in X_T$, which is some fixed closest hub of X_T to T', breaking ties arbitrarily. Moreover, given a hierarchical decomposition $\mathcal{X}_T = \{\mathcal{U}_0, \ldots, \mathcal{U}_{\lambda(X_T)}\}$ of X_T, we define for every i the *connecting i-cluster* of a child town T' of T to be the set $U \in \mathcal{U}_\ell$ on level $\ell = \min\{i, \lambda(X_T)\}$ containing the connecting hub of T'. We then follow the steps below, after choosing μ from the interval $(0, 1]$ uniformly at random (cf. Figure 1):

1. Given a town $T \in \mathcal{T}$, we apply Lemma 4 to find a randomized hierarchical decomposition $\mathcal{X}_T = \{\mathcal{U}_0, \ldots, \mathcal{U}_{\lambda(X_T)}\}$ of the hubs X_T of T.
2. Using $\mathcal{X}_T$, we define a randomized partial decomposition of $T \cap W$ as follows. For any i and $U \in \mathcal{U}_{\min\{i,\lambda(X_T)\}}$, let the set $A_i^U \subseteq T \cap W$ be the union of all $T' \cap W$ where T' is a child town of T with the following two properties:
   a. U is the connecting i-cluster of T', and
   b. $\mathrm{dist}(T', V \setminus T') \leq \mu 2^i$.

   Hence $\mathcal{A}_i^U$ contains all towns somewhat close to U, and with small diameter due to Lemma 9. We let $\mathcal{A}_i^T$ be the set containing each non-empty A_i^U.
3. Now, the hierarchical decomposition $\mathcal{D} = \{\mathcal{B}_0, \ldots, \mathcal{B}_{\lambda(W)}\}$ of W can be constructed inductively as follows. At the highest level $\lambda(W)$ of $\mathcal{D}$, W is partitioned in a single set: $\mathcal{B}_{\lambda(W)} = \{W\}$. Now, to decompose a part $B \in \mathcal{B}_{i+1}$ at level $i+1$, we do the following. Let $T \in \mathcal{T}$ be the inclusion-wise minimal town for which $B \subseteq T$. The "small" subtowns of T lying inside B are grouped according to step (2) (note that $\mathrm{dist}(T', V \setminus T')$ also bounds the diameter of T' by Lemma 9), and the other ones form individual subparts. More formally, the set $\mathcal{B}_i$ contains every part $A \in \mathcal{A}_i^T$ for which $A \subseteq B$, and also every set $T' \cap W$, where T' is a child town of T for which $T' \cap W \subseteq B$ and $T' \cap W$ was not covered by the previously added parts of $\mathcal{A}_i^T$, i.e., $T' \cap W \cap A = \emptyset$ for every $A \in \mathcal{A}_i^T$.

To prove that the constructed decomposition $\mathcal{D}$ has the desired properties –i.e. that it is indeed a hierarchical decomposition, with parts of bounded diameter and small scaling probability factor –, we begin with some auxiliary lemmas, of which the first one bounds the distance of a town to its connecting hub.

▶ **Lemma 12.** *If T' is a child town of T with connecting hub $x \in X_T$, then* $\operatorname{dist}(x, T') \leq (1 + 2\rho) \operatorname{dist}(T', V \setminus T')$.

Proof. Let T'' be the closest sibling town to T', and let $u \in T'$ and $v \in T''$ be the vertices defining the distance from T' to T'', i.e., $\operatorname{dist}(u, v) = \operatorname{dist}(T', T'') = \operatorname{dist}(T', V \setminus T')$. By Theorem 8, there is a hub $y \in X_T$ for which $\operatorname{dist}(u, y) + \operatorname{dist}(y, v) \leq (1 + 2\rho) \cdot \operatorname{dist}(u, v) = (1 + 2\rho) \cdot \operatorname{dist}(T', V \setminus T')$. This implies $\operatorname{dist}(y, T') \leq \operatorname{dist}(u, y) \leq (1 + 2\rho) \cdot \operatorname{dist}(T', V \setminus T')$. Since the connecting hub x of T' is at least as close to T' as y, the claim follows. ◀

Based on the above lemma, we next prove a key property that the diameter of any part of $\mathcal{B}_i \in \mathcal{D}$ is bounded.

▶ **Lemma 13.** *If $\rho \leq 1/2$, then the diameter of any part of $\mathcal{B}_i \in \mathcal{D}$ is less than 2^{i+4}.*

Proof. On the highest level $\lambda(W)$ of $\mathcal{D}$ the only part of $\mathcal{B}_{\lambda(W)}$ is W itself. As $\lambda(W) = \lceil \log_2 \operatorname{diam}(W) \rceil$ we get $\operatorname{diam}(W) \leq 2^{\lambda(W)+1}$, as required.

For any level $i < \lambda(W)$, a set in $\mathcal{B}_i$ is equal to a set $A \in \mathcal{A}_i^T$ for some town $T \in \mathcal{T}$ or it is equal to some set $T' \cap W$ for a child town T' of T. In the former case, the set A is equal to a set A_i^U for some cluster $U \in \mathcal{U}_\ell$ where $\ell = \min\{i, \lambda(X_T)\}$ and $\mathcal{U}_\ell \in \mathcal{X}_T$. The set A_i^U contains the union of sets $T' \cap W$ for child towns T' of T, for which their connecting hubs lie in U and $\operatorname{dist}(T', V \setminus T') \leq \mu 2^i \leq 2^i$, as $\mu \leq 1$. Thus from Lemma 12 we get $\operatorname{dist}(U, T') \leq (1 + 2\rho)2^i$, and by Lemma 9 we have $\operatorname{diam}(T') < \operatorname{dist}(T', V \setminus T') \leq 2^i$. The cluster U has diameter less than 2^{i+1} by Lemma 4, since it is part of the hierarchical decomposition $\mathcal{X}_T$ and lies on level $\ell \leq i$. Let u and v be the vertices of A_i^U defining the diameter of A_i^U, i.e., $\operatorname{dist}(u, v) = \operatorname{diam}(A_i^U)$. We may reach v from u by first crossing the child town T' that u lies in, then passing over to U, then crossing U, after which we pass over to the child town T'' containing v, and finally crossing this child town as well to reach v. Hence, assuming that $\rho \leq 1/2$ the diameter of A_i^U is bounded by

$$\begin{aligned}\operatorname{dist}(u, v) &\leq \operatorname{diam}(T') + \operatorname{dist}(U, T') + \operatorname{diam}(U) + \operatorname{dist}(U, T'') + \operatorname{diam}(T'') \\ &< 2 \cdot 2^i + 2 \cdot (1 + 2\rho)2^i + 2^{i+1} = (6 + 4\rho)2^i \leq 2^{i+3}\end{aligned}$$

Now consider the other case, when a set $B \in \mathcal{B}_i$ on level $i < \lambda(W)$ is equal to some set $T' \cap W$ for a child town T' of a town T. For such a child town T' there is no enforced upper bound on the distance to other child towns as before, and thus it is necessary to be more careful to bound the diameter of the part. Starting with $B = B_i$, let $B_i \subseteq B_{i+1} \subseteq \ldots \subseteq B_j$ be the longest chain of parts of increasing levels that are of the same type as B. More concretely, for every $\ell \in \{i, i+1, \ldots, j\}$ we have $B_\ell \in \mathcal{B}_\ell$ and B_ℓ is equal to some set $T'_\ell \cap W$ for a child town T'_ℓ of the inclusion-wise minimal town T_ℓ containing $B_{\ell+1}$. Note that in particular $j < \lambda(W)$. As we chose the longest such chain, on the next level $j + 1$ there is no such set containing B_j, which means that the set $B_{j+1} \in \mathcal{B}_{j+1}$ for which $B_j \subseteq B_{j+1}$ is either equal to a set $A \in \mathcal{A}_{j+1}^{T_{j+1}}$ for some town T_{j+1}, or $j + 1 = \lambda(W)$. In either case, from above we get $\operatorname{diam}(B_{j+1}) \leq 2^{j+4}$.

Note that for any $\ell \in \{i, i + 1, \ldots, j - 1\}$ the town T'_ℓ is a descendant town of $T'_{\ell+1}$, since $B_{\ell+1}$ is contained in $T'_{\ell+1}$ and T'_ℓ is a child town of the inclusion-wise minimal town T_ℓ containing $B_{\ell+1}$. By Theorem 8 and Lemma 9 we thus get $\operatorname{diam}(T'_\ell) \leq \operatorname{diam}(T'_{\ell+1})/2$,

which implies $\operatorname{diam}(T_i') \leq \operatorname{diam}(T_j')/2^{j-i}$. The set $B = B_i$ is contained in T_i', which means $\operatorname{diam}(B) \leq \operatorname{diam}(T_i')$. The town T_j is the inclusion-wise minimal town containing B_{j+1}, while at the same time the child town T_j' of T_j contains B_j. As $B_j \subseteq B_{j+1}$, this means that B_{j+1} both contains vertices inside and outside of T_j', and so $\operatorname{dist}(T_j', V \setminus T_j') \leq \operatorname{diam}(B_{j+1})$. By Lemma 9 we know that $\operatorname{diam}(T_j') \leq \operatorname{dist}(T_j', V \setminus T_j')$, and putting all these inequalities together we obtain

$$\begin{aligned}\operatorname{diam}(B) \leq \operatorname{diam}(T_i') \leq \operatorname{diam}(T_j')/2^{j-i} &\leq \operatorname{dist}(T_j', V \setminus T_j')/2^{j-i} \\ &\leq \operatorname{diam}(B_{j+1})/2^{j-i} \leq 2^{j+4}/2^{j-i} = 2^{i+4}.\end{aligned}$$ ◀

Using Lemma 13 it is not hard to prove the correctness of $\mathcal{D}$, which we turn to next. All statements marked with "⋆" are deferred to the full version of the paper, due to space constraints.

▶ **Lemma 14 (⋆).** *The tuple $\mathcal{D} = \{\mathcal{B}_0, \ldots, \mathcal{B}_{\lambda(V)}\}$ is a hierarchical decomposition of W.*

We now turn to proving the properties of Lemma 5, starting with the scaling probability.

▶ **Lemma 15.** *The decomposition $\mathcal{D}$ has scaling probability factor $\sigma = (h \log(1/\rho))^{O(1)}$.*

Proof. To prove the claim, we need to prove that for any $v \in W$, radius r, and level i, the probability that $\mathcal{D}$ cuts the ball $\beta_v(r)$ at level i is at most $(h \log(1/\rho))^{O(1)} \cdot r/2^i$. If $\mathcal{D}$ cuts $\beta_v(r)$ at level i, it means that $\beta_v(r)$ is fully contained in a part at level $i+1$: let $T \in \mathcal{T}$ be the inclusion-wise minimal town containing that part. There are two cases to consider: either $\beta_v(r)$ is cut by "small" parts, i.e. there exist two distinct parts $A, A' \in \mathcal{A}_i^T$ such that $v \in A$ and $u \in A'$ for some $u \in W \cap \beta_v(r)$, or not.

We start with the latter case, when $\beta_v(r)$ is not cut by small parts. If $\mathcal{D}$ cuts the ball at level i, there are distinct parts $B, B' \in \mathcal{B}_i$ such that $v \in B$ and $u \in B'$ for some $u \in W \cap \beta_v(r)$. Assume w.l.o.g. that $B \notin \mathcal{A}_i^T$ (which is possible to assume since $\beta_v(r)$ is not cut by small parts). By construction of the decomposition, there must be a child town T' of T, for which $B = T' \cap W$ and $\operatorname{dist}(T', V \setminus T') > \mu 2^i$. Note that $r \geq \operatorname{dist}(v, u) \geq \operatorname{dist}(T', B'') \geq \operatorname{dist}(T', V \setminus T') \geq \mu 2^i$, and hence $\mu \leq r/2^i$. The decomposition $\mathcal{D}$ can therefore only cut $\beta_v(r)$ on level i if $\mu < r/2^i$. Since μ is chosen uniformly at random from the interval $(0, 1]$, the probability is less than $r/2^i$.

We now turn to the other case, when $\beta_v(r)$ is cut by two small parts A_1 and A_2. The town T must have two child towns T_1 and T_2 for which $v \in T_1 \cap W \subseteq A_1$ and $u \in T_2 \cap W \subseteq A_2$. Let x_1 and x_2 be the connecting hubs of T_1 and T_2. The decomposition $\mathcal{D}$ cuts v and u on level i if and only if $\mathcal{X}_T$ cuts x_1 and x_2 on level $\ell = \min\{i, \lambda(X_T)\}$. Indeed, let U_1 and U_2 be the connecting i-clusters of T_1 and T_2: then $A_1 = A^i_{U_1}$ and $A_2 = A^i_{U_2}$, with $x_1 \in U_1, x_2 \in U_2$. Thus $\mathcal{D}$ cuts v and u on level i if and only if $U_1 \neq U_2$, i.e., if and only if $\mathcal{X}_T$ cuts x_1 and x_2 on level $\ell = \min\{i, \lambda(X_T)\}$.

To compute the probability that x_1 and x_2 are cut, it is necessary to bound the distance between them. As $v \in T_1$ and $u \in T_2$ while $u \in \beta_v(r)$, for each $j \in \{1, 2\}$ we have $\operatorname{dist}(T_j, V \setminus T_j) \leq \operatorname{dist}(T_1, T_2) \leq r$. By Lemma 12 the distance between T_j and its connecting hub $x_j \in X_T$ is thus at most $(1 + 2\rho)r$. Also, by Lemma 9 we have $\operatorname{diam}(T_j) < \operatorname{dist}(T_j, V \setminus T_j) \leq r$, and we get

$$\operatorname{dist}(x_1, x_2) \leq \operatorname{dist}(x_1, T_1) + \operatorname{diam}(T_1) + \operatorname{dist}(T_1, T_2) + \operatorname{diam}(T_2) + \operatorname{dist}(T_2, x_2) \leq (5 + 4\rho)r.$$

We can reformulate the above as follows: if $\mathcal{D}$ cuts the ball $\beta_v(r)$ at level i, and $\beta_v(r)$ is cut by some "small" parts A_1 and A_2, then $\mathcal{X}_T$ cuts the ball $\beta_{x_1}((5 + 4\rho)r)$ on level i, where x_1 is the hub defined for v above. We know that the probability of the latter event

is at most $2^{O(d)}(5+4\rho)r/2^i$ by Lemma 4, where $d = O(\log(h\log(1/\rho)))$ is the doubling dimension of X_T by Theorem 8. Hence the probability that $\mathcal{D}$ cuts the ball $\beta_v(r)$ at level i is bounded by $(h\log(1/\rho))^{O(1)} \cdot r/2^i$. Taking a union bound over the two considered cases proves the claim. ◀

To prove the remaining property of Lemma 5 for $\mathcal{D}$, for each $B \in \mathcal{B}_i$ we need to choose an interface I_B from the whole vertex set V. For this we use a carefully chosen net (see Definition 6) of the hubs of the inclusion-wise minimal town T containing B, as formalized in the following lemma.

▶ **Lemma 16.** *Given $B \in \mathcal{B}_i$ for some $\mathcal{B}_i \in \mathcal{D}$ and $i \geq 1$, let $T \in \mathcal{T}$ be the inclusion-wise minimal town containing B. We define the interface I_B to be a $\rho 2^i$-net of the set $Y_B = \{x \in X_T \mid \operatorname{dist}(x,B) \leq (1+2\rho)\operatorname{diam}(B)\}$. The interface I_B has the conciseness and preciseness properties of Lemma 5 for $\rho \leq 1/2$.*

Proof. We first prove that I_B is precise. Consider two vertices $u, v \in B$ that are cut at level $i-1$ by $\mathcal{D}$. This means there are two distinct parts $B', B'' \in \mathcal{B}_{i-1}$ on this level such that $v \in B'$ and $u \in B''$. By definition, both B' and B'' are unions of sets $T' \cap W$ where T' is a child town of the inclusion-wise minimal town T containing B. Also $B' \cap B'' = \emptyset$ by Lemma 14. This means that T has two child towns T_1 and T_2 for which $v \in T_1 \cap W \subseteq B'$ and $u \in T_2 \cap W \subseteq B''$. By Theorem 8, there is an approximate core hub $x \in X_T$ such that $\operatorname{dist}(u,x) + \operatorname{dist}(x,v) \leq (1+2\rho)\operatorname{dist}(u,v)$. In particular, $\operatorname{dist}(x,B) \leq \operatorname{dist}(u,x) \leq (1+2\rho)\operatorname{dist}(u,v) \leq (1+2\rho)\operatorname{diam}(B)$, as $u, v \in B$. This means that $x \in Y_B$. Since I_B is a $\rho 2^i$-net of Y_B, there is a hub $p \in I_B$ for which $\operatorname{dist}(x,p) \leq \rho 2^i$. By Lemma 13 we have $\operatorname{dist}(u,v) \leq \operatorname{diam}(B) \leq 2^{i+4}$ if $\rho \leq 1/2$, and so I_B is precise:

$$\begin{aligned}\operatorname{dist}(u,p) + \operatorname{dist}(p,v) &\leq \operatorname{dist}(u,x) + 2\cdot\operatorname{dist}(x,p) + \operatorname{dist}(x,v)\\ &\leq (1+2\rho)\operatorname{dist}(u,v) + \rho 2^{i+1} \leq \operatorname{dist}(u,v) + 2\rho\cdot 2^{i+4} + \rho 2^{i+1} \leq \operatorname{dist}(u,v) + 34\cdot\rho 2^i,\end{aligned}$$

To prove conciseness, recall that $\operatorname{diam}(B) \leq 2^{i+4}$ by Lemma 13, which means that $\operatorname{diam}(Y_B) \leq \operatorname{diam}(B) + 2(1+2\rho)\operatorname{diam}(B) \leq 5\cdot 2^{i+4}$ for $\rho \leq 1/2$. Since I_B is a $\rho 2^i$-net of Y_B, Lemma 7 implies $|I_B| \leq 2^{d\cdot\lceil\log_2(80/\rho)\rceil}$, where d is the doubling dimension of Y_B. Theorem 8 says that X_T has doubling dimension $O(\log(h\log(1/\rho)))$, and as $Y_B \subseteq X_T$ the same asymptotic bound holds for the doubling dimension d of Y_B by Lemma 7. Therefore we get $|I_B| \leq (h\log(1/\rho))^{O(\log(1/\rho))} = (h/\rho)^{O(1)}$, which concludes the proof. ◀

4 The algorithm

Let an instance $\mathcal{I}$ of the k-Clustering^q or Facility Location^q problem on a shortest-path metric (V, dist) of a graph G with highway dimension h, and maximum demand $X = \max_{v\in V}\chi_{\mathcal{I}}(v)$ be given. The algorithm performs the following steps:

1. compute a town decomposition $\mathcal{T}$ together with the hubs for each town as given by Theorem 8.
2. compute a hierarchical decomposition $\mathcal{D}$ according to Lemma 5, while simultaneously converting $\mathcal{I}$ into a *coarse* instance w.r.t. $\mathcal{D}$, meaning that there is a subset $W \subseteq V$ for which
   - the clients and facilities of $\mathcal{I}$ are contained in W, i.e., $F \cup \{v \in V \mid \chi_{\mathcal{I}}(v) > 0\} \subseteq W$, and
   - every part of $\mathcal{D}$ on level at most $\xi(W) = \lfloor \lambda(W) - 2\log_2(nX/\varepsilon)\rfloor$ has at most one facility, i.e., $|B \cap F| \leq 1$ for every $B \in \mathcal{B}_{\xi(W)}$.

3. compute the instance $\mathcal{I}_{\mathcal{D}}$ of small distortion as given by Lemma 11.
4. run a dynamic program on $\mathcal{I}_{\mathcal{D}}$ as given in Section 4.2, to compute an *optimum rounded interface-respecting solution* (see Section 4.1 for a formal definition), and output it as a solution to the input instance.

In a nutshell, the coarseness of the instances guarantees that only a logarithmic number of levels need to be considered by the dynamic program. This step loses a $(1+\varepsilon)$-factor in the solution quality. The dynamic program is only able to compute highly structured solutions, which are captured by the notion of *rounding* and *interface-respecting*. Due to this, another $(1+\varepsilon)$-factor in the solution quality is lost. In Section 4.1 we prove that the output of the dynamic program is a near-optimal solution to the input instance (proving Theorem 2), and we also detail step (4) of the algorithm. Then in Section 4.2 we describe the details of the dynamic program.

4.1 Approximating the distances

One caveat of the dynamic program is that the runtime is only polynomial if the the recursion depth is logarithmic. However when computing our decomposition on the whole metric (V, dist), the number of levels is $\lambda(V)+1 = \lceil \log_2 \mathrm{diam}(V) \rceil + 1$, which can be linear in the input size. Standard techniques can be used to reduce the number of levels to $O(\log(n/\varepsilon))$ when aiming for a $(1+\varepsilon)$-approximation by preprocessing the input metric. However, for graphs of bounded highway dimension these general techniques change the hub sets and we would have to be careful to maintain the properties we need, as given by Theorem 8. Therefore we adapt the standard techniques to our setting via the notion of *coarse* instances.

The following lemma shows that we can reduce any instance to a set of coarse ones, for which, as we will see, our dynamic program only needs to consider the highest $2\log_2(nX/\varepsilon)$ levels.

▶ **Lemma 17 (⋆).** *Let $\mathcal{I}$ be an instance of k-Clustering^q or Facility Location^q on a graph G of highway dimension h. There are polynomial-time computable instances $\mathcal{I}_1, \ldots, \mathcal{I}_b$ and respective hierarchical decompositions $\mathcal{D}_1, \ldots, \mathcal{D}_b$ with the properties given in Lemma 5 for any $\rho \leq 1/2$, such that for each $i \in \{1, \ldots, b\}$ the instance $\mathcal{I}_i$ is also defined on G and is coarse w.r.t. $\mathcal{D}_i$. Furthermore, if an α-approximation can be computed for each of the instances $\mathcal{I}_1, \ldots, \mathcal{I}_b$ in polynomial time, then for any $\varepsilon > 0$ a $(1+O(\varepsilon))\alpha$-approximation can be computed for $\mathcal{I}$ in polynomial time.*

Lemma 17 implies that if there is a PTAS for coarse instances, we also have a PTAS in general. Hence from now on we assume that the given instance $\mathcal{I}$ is coarse w.r.t. a hierarchical decomposition $\mathcal{D}$ of some subset W of the vertices of the input graph G, where $\mathcal{D}$ has bounded scaling probability factor and concise and precise interface sets in G according to Lemma 5 (for some value $\rho > 0$ specified later)

The next step of the algorithm is to compute a new instance $\mathcal{I}_{\mathcal{D}}$ with small distortion as given by Lemma 11. Recall that $\mathcal{I}_{\mathcal{D}}$ is obtained from $\mathcal{I}$ by moving badly cut clients to facilities of L. In particular, the instance $\mathcal{I}_{\mathcal{D}}$ is also coarse w.r.t. $\mathcal{D}$, which means that we may run our dynamic program on $\mathcal{I}_{\mathcal{D}}$.

The dynamic program exploits the interface sets of $\mathcal{D}$ by computing a near-optimum "interface-respecting" solution to $\mathcal{I}_{\mathcal{D}}$, i.e., a solution where clients are connected to facilities through interface points. Moreover, for the dynamic program to run in polynomial time it can only estimate the distances between interface points and facilities to a certain precision. In general, we denote by $\langle x \rangle_i = \min\{(35+\delta)\rho 2^i \mid \delta \in \mathbb{N} \text{ and } \rho\delta 2^i \geq x\}$ the value of x rounded to

the next multiple of $\rho 2^i$ and shifted by $35\rho 2^i$. We then define the *rounded interface-respecting distance* $\text{dist}'(v,u)$ from a vertex v to another vertex u as follows. If v and u are not cut at any level, i.e., $v = u$, then $\text{dist}'(v,u) = 0$. Otherwise, if $i \geq 1$ is the level of $\mathcal{D}$ such that there is a part $B \in \mathcal{B}_i$ with $v, u \in B$, and $\mathcal{D}$ cuts v and u at level $i-1$, we let

$$\text{dist}'(v,u) = \min\left\{ \text{dist}(v,p) + \langle \text{dist}(p,u) \rangle_i \mid p \in I_B \right\}.$$

Note that $\text{dist}'(\cdot,\cdot)$ does not necessarily fulfill the triangle inequality, and is also not symmetric. We therefore need the bounds of the following lemma.

▶ **Lemma 18.** *For any level $i \geq 1$ and vertices v and u that are cut by $\mathcal{D}$ on level $i-1$ we have $\text{dist}'(v,u) \leq \text{dist}(v,u) + 70 \cdot \rho 2^i$. Let $B \in \mathcal{B}_j$ be the part on some level $j \geq i$ with $v, u \in B$. For any $p \in I_B$ we have $\text{dist}'(v,u) \leq \text{dist}(v,u) + \langle \text{dist}(p,u) \rangle_j$.*

Proof. Let $B' \in \mathcal{B}_i$ be the part on level i containing both v and u. By Lemma 5 there is an interface point $p' \in I_{B'}$ such that $\text{dist}(v,p') + \text{dist}(p',u) \leq \text{dist}(v,u) + 34 \cdot \rho 2^i$. By definition of the rounding we also have $\langle \text{dist}(p',u) \rangle_i \leq \text{dist}(p',u) + 36 \cdot \rho 2^i$. Hence $\text{dist}'(v,u) \leq \text{dist}(v,p') + \langle \text{dist}(p',u) \rangle_i \leq \text{dist}(v,p') + \text{dist}(p',u) + 36 \cdot \rho 2^i \leq \text{dist}(v,u) + 70 \cdot \rho 2^i$.

The second part is obvious if $j = i$ from the definition of $\text{dist}'(v,u)$. If $j \geq i+1$, we use the above bound on $\text{dist}'(v,u)$ together with the additive shift of the rounding and the triangle inequality of $\text{dist}(\cdot,\cdot)$ to obtain

$$\begin{aligned}\text{dist}'(v,u) &\leq \text{dist}(v,u) + 70 \cdot \rho 2^i \leq \text{dist}(v,p) + \text{dist}(p,u) + 70 \cdot \rho 2^{j-1} \\ &\leq \text{dist}(v,p) + \langle \text{dist}(p,u) \rangle_j - 35 \cdot \rho 2^j + 70 \cdot \rho 2^{j-1} = \text{dist}(v,p) + \langle \text{dist}(p,u) \rangle_j. \text{◀}\end{aligned}$$

For any non-empty set S of facilities, we define $\text{dist}'(v,S) = \min_{f \in S}\{\text{dist}'(v,S)\}$, and for empty sets we let $\text{dist}'(v,\emptyset) = \infty$. Analogous to $\text{cost}_{\mathcal{I}_0}(S)$, for a solution S to some instance $\mathcal{I}_0$ we also define $\text{cost}'_{\mathcal{I}_0}(S)$ using $\text{dist}'(\cdot,\cdot)$ as

$$\text{cost}'_{\mathcal{I}_0}(S) = \sum_{v \in V} \chi_{\mathcal{I}_\mathcal{D}}(v) \cdot \text{dist}'(v,S)^q.$$

We show the following lemma, which translates between $\text{cost}'_{\mathcal{I}_\mathcal{D}}$ and $\text{cost}_\mathcal{I}$, and is implied by the preciseness of the interface sets and the fact that $\mathcal{I}_\mathcal{D}$ has small distortion. Recall that the set of facilities is the same in $\mathcal{I}$ and $\mathcal{I}_\mathcal{D}$, i.e., a solution to one of these instances is also a solution to the other.

▶ **Lemma 19 (⋆).** *Let $\mathcal{I}$ be an instance of k-Clustering^q or Facility Location^q with optimum solution OPT and approximate solution L. Let $\mathcal{I}_\mathcal{D}$ be an instance of small distortion for some $0 < \varepsilon < 1/2$, computed from L and a hierarchical decomposition $\mathcal{D}$ with precise interface sets for $\rho \leq \frac{\varepsilon^{q+4+1/q}}{280\sigma(q+1)^q}$ according to Lemma 5. For the witness solution $\hat{S}$ of $\mathcal{I}_\mathcal{D}$ we have $\text{cost}'_{\mathcal{I}_\mathcal{D}}(\hat{S}) \leq (1+O(\varepsilon))\,\text{cost}_\mathcal{I}(OPT) + O(\varepsilon)\,\text{cost}_\mathcal{I}(L)$. Moreover, for any solution S we have $\text{cost}_\mathcal{I}(S) \leq (1+O(\varepsilon))\,\text{cost}'_{\mathcal{I}_\mathcal{D}}(S) + O(\varepsilon)\,\text{cost}_\mathcal{I}(L)$.*

The next lemma states the properties of the dynamic program that for any coarse instance $\mathcal{I}_0$ computes an *optimal rounded interface-respecting solution*, which formally is a subset OPT' of facilities that minimizes $\text{cost}'_{\mathcal{I}_0}(\text{OPT}')$ with $|\text{OPT}'| \leq k$ for k-Clustering^q, while for Facility Location^q it minimizes $\text{cost}'_{\mathcal{I}_0}(\text{OPT}') + \sum_{f \in \text{OPT}'} w_f$. This step of the algorithm exploits the conciseness of the interface sets and the coarseness of the instance to bound the runtime. We prove the following lemma in Section 4.2.

▶ **Lemma 20.** *Let $\mathcal{I}_0$ be an instance of k-Clustering^q or Facility Location^q that for some $\varepsilon > 0$ is coarse w.r.t. a hierarchical decomposition $\mathcal{D}$ with concise interface sets for some $1/2 \geq \rho > 0$ according to Lemma 5. An optimum rounded interface-respecting solution for $\mathcal{I}_0$ can be computed in $(nX/\varepsilon)^{(h/\rho)^{O(1)}}$ time.*

We are now ready to put together the above lemmas to prove Theorem 2. Due to space constraints however, the formal proof is deferred to the full version of the paper.

4.2 The dynamic program (proof of Lemma 20)

We describe the algorithm for k-Clustering^q, and only mention in the end how to modify the algorithm to compute a solution for Facility Location^q.

The solution is computed by a dynamic program recursing on the decomposition $\mathcal{D}$. Let W be the vertex set that $\mathcal{D}$ decomposes, and which contains all clients and facilities of the coarse instance $\mathcal{I}$. Roughly speaking, the table of the dynamic program will have an entry for every part $B \in \mathcal{B}_i$ of $\mathcal{D}$ on all levels $i \geq \xi(W)$, for which it will estimate the distance from each interface point on all higher levels $j \geq i+1$ to the closest facility of the optimum solution. That is, if $\tilde{B} \in \mathcal{B}_j$ is a higher-level part for which $B \subseteq \tilde{B}$, then the distances from all interface points $I_{\tilde{B}}$ to facilities of the solution in $\tilde{B}$ will be estimated.

Here the estimation happens in two ways. First off, the distances to facilities outside of B have to be guessed. That is, there is an *external* distance function d_j^+ that assigns a distance to each interface point of $I_{\tilde{B}}$, anticipating the distance from such a point to the closest facility of $\tilde{B}$, if this facility lies outside of B. In order to verify whether the guess was correct, each entry for a part B on level i also provides an *internal* distance function d_j^-, which stores the distance from each interface point of $I_{\tilde{B}}$ on level $j \geq i+1$ to the closest facility, if the facility is guessed to lie inside of B.

The other way in which distances are estimated concerns the preciseness with which they are stored. The distance functions d_j^+ and d_j^- will only take rounded values $\langle x \rangle_j$ where $0 < x \leq 2^{j+5}$, or ∞ if no facility at the appropriate distance exists. In particular, if the facility of the solution in $\tilde{B}$ that is closest to $p \in I_{\tilde{B}}$ lies outside of B then $d_j^-(p) = \infty$, and if it lies inside of B then $d_j^+(p) = \infty$. If there is no facility of the solution in $\tilde{B}$ then both distance functions d_j^+ and d_j^- are set to ∞ for all $p \in I_{\tilde{B}}$. Note that this means that at least one of $d_j^+(p)$ and $d_j^-(p)$ is always set to ∞. Note also that the finite values in the domains of the distance functions admit to store the rounded distance to any facility in $\tilde{B}$ on level j, since the diameter of $\tilde{B}$ is at most 2^{j+4} by Lemma 13, and the distance from any $p \in I_{\tilde{B}}$ to $\tilde{B}$ is at most $(1+2\rho)\operatorname{diam}(\tilde{B})$ by Lemma 16, i.e., for any $f \in \tilde{B} \cap F$ we have $\operatorname{dist}(p,f) \leq (1+2\rho)2^{j+4} \leq 2^{j+5}$ using $\rho \leq 1/2$.

Formal definition of the table. Let us denote by I_B^j the interface set of the part $\tilde{B} \in \mathcal{B}_j$ on level $j \geq i+1$ containing $B \in \mathcal{B}_i$, i.e., $I_B^j = I_{\tilde{B}}$. Every entry of the dynamic programming table T is defined by a part $B \in \mathcal{B}_i$ of $\mathcal{D}$ on a level $i \in \{\xi(W), \ldots, \lambda(W)\}$, and two distance functions $d_j^+, d_j^- : I_B^j \to \{\langle x \rangle_j \mid 0 < x \leq 2^{j+5}\} \cup \{\infty\}$ for each $j \in \{i+1, \ldots, \lambda(W)\}$, such that $\max\{d_j^+(p), d_j^-(p)\} = \infty$ for all $p \in I_B^j$. Additionally, each entry comes with an integer $k' \in \{0, \ldots, k\}$, which is a guess on the number of facilities that the optimum solution contains in B.

In an entry $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ we store the rounded interface-respecting cost of connecting the clients of B to facilities that adhere to the distance functions. More concretely, let $S \subseteq F \cap B$ be any subset of facilities in B. We say that S is *compatible* with an entry $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ if $|S| = k'$, and for any $j \geq i+1$ the values of the distance functions for every interface point $p \in I_B^j$ are set to either

- $d_j^-(p) = \langle \operatorname{dist}(p,S) \rangle_j$ and $d_j^+(p) = \infty$, or
- $d_j^+(p) \leq \langle \operatorname{dist}(p,S) \rangle_j$ and $d_j^-(p) = \infty$.

Recall that $\text{dist}(v, \emptyset) = \infty$, and so the empty set $S = \emptyset$ is compatible with an entry $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ if $k' = 0$, and the values of all internal distance functions are set to ∞. Over all sets $S \subseteq F \cap B$ compatible with the entry $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ for $B \in \mathcal{B}_i$, the entry should store the minimum value of $C_B(S)$, which is defined as

$$C_B(S) = \sum_{v \in B} \chi_{\mathcal{I}_0}(v) \cdot \min \Big\{ \text{dist}'(v, S), \min_{\substack{j \geq i+1 \\ p \in I_B^j}} \big\{ \text{dist}(v, p) + d_j^+(p) \big\} \Big\}.$$

If there is no compatible set $S \subseteq F \cap B$ for the entry, then $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}] = \infty$.

On the highest level $i = \lambda(W)$, there are no distance functions to adhere to on levels $j \geq i + 1$, and thus any set $S \subseteq W$ of facilities is compatible with the entry for $B = W$ and $k' = |S|$. Furthermore, $\text{cost}'_{\mathcal{I}_0}(S)$ is equal to $C_W(S)$, since W contains all clients and facilities of the coarse instance $\mathcal{I}_0$. In particular, the entry of T for which $k' = k$ and $B = W$, will contain the objective function value of the optimum rounded interface-respecting solution to $\mathcal{I}_0$. Hence if we can compute the table T we can also output the optimum rounded interface-respecting solution via this entry.

Computing the table. We begin with a part $B \in \mathcal{B}_{\xi(W)}$ on the lowest considered level $\xi(W)$, for which we know that B contains at most one facility, as $\mathcal{I}_0$ is coarse. If B contains no facility, then only $S = \emptyset$ can be compatible with the entry $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ and computing the value of the entry is straightforward given the definition of $C_B(S)$, where all incompatible entries are set to ∞. If B contains one facility f, then any compatible set S is either empty or only contains f. We can thus check whether either of the two options is compatible with the entry $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ by checking if k' is set to 0 or 1, respectively, and checking that all values of the internal distance function are set correctly. Thereafter we can again use the definition of $C_B(S)$ to compute the values for both possible sets S and store them in the respective compatible entries. All incompatible entries are set to ∞.

Now fix a part $B \in \mathcal{B}_i$ that lies on a level $i > \xi(W)$. We show how to compute all entries $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ for all values k' and distance functions. By induction we have already computed the correct values of all entries of T for parts $B' \in \mathcal{B}_{i-1}$ where $B' \subseteq B$. We order these parts arbitrarily, so that $B'_1, \ldots, B'_b$ are the parts of $\mathcal{B}_{i-1}$ contained in B. We then define an auxiliary table $\hat{T}$ that is similar to the table T, but should compute the best compatible facility set in the union $B'_{\leq \ell} = \bigcup_{h=1}^{\ell} B'_h$ of the first ℓ subparts of B. Accordingly, $\hat{T}$ has an entry for each union of parts $B'_{\leq \ell}$, each $k' \in \{0, \ldots, k\}$, and distance functions $d_j^+, d_j^- : I_B^j \to \{\langle x \rangle_j \mid 0 < x \leq 2^{j+5}\} \cup \{\infty\}$ for each $j \in \{i, \ldots, \lambda(W)\}$, such that $\max\{d_j^+(p), d_j^-(p)\} = \infty$ for all $p \in I_B^j$. Here, naturally, $I_B^i = I_B$, i.e., the entry also takes the interface set of B into account.

Analogous to before, a set $S \subseteq F \cap B'_{\leq \ell}$ of facilities in the union is *compatible* with an entry $\hat{T}[B'_{\leq \ell}, k', (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$ if $|S| = k'$, and for any $j \geq i$ the values of the distance functions for every interface point $p \in I_B^j$ are set to either

- $d_j^-(p) = \langle \text{dist}(p, S) \rangle_j$ and $d_j^+(p) = \infty$, or
- $d_j^+(p) \leq \langle \text{dist}(p, S) \rangle_j$ and $d_j^-(p) = \infty$.

Over all sets $S \subseteq F \cap B'_{\leq \ell}$ compatible with $\hat{T}[B'_{\leq \ell}, k', (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$, the entry should store the minimum value of $\hat{C}_{\leq \ell}(S)$, which is defined as

$$\hat{C}_{\leq \ell}(S) = \sum_{v \in B'_{\leq \ell}} \chi_{\mathcal{I}_0}(v) \cdot \min \Big\{ \text{dist}'(v, S), \min_{\substack{j \geq i \\ p \in I_B^j}} \big\{ \text{dist}(v, p) + d_j^+(p) \big\} \Big\}.$$

If there is no compatible set $S \subseteq F \cap B'_{\leq \ell}$ for the entry, then $\hat{T}[B'_{\leq \ell}, k', (d_j^+, d_j^-)_{j=i}^{\lambda(W)}] = \infty$.

To compute T using the auxiliary table $\hat{T}$, note that since $B = B'_{\leq b}$, any set $S \subseteq F \cap B$ is compatible with the entry $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ if and only if it is compatible with a corresponding entry $\hat{T}[B'_{\leq b}, k', (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$ for some internal distance function d_i^- on level i. Furthermore, if $d_i^+(p) = \infty$ for all $p \in I^i$, then $C_B(S) = \hat{C}_{\leq b}(S)$ for such a set S. Therefore we can easily compute the entry $T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ from $\hat{T}$ by setting

$$T[B, k', (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}] = \min_{d_i^-} \Big\{ \hat{T}[B'_{\leq b}, k', (d_j^+, d_j^-)_{j=i}^{\lambda(W)}] \mid \forall p \in I_B^i : d_i^+(p) = \infty \Big\}.$$

Computing the auxiliary table. Also computing an entry of $\hat{T}$ for $B'_{\leq 1}$ is easy using the entries of T for B'_1, since $B'_1 = B'_{\leq 1}$ and so (taking the index shift of i into account) we have

$$\hat{T}[B'_{\leq 1}, k', (d_j^+, d_j^-)_{j=i}^{\lambda(W)}] = T[B'_1, k', (d_j^+, d_j^-)_{j=i}^{\lambda(W)}].$$

To compute entries of $\hat{T}$ for some $B'_{\leq \ell}$ where $\ell \geq 2$, we combine entries of table T for B'_ℓ with entries of table $\hat{T}$ for $B'_{\leq \ell-1}$. However we will only combine entries with distance functions that imply compatible solutions. More concretely, we say that distance functions $(d_j^+, d_j^-)_{j=i}^{\lambda(W)}$ for $B'_{\leq \ell}$, $(\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}$ for B'_ℓ, and $(\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}$ for $B'_{\leq \ell-1}$ are *consistent* if for every level $j \geq i$ and $p \in I_B^j$ we have one of

1. $d_j^+(p) = \delta_j^+(p) = \beta_j^+(p)$ and $d_j^-(p) = \delta_j^-(p) = \beta_j^-(p) = \infty$, or
2. $d_j^-(p) = \delta_j^-(p) = \beta_j^+(p)$ and $d_j^+(p) = \delta_j^+(p) = \beta_j^-(p) = \infty$, or
3. $d_j^-(p) = \delta_j^+(p) = \beta_j^-(p)$ and $d_j^+(p) = \delta_j^-(p) = \beta_j^+(p) = \infty$.

The algorithm now considers all sets of consistent distance functions to compute an entry $\hat{T}[B'_{\leq \ell}, k', (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$ for $\ell \geq 2$ by setting it to

$$\min \big\{ T[B'_\ell, k'', (\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}] + \hat{T}[B'_{\leq \ell-1}, k' - k'', (\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}] \mid k'' \in \{0, \ldots, k'\} \text{ and } (d_j^+, d_j^-)_{j=i}^{\lambda(W)}, (\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}, (\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)} \text{ are consistent} \big\} \quad (1)$$

We now prove the correctness using two lemmas. The following lemma implies that if we only consider consistent distance functions to compute entries recursively, then the entries will store values for compatible solutions.

▶ **Lemma 21.** *Let $(d_j^+, d_j^-)_{j=i}^{\lambda(W)}$ for $B'_{\leq \ell}$, $(\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}$ for B'_ℓ, and $(\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}$ for $B'_{\leq \ell-1}$ be consistent distance functions, and let $S_1 = B'_\ell \cap F$ and $S_2 = B'_{\leq \ell-1} \cap F$ be facility sets. If S_1 is compatible with entry $T[B'_\ell, |S_1|, (\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}]$ and S_2 is compatible with entry $\hat{T}[B'_{\leq \ell-1}, |S_2|, (\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}]$, then the union $S = S_1 \cup S_2$ is compatible with entry $\hat{T}[B'_{\leq \ell}, |S|, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$. Moreover, $\hat{C}_{\leq \ell}(S) = C_{B'_\ell}(S_1) + \hat{C}_{\leq \ell-1}(S_2)$.*

Proof. To prove compatibility of S with the entry $\hat{T}[B'_{\leq \ell}, |S|, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$, it suffices to show that the distance functions are set correctly. Fix a level $j \geq i$ and an interface point $p \in I_B^j$. There are three cases to consider, according to the definition of consistency of the distance functions. In the first case, all three internal distance functions are set to ∞, and all external distance functions are set to the same value. In particular, since S_1 and S_2 are compatible with their respective entries, we have $d_j^+(p) = \delta_j^+(p) = \beta_j^+(p) \leq \min\{\langle \text{dist}(p, S_1) \rangle_j, \langle \text{dist}(p, S_2) \rangle_j\} = \langle \text{dist}(p, S) \rangle_j$, as $S = S_1 \cup S_2$. In the second case, $\beta_j^-(p) = \delta_j^+(p) = \infty$ and so $\beta_j^+(p) \leq \langle \text{dist}(p, S_2) \rangle_j$, since S_2 is compatible with its entry, and

$\delta_j^-(p) = \langle \text{dist}(p, S_1)\rangle_j$, since S_1 is compatible with its entry. Since we also have $\beta_j^+(p) = \delta_j^-(p)$ we get $\langle \text{dist}(p, S_1)\rangle_j \leq \langle \text{dist}(p, S_2)\rangle_j$, and hence $\langle \text{dist}(p, S)\rangle_j = \langle \text{dist}(p, S_1)\rangle_j$. Consistency furthermore implies $d_j^-(p) = \delta_j^-(p) = \langle \text{dist}(p, S)\rangle_j$ and $d_j^+(p) = \infty$. The third case is analogous to the second, and therefore S is compatible with its entry.

For the second part, we consider the contributions of vertices to the terms $\hat{C}_{\leq \ell}(S)$, $C_{B'_\ell}(S_1)$, and $\hat{C}_{\leq \ell-1}(S_2)$, and show that they are the same for $\hat{C}_{\leq \ell}(S)$ and for $C_{B'_\ell}(S_1) + \hat{C}_{\leq \ell-1}(S_2)$. For this we first fix a vertex $v \in B'_{\leq \ell-1}$, and in the following distinguish the cases where its contribution to $\hat{C}_{\leq \ell-1}(S_2)$ and $\hat{C}_{\leq \ell}(S)$ is due to a facility or an interface point.

The first case is that $\text{dist}'(v, S_2) \leq \min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + \beta_j^+(p)\}$, i.e., the contribution of v to $\hat{C}_{\ell-1}(S_2)$ is given by a facility of S_2. Note that the consistency of the distance functions always implies that $\beta_j^+(p) = d_j^+(p)$ or $d_j^+(p) = \infty$ for any level $j \geq i$ and interface point $p \in I^j$, and so $\min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + \beta_j^+(p)\} \leq \min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + d_j^+(p)\}$. At the same time $\text{dist}'(v, S) \leq \text{dist}'(v, S_2)$ as $S_2 \subseteq S$. We hence get that $\text{dist}'(v, S) \leq \min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + d_j^+(p)\}$, i.e., the contribution of v to $\hat{C}_\ell(S)$ is also given by a facility of S in this case. Thus to show that the contribution of v to $\hat{C}_{\ell-1}(S_2)$ and $\hat{C}_\ell(S)$ is the same, we need to show that $\text{dist}'(v, S) = \text{dist}'(v, S_2)$. Note that this is implied if $\text{dist}'(v, S) \geq \min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + \beta_j^+(p)\}$, since we have $\text{dist}'(v, S) \leq \text{dist}'(v, S_2) \leq \min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + \beta_j^+(p)\}$. Thus the following proves the claim, using that the contribution of v to $\hat{C}_\ell(S)$ is given by a facility of S.

▷ Claim 22. For $v \in B'_{\leq \ell-1}$, if $\text{dist}'(v, S) \leq \min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + d_j^+(p)\}$ then we have $\text{dist}'(v, S) = \text{dist}'(v, S_2)$ or $\text{dist}'(v, S) \geq \min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + \beta_j^+(p)\}$.

Proof. Given $\text{dist}'(v, S) \leq \min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + d_j^+(p)\}$, assume to the contrary that we have $\text{dist}'(v, S) < \min_{j \geq i,\, p \in I_B^j}\{\text{dist}(v, p) + \beta_j^+(p)\}$ and $\text{dist}'(v, S) \neq \text{dist}'(v, S_2)$, which, as $S = S_1 \cup S_2$, means $\text{dist}'(v, S) < \text{dist}'(v, S_2)$. The latter inequality implies that the value of $\text{dist}'(v, S)$ is obtained for some facility $f \in S_1 \subseteq B'_\ell$. In particular, $v \in B'_{\leq \ell-1}$ and $f \in B'_\ell$ are cut at level $i - 1$, and so there is an interface point $p \in I_B^i$ such that $\text{dist}'(v, S) = \text{dist}(v, p) + \langle \text{dist}(p, f)\rangle_i$, and f is the closest facility to p in S, i.e, $\langle \text{dist}(p, S)\rangle_i = \langle \text{dist}(p, f)\rangle_i$. Using the former of the assumed inequalities we get $\text{dist}(v, p) + \langle \text{dist}(p, f)\rangle_i = \text{dist}'(v, S) < \text{dist}(v, p) + \beta_i^+(p)$, and so we can conclude that $\langle \text{dist}(p, f)\rangle_i < \beta_i^+(p)$.

Using the inequality of the premise of the claim, we also get $\text{dist}(v, p) + \langle \text{dist}(p, f)\rangle_i = \text{dist}'(v, S) \leq \text{dist}(v, p) + d_i^+(p)$, i.e. $\langle \text{dist}(p, f)\rangle_i \leq d_i^+(p)$. Since S is compatible with entry $\hat{T}[B'_{\leq \ell}, |S|, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$, we have $d_i^+(p) = \infty$ or $d_i^+(p) \leq \langle \text{dist}(p, S)\rangle_i$. In the latter case we would have $d_i^+(p) \leq \langle \text{dist}(p, S)\rangle_i = \langle \text{dist}(p, f)\rangle_i < \beta_i^+(p)$, which however cannot happen if the distance functions are consistent. Thus compatibility of S implies $d_i^+(p) = \infty$ and $d_i^-(p) = \langle \text{dist}(p, f)\rangle_i$. In particular, we can conclude that $d_i^-(p)$ has a finite value (as f exists) and $\beta_i^+(p)$ differs from $d_i^-(p)$. This can only mean that the third of the consistency properties applies to p at level i, and so $\beta_i^-(p) = d_i^-(p) = \langle \text{dist}(p, f)\rangle_i$.

In particular, also $\beta_i^-(p)$ has a finite value, and using the compatibility of S_2 with entry $\hat{T}[B'_{\leq \ell-1}, |S_2|, (\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}]$, we can conclude that there exists a facility $f' \in S_2 \subseteq B'_{\leq \ell-1}$ with $\langle \text{dist}(p, f')\rangle_i = \beta_i^-(p) = \langle \text{dist}(p, f)\rangle_i$. Now let $j \leq i$ be the level for which $v \in B'_{\leq \ell-1}$ and $f' \in B'_{\leq \ell-1}$ are cut at level $j - 1$ by $\mathcal{D}$. Lemma 18 implies $\text{dist}'(v, f') \leq \text{dist}(v, p) + \langle \text{dist}(p, f')\rangle_i$, but then we have

$$\text{dist}'(v, S_2) \leq \text{dist}'(v, f') \leq \text{dist}(v, p) + \langle \text{dist}(p, f')\rangle_i = \text{dist}(v, p) + \langle \text{dist}(p, f)\rangle_i = \text{dist}'(v, S),$$

which is a contradiction to $\text{dist}'(v, S) < \text{dist}'(v, S_2)$. ◁

The next case we consider is that $\min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+d_j^+(p)\} < \text{dist}'(v,S)$, i.e., the contribution of v to $\hat{C}_\ell(S)$ is given by an interface point. As observed before, we have $\min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\} \leq \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+d_j^+(p)\}$ and $\text{dist}'(v,S) \leq \text{dist}'(v,S_2)$, which implies $\min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\} < \text{dist}'(v,S_2)$, i.e. in this case the contribution of v to $\hat{C}_{\ell-1}(S_2)$ is also given by an interface point. Note that it also implies $\text{dist}'(v,S) > \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$, and thus the following claim shows that the contribution of v to $\hat{C}_\ell(S)$ and $\hat{C}_{\ell-1}(S_2)$ is the same.

▷ Claim 23. For $v \in B'_{\leq \ell-1}$, if $\text{dist}'(v,S) > \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$ then we have $\min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\} = \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+d_j^+(p)\}$.

Proof. Given $\text{dist}'(v,S) > \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$, assume to the contrary that $\min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\} \neq \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+d_j^+(p)\}$. As observed before, the consistency of the distance functions always implies $\beta_j^+(p) = d_j^+(p)$ or $d_j^+(p) = \infty$, and thus we must have $\min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\} < \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+d_j^+(p)\}$. Let $j \geq i$ and $p \in I_B^j$ be the level and interface point for which the minimum of the former term of this inequality is obtained. The inequality then implies $\beta_j^+(p) < d_j^+(p)$ for this particular point p and level j, which can only be the case if $\beta_j^+(p) < \infty$ and $d_j^+(p) = \infty$. The values of $\beta_j^+(p)$ and $d_j^+(p)$ can only differ if the second of the consistency properties applies to p at level j, and so $\beta_j^+(p) = d_j^-(p)$. Since $\beta_j^+(p) < \infty$, the compatibility of S with entry $\hat{T}[B'_{\leq \ell}, |S|, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$, implies $\beta_j^+(p) = d_j^-(p) = \langle \text{dist}(p,S)\rangle_j$.

Now let $f \in S \subseteq B'_{\leq \ell}$ be the facility for which $\text{dist}'(v,S) = \text{dist}'(v,f)$ (which exists as $d_j^-(p) < \infty$). Let $j' \leq i$ be the level for which $v \in B'_{\leq \ell-1}$ and $f \in B'_{\leq \ell}$ are cut at level $j'-1$ by $\mathcal{D}$. By Lemma 18 we have $\text{dist}'(v,f) \leq \text{dist}(v,p) + \langle \text{dist}(p,f)\rangle_j$, since $j' \leq j$ and the part $B \in \mathcal{B}_i$ containing v and f is itself contained in some part $\tilde{B} \in \mathcal{B}_j$ with $v, f \in \tilde{B}$ and $p \in I_{\tilde{B}}$. But then,

$$\text{dist}'(v,S) \leq \text{dist}(v,p) + \langle \text{dist}(p,f)\rangle_j = \text{dist}(v,p) + \langle \text{dist}(p,S)\rangle_j = \text{dist}(v,p) + \beta_j^+(p).$$

However the last term is equal to $\min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$, which gives a contradiction to our premise $\text{dist}'(v,S) > \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$. ◁

So far we considered the case when the contribution of v to $\hat{C}_{\ell-1}(S_2)$ is given by a facility, or when the contribution of v to $\hat{C}_\ell(S)$ is given by an interface point. Thus the last case we consider is when the contribution of v to $\hat{C}_{\ell-1}(S_2)$ is given by an interface point, and the contribution of v to $\hat{C}_\ell(S)$ is given by a facility, i.e., $\min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\} < \text{dist}'(v,S_2)$ and $\text{dist}'(v,S) \leq \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+d_j^+(p)\}$. We need to show that $\text{dist}'(v,S) = \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$. First assume $\text{dist}'(v,S) > \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$. Due to Claim 23 this would imply $\text{dist}'(v,S) > \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+d_j^+(p)\}$, which however contradicts our assumption to the contrary, i.e., that the contribution of v to $\hat{C}_\ell(S)$ is given by a facility. Hence we must instead have $\text{dist}'(v,S) \leq \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$.

According to Claim 22, our assumption that $\text{dist}'(v,S) \leq \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+d_j^+(p)\}$ implies $\text{dist}'(v,S) = \text{dist}'(v,S_2)$ or $\text{dist}'(v,S) \geq \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$. In the former case, together with our assumption that the contribution of v to $\hat{C}_{\ell-1}(S_2)$ is given by an interface point, we would get $\text{dist}'(v,S) > \min_{j\geq i,\ p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$, for which we saw above that this leads to a contradiction via Claim 23. Hence we are left with the other

implication of Claim 22, i.e., $\text{dist}'(v,S) \geq \min_{j\geq i,\, p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$. This together with our conclusion from above, i.e., $\text{dist}'(v,S) \leq \min_{j\geq i,\, p\in I_B^j}\{\text{dist}(v,p)+\beta_j^+(p)\}$, means that the contribution of v to $\hat{C}_\ell(S)$ and $\hat{C}_{\ell-1}(S_2)$ is the same.

By analogous arguments, the contribution of any $v \in B'_\ell$ to $C_{B'_\ell}(S_1)$ is the same as its contribution to $\hat{C}_{\leq\ell}(S)$. Since B'_ℓ and $B'_{\leq\ell-1}$ partition the set $B'_{\leq\ell}$, this means that $\hat{C}_{\leq\ell}(S) = C_{B'_\ell}(S_1) + \hat{C}_{\leq\ell-1}(S_2)$, as required. ◀

The next lemma implies that the compatible facility set minimizing $\hat{C}_{\leq\ell}(S)$ is considered as a solution when recursing over consistent distance functions.

▶ **Lemma 24** (⋆). *Let $S = B'_{\leq\ell} \cap F$ be a facility set of $B'_{\leq\ell}$ that is compatible with entry $\hat{T}[B'_{\leq\ell}, |S|, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$, and let $S_1 = S \cap B'_\ell$ and $S_2 = S \cap B'_{\leq\ell-1}$. Then there exist distance functions $(\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}$ for B'_ℓ, and $(\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}$ for $B'_{\leq\ell-1}$ such that*

- *$(d_j^+, d_j^-)_{j=i}^{\lambda(W)}$, $(\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}$, and $(\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}$ are consistent, and*
- *the set S_1 is compatible with entry $T[B'_\ell, |S_1|, (\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}]$ and S_2 is compatible with entry $\hat{T}[B'_{\leq\ell-1}, |S_2|, (\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}]$.*

To argue that the algorithm sets the value of $\hat{T}[B'_{\leq\ell}, k', (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$ correctly via (1), consider a set $S \subseteq B'_{\leq\ell}$ that is compatible with this entry and minimizes $\hat{C}_{\leq\ell}(S)$. By induction, Lemma 24 implies $T[B'_\ell, |S_1|, (\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}] \leq C_{B'_\ell}(S_1)$ and $\hat{T}[B'_{\leq\ell-1}, |S_2|, (\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}] \leq \hat{C}_{\leq\ell-1}(S_2)$, where $S_1 = S \cap B'_\ell$ and $S_2 = S \cap B'_{\leq\ell-1}$. From (1) we therefore obtain $\hat{T}[B'_{\leq\ell}, |S|, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}] \leq C_{B'_\ell}(S_1) + \hat{C}_{\leq\ell-1}(S_2)$. By Lemma 21 only compatible sets are stored in an entry by induction, and so the definition of S implies $\hat{T}[B'_{\leq\ell}, |S|, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}] = \hat{C}_{\leq\ell}(S)$, as required.

Bounding the runtime. To bound the size of the tables T and $\hat{T}$, note that since there are $\lambda(W) - \xi(W) + 1 \leq 2\log_2(nX/\varepsilon) + 2$ considered levels i, and each level $\mathcal{B}_i$ of $\mathcal{D}$ is a partition of W where $|W| \leq n$, there are at most $O(n\log(nX/\varepsilon))$ parts B considered by T in total. The other table $\hat{T}$ considers the same number of parts, since a set $B'_{\leq\ell}$ can be uniquely mapped to the part B'_ℓ. The number of possible values for k' is $k+1 = O(n)$. The domain $\{\langle x\rangle_j \mid 0 < x \leq 2^{j+5}\} \cup \{\infty\}$ of a distance function for level j has at most $\lceil 2^{j+5}/(\rho 2^j)\rceil + 1 = O(1/\rho)$ values, since $\langle x\rangle_j$ rounds a value to a multiple of $\rho 2^j$. The conciseness of the interface sets means that $|I_B^j| \leq (h/\rho)^{O(1)}$ according to Lemma 5. Hence there are at most $O(1/\rho)^{(h/\rho)^{O(1)}} = 2^{(h/\rho)^{O(1)}}$ possible distance functions. Since each entry of the table stores two distance functions for each of at most $2\log_2(nX/\varepsilon)+2$ levels, the total number of entries of T and $\hat{T}$ is at most

$$O(n\log(nX/\varepsilon)) \cdot n \cdot (2^{(h/\rho)^{O(1)}})^{O(\log(nX/\varepsilon))} = (nX/\varepsilon)^{(h/\rho)^{O(1)}}.$$

Computing an entry of a table is dominated by (1). Going through all values $k' \leq n$ and all possible consistent distance functions to compute (1), takes $n \cdot 2^{(h/\rho)^{O(1)}}$ time, as there are $2^{(h/\rho)^{O(1)}}$ possible distance functions. Hence the total runtime is $(nX/\varepsilon)^{(h/\rho)^{O(1)}}$, proving Lemma 20.

The Facility Location^q problem. To compute an optimum rounded interface-respecting solution to Facility Location^q, the tables T and $\hat{T}$ can ignore the number of open facilities k', i.e., they have respective entries $T[B, (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ and $\hat{T}[B'_{\leq\ell}, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$.

Accordingly, compatibility of facility sets with entries is defined as before, but ignoring the sizes of the sets. The value stored in each entry now also takes the opening costs of facilities into account. That is, for any set of facilities $S \subseteq F \cap B$ in a part B we define

$$C_B(S) = \sum_{v \in B} \chi_{\mathcal{I}_0}(v) \cdot \min\Big\{ \operatorname{dist}'(v,S), \min_{\substack{j \geq i+1 \\ p \in I_B^j}} \big\{ \operatorname{dist}(v,p) + d_j^+(p) \big\} \Big\} + \sum_{f \in S} w_f,$$

and an entry $T[B, (d_j^+, d_j^-)_{j=i+1}^{\lambda(W)}]$ stores the minimum value of $C_B(S)$ over all sets S compatible with the entry, or ∞ if no such set exists. For $S \subseteq F \cap B'_{\leq \ell}$ in a union of subparts $B'_{\leq \ell}$ we define

$$\hat{C}_{\leq \ell}(S) = \sum_{v \in B'_{\leq \ell}} \chi_{\mathcal{I}_0}(v) \cdot \min\Big\{ \operatorname{dist}'(v,S), \min_{\substack{j \geq i \\ p \in I_B^j}} \big\{ \operatorname{dist}(v,p) + d_j^+(p) \big\} \Big\} + \sum_{f \in S} w_f,$$

and an entry $\hat{T}[B'_{\leq \ell}, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}]$ stores the minimum value of $\hat{C}_{\leq \ell}(S)$ over all sets S compatible with the entry, or ∞ if no such set exists.

The entries of the tables can be computed in the same manner as before, but ingoring the sets sizes. In particular, the most involved recursion becomes

$$\hat{T}[B'_{\leq \ell}, (d_j^+, d_j^-)_{j=i}^{\lambda(W)}] = \min \big\{ T[B'_\ell, (\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}] + \hat{T}[B'_{\leq \ell-1}, (\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)}] \;\big|\; (d_j^+, d_j^-)_{j=i}^{\lambda(W)}, (\delta_j^+, \delta_j^-)_{j=i}^{\lambda(W)}, (\beta_j^+, \beta_j^-)_{j=i}^{\lambda(W)} \text{ are consistent} \big\}.$$

Note that if $S_1 = B'_\ell \cap F$ and $S_2 = B'_{\leq \ell-1} \cap F$ then these two sets are disjoint, and so $\sum_{f \in S} w_f = \sum_{f \in S_1} w_f + \sum_{f \in S_2} w_f$ for the union $S = S_1 \cup S_2$. Hence when proving $\hat{C}_{\leq \ell}(S) = C_{B'_\ell}(S_1) + \hat{C}_{\leq \ell-1}(S_2)$ for Lemma 21, we can ignore the facility opening costs, and the proof remains the same as before. All other arguments carry over, and thus an optimum rounded interface-respecting solution for an instance of Facility Location^q can also be computed in $(nX/\varepsilon)^{(h/\rho)^{O(1)}}$ time.

5 Hardness for graphs of highway dimension 1

For both k-Clustering^q and Facility Location^q we present the same reduction from the NP-hard satisfiability problem (SAT), in which a boolean formula φ in conjunctive normal form is given, and a satisfying assignment of its variables needs to be found.

For a given SAT formula φ with k variables and ℓ clauses we construct a graph G_φ as follows. For each variable x we introduce a path $P_x = (t_x, u_x, f_x)$ with two edges of length 1 each. The two endpoints t_x and f_x are facilities of F and the additional vertex u_x is a client, i.e., $\chi(u_x) = 1$. For each clause C_i, where $i \in [\ell]$, we introduce a vertex v_i and add the edge $v_i t_x$ for each variable x such that C_i contains x as a positive literal, and we add the edge $v_i f_x$ for each x for which C_i contains x as a negative literal. Every edge incident to v_i has length $(11c)^i$ for the constant $c > 4$ due to Definition 1, and v_i is also a client, i.e., $\chi(v_i) = 1$. In case of Facility Location^q, every facility $f \in F$ has cost $w_f = 1$, i.e., we construct an instance of the uniform version of the problem.

▶ **Lemma 25.** *The constructed graph G_φ has highway dimension 1.*

Proof. Fix a scale $r > 0$ and let $i = \lfloor \log_{11c}(r/5) + 1 \rfloor$. Note that $\beta_w(cr)$ cannot contain any edge incident to a vertex v_j for $j \geq i+1$, since the length of every such edge is $(11c)^j \geq 11cr/5 > 2cr$ and the diameter of $\beta_w(cr)$ is at most $2cr$. Thus if $\beta_w(cr)$ contains a vertex v_j for $j \geq i+1$, then $\beta_w(cr)$ contains only v_j, and there is nothing to prove. Note

also that any path in $\beta_w(cr)$ that does not use v_i has length at most $2 + \sum_{j=1}^{i-1}(2(11c)^j + 2)$, since any such path can contain at most two edges incident to a vertex v_j and the paths P_x of length 2 are connected only through edges incident to vertices v_j. The length of such a path is thus strictly shorter than

$$2 + 2\left(\frac{(11c)^i}{11c-1} - 1\right) + 2i \leq 5(11c)^{i-1} \leq r,$$

where the first inequality holds since $i \geq 1$ and $c > 4$. Hence the only paths that need to be hit by hubs on scale r are those passing through v_i, which can clearly be done using only one hub, namely v_i. ◀

To finish the reduction for k-Clustering^q, we claim that there is a satisfying assignment for φ if and only if there is a solution for G_φ with cost at most $k + \sum_{i=1}^{\ell}(11c)^{iq}$. If there is a satisfying assignment for φ we open each facility t_x for variables x that are set to true, and we open each facility f_x for variables x that are set to false. This opens exactly k facilities and the cost of the solution is $k + \sum_{i=1}^{\ell}(11c)^{iq}$, since each of the k vertices u_x is assigned to either t_x or f_x at distance 1, and vertex v_i is assigned to a vertex t_x or f_x at distance $(11c)^i$ that corresponds to a literal of C_i that is true.

Conversely, assume there is a solution to k-Clustering^q of cost at most $k + \sum_{i=1}^{\ell}(11c)^{iq}$ in G_φ. Note that the minimum distance from any u_x to a facility is 1, while the minimum distance from any v_i to a facility is $(11c)^i$. Thus any solution must have cost at least $k + \sum_{i=1}^{\ell}(11c)^{iq}$, so that the assumed solution must open a facility at minimum distance for each client of G_φ. In particular, for each variable x, at least one of the facilities t_x and f_x is opened by the solution. Moreover, as only k facilities can be opened and there are k variables, exactly one of t_x and f_x is opened for each x. Thus the k-Clustering^q solution in G_φ can be interpreted as an assignment for φ, where we set a variable x to true if t_x is opened, and we set it to false if f_x is opened. Since also for each v_i the solution opens a facility at minimum distance, there must be a variable in C_i that is set so that its literal in C_i is true, i.e., the assignment satisfies φ. Thus due to the above lemma bounding the highway dimension of G_φ, we obtain the Theorem 3 for k-Clustering^q.

For Facility Location^q we claim that there is a satisfying assignment for φ if and only if there is a solution for G_φ of cost at most $2k + \sum_{i=1}^{\ell}(11c)^{iq}$. In fact the arguments are exactly the same as for k-Clustering^q above: if there is a satisfying assignment then a solution for Facility Location^q of cost $2k + \sum_{i=1}^{\ell}(11c)^{iq}$ exists, by opening the k facilities corresponding to the assignment of cost 1 each. Conversely, any solution has cost at least $k + \sum_{i=1}^{\ell}(11c)^{iq}$ due to the edge lengths, and at least k facilities need to be opened, one for each variable gadget. This gives a minimum cost of $2k + \sum_{i=1}^{\ell}(11c)^{iq}$, and any such solution corresponds to a satisfying assignment of φ. This proves Theorem 3 for uniform Facility Location^q.

References

1. I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *SODA*, pages 782–793, 2010.
2. Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *SIAM Journal on Computing*, pages FOCS17–97, 2019.
3. Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 106–113, New York, NY, USA, 1998. ACM. doi:10.1145/276698.276718.

4 Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of euclidean k-means. In *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, pages 754–767, 2015.

5 Yair Bartal and Lee-Ad Gottlieb. A linear time approximation scheme for euclidean TSP. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2013.

6 A. Becker, P. N. Klein, and D. Saulpic. Polynomial-time approximation schemes for k-center and bounded-capacity vehicle routing in metrics with bounded highway dimension. In *ESA*, pages 8:1–8:15, 2018.

7 J. Blum. Hierarchy of transportation network parameters and hardness results. In *IPEC*, pages 4:1–4:15, 2019.

8 Vladimir Braverman, Shaofeng H-C Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in excluded-minor graphs and beyond. *arXiv preprint*, 2020. `arXiv:2004.07718`.

9 Jaroslaw Byrka, Thomas Pensyl, Bartosz Rybicki, Srinivasan Aravind, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 737–756, 2015.

10 Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-linear time approximations schemes for clustering in doubling metrics. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 540–559, 2019. `doi:10.1109/FOCS.2019.00041`.

11 Vincent Cohen-Addad, Philip N Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. *SIAM Journal on Computing*, 48(2):644–667, 2019.

12 Y. Disser, A. E. Feldmann, M. Klimm, and J. Könemann. Travelling on graphs with small highway dimension. In *Graph-Theoretic Concepts in Computer Science - 45th International Workshop, WG*, volume 11789, pages 175–189. Springer, 2019.

13 A. E. Feldmann. Fixed-parameter approximations for k-center problems in low highway dimension graphs. *Algorithmica*, 81(3):1031–1052, 2019.

14 A. E. Feldmann, W. S. Fung, J. Könemann, and I. Post. A $(1 + \varepsilon)$-embedding of low highway dimension graphs into bounded treewidth graphs. *SIAM Journal on Computing*, 47(4):1275–1734, 2018.

15 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. *SIAM J. Comput.*, 48(2):452–480, 2019. `doi:10.1137/17M1127181`.

16 Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, 2003.

17 Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *CoRR*, abs/0809.2554, 2008. `arXiv:0809.2554`.

18 Stavros G Kolliopoulos and Satish Rao. A nearly linear-time approximation scheme for the euclidean k-median problem. *SIAM Journal on Computing*, 37(3):757–782, 2007.

19 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013.

20 K. Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 281–290. ACM, 2004. `doi:10.1145/1007352.1007399`.

Coresets for the Nearest-Neighbor Rule

Alejandro Flores-Velazco
Department of Computer Science, University of Maryland, College Park, MD, USA
afloresv@cs.umd.edu

David M. Mount
Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA
mount@umd.edu

Abstract

Given a training set P of *labeled* points, the *nearest-neighbor rule* predicts the class of an *unlabeled* query point as the label of its closest point in the set. To improve the time and space complexity of classification, a natural question is how to reduce the training set without significantly affecting the accuracy of the nearest-neighbor rule. *Nearest-neighbor condensation* deals with finding a subset $R \subseteq P$ such that for every point $p \in P$, p's nearest-neighbor in R has the same label as p. This relates to the concept of *coresets*, which can be broadly defined as subsets of the set, such that an exact result on the coreset corresponds to an approximate result on the original set. However, the guarantees of a coreset hold for any query point, and not only for the points of the training set.

This paper introduces the concept of coresets for nearest-neighbor classification. We extend existing criteria used for condensation, and prove sufficient conditions to correctly classify any query point when using these subsets. Additionally, we prove that finding such subsets of minimum cardinality is NP-hard, and propose quadratic-time approximation algorithms with provable upper-bounds on the size of their selected subsets. Moreover, we show how to improve one of these algorithms to have subquadratic runtime, being the first of this kind for condensation.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases coresets, nearest-neighbor rule, classification, nearest-neighbor condensation, training-set reduction, approximate nearest-neighbor, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.47

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.06650`.

Supplementary Material Source code is available at `https://github.com/afloresv/nnc`.

Funding Research partially supported by NSF grant CCF-1618866.

Acknowledgements Thanks to Prof. Emely Arráiz for suggesting the problem of condensation while the first author was a student at Universidad Simón Bolívar, Venezuela. Thanks to Ahmed Abdelkader for the helpful discussions and valuable suggestions.

1 Introduction

In non-parametric classification, we are given a *training set* P consisting of n points in a metric space $(\mathcal{X}, \mathsf{d})$, with domain $\mathcal{X}$ and distance function $\mathsf{d} : \mathcal{X}^2 \to \mathbb{R}^+$. Additionally, P is partitioned into a finite set of *classes* by associating each point $p \in P$ with a *label* $l(p)$, indicating the class to which it belongs. Given an *unlabeled* query point $q \in \mathcal{X}$, the goal of a *classifier* is to predict q's label using the training set P.

The *nearest-neighbor rule* is among the best-known classification techniques [19]. It assigns a query point the label of its closest point in P, according to the metric d. The nearest-neighbor rule exhibits good classification accuracy both experimentally and theoretically [14, 15, 36], but it is often criticized due to its high space and time complexities. Clearly, the training set

P must be stored to answer nearest-neighbor queries, and the time required for such queries depends to a large degree on the size and dimensionality of the data. These drawbacks inspire the question of whether it is possible replace P with a significantly smaller subset, without significantly reducing the classification accuracy under the nearest-neighbor rule. This problem is called *nearest-neighbor condensation* [22, 25, 34, 37].

There are obvious parallels between condensation and the concept of *coresets* in geometric approximation [1, 17, 23, 33]. Intuitively, a *coreset* is small subset of the original data, that well approximates some statistical properties of the original set. Coresets have also been applied to many problems in machine learning, such as clustering and neural network compression [8, 11, 18, 29]. This includes recent results on coresets for the SVM classifier [38].

This paper presents the first approach to compute coresets for the nearest-neighbor rule, leveraging its resemblance to the problem of nearest-neighbor condensation. We also present one of the first results on practical condensation algorithms with theoretical guarantees.

Preliminaries. Given any point $q \in \mathcal{X}$ in the metric space, its nearest-neighbor, denoted $\mathrm{nn}(q)$, is the closest point of P according the the distance function d. The distance from q to its nearest-neighbor is denoted by $\mathsf{d}_{\mathrm{nn}}(q, P)$, or simply $\mathsf{d}_{\mathrm{nn}}(q)$ when P is clear. Given a point $p \in P$ from the training set, its nearest-neighbor in P is point p itself. Additionally, any point of P whose label differs from p's is called an *enemy* of p. The closest such point is called p's *nearest-enemy*, and the distance to this point is called p's *nearest-enemy distance*. These are denoted by $\mathrm{ne}(p)$ and $\mathsf{d}_{\mathrm{ne}}(p, P)$ (or simply $\mathsf{d}_{\mathrm{ne}}(p)$), respectively.

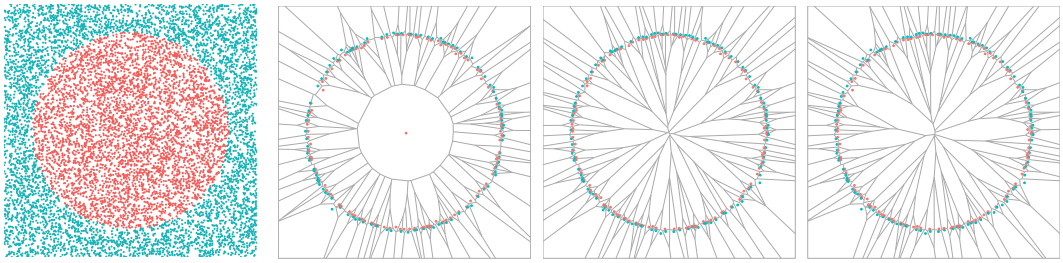
Clearly, the size of a coreset for nearest-neighbor classification depends on the spatial characteristics of the classes in the training set. For example, it is much easier to find a small coreset for two spatially well separated clusters than for two classes that have a high degree of overlap. To model the intrinsic complexity of nearest-neighbor classification, we define κ to be the number of nearest-enemy points of P, i.e., the cardinality of set $\{\mathrm{ne}(p) \mid p \in P\}$.

Through a suitable uniform scaling, we may assume that the *diameter* of P (that is, the maximum distance between any two points in the training set) is 1. The *spread* of P, denoted as Δ, is the ratio between the largest and smallest distances in P. Define the *margin* of P, denoted γ, to be the smallest nearest-enemy distance in P. Clearly, $1/\gamma \leq \Delta$.

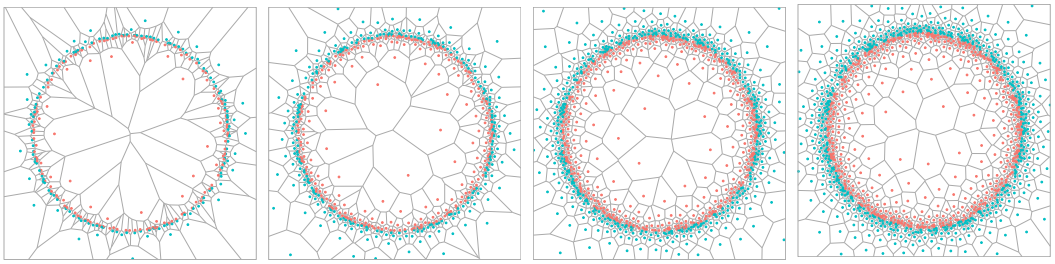
A metric space $(\mathcal{X}, \mathsf{d})$ is said to be *doubling* [26] if there exist some bounded value λ such that any metric ball of radius r can be covered with at most λ metric balls of radius $r/2$. Its *doubling dimension* is the base-2 logarithm of λ, denoted as $\mathrm{ddim}(\mathcal{X}) = \log \lambda$. Throughout, we assume that $\mathrm{ddim}(\mathcal{X})$ is a constant, which means that multiplicative factors depending on $\mathrm{ddim}(\mathcal{X})$ may be hidden in our asymptotic notation. Many natural metric spaces of interest are doubling, including d-dimensional Euclidean space whose doubling dimension is $\Theta(d)$. It is well know that for any subset $R \subseteq \mathcal{X}$ with some spread Δ_R, the size of R is bounded by $|R| \leq \lceil \Delta_R \rceil^{\mathrm{ddim}(\mathcal{X})+1}$.

Related Work. A subset $R \subseteq P$ is said to be *consistent* [25] if and only if for every $p \in P$ its nearest-neighbor in R is of the same class as p. Intuitively, R is consistent if and only if all points of P are correctly classified using the nearest-neighbor rule over R. Formally, the problem of *nearest-neighbor condensation* consists of finding a consistent subset of P.

Another criterion used for condensation is known as *selectiveness* [34]. A subset $R \subseteq P$ is said to be *selective* if and only if for all $p \in P$ its nearest-neighbor in R is closer to p than its nearest-enemy in P. Clearly, any selective subset is also consistent. Observe that these condensation criteria ensure that every point in the training set will be correctly classified after condensation, but they do not imply the same for arbitrary points in the metric space.



(a) Training set (10^4 pts). **(b)** FCNN (222 pts). **(c)** VSS (233 pts). **(d)** RSS (233 pts).



(e) 0.1-RSS (300 pts). **(f)** 0.5-RSS (540 pts). **(g)** 1-RSS (846 pts). **(h)** $\sqrt{2}$-RSS (1066 pts).

Figure 1 An illustrative example of the subsets selected by different condensation algorithms from an initial training set P in $\mathbb{R}^2$ of 10^4 points. FCNN, VSS, and RSS, are known algorithms for this problem, while α-RSS is proposed in this paper, along with new condensation criteria. The subsets selected by α-RSS depend on the parameter $\alpha \geq 0$, here assigned to the values $\alpha = \{0.1, 0.5, 1, \sqrt{2}\}$.

It is known that the problems of computing consistent and selective subsets of minimum cardinality are both NP-hard [28, 39, 40]. An approximation algorithm called NET [22] was proposed for the problem of finding minimum cardinality consistent subsets, along with almost matching hardness lower-bounds. The algorithm simply computes a γ-net of P, where γ is the minimum nearest-enemy distance in P, which clearly results in a consistent subset of P (also selective). In practice, γ tends to be small, which results in subsets of much higher cardinality than needed. To overcome this issue, the authors proposed a post-processing pruning technique to further reduce the selected subset. Even with the extra pruning, NET is often outperformed on typical data sets by more practical heuristics with respect to runtime and selection size. More recently, a subexponential-time algorithm was proposed [10] for finding minimum cardinality consistent subsets of point sets $P \subset \mathbb{R}^2$ in the plane, along with other case-specific algorithms for special instances of the problem in $\mathbb{R}^2$. On the other hand, less is known about computing minimum cardinality selective subsets: there is only a worst-case exponential time algorithm called SNN [34] for computing such optimal subsets.

Most research has focused on proposing practical heuristics to find either consistent or selective subsets of P (for comprehensive surveys see [27, 37]). CNN (*Condensed Nearest-Neighbor*) [25] was the first algorithm proposed to compute consistent subsets. Even though it has been widely used in the literature, CNN suffers from several drawbacks: its running time is cubic in the worst-case, and the resulting subset is *order-dependent*, meaning that the result is determined by the order in which points are considered by the algorithm. Alternatives include FCNN (*Fast* CNN) [3] and MSS (*Modified Selective Subset*) [7], which compute consistent and selective subsets respectively. Both algorithms run in $\mathcal{O}(n^2)$ worst-case time, and are order-independent. While such heuristics have been extensively studied experimentally [21], theoretical results are scarce. Recently, it was shown [20] that the size of the subsets selected by MSS cannot be bounded, while for FCNN it is still unknown whether is possible to establish any bound. The same paper [20] proposes two new algorithms, namely

RSS (*Relaxed Selective Subset*) and VSS (*Voronoi Selective Subset*), to find selective subsets of P in $\mathcal{O}(n^2)$ worst-case time. Both algorithms provide some guarantees on its selection size in Euclidean space.

Contributions. As mentioned in the previous section, consistency and selectivity imply correct classification to points of the training set, but not to arbitrary points of the metric space (This is striking since this is the fundamental purpose of classification!). In this paper, we introduce the concept of a coreset for classification with the nearest-neighbor rule, which provides approximate guarantees on correct classification for all query points. We demonstrate their existence, analyze their size, and discuss their efficient computation.

We say that a subset $R \subseteq P$ is an *ε-coreset for the nearest-neighbor rule* on P, if and only if for every query point $q \in \mathcal{X}$, the class of its exact nearest-neighbor in R is the same as the class of some ε-approximate nearest-neighbor of q in P (see Section 2 for definitions). Recalling the concepts of κ and γ introduced in the preliminaries, here is our main result:

▶ **Theorem 1.** *Given a training set P in a doubling metric space $(\mathcal{X}, \mathsf{d})$, there exist an ε-coreset for the nearest-neighbor rule of size $\mathcal{O}(\kappa \log \frac{1}{\gamma} (1/\varepsilon)^{\mathrm{ddim}(\mathcal{X})+1})$, and this coreset can be computed in subquadratic worst-case time.*

Here is a summary of our principal results:

- We extend the criteria used for nearest-neighbor condensation, and identify sufficient conditions to guarantee the correct classification of any query point after condensation.
- We prove that finding minimum-cardinality subsets with this new criteria is NP-hard.
- We provide quadratic-time approximation algorithms with provable upper-bounds on the sizes of their selected subsets, and we show that the running time of one such algorithm can be improved to be subquadratic.

Our subquadratic-time algorithm is the first with such worst-case runtime for the problem of nearest-neighbor condensation.

2 Coreset Characterization

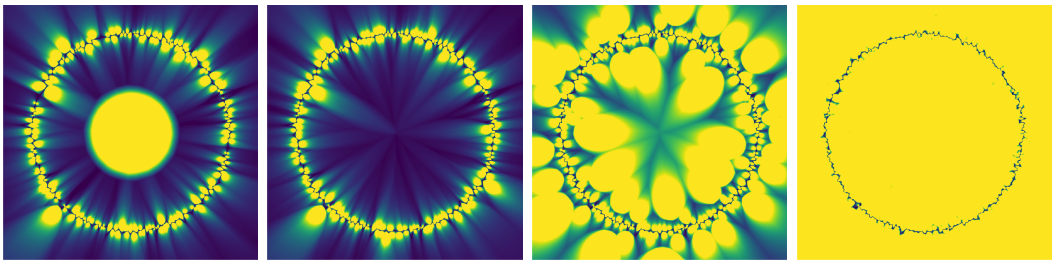
In practice, nearest-neighbors are usually not computed exactly, but rather approximately. Given an approximation parameter $\varepsilon \geq 0$, an *ε-approximate* nearest-neighbor or ε-ANN query returns any point whose distance from the query point is within a factor of $(1 + \varepsilon)$ times the true nearest-neighbor distance.

Intuitively, a query point should be easier to classify if its nearest-neighbor is significantly closer than its nearest-enemy. This intuition can be formalized through the concept of the *chromatic density* [31] of a query point $q \in \mathcal{X}$ with respect to a set $R \subseteq P$, defined as:

$$\delta(q, R) = \frac{\mathsf{d}_{\mathrm{ne}}(q, R)}{\mathsf{d}_{\mathrm{nn}}(q, R)} - 1. \tag{1}$$

Clearly, if $\delta(q, R) > \varepsilon$ then q will be correctly classified[1] by an ε-ANN query over R, as all possible candidates for the approximate nearest-neighbor belong to the same class as q's true nearest-neighbor. However, as evidenced in Figures 2a and 2b, one side effect of existing condensation algorithms is a significant reduction in the chromatic density of query points. Consequently, we propose new criteria and algorithms that maintain high chromatic densities after condensation, which are then leveraged to build coresets for the nearest-neighbor rule.

[1] By *correct classification*, we mean that the classification is the same as the classification that results from applying the nearest-neighbor rule exactly on the entire training set P.



(a) FCNN. **(b)** RSS. **(c)** 0.1-RSS. **(d)** 0.5-RSS.

Figure 2 Heatmap of *chromatic density* values of points in $\mathbb{R}^2$ w.r.t. the subsets computed by different condensation algorithms: FCNN, RSS, and α-RSS (see Figure 1). *Yellow* • corresponds to chromatic density values ≥ 0.5, while *blue* • corresponds to 0. Evidently, α-RSS helps maintaining high chromatic density values when compared to standard condensation algorithms.

2.1 Approximation-Sensitive Condensation

The decision boundaries of the nearest-neighbor rule (that is, points q such that $\mathsf{d}_{\text{ne}}(q, P) = \mathsf{d}_{\text{nn}}(q, P)$) are naturally characterized by points that separate clusters of points of different classes. As illustrated in Figures 1b-1d, condensation algorithms tend to select such points. However, this behavior implies a significant reduction of the chromatic density of query points that are far from such boundaries (see Figures 2a-2b).

A natural way to define an approximate notion of consistency is to ensure that all points in P are correctly classified by ANN queries over the condensed subset R. Given a condensation parameter $\alpha \geq 0$, we define a subset $R \subseteq P$ to be:

α-consistent if $\forall\ p \in P,\ \mathsf{d}_{\text{nn}}(p, R) < \mathsf{d}_{\text{ne}}(p, R)/(1+\alpha)$.

α-selective if $\forall\ p \in P,\ \mathsf{d}_{\text{nn}}(p, R) < \mathsf{d}_{\text{ne}}(p, P)/(1+\alpha)$.

It is easy to see that the standard forms arise as special cases when $\alpha = 0$. These new condensation criteria imply that $\delta(p, R) > \alpha$ for every $p \in P$, meaning that p is correctly classified using an α-ANN query on R. Note that any α-selective subset is also α-consistent. Such subsets always exist for any $\alpha \geq 0$ by taking $R = P$. Current condensation algorithms cannot guarantee either α-consistency or α-selectiveness unless α is equal to zero. Therefore, the central algorithmic challenge is how to efficiently compute such sets whose sizes are significantly smaller than P. We propose new algorithms to compute such subsets, which showcase how to maintain high chromatic density values after condensation, as evidenced in Figures 2c and 2d. This empirical evidence is matched with theoretical guarantees for α-consistent and α-selective subsets, described in the following section.

2.2 Guarantees on Classification Accuracy

These newly defined criteria for nearest-neighbor condensation enforce lower-bounds on the chromatic density of any point of P after condensation. However, this doesn't immediately imply having similar lower-bounds for unlabeled query points of $\mathcal{X}$. In this section, we prove useful bounds on the chromatic density of query points, and characterize sufficient conditions to correctly classify some of these query points after condensation.

Intuitively, the chromatic density determines how easy it is to correctly classify a query point $q \in \mathcal{X}$. We show that the "ease" of classification of q after condensation (i.e., $\delta(q, R)$) depends on both the condensation parameter α, and the chromatic density of q before condensation (i.e., $\delta(q, P)$). This result is formalized in the following lemma:

▶ **Lemma 2.** *Let $q \in \mathcal{X}$ be a query point, and R an α-consistent subset of P, for $\alpha \geq 0$. Then, q's chromatic density with respect to R is:*

$$\delta(q, R) > \frac{\alpha\,\delta(q, P) - 2}{\delta(q, P) + \alpha + 3}.$$

Proof. The proof follows by analyzing q's nearest-enemy distance in R. To this end, consider the point $p \in P$ that is q's nearest-neighbor in P. There are two possible cases:

Case 1: If $p \in R$, clearly $\mathsf{d}_{\mathrm{nn}}(q, R) = \mathsf{d}_{\mathrm{nn}}(q, P)$. Additionally, it is easy to show that after condensation, q's nearest-enemy distance can only increase: i.e., $\mathsf{d}_{\mathrm{ne}}(q, P) \leq \mathsf{d}_{\mathrm{ne}}(q, R)$. This implies that $\delta(q, R) \geq \delta(q, P)$.

Case 2: If $p \notin R$, we can upper-bound q's nearest-neighbor distance in R as follows:

Since R is an α-consistent subset of P, we know that there exists a point $r \in R$ such that $\mathsf{d}(p, r) < \mathsf{d}_{\mathrm{ne}}(p, R)/(1 + \alpha)$. By the triangle inequality and the definition of nearest-enemy, $\mathsf{d}_{\mathrm{ne}}(p, R) \leq \mathsf{d}(p, \mathrm{ne}(q, R)) \leq \mathsf{d}(q, p) + \mathsf{d}_{\mathrm{ne}}(q, R)$. Additionally, applying the definition of chromatic density on q and knowing that $\mathsf{d}_{\mathrm{ne}}(q, P) \leq \mathsf{d}_{\mathrm{ne}}(q, R)$, we have $\mathsf{d}(q, p) = \mathsf{d}_{\mathrm{nn}}(q, P) \leq \mathsf{d}_{\mathrm{nn}}(q, R) = \mathsf{d}_{\mathrm{ne}}(q, R)/(1 + \delta(q, P))$. Therefore:

$$\begin{aligned}\mathsf{d}_{\mathrm{nn}}(q, R) &\leq \mathsf{d}(q, r) \leq \mathsf{d}(q, p) + \mathsf{d}(p, r) \\ &< \mathsf{d}(q, p) + \frac{\mathsf{d}(q, p) + \mathsf{d}_{\mathrm{ne}}(q, R)}{1 + \alpha} \leq \left(\frac{\delta(q, P) + \alpha + 3}{(1 + \alpha)(1 + \delta(q, P))}\right) \mathsf{d}_{\mathrm{ne}}(q, R).\end{aligned}$$

Finally, from the definition of $\delta(q, R)$, we have:

$$\delta(q, R) = \frac{\mathsf{d}_{\mathrm{ne}}(q, R)}{\mathsf{d}_{\mathrm{nn}}(q, R)} - 1 > \frac{(1 + \alpha)(1 + \delta(q, P))}{\delta(q, P) + \alpha + 3} - 1 = \frac{\alpha\,\delta(q, P) - 2}{\delta(q, P) + \alpha + 3}.$$ ◀

The above result can be leveraged to define a coreset, in the sense that an exact result on the coreset corresponds to an approximate result on the original set. As previously defined, we say that a set $R \subseteq P$ is an *ε-coreset for the nearest-neighbor rule* on P, if and only if for every query point $q \in \mathcal{X}$, the class of q's exact nearest-neighbor in R is the same as the class of any of its ε-approximate nearest-neighbors in P.

▶ **Lemma 3.** *Any ε-coreset for the nearest-neighbor rule is an α-consistent subset, for $\alpha \geq 0$.*

Proof. Consider any ε-coreset $C \subseteq P$ for the nearest-neighbor rule on P. Since the approximation guarantee holds for any point in $\mathcal{X}$, it holds for any $p \in P \setminus C$. We know p's nearest-neighbor in the original set P is p itself, thus making $\mathsf{d}_{\mathrm{nn}}(p, P)$ zero. This implies that p must be correctly classified by a nearest-neighbor query on C, that is, $\mathsf{d}_{\mathrm{nn}}(p, C) < \mathsf{d}_{\mathrm{ne}}(p, C)$, which is the definition of α-consistency for any $\alpha \geq 0$. ◀

▶ **Theorem 4.** *Any $2/\varepsilon$-selective subset is an ε-coreset for the nearest-neighbor rule.*

Proof. Let R be an α-selective subset of P, where $\alpha = 2/\varepsilon$. Consider any query point $q \in \mathcal{X}$ in the metric space. It suffices to show that its nearest-neighbor in R is of the same class as any ε-approximate nearest-neighbor in P. To this end, consider q's chromatic density with respect to both P and R, denoted as $\delta(q, P)$ and $\delta(q, R)$, respectively. We identify two cases:

Case 1 (Correct-Classification guarantee): If $\delta(q, P) \geq \varepsilon$.

Consider the bound derived in Lemma 2. Since $\alpha \geq 0$, and by our assumption that $\delta(q, P) \geq \varepsilon > 0$, setting $\alpha = 2/\varepsilon$ implies that $\delta(q, R) > 0$. This means that the nearest-neighbor of q in R belongs to the same class as the nearest-neighbor of q in P. Intuitively, this guarantees that q is correctly classified by the nearest-neighbor rule in R.

Case 2 (ε-Approximation guarantee): If $\delta(q,P) < \varepsilon$.

Let $p \in P$ be q's nearest-neighbor in P, thus $\mathsf{d}(q,p) = \mathsf{d}_{\mathrm{nn}}(q,P)$. Since R is α-selective, there exists a point $r \in R$ such that $\mathsf{d}(p,r) = \mathsf{d}_{\mathrm{nn}}(p,R) < \mathsf{d}_{\mathrm{ne}}(p,P)/(1+\alpha)$. Additionally, by the triangle inequality and the definition of nearest-enemies, we have

$$\mathsf{d}_{\mathrm{ne}}(p,P) \leq \mathsf{d}(p,\mathrm{ne}(q,P)) \leq \mathsf{d}(p,q) + \mathsf{d}(q,\mathrm{ne}(q,P)) = \mathsf{d}_{\mathrm{nn}}(q,P) + \mathsf{d}_{\mathrm{ne}}(q,P).$$

From the definition of chromatic density, $\mathsf{d}_{\mathrm{ne}}(q,P) = (1+\delta(q,P))\,\mathsf{d}_{\mathrm{nn}}(q,P)$. Together, these inequalities imply that $(1+\alpha)\,\mathsf{d}(p,r) \leq (2+\delta(q,P))\,\mathsf{d}_{\mathrm{nn}}(q,P)$. Therefore:

$$\mathsf{d}_{\mathrm{nn}}(q,R) \leq \mathsf{d}(q,r) \leq \mathsf{d}(q,p) + \mathsf{d}(p,r) \leq \left(1 + \frac{2+\delta(q,P)}{1+\alpha}\right)\mathsf{d}_{\mathrm{nn}}(q,P).$$

Now, assuming $\delta(q,P) < \varepsilon$ and setting $\alpha = 2/\varepsilon$, imply that $\mathsf{d}_{\mathrm{nn}}(q,R) < (1+\varepsilon)\,\mathsf{d}_{\mathrm{nn}}(q,P)$. Therefore, the nearest-neighbor of q in R is an ε-approximate nearest-neighbor of q in P.

Cases 1 and 2 imply that setting $\alpha = 2/\varepsilon$ is sufficient to ensure that the nearest-neighbor rule classifies any query point $q \in \mathcal{X}$ with the class of one of its ε-approximate nearest-neighbors in P. Therefore, R is an ε-coreset for the nearest-neighbor rule on P. ◀

So far, we have assumed that nearest-neighbor queries over R are computed exactly, as this is the standard notion of coresets. However, it is reasonable to compute nearest-neighbors approximately even for R. How should the two approximations be combined to achieve a desired final degree of accuracy? Consider another approximation parameter ξ, where $0 \leq \xi < \varepsilon$. We say that a set $R \subseteq P$ is an (ξ,ε)-*coreset* for the approximate nearest-neighbor rule on P, if and only if for every query point $q \in \mathcal{X}$, the class of any of q's ξ-approximate nearest-neighbor in R is the same as the class of any of its ε-approximate nearest-neighbors in P. The following result generalizes Theorem 4 to accommodate for ξ-ANN queries after condensation.

▶ **Theorem 5.** *Any α-selective subset is an (ξ,ε)-coreset for the approximate nearest-neighbor rule when $\alpha = \Omega(1/(\varepsilon-\xi))$.*

Proof. This follows from similar arguments to the ones described in the proof of Theorem 4. Instead, here we set $\alpha = (\varepsilon\xi + 3\xi + 2)/(\varepsilon - \xi)$. Let R be an α-selective subset of P, and $q \in \mathcal{X}$ any query point in the metric space, consider the same two cases:

Case 1 (Correct-Classification guarantee): If $\delta(q,P) \geq \varepsilon$.

Consider the bound derived in Lemma 2. By our assumption that $\delta(q,P) \geq \varepsilon > 0$, and since $\alpha \geq 0$, the following inequality holds true:

$$\delta(q,R) > \frac{\alpha\,\delta(q,P) - 2}{\delta(q,P) + \alpha + 3} \geq \frac{\alpha\varepsilon - 2}{\varepsilon + \alpha + 3}$$

Based on this, it is easy to see that the assignment of $\alpha = (\varepsilon\xi + 3\xi + 2)/(\varepsilon - \xi)$ implies that $\delta(q,R) > \xi$, meaning that any of q's ξ-approximate nearest-neighbors in R belong to the same class as q's nearest-neighbor in P. Intuitively, this guarantees that q is correctly classified by the ξ-ANN rule in R.

Case 2 (ε-Approximation guarantee): If $\delta(q,P) < \varepsilon$.

The assignment of α implies that $\mathsf{d}_{\mathrm{nn}}(q,R) < \frac{1+\varepsilon}{1+\xi}\,\mathsf{d}_{\mathrm{nn}}(q,P)$. This means that an ξ-ANN query for q in R, will return one of q's ε-approximate nearest-neighbors in P.

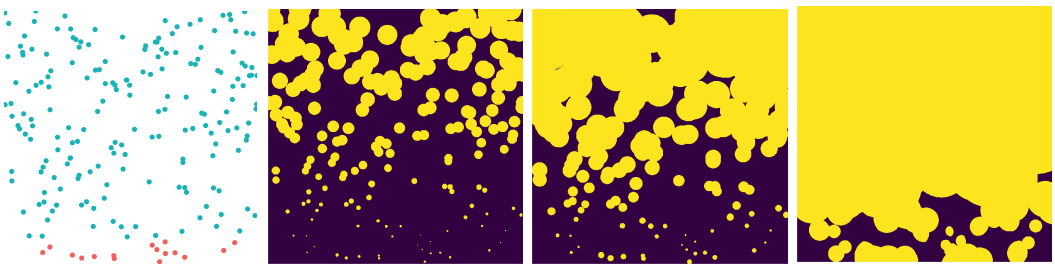
All together, this implies that R is an (ξ,ε)-coreset for the nearest-neighbor rule on P. ◀

In contrast with standard condensation criteria, these new results provide guarantees on either approximation or the correct classification, of any query point in the metric space. This is true even for query points that were "hard" to classify with the entire training set, formally defined as query points with low chromatic density. Consequently, Theorems 4 and 5 show that α must be set to large values if we hope to provide any sort of guarantees for these query points. However, better results can be achieved by restricting the set of points that are guaranteed to be correctly classified. This relates to the notion of *weak* coresets, which provide approximation guarantees only for a subset of the possible queries. Given $\beta \geq 0$, we define $\mathcal{Q}_\beta$ as the set of query points in $\mathcal{X}$ whose chromatic density with respect to P is at least β (i.e., $\mathcal{Q}_\beta = \{q \in \mathcal{X} \mid \delta(q, P) \geq \beta\}$). The following result describes the trade-off between α and β to guarantee the correct classification of query points in $\mathcal{Q}_\beta$ after condensation.

▶ **Theorem 6.** *Any α-consistent subset is a weak ε-coreset for the nearest-neighbor rule for queries in $\mathcal{Q}_\beta$, for $\beta = 2/\alpha$. Moreover, all query points in $\mathcal{Q}_\beta$ are correctly classified.*

The proof of this theorem is rather simple, and follows the same arguments outlined in Case 1 of the proof of Theorem 4. Basically, we use Lemma 2 to show that for any query point $q \in \mathcal{Q}_\beta$, q's chromatic density after condensation is greater than zero if $\alpha\beta \geq 2$. Note that ε plays no role in this result, as the guarantee on query points of $\mathcal{Q}_\beta$ is of correct classification (i.e., the class of its *exact* nearest-neighbor in P), rather than an approximation.

The trade-off between α and β is illustrated in Figure 3. From an initial training set $P \subset \mathbb{R}^2$ (Figure 3a), we show the regions of $\mathbb{R}^2$ that comprise the sets $\mathcal{Q}_\beta$ for $\beta = 2/\alpha$, using $\alpha = \{0.1, 0.2, \sqrt{2}\}$ (Figures 3b-3d). While evidently, increasing α guarantees that more query points will be correctly classified after condensation, this example demonstrates a phenomenon commonly observed experimentally: most query points lie far from enemy points, and thus have high chromatic density with respect to P. Therefore, while Theorem 4 states that α must be set to $2/\varepsilon$ to provide approximation guarantees on all query points, Theorem 6 shows that much smaller values of α are sufficient to provide guarantees on some query points, as evidenced in the example in Figure 3.



(a) Training set (200 pts). **(b)** $\mathcal{Q}_{2/\alpha}$ for $\alpha = 0.1$. **(c)** $\mathcal{Q}_{2/\alpha}$ for $\alpha = 0.2$. **(d)** $\mathcal{Q}_{2/\alpha}$ for $\alpha = \sqrt{2}$.

Figure 3 Depiction of the $\mathcal{Q}_\beta$ sets for which any α-consistent subset is weak coreset ($\beta = 2/\alpha$). Query points in the *yellow* • areas are inside $\mathcal{Q}_\beta$, and thus correctly classified after condensation. Query points in the *blue* • areas are not in $\mathcal{Q}_\beta$, and have no guarantee of correct classification.

These results establish a clear connection between the problem of condensation and that of finding coresets for the nearest-neighbor rule, and provides a roadmap to prove Theorem 1. This is the first characterization of sufficient conditions to correctly classify any query point in $\mathcal{X}$ after condensation, and not just the points in P (as the original consistency criteria implies). In the following section, these existential results are matched with algorithms to compute α-selective subsets of P of bounded cardinality.

3 Coreset Computation

3.1 Hardness Results

Define Min-α-CS to be the problem of computing an α-consistent subset of minimum cardinality for a given training set P. Similarly, let Min-α-SS be the corresponding optimization problem for α-selective subsets. Following known results from standard condensation [28, 39, 40], when α is set to zero, the Min-0-CS and Min-0-SS problems are both known to be NP-hard. Being special cases of the general problems just defined, this implies that both Min-α-CS and Min-α-SS are NP-hard.

In this section, we present results related to the hardness of approximation of both problems, along with simple algorithmic approaches with tight approximation factors.

▶ **Theorem 7.** *The Min-α-CS problem is* NP*-hard to approximate in polynomial time within a factor of* $2^{(\mathrm{ddim}(\mathcal{X})\log((1+\alpha)/\gamma))^{1-o(1)}}$.

The full proof is omitted, as it follows from a modification of the hardness bounds proof for the Min-0-CS problem described in [22], which is based on a reduction from the *Label Cover* problem. Proving Theorem 7 involves a careful adjustment of the distances in this reduction, so that all the points in the construction have chromatic density at least α. Consequently, this implies that the minimum nearest-enemy distance is reduced by a factor of $1/(1+\alpha)$, explaining the resulting bound for Min-α-CS.

The NET algorithm [22] can also be generalized to compute α-consistent subsets of P as follows. We define α-NET as the algorithm that computes a $\gamma/(1+\alpha)$-net of P, where γ is the smallest nearest-enemy distance in P. The covering property of nets [24] implies that the resulting subset is α-consistent, while the packing property suggests that its cardinality is $\mathcal{O}\left(((1+\alpha)/\gamma)^{\mathrm{ddim}(\mathcal{X})+1}\right)$, implying a tight approximation to the Min-α-CS problem.

▶ **Theorem 8.** *The Min-α-SS problem is* NP*-hard to approximate in polynomial time within a factor of* $(1-o(1))\ln n$ *unless* NP $\subseteq$ DTIME$(n^{\log\log n})$.

Proof. The result follows from the hardness of another related covering problem: the minimum *dominating set* [16, 30, 32]. We describe a simple L-reduction from any instance of this problem to an instance of Min-α-SS, which preserves the approximation ratio.

1. Consider any instance of minimum dominating set, consisting of the graph $G = (V, E)$.
2. Generate a new edge-weighted graph G' as follows:
   Create two copies of G, namely $G_{\mathsf{r}} = (V_{\mathsf{r}}, E_{\mathsf{r}})$ and $G_{\mathsf{b}} = (V_{\mathsf{b}}, E_{\mathsf{b}})$, of *red* and *blue* nodes respectively. Set all edge-weights of G_{r} and G_{b} to be 1. Finally, connect each red node v_{r} to its corresponding blue node v_{b} by an edge $\{v_{\mathsf{r}}, v_{\mathsf{b}}\}$ of weight $1+\alpha+\xi$ for a sufficiently small constant $\xi > 0$. Formally, G' is defined as the edge-weighted graph $G' = (V', E')$ where the set of nodes is $V' = V_{\mathsf{r}} \cup V_{\mathsf{b}}$, the set of edges is $E' = E_{\mathsf{r}} \cup E_{\mathsf{r}} \cup \{\{v_{\mathsf{r}}, v_{\mathsf{b}}\} \mid v \in V\}$, and an edge-weight function $w : E' \to \mathbb{R}^+$ where $w(e) = 1$ iff $e \in E_{\mathsf{r}} \cup E_{\mathsf{b}}$, and $w(e) = 1+\alpha+\xi$ otherwise.
3. A labeling function l where $l(v) = \textit{red}$ iff $v \in V_{\mathsf{r}}$, and $l(v) = \textit{blue}$ iff $v \in V_{\mathsf{b}}$.
4. Compute the shortest-path metric of G', denoted as $\mathsf{d}_{G'}$.
5. Solve the Min-α-SS problem for the set V', on metric $\mathsf{d}_{G'}$, and the labels defined by l.

A dominating set of G consists of a subset of nodes $D \subseteq V$, such that every node $v \in V \setminus D$ is adjacent to a node in D. Given any dominating set $D \subseteq V$ of G, it is easy to see that the subset $R = \{v_{\mathsf{r}}, v_{\mathsf{b}} \mid v \in D\}$ is an α-selective subset of V', where $|R| = 2|D|$.

Similarly, given an α-selective subset $R \subseteq V'$, there is a corresponding dominating set D of G, where $|D| \leq |R|/2$, as D can be either $R \cap V_{\mathsf{r}}$ or $R \cap V_{\mathsf{b}}$. Therefore, MIN-α-SS is as hard to approximate as the minimum dominating set problem. ◀

There is a clear connection between the MIN-α-SS problem and covering problems, in particular that of finding an optimal hitting set. Given a set of elements U and a family C of subsets of U, a *hitting set* of (U, C) is a subset $H \subseteq U$ such that every set in C contains at least one element of H. Therefore, let $N_{p,\alpha}$ be the set of points of P whose distance to p is less than $\mathsf{d}_{\mathrm{ne}}(p)/(1+\alpha)$, then any hitting set of $(P, \{N_{p,\alpha} \mid p \in P\})$ is also an α-selective subset of P, and vice versa. This simple reduction implies a $\mathcal{O}(n^3)$ worst-case time $\mathcal{O}(\log n)$-approximation algorithm for MIN-α-SS, based on the classic greedy algorithm for set cover [12, 35]. Call this approach α-HSS or *α-Hitting Selective Subset*. It follows from Theorem 8 that for training sets in general metric spaces, this is the best approximation possible under standard complexity assumptions.

While both α-NET and α-HSS compute tight approximations of their corresponding problems, their performance in practice does not compare to heuristic approaches for standard condensation (see Section 4 for experimental results). Therefore, in the next section, we consider one such heuristic and extend it to compute subsets with the newly defined criteria.

3.2 A Practical Algorithm

For standard condensation, the RSS algorithm was recently proposed [20] to compute selective subsets. It runs in quadratic worst-case time and exhibits good performance in practice. The selection process of this algorithm is heuristic in nature and can be described as follows: beginning with an empty set, the points in $p \in P$ are examined in increasing order with respect to their nearest-enemy distance $\mathsf{d}_{\mathrm{ne}}(p)$. The point p is added to the subset R if $\mathsf{d}_{\mathrm{nn}}(p, R) \geq \mathsf{d}_{\mathrm{ne}}(p)$. It is easy to see that the resulting subset is selective.

We define a generalization, called α-RSS, to compute α-selective subsets of P. The condition to add a point $p \in P$ to the selected subset checks if any previously selected point is closer to p than $\mathsf{d}_{\mathrm{ne}}(p)/(1+\alpha)$, instead of just $\mathsf{d}_{\mathrm{ne}}(p)$. See Algorithm 1 for a formal description, and Figure 4 for an illustration. It is easy to see that this algorithm computes an α-selective subset, while keeping the quadratic time complexity of the original RSS algorithm.

Algorithm 1 α-RSS.

Input: Initial training set P and parameter $\alpha \geq 0$
Output: α-selective subset $R \subseteq P$

```
R ← φ
Let {p_i}_{i=1}^n be the points of P sorted increasingly w.r.t. d_ne(p_i)
foreach p_i ∈ P, where i = 1...n do
    if d_nn(p_i, R) ≥ d_ne(p_i)/(1+α) then
        R ← R ∪ {p_i}
return R
```

Naturally, we want to analyze the number of points this algorithm selects. The remainder of this section establishes upper-bounds and approximation guarantees of the α-RSS algorithm for any doubling metric space, with improved results in the Euclidean space. This resolves the open problem posed in [20] of whether RSS computes an approximation of the MIN-0-CS and MIN-0-SS problems.

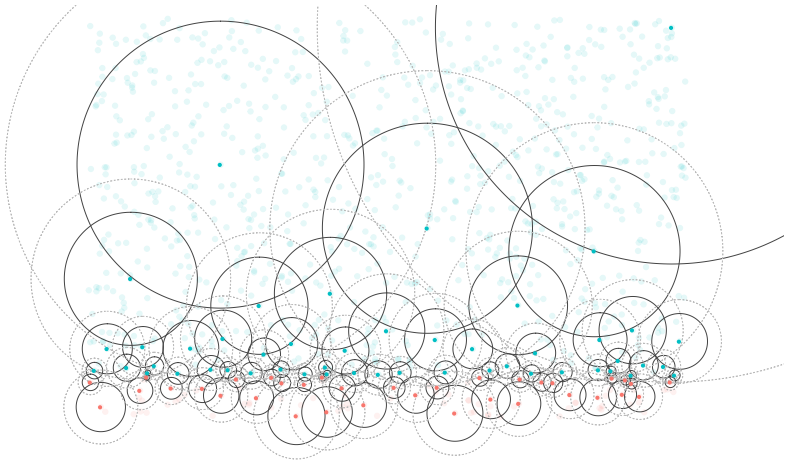


Figure 4 Selection of α-RSS for α=0.5. Faded points are not selected, while selected points are drawn along with a ball of radius $\mathsf{d}_{\mathrm{ne}}(p)$ (dotted outline) and a ball of radius $\mathsf{d}_{\mathrm{ne}}(p)/(1+\alpha)$ (solid outline). A point p is selected if no previously selected point is closer to p than $\mathsf{d}_{\mathrm{ne}}(p)/(1+\alpha)$.

Size in Doubling spaces. First, we consider the case where the underlying metric space $(\mathcal{X}, \mathsf{d})$ of P is doubling. The following results depend on the doubling dimension $\mathrm{ddim}(\mathcal{X})$ of the metric space (which is assumed to be constant), the margin γ (the smallest nearest-enemy distance of any point in P), and κ (the number of nearest-enemy points in P).

▶ **Theorem 9.** *α-RSS computes a tight approximation for the Min-α-CS problem.*

Proof. This follows from a direct comparison to the resulting subset of the α-NET algorithm from the previous section. For any point p selected by α-NET, let $B_{p,\alpha}$ be the set of points of P "covered" by p, that is, whose distance to p is at most $\gamma/(1+\alpha)$. By the covering property of ε-nets, this defines a partition on P when considering every point p selected by α-NET.

Let R be the set of points selected by α-RSS, we analyze the size of $B_{p,\alpha} \cap R$, that is, for any given $B_{p,\alpha}$ how many points could have been selected by the α-RSS algorithm. Let $a, b \in B_{p,\alpha} \cap R$ be any two such points, where without loss of generality, $\mathsf{d}_{\mathrm{ne}}(a) \leq \mathsf{d}_{\mathrm{ne}}(b)$. By the selection process of the algorithm, we know that $\mathsf{d}(a,b) \geq \mathsf{d}_{\mathrm{ne}}(b)/(1+\alpha) \geq \gamma/(1+\alpha)$. A simple packing argument in doubling metrics implies that $|B_{p,\alpha} \cap R| \leq 2^{\mathrm{ddim}(\mathcal{X})+1}$. Altogether, we have that the size of the subset selected by α-RSS is $\mathcal{O}\left((2(1+\alpha)/\gamma)^{\mathrm{ddim}(\mathcal{X})+1}\right)$. ◀

▶ **Theorem 10.** *α-RSS computes an $\mathcal{O}\left(\log\left(\min\left(1+2/\alpha, 1/\gamma\right)\right)\right)$-factor approximation for the Min-α-SS problem. For $\alpha = \Omega(1)$, this is a constant-factor approximation.*

Proof. Let OPT_α be the optimum solution to the Min-α-SS problem, i.e., the minimum cardinality α-selective subset of P. For every point $p \in \mathrm{OPT}_\alpha$ in such solution, define $S_{p,\alpha}$ to be the set of points in P "covered" by p, or simply $S_{p,\alpha} = \{r \in P \mid \mathsf{d}(r,p) < \mathsf{d}_{\mathrm{ne}}(r)/(1+\alpha)\}$. Additionally, let R be the set of points selected by α-RSS, define $R_{p,\sigma}$ to be the points selected by α-RSS which also belong to $S_{p,\alpha}$ and whose nearest-enemy distance is between σ and 2σ, for $\sigma \in [\gamma, 1]$. That is, $R_{p,\sigma} = \{r \in R \cap S_{p,\alpha} \mid \mathsf{d}_{\mathrm{ne}}(r) \in [\sigma, 2\sigma)\}$. Clearly, these subsets define a partitioning of R for all $p \in \mathrm{OPT}_\alpha$ and values of $\sigma = \gamma\, 2^i$ for $i = \{0, 1, 2, \ldots, \lceil \log \frac{1}{\gamma} \rceil\}$.

However, depending on α, some values of σ would yield empty $R_{p,\sigma}$ sets. Consider some point $q \in S_{p,\alpha}$, we can bound its nearest-enemy distance with respect to the nearest-enemy distance of point p. In particular, by leveraging simple triangle-inequality arguments, it is possible to prove that $\frac{1+\alpha}{2+\alpha}\,\mathsf{d}_{\mathrm{ne}}(p) \leq \mathsf{d}_{\mathrm{ne}}(q) \leq \frac{1+\alpha}{\alpha}\,\mathsf{d}_{\mathrm{ne}}(p)$. Therefore, the values of σ for which $R_{p,\sigma}$ sets are not empty, are $\sigma = 2^j\,\frac{1+\alpha}{2+\alpha}\,\mathsf{d}_{\mathrm{ne}}(p)$ for $j = \{0, \ldots, \lceil \log(1 + 2/\alpha) \rceil\}$.

The proof now follows by bounding the size of $R_{p,\sigma}$ which can be achieved by bounding its spread. Thus, lets consider the smallest and largest pairwise distances among points in $R_{p,\sigma}$. Take any two points $a, b \in R_{p,\sigma}$ where without loss of generality, $\mathsf{d}_{\mathrm{ne}}(a) \leq \mathsf{d}_{\mathrm{ne}}(b)$. Note that points selected by α-RSS cannot be "too close" to each other; that is, as a and b were selected by the algorithm, we know that $(1+\alpha)\,\mathsf{d}(a,b) \geq \mathsf{d}_{\mathrm{ne}}(b) \geq \sigma$. Therefore, the smallest pairwise distance in $R_{p,\sigma}$ is at least $\sigma/(1+\alpha)$. Additionally, by the triangle inequality, we can bound the maximum pairwise distance using their distance to p as $\mathsf{d}(a,b) \leq \mathsf{d}(a,p) + \mathsf{d}(p,b) \leq 4\sigma/(1+\alpha)$. Then, by the packing properties of doubling spaces, the size of $R_{p,\sigma}$ is at most $4^{\mathrm{ddim}(\mathcal{X})+1}$.

Altogether, for every $p \in \mathrm{OPT}_\alpha$ there are up to $\lceil \log(\min(1 + 2/\alpha, 1/\gamma)) \rceil$ non-empty $R_{p,\sigma}$ subsets, each containing at most $4^{\mathrm{ddim}(\mathcal{X})+1}$ points. In doubling spaces with constant doubling dimension, the size of these subsets is also constant. ◀

While these results are meaningful from a theoretical perspective, it is also useful to establishing bounds in terms of the geometry of the learning space, which is characterized by the boundaries between points of different classes. Thus, using similar packing arguments as above, we bound the selection size of the algorithm with respect to κ.

▶ **Theorem 11.** α-RSS *selects* $\mathcal{O}\left(\kappa \log \frac{1}{\gamma}\ (1+\alpha)^{\mathrm{ddim}(\mathcal{X})+1}\right)$ *points.*

Proof. This follows from similar arguments to the ones used to prove Theorem 10, using an alternative charging scheme for each nearest-enemy point in the training set. Consider one such point $p \in \{\mathrm{ne}(r) \mid r \in P\}$ and a value $\sigma \in [\gamma, 1]$, we define $R'_{p,\sigma}$ to be the subset of points from α-RSS whose nearest-enemy is p, and their nearest-enemy distance is between σ and 2σ. That is, $R'_{p,\sigma} = \{r \in R \mid \mathrm{ne}(r) = p \wedge \mathsf{d}_{\mathrm{ne}}(r) \in [\sigma, 2\sigma)\}$. These subsets partition R for all nearest-enemy points of P, and values of $\sigma = \gamma\, 2^i$ for $i = \{0, 1, 2, \ldots, \lceil \log \frac{1}{\gamma} \rceil\}$.

For any two points $a, b \in R'_{p,\sigma}$, the selection criteria of α-RSS implies some separation between selected points, which can be used to prove that $\mathsf{d}(a,b) \geq \sigma/(1+\alpha)$. Additionally, we know that $\mathsf{d}(a,b) \leq \mathsf{d}(a,p) + \mathsf{d}(p,b) = \mathsf{d}_{\mathrm{ne}}(a) + \mathsf{d}_{\mathrm{ne}}(b) \leq 4\sigma$. Using a simple packing argument, we have that $|R'_{p,\sigma}| \leq \lceil 4(1+\alpha) \rceil^{\mathrm{ddim}(\mathcal{X})+1}$.

Altogether, by counting all sets $R'_{p,\sigma}$ for each nearest-enemy in the training set and values of σ, the size of R is upper-bounded by $|R| \leq \kappa \lceil \log 1/\gamma \rceil \lceil 4(1+\alpha) \rceil^{\mathrm{ddim}(\mathcal{X})+1}$. Based on the assumption that $\mathrm{ddim}(\mathcal{X})$ is constant, this completes the proof. ◀

As a corollary, this result implies that when $\alpha = 2/\varepsilon$, the α-selective subset computed by α-RSS contains $\mathcal{O}\left(\kappa \log 1/\gamma\ (1/\varepsilon)^{\mathrm{ddim}(\mathcal{X})+1}\right)$ points. This establishes the size bound on the ε-coreset given in Theorem 1, which can be computed using the α-RSS algorithm.

Size in Euclidean space. In the case where $P \subset \mathbb{R}^d$ lies in d-dimensional Euclidean space, the analysis of α-RSS can be further improved, leading to a constant-factor approximation of Min-α-SS for any value of $\alpha \geq 0$, and reduced dependency on the dimensionality of P.

▶ **Theorem 12.** α-RSS *computes an* $\mathcal{O}(1)$*-approximation for the* Min-α*-SS problem in* $\mathbb{R}^d$.

Proof. Similar to the proof of Theorem 10, define $R_p = S_{p,\alpha} \cap R$ as the points selected by α-RSS that are "covered" by p in the optimum solution OPT_α. Consider two such points $a, b \in R_p$ where without loss of generality, $\mathsf{d}_{\mathrm{ne}}(a) \leq \mathsf{d}_{\mathrm{ne}}(b)$. By the definition of $S_{p,\alpha}$ we know that $\mathsf{d}(a,p) < \mathsf{d}_{\mathrm{ne}}(a)/(1+\alpha)$, and similarly with b. Additionally, from the selection of the algorithm we know that $\mathsf{d}(a,b) \geq \mathsf{d}_{\mathrm{ne}}(b)/(1+\alpha)$. Overall, these inequalities imply that the angle $\angle apb \geq \pi/3$. By a simple packing argument, the size of R_p is bounded by the *kissing number* in d-dimensional Euclidean space, or simply $\mathcal{O}((3/\pi)^{d-1})$. Therefore, we have that $|R| \leq \sum_p |R_p| = |\mathrm{OPT}_\alpha|\ \mathcal{O}((3/\pi)^{d-1})$. Assuming d is constant, this completes the proof. ◀

Furthermore, a similar constant-factor approximation can be achieved for any training set P in ℓ_p space for $p \geq 3$. This follows analogously to the proof of Theorem 12, exploiting the bounds between ℓ_p and ℓ_2 metrics, where $1/\sqrt{d}\ \|v\|_p \leq \|v\|_2 \leq \sqrt{d}\ \|v\|_p$. This would imply that the angle between any two points in α-RSS_p is $\Omega(1/d)$. Therefore, it shows that α-RSS achieves an approximation factor of $\mathcal{O}(d^{d-1})$, or simply $\mathcal{O}(1)$ for constant dimension.

Similarly to the case of doubling spaces, we also establish upper-bounds in terms of κ for the selection size of the algorithm in Euclidean space. The following result improves the exponential dependence on the dimensionality of P (from $\mathsf{ddim}(\mathbb{R}^d) = \Theta(d)$ to $d-1$), while keeping the dependency on the margin γ, which contrast with the approximation factor results.

▶ **Theorem 13.** *In Euclidean space $\mathbb{R}^d$, α-RSS selects $\mathcal{O}\left(\kappa \log \frac{1}{\gamma}\ (1+\alpha)^{d-1}\right)$ points.*

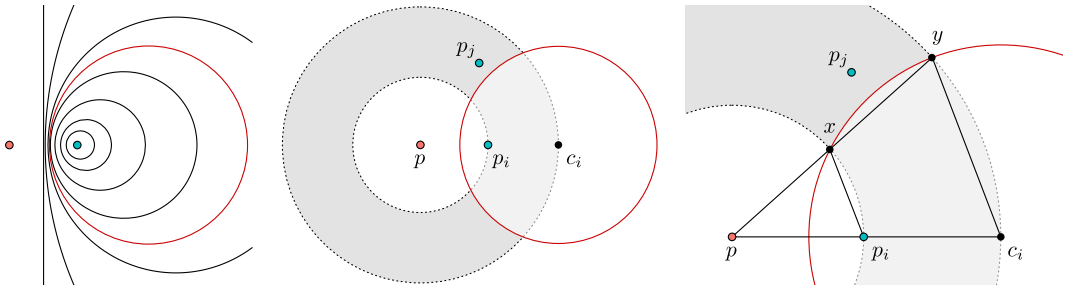
Proof. Let p be any nearest-enemy point of P and $\sigma \in [\gamma, 1]$, similarly define $R'_{p,\sigma}$ to be the set of points selected by α-RSS whose nearest-enemy is p and their nearest-enemy distance is between σ and $b\sigma$, for $b = \frac{(1+\alpha)^2}{\alpha(2+\alpha)}$. Equivalently, these subsets define a partitioning of R for all nearest-enemy points p and values of $\sigma = \gamma\, b^k$ for $k = \{0, 1, 2, \ldots, \lceil \log_b \frac{1}{\gamma} \rceil\}$. Thus, the proof follows from bounding the minimum angle between points in these subsets. For any two such points $p_i, p_j \in R'_{p,\sigma}$, we lower bound the angle $\angle p_i p p_j$. Assume without loss of generality that $\mathsf{d}_{\mathrm{ne}}(p_i) \leq \mathsf{d}_{\mathrm{ne}}(p_j)$. By definition of the partitioning, we also know that $\mathsf{d}_{\mathrm{ne}}(p_j) \leq b\sigma \leq b\, \mathsf{d}_{\mathrm{ne}}(p_i)$. Therefore, altogether we have that $\mathsf{d}_{\mathrm{ne}}(p_i) \leq \mathsf{d}_{\mathrm{ne}}(p_j) \leq b\, \mathsf{d}_{\mathrm{ne}}(p_i)$.

First, consider the set of points whose distance to p_i is $(1+\alpha)$ times their distance to p, which defines a multiplicative weighted bisector [6] between points p and p_i, with weights equal to 1 and $1/(1+\alpha)$ respectively. This is characterized as a d-dimensional ball (see Figure 5a) with center $c_i = (p_i - p)\, b + p$ and radius $\mathsf{d}_{\mathrm{ne}}(p_i)\, b/(1+\alpha)$. Thus p, p_i and c_i are collinear, and the distance between p and c_i is $\mathsf{d}(p, c_i) = b\, \mathsf{d}_{\mathrm{ne}}(p_i)$. In particular, let's consider the relation between p_j and such bisector. As p_j was selected by the algorithm after p_i, we know that $(1+\alpha)\, \mathsf{d}(p_j, p_i) \geq \mathsf{d}_{\mathrm{ne}}(p_j)$ where $\mathsf{d}_{\mathrm{ne}}(p_j) = \mathsf{d}(p_j, p)$. Therefore, clearly p_j lies either outside or in the surface of the weighted bisector between p and p_i (see Figure 5b).

For angle $\angle p_i p p_j$, we can frame the analysis to the plane defined by p, p_i and p_j. Let x and y be two points in this plane, such that they are the intersection points between the weighted bisector and the balls centered at p of radii $\mathsf{d}_{\mathrm{ne}}(p_i)$ and $b\, \mathsf{d}_{\mathrm{ne}}(p_i)$ respectively (see Figure 5c). By the convexity of the weighted bisector between p and p_i, we can say that $\angle p_i p p_j \geq \min(\angle xpp_i, \angle ypc_j)$. Now, consider the triangles $\triangle pxp_i$ and $\triangle pyc_i$. By the careful selection of b, these triangles are both isosceles and similar. In particular, for $\triangle pxp_i$ the two sides incident to p have length equal to $\mathsf{d}_{\mathrm{ne}}(p_i)$, and the side opposite to p has length equal to $\mathsf{d}_{\mathrm{ne}}(p_i)/(1+\alpha)$. For $\triangle pyc_i$, the side lengths are $b\, \mathsf{d}_{\mathrm{ne}}(p_i)$ and $\mathsf{d}_{\mathrm{ne}}(p_i)\, b/(1+\alpha)$. Therefore, the angle $\angle p_i p p_j \geq \angle xpp_i \geq 1/(1+\alpha)$.

By a simple packing argument based on this minimum angle, we have that the size of $R'_{p,\sigma}$ is $\mathcal{O}((1+\alpha)^{d-1})$. All together, following the defined partitioning, we have that:

$$|R| = \sum_p \sum_{k=0}^{\lceil \log_b \frac{1}{\gamma} \rceil} |R'_{p,b^k}| \leq \kappa \left\lceil \log_b \frac{1}{\gamma} \right\rceil \mathcal{O}\left((1+\alpha)^{d-1}\right)$$



(a) Multiplicatively weighted bisectors for different weights.

(b) Position of point p_j w.r.t. the weighted bisector between points p and p_i.

(c) The intersection points x and y between the weighted bisector and the limit balls of $R_{p,\sigma}$.

Figure 5 Construction for the analysis of the minimum angle between two points in $R'_{p,\sigma}$ w.r.t. some nearest-enemy point $p \in P$. Let points $p_i, p_j \in R'_{p,\sigma}$, we analyze the angle $\angle p_i p p_j$.

For constant α and d, the size of α-RSS is $\mathcal{O}(\kappa \log \frac{1}{\gamma})$. Moreover, when α is zero α-RSS selects $\mathcal{O}(\kappa\ c^{d-1})$, matching the previously known bound for RSS in Euclidean space. ◀

3.3 Subquadratic Algorithm

In this section we present a subquadratic implementation for the α-RSS algorithm, which completes the proof of our main result, Theorem 1. Among algorithms for nearest-neighbor condensation, FCNN achieves the best worst-case time complexity, running in $\mathcal{O}(nm)$ time, where $m = |R|$ is the size of the selected subset.

The α-RSS algorithm consists of two main stages: computing the nearest-enemy distances of all points in P (and sorting the points based on these), and the selection process itself. The first stage requires a total of n nearest-enemy queries, plus additional $\mathcal{O}(n \log n)$ time for sorting. The second stage performs n nearest-neighbor queries on the current selected subset R, which needs to be updated m times. In both cases, using exact nearest-neighbor search would degenerate into linear search due to the *curse of dimensionality*. Thus, the first and second stage of the algorithm would need $\mathcal{O}(n^2)$ and $\mathcal{O}(nm)$ worst-case time respectively.

These bottlenecks can be overcome by leveraging approximate nearest-neighbor techniques. Clearly, the first stage of the algorithm can be improved by computing nearest-enemy distances approximately, using as many ANN structures as classes there are in P, which is considered to be a small constant. Therefore, by also applying a simple brute-force search for nearest-neighbors in the second stage, result (i) of the next theorem follows immediately. Moreover, by combining this with standard techniques for static-to-dynamic conversions [9], we have result (ii) below. Denote this variant of α-RSS as (α, ξ)-RSS, for a parameter $\xi \geq 0$.

▶ **Theorem 14.** *Given a data structure for ξ-ANN searching with construction time t_c and query time t_q (which potentially depend on n and ξ), the (α, ξ)-RSS variant can be implemented with the following worst-case time complexities, where m is the size of the selected subset.*

(i) $\mathcal{O}\left(t_c + n\left(t_q + m + \log n\right)\right)$

(ii) $\mathcal{O}\left(\left(t_c + n\, t_q\right) \log n\right)$

More generally, if we are given an additional data structure for dynamic ξ-ANN searching with construction time t'_c, query time t'_q, and insertion time t'_i, the overall running time will be $\mathcal{O}\left(t_c + t'_c + n\left(t_q + t'_q + \log n\right) + m\, t'_i\right)$. Indeed, this can be used to obtain (ii) from the static-to-dynamic conversions [9], which propose an approach to convert static search

structures into dynamic ones. These results directly imply implementations of (α, ξ)-RSS with subquadratic worst-case time complexities, based on ANN techniques [4, 5] for low-dimensional Euclidean space, and using techniques like LSH [2] that are suitable for ANN in high-dimensional Hamming and Euclidean spaces. More generally, subquadratic runtimes can be achieved by leveraging techniques [13] for dynamic ANN search in doubling spaces.

Dealing with uncertainty. Such implementation schemes for α-RSS would incur an approximation error (of up to $1+\xi$) on the computed distances: either only during the first stage if (i) is implemented, or during both stages if (ii) or the dynamic-structure scheme are implemented. The uncertainty introduced by these approximate queries, imply that in order to guarantee finding α-selective subsets, we must modify the condition for adding point during the second stage of the algorithm. Let $\mathsf{d}_{\mathrm{ne}}(p, \xi)$ denote the ξ-approximate nearest-enemy distance of p computed in the first stage, and let $\mathsf{d}_{\mathrm{nn}}(p, R, \xi)$ denote the ξ-approximate nearest-neighbor distance of p over points of the current subset (computed in the second stage). Then, (α, ξ)-RSS adds a point p into the subset if $(1+\xi)(1+\alpha)\,\mathsf{d}_{\mathrm{nn}}(p, R, \xi) \geq \mathsf{d}_{\mathrm{ne}}(p, \xi)$.

By similar arguments to the ones described in Section 3.2, size guarantees can be extended to (α, ξ)-RSS. First, the size of the subset selected by (α, ξ)-RSS, in terms of the number of nearest-enemy points in the set, would be bounded by the size of the subset selected by $\hat{\alpha}$-RSS with $\hat{\alpha} = (1+\alpha)(1+\xi)^2 - 1$. Additionally, the approximation factor of (α, ξ)-RSS in both doubling and Euclidean metric spaces would increase by a factor of $\mathcal{O}((1+\xi)^{2(\mathrm{ddim}(\mathcal{X})+1)})$.

This completes the proof of Theorem 1.

4 Experimental Evaluation

In order to get a clearer impression of the relevance of these results in practice, we performed experimental trials on several training sets, both synthetically generated and widely used benchmarks. First, we consider 21 training sets from the UCI *Machine Learning Repository*[2] which are commonly used in the literature to evaluate condensation algorithms [21]. These consist of a number of points ranging from 150 to 58000, in d-dimensional Euclidean space with d between 2 and 64, and 2 to 26 classes. We also generated some synthetic training sets, containing 10^5 uniformly distributed points, in 2 to 3 dimensions, and 3 classes. All training sets used in these experimental trials are summarized in Table 1. The implementation of the algorithms, training sets used, and raw results, are publicly available[3].

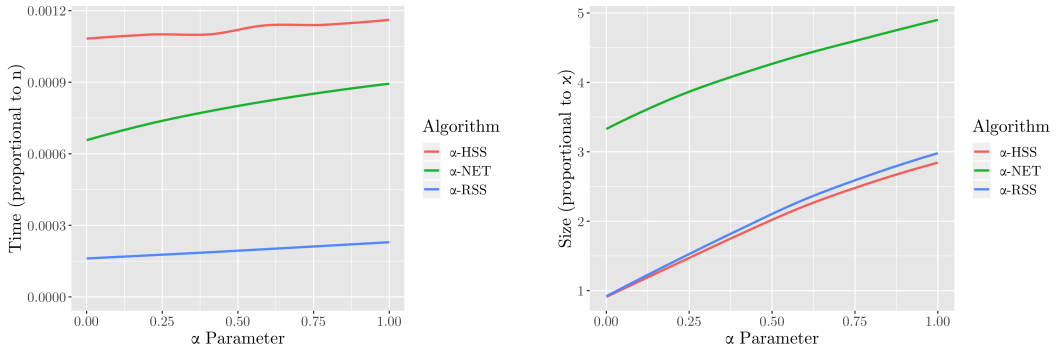
These experimental trials compare the performance of different condensation algorithms when applied to vastly different training sets. We use two measures of comparison on these algorithms: their runtime in the different training sets, and the size of the subset selected. Clearly, these values might differ greatly on training sets whose size are too distinct. Therefore, before comparing the raw results, these are normalized. The runtime of an algorithm for a given training set is normalized by dividing it by n, the size of the training set. The size of the selected subset is normalized by dividing it by κ, the number of nearest-enemy points in the training set, which characterizes the complexity of the boundaries between classes.

Algorithm Comparison. The first experiment evaluates the performance of the three algorithms proposed in this paper: α-RSS, α-HSS, and α-NET. The evaluation is carried out by varying the value of the α parameter from 0 to 1, to understand the impact of increasing

[2] `https://archive.ics.uci.edu/ml/index.php`
[3] `https://github.com/afloresv/nnc/`

this parameter. The implementation of α-HSS uses the well-known greedy algorithm for set cover [12], and solves the problem using the reduction described in Section 3.1. In the other hand, recall that the original NET algorithm (for $\alpha = 0$) implements an extra pruning technique to further reduce the training set after computing the γ-net. To do a fair comparison between the techniques, we implemented the α-NET algorithm with a modified version of this pruning techinque that guarantees that the selected subset is still α-selective.

The results show that α-RSS outperforms the other algorithms in terms of running time by a big margin, and irrespective of the value of α (see Figure 6a). Additionally, the number of points selected by α-RSS is comparable to α-HSS, which guarantees the best possible approximation factor in general metrics, while α-NET is significantly outperformed.

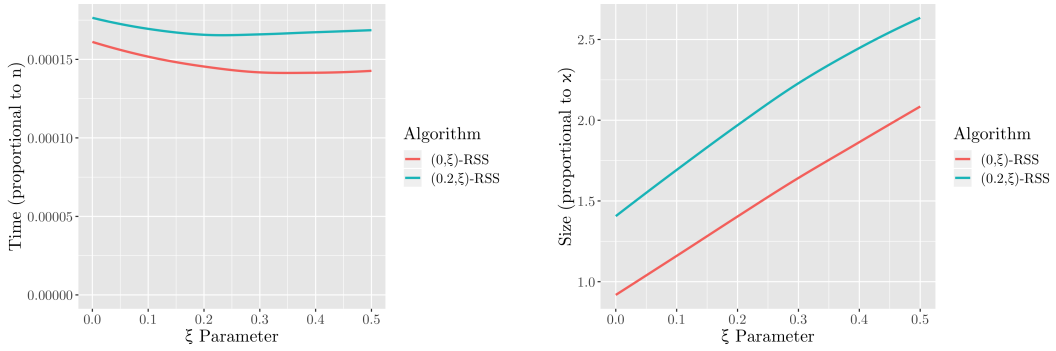


(a) Running time.

(b) Size of the selected subsets.

Figure 6 Comparison α-RSS, α-NET, and α-HSS, for different values of α.

Subquadratic Approach. Using the same experimental framework, we evaluate performance of the subquadratic implementation (α, ξ)-RSS described in Section 3.3. In this case, we change the value of parameter ξ to assess its effect on the running time and selection size over the algorithm, for two different values of α (see Figure 7). The results show an expected increase of the number of selected points, while significantly improving its running time.



(a) Running time.

(b) Size of the selected subsets.

Figure 7 Evaluating the effect of increasing the parameter ξ on (α, ξ)-RSS for $\alpha = \{0, 0.2\}$.

Table 1 Training sets used to evaluate the performance of condensation algorithms. Indicates the number of points n, dimensions d, classes c, nearest-enemy points κ (also in percentage *w.r.t.* n).

Training set	n	d	c	κ (%)
banana	5300	2	2	811 (15.30%)
cleveland	297	13	5	125 (42.09%)
glass	214	9	6	87 (40.65%)
iris	150	4	3	20 (13.33%)
iris2d	150	2	3	13 (8.67%)
letter	20000	16	26	6100 (30.50%)
magic	19020	10	2	5191 (27.29%)
monk	432	6	2	300 (69.44%)
optdigits	5620	64	10	1245 (22.15%)
pageblocks	5472	10	5	429 (7.84%)
penbased	10992	16	10	1352 (12.30%)
pima	768	8	2	293 (38.15%)
ring	7400	20	2	2369 (32.01%)
satimage	6435	36	6	1167 (18.14%)
segmentation	2100	19	7	398 (18.95%)
shuttle	58000	9	7	920 (1.59%)
thyroid	7200	21	3	779 (10.82%)
twonorm	7400	20	2	1298 (17.54%)
wdbc	569	30	2	123 (21.62%)
wine	178	13	3	37 (20.79%)
wisconsin	683	9	2	35 (5.12%)
v-100000-2-3-15	100000	2	3	1909 (1.90%)
v-100000-2-3-5	100000	2	3	788 (0.78%)
v-100000-3-3-15	100000	3	3	7043 (7.04%)
v-100000-3-3-5	100000	3	3	3738 (3.73%)
v-100000-4-3-15	100000	4	3	13027 (13.02%)
v-100000-4-3-5	100000	4	3	10826 (10.82%)
v-100000-5-3-15	100000	5	3	22255 (22.25%)
v-100000-5-3-5	100000	5	3	17705 (17.70%)

References

1 Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

2 Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. *arXiv preprint*, 2018. `arXiv:1806.09823`.

3 Fabrizio Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.

4 Sunil Arya, Guilherme D Da Fonseca, and David M Mount. Approximate polytope membership queries. *SIAM Journal on Computing*, 47(1):1–51, 2018.

5 Sunil Arya, Theocharis Malamatos, and David M Mount. Space-time tradeoffs for approximate nearest neighbor searching. *Journal of the ACM (JACM)*, 57(1):1, 2009.

6 Franz Aurenhammer and Herbert Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition*, 17(2):251–257, 1984.

7 Ricardo Barandela, Francesc J Ferri, and J Salvador Sánchez. Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(06):787–806, 2005.

8 Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. *arXiv preprint*, 2018. `arXiv:1804.05345`.

9 Jon Louis Bentley and James B Saxe. Decomposable searching problems I. Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

10 Ahmad Biniaz, Sergio Cabello, Paz Carmi, Jean-Lou De Carufel, Anil Maheshwari, Saeed Mehrabi, and Michiel Smid. On the minimum consistent subset problem. In *WADS*, 2019.

11 Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *arXiv preprint*, 2016. arXiv:1612.00889.

12 Václav Chvatal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 1979.

13 Richard Cole and Lee-Ad Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 574–583, 2006.

14 Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 1967.

15 Luc Devroye. On the inequality of cover and hart in nearest neighbor discrimination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1:75–78, 1981.

16 Uriel Feige. A threshold of ln n for approximating set cover. *JACM*, 1998.

17 Dan Feldman. Core-sets: Updated survey. In *Sampling Techniques for Supervised or Unsupervised Tasks*, pages 23–44. Springer, 2020.

18 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578, 2011.

19 Evelyn Fix and Joseph L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine*, Technical Report 4(3):477+, January 1951.

20 Alejandro Flores-Velazco and David M. Mount. Guarantees on nearest-neighbor condensation heuristics. In Zachary Friggstad and Jean-Lou De Carufel, editors, *Proceedings of the 31st Canadian Conference on Computational Geometry, CCCG 2019, August 8-10, 2019, University of Alberta, Edmonton, Alberta, Canada*, pages 87–93, 2019.

21 Salvador Garcia, Joaquin Derrac, Jose Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE TPAMI*, 2012.

22 Lee-Ad Gottlieb, Aryeh Kontorovich, and Pinhas Nisnevitch. Near-optimal sample compression for nearest neighbors. In *Advances in Neural Information Processing Systems*, 2014.

23 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004.

24 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

25 Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE Trans. Inf. Theor.*, 1968.

26 Juha Heinonen. *Lectures on analysis on metric spaces*. Springer Science & Business Media, 2012.

27 Norbert Jankowski and Marek Grochowski. Comparison of instances selection algorithms I. Algorithms survey. In *Artificial Intelligence and Soft Computing-ICAISC*. Springer, 2004.

28 Kamyar Khodamoradi, Ramesh Krishnamurti, and Bodhayan Roy. Consistent subset problem with two labels. In *Conference on Algorithms and Discrete Applied Mathematics*, 2018.

29 Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. *arXiv preprint*, 2019. arXiv:1911.07412.

30 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

31 David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. Chromatic nearest neighbor searching: A query sensitive approach. *Computational Geometry*, 2000.

32 Azaria Paz and Shlomo Moran. Non deterministic polynomial optimization problems and their approximations. *Theoretical Computer Science*, 15(3):251–277, 1981.

33 Jeff M. Phillips. Coresets and sketches, 2016. arXiv:1601.00617.

34 G. L. Ritter, H. B. Woodruff, S. R. Lowry, and T. L. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 1975.

35 Petr Slavík. A tight analysis of the greedy algorithm for set cover. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*, STOC, 1996.

36 Charles J Stone. Consistent nonparametric regression. *The annals of statistics*, pages 595–620, 1977.

37 Godfried Toussaint. Open problems in geometric methods for instance-based learning. In *JCDCG*, volume 2866 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/978-3-540-44400-8_29.

38 Murad Tukan, Cenk Baykal, Dan Feldman, and Daniela Rus. On coresets for support vector machines. *arXiv preprint*, 2020. arXiv:2002.06469.

39 Gordon Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SoCG, pages 224–233, New York, NY, USA, 1991. ACM.

40 Anastasiya V. Zukhba. NP-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recog. Image Anal.*, 20(4):484–494, 2010.

Kernelization of Whitney Switches

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway
Fedor.Fomin@uib.no

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
Petr.Golovach@uib.no

Abstract

A fundamental theorem of Whitney from 1933 asserts that 2-connected graphs G and H are 2-isomorphic, or equivalently, their cycle matroids are isomorphic, if and only if G can be transformed into H by a series of operations called Whitney switches. In this paper we consider the quantitative question arising from Whitney's theorem: Given 2-isomorphic graphs, can we transform one into another by applying at most k Whitney switches? This problem is already NP-complete for cycles, and we investigate its parameterized complexity. We show that the problem admits a kernel of size $\mathcal{O}(k)$, and thus, is fixed-parameter tractable when parameterized by k.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Whitney switch, 2-isomorphism, Parameterized Complexity, kernelization

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.48

Related Version The full version of the paper is available at `https://arxiv.org/abs/2006.13684`.

Funding The research leading to these results has received funding from the Research Council of Norway via the project "MULTIVAL" (grant no. 263317).

Acknowledgements We are grateful to Erlend Raa Vågset for fruitful discussions that initiated the research resulted in the paper.

1 Introduction

A fundamental result of Whitney from 1933 [35], asserts that every 2-connected graph is completely characterized, up to a series of Whitney switches (also known as 2-switches), by its edge set and cycles. This theorem is one of the cornerstones of Matroid Theory, since it provides an exact characterization of two graphs having isomorphic cycle matroids [32]. In graph drawing and graph embeddings, this theorem (applied to dual graphs) is used to characterize all drawings of a planar graph [8].

A *Whitney switch* is an operation, that from a 2-connected graph G, constructs graph G' as follows. Let $\{u, v\}$ be two vertices of G, whose removal separates G into two disjoint subgraphs G_1 and G_2. The graph G' is obtained by flipping the neighbors of u and v in the set of vertices of G_2. In other words, for every vertex $w \in V(G_2)$, if w was adjacent to u in G, in graph G' edge uw is replaced by vw. Similarly, if w was adjacent to v in G, we replace vw by uw. See Figure 1 for an example.

If we view the graph G as a graph with labelled edges, then a Whitney switch transforms G into a graph G' with the same set of labelled edges, however graphs G and G' are not necessarily isomorphic. On the other hand, graphs G and G' have the same set of cycles in the following sense: a set of (labelled) edges forms a cycle in G if and only if it forms a cycle in G'. (In other words, the cycle matroids of G and G' are isomorphic.) Whitney's

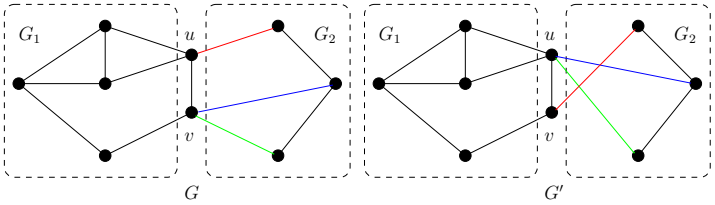


Figure 1 G' is obtained from G by the Whitney switch with respect to the partition of $G - \{u, v\}$ into G_1 and G_2.

theorem claims that the opposite is also true: if there is a cycle-preserving mapping between graphs G and G' then one graph can be transformed into another by a sequence of Whitney switches. To state the theorem of Whitney more precisely, we need to define 2-isomorphisms.

We say that 2-connected graphs G and H are 2-*isomorphic* if there is a bijection $\varphi\colon E(G) \to E(H)$ such that φ and φ^{-1} preserve cycles, that is, for every cycle C of G, C is mapped to a cycle of H by φ and, symmetrically, every cycle of H is mapped to a cycle of G by φ^{-1}. We refer to φ as to 2-*isomorphism* from G to H. An isomorphism $\psi\colon V(G) \to V(H)$ is a φ-*isomorphism* if for every edge $uv \in E(G)$, $\varphi(uv) = \psi(u)\psi(v)$, and G and H are φ-isomorphic if there is an isomorphism G to H that is a φ-isomorphism. Let us note that if G is 3-connected and 2-isomorphic to H under φ then G and H are φ-isomorphic [29, Lemma 1]. But for 2-connected graphs this is not true. For example, the graphs in Fig. 1 are not isomorphic but are 2-isomorphic. Moreover, even isomorphic graphs with 2-isomorphism φ not always have a φ-isomorphism. For example, for the 2-isomorphism φ in Fig. 2 mapping a cycle G into another cycle H (we view these cycles as labelled graphs), there is no φ-isomorphism. (For every φ-isomorphism edges $\varphi(a)$ and $\varphi(b)$ should have an endpoint in common.) On the other hand, graph G' obtained from G by Whitney switch (for vertices u and v) is φ-isomorphic to H.

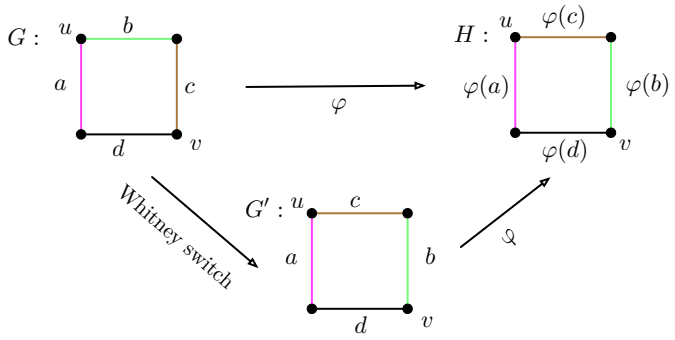


Figure 2 Graph G is not φ-isomorphic to H but its Whitney switch G' is.

▶ **Theorem 1** (Whitney's theorem [35]). *If there is a* 2*-isomorphism* φ *from graph* G *to graph* H*, then* G *can be transformed by a sequence of Whitney switches to a graph* G' *which is* φ*-isomorphic to* H*.*

However, Whitney's theorem does not provide an answer to the following question: Given a 2-isomorphism φ from graph G to graph H, *what is the minimum number of Whitney switches required to transform* G *to a graph* φ*-isomorphic to* H*?* Truemper in [29] proved that $n - 2$ switches always suffices, where n is the number of vertices in G. He also proved

that this upper bound it tight, that is, there are graphs G and H for which $n-2$ switches are necessary. In this paper we study the algorithmic complexity of the following problem about Whitney switches.

WHITNEY SWITCHES

Input:	2-Isomorphic n-vertex graphs G and H with a 2-isomorphism $\varphi\colon E(G) \to E(H)$, and a nonnegative integer k.
Task:	Decide whether it is possible to obtain from G a graph G' that φ-isomorphic to H by at most k Whitney switches.

The departure point for our study is an easy reduction (Theorem 4) from SORTING BY REVERSALS that establishes NP-completeness of WHITNEY SWITCHES even when input graphs G and H are cycles. Our main algorithmic result is the following theorem (we postpone the definition of a kernel till Section 2). Informally, it means that the instance of the problem can be compressed in polynomial time to an equivalent instance with two graphs on $\mathcal{O}(k)$ vertices. It also implies that WHITNEY SWITCHES is fixed-parameter tractable parameterized by k.

▶ **Theorem 2.** *WHITNEY SWITCHES admits a kernel with $\mathcal{O}(k)$ vertices and is solvable in $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ time.*

While Theorem 2 is not restricted to planar graphs, pipelined with the well-known connection of planar embeddings and Whitney switches, it can be used to obtain interesting algorithmic consequences about distance between planar embeddings of a graph. Recall that graphs G and G^* are called abstractly dual if there is a bijection $\pi\colon E(G) \to E(G^*)$ such that edge set $E \subseteq E(G)$ forms a cycle in G if and only if $\pi(E)$ is a minimal edge-cut in G^*. By another classical theorem of Whitney [34], a graph G has a dual graph if and only if G is planar. Moreover, an embedding of a planar graph into a sphere is uniquely defined by the planar graph G and edges of the faces, or equivalently, its dual graph G^*. While every 3-connected planar graph has a unique embedding into the sphere, a 2-connected graph can have several non-equivalent embeddings, and hence several non-isomorphic dual graphs. If G_1^* and G_2^* are dual graphs of graph G, then G_1^* is 2-isomorphic to G_2^*. Then by Theorem 1, by a sequence of Whitney switches G_1^* can be transformed into G_2^*, or equivalently, the embedding of G corresponding to G_1^* can be transformed to embedding of G corresponding to G_2^*. We refer to the survey of Carsten Thomassen [28, Section 2.2] for more details. By Theorem 2, we have that given two planar embeddings of a (labelled) 2-connected graph G, deciding whether one embedding can be transformed into another by making use of at most k Whitney switches, admits a kernel of size $\mathcal{O}(k)$ and is fixed-parameter tractable.

Related work. Whitney's theorem had a strong impact on the development of modern graph and matroid theories. While the original proof is long, a number of simpler proofs are known in the literature. In particular, the work of Truemper in [29], whose proof of Whitney's theorem is based on applications of Tutte decomposition [30, 31]. This is also the approach we adapt in our work.

The well-studied problem similar in spirit to WHITNEY SWITCHES is the problem of computing the flip distance for triangulations of a set of points. The parameterized complexity of this problem was studied in [10, 23]. As we already have mentioned, WHITNEY SWITCHES for planar graphs is equivalent to the problem of computing the Whitney switch distance between planar embeddings. We refer to the survey of Bose and Hurtado [4] for the discussion of the relations between geometric and graph variants. The problem is known to be NP-complete [22] and FPT parameterized by the number of flips [19]. For the special case when

the set of points defines a convex polygon, the problem of computing the flip distance between triangulations is equivalent to computing the rotation distance between two binary trees. For that case linear kernels are known [10, 23] but for the general case the existence of a polynomial kernel is open.

Whitney Switches also can be seen as a reconfiguration problem, study of reconfiguration problems is a popular trend in parameterized complexity, see e.g. [24, 21].

Overview of the proof of Theorem 2. The main tool in the construction of the kernel is the classical Tutte decompositions [30, 31] We postpone the formal definition till Section 2. Informally, the Tutte decomposition of a 2-connected graph represents the vertex separators of size two in a tree-like structure. Each node of this tree represents a part of the graph (or *bag*) that is either a 3-connected graph or a cycle, and each edge corresponds to a separator of size two. Then a 2-isomorphism of G and H allows to establish an isomorphism of the trees representing the Tutte decompositions of the input graphs. After that, potential Whitney switches can be divided into two types: the switches with respect to separators corresponding to the edges of the trees and the switches with respect to separators formed by nonadjacent vertices of a cycle-bag. The switches of the first type are relatively easy to analyze and we can identify necessary switches of this type. The "troublemakers" that make the problem hard are switches of the second type. To deal with them, we use the structural results about sorting of permutations by reversals of Hannenhalli and Pevzner [17] adapted for our purposes. This allows us to identify a set of vertices of size $\mathcal{O}(k)$ that potentially can be used for Whitney switches transforming G to H. Given such a set of crucial vertices, we simplify the structure of the input graphs and then reduce their size.

Organization of the paper. In Section 2, we give basic definitions. In Section 3, we discuss the Sorting by Reversals problem for permutations that is closely related to Whitney Switches. Section 4 contains structural results used by our kernelization algorithm, and in Section 5, we give the algorithm itself. We conclude in Section 6 by discussing further directions of research. Due to space constraints, the proofs are either omitted or just sketched. The details are given in the full version of the paper [15].

2 Preliminaries

Graphs. All graphs considered in this paper are finite undirected graphs without loops or multiple edges, unless it is specified explicitly that we consider directed graphs (in Section 6 we deal with tournaments). We follow the standard graph theoretic notation and terminology (see, e.g., [13]). For each of the graph problems considered in this paper, we let $n = |V(G)|$ and $m = |E(G)|$ denote the number of vertices and edges, respectively, of the input graph G if it does not create confusion. A pair (A, B), where $A, B \subseteq V(G)$, is a *separation* of G if $A \cup B = V(G)$, $A \setminus B \neq \emptyset$, $B \setminus A \neq \emptyset$ and G has no edge uv with $u \in A \setminus B$ and $v \in B \setminus A$; $|A \cap B|$ is the *order* of the separation. If the order is 2, then we say that (A, B) is a *Whitney separation*. A set $S \subseteq V(G)$ is a *separator* of G if there is a separation (A, B) of G with $S = A \cap B$.

Whitney switches. It is convenient to define Whitney switches using separations. Let G be a 2-connected graph. Let also (A, B) be a Whitney separation of G with $A \cap B = \{u, v\}$. The *Whitney switch* operation with respect to (A, B) transforms G as follows: take $G[A]$ and $G[B]$ and identify the vertex u of $G[A]$ with the vertex v of $G[B]$ and, symmetrically, v of $G[A]$ with

u of $G[B]$; if u and v are adjacent in G, then the edges uv of $G[A]$ and $G[B]$ are identified as well. Let G' be the obtained graph. We define the mapping $\sigma_{(A,B)}\colon E(G) \to E(G')$ that maps the edges of $G[A]$ and $G[B]$, respectively, to themselves. It is easy to see that $\sigma_{(A,B)}$ is a 2-isomorphism of G to G'. Therefore, if φ is a 2-isomorphism of G to a graph H, then $\varphi \circ \sigma^{-1}_{(A,B)}$ is a 2-isomorphism of G' to H. To simplify notation, we assume, if it does not create confusion, that the sets of edges of G and G' are identical and we only change incidences by switching. In particular, under this assumption, we have that $\varphi \circ \sigma^{-1}_{(A,B)} = \varphi$. We also assume that the graphs G and G' have the same sets of vertices.

Tutte decomposition. Our kernelization algorithm for Whitney Switches is based on the classical result of Tutte [30, 31] about decomposing of 2-connected graphs via separators of size two. Following Courcelle [11], we define Tutte decompositions in the terms of tree decompositions.

A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$, where T is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$, called a bag, such that the following three conditions hold:

(T1) $\bigcup_{t\in V(T)} X_t = V(G)$, that is, every vertex of G is in at least one bag,

(T2) for every $uv \in E(G)$, there exists a node t of T such that the bag X_t contains both u and v,

(T3) for every $v \in V(G)$, the set $T_v = \{t \in V(T) \mid v \in X_t\}$, i.e., the set of nodes whose corresponding bags contain v, induces a connected subtree of T.

To distinguish between the vertices of the decomposition tree T and the vertices of the graph G, we will refer to the vertices of T as *nodes*.

Let $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$ be a tree decomposition of G. The *torso* of X_t for $t \in V(T)$ is the graph obtained from $G[X_t]$ by additionally making adjacent every two distinct vertices $u, v \in X_t$ such that there is $t' \in V(T)$ adjacent to t with $u, v \in X_t \cap X_{t'}$. For adjacent $t, t' \in V(T)$, $X_t \cap X_{t'}$ is the *adhesion set* of the bags X_t and $X_{t'}$ and $|X_t \cap X_{t'}|$ is the *adhesion* of the bags. The maximum adhesion of adjacent bags is called the *adhesion* of the tree decomposition.

Let G be a 2-connected graph. A tree decomposition $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$ is said to be a *Tutte decomposition* if $\mathcal{T}$ is a tree decomposition of adhesion 2 such that there is a partition $(W_2, W_{\geq 3})$ of $V(T)$ such that the following holds:

(T4) $|X_t| = 2$ for $t \in W_2$ and $|X_t| \geq 3$ for $t \in W_{\geq 3}$,

(T5) the torso of X_t is either a 3-connected graph or a cycle for every $t \in W_{\geq 3}$,

(T6) for every $t \in W_2$, $d_T(t) \geq 2$ and $t' \in W_{\geq 3}$ for each neighbor t' of t,

(T7) for every $t \in W_{\geq 3}$, $t' \in W_2$ for each neighbor t' of t,

(T8) if $t \in W_2$ and $d_T(t) = 2$, then for the neighbors t' and t'' of t, either the torso of t' or the torso of t'' is a 3-connected graph or the vertices of X_t are adjacent in G.

Notice that the bags X_t for $t \in W_2$ are distinct separators of G of size two, and $X_t \subseteq X_{t'}$ for $t \in W_2$ and $t' \in N_T(t)$. Observe also that if $\{u, v\}$ is a separator of G of size two, then either $\{u, v\} = X_t$ for some $t \in W_2$ or $u, v \in X_t$ for $t \in W_{\geq 3}$ such that the torso of X_t is a cycle and u and v are nonadjacent vertices of the torso.

Combining the results of Tutte [30, 31] and of Hopcroft and Tarjan [18], we state the following proposition.

▶ **Proposition 3** ([30, 31, 18]). *A 2-connected graph G has a unique Tutte decomposition that can be constructed in linear time.*

Parameterized Complexity and Kernelization. We refer to the books [12, 14, 16] for the detailed introduction to the field. Here we only give the most basic definitions. In the Parameterized Complexity theorey, the computational complexity is measured as a function of the input size n of a problem and an integer *parameter* k associated with the input. A parameterized problem is said to be *fixed parameter tractable* (or FPT) if it can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some function f. A *kernelization* algorithm for a parameterized problem Π is a polynomial algorithm that maps each instance (I, k) of Π to an instance (I', k') of Π such that (i) (I, k) is a yes-instance of Π if and only if (I', k') is a yes-instance of Π, and (ii) $|I'| + k'$ is bounded by $f(k)$ for a computable function f. Respectively, (I', k') is a *kernel* and f is its *size*. A kernel is *polynomial* if f is polynomial. It is common to present a kernelization algorithm as a series of *reduction rules*. A reduction rule for a parameterized problem is an algorithm that takes an instance of the problem and computes in polynomial time another instance that is more "simple" in a certain way. A reduction rule is *safe* if the computed instance is equivalent to the input instance.

3 Sorting by reversals

Sorting by reversals is the classical problem with many applications including bioinformatics. We refer to the book of Pevzner [25] for the detailed survey of results and applications of this problem. This problem is also strongly related to Whitney Switches– solving the problem for two cycles is basically the same as sorting circular permutations by reversals. First we use this relation to observe the NP-completeness. But we also need to establish some structural properties of sorting by reversals which will be used in kernelization algorithm.

Let $\pi = (\pi_1, \ldots, \pi_n)$ be a permutation of $\{1, \ldots, n\}$, that is, a bijective mapping of $\{1, \ldots, n\}$ to itself. Throughout this section, all considered permutations are permutations of $\{1, \ldots, n\}$. For $1 \leq i \leq j \leq n$, the *reversal* $\rho(i, j)$ reverse the order of elements $\pi_i, \ldots, \pi_j$ and transforms π into

$$\rho(i, j) \circ \pi = (\pi_1, \ldots, \pi_{i-1}, \pi_j, \pi_{j-1}, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi_n).$$

The *reversal distance* $d(\pi, \sigma)$ between two permutations π and σ is the minimum number of reversals needed to transform π to σ. For a permutation π, $d(\pi) = d(\pi, \iota)$, where ι is the identity permutation; note that $d(\pi, \sigma) = d(\sigma^{-1} \circ \pi, \iota)$ and this means that computing the reversal distance can be reduced to sorting a permutation by the minimum number of reversals.

These definitions can be extended for circular permutations (further, we may refer to usual permutations as *linear* to avoid confusion). We say that $\pi^c = (\pi_1, \ldots, \pi_n)$ is a *circular* permutation if π^c is the class of the permutations that can be obtained from the linear permutation $(\pi_1, \ldots, \pi_n)$ by *rotations* and *reflections*, that is, all the permutations

$$(\pi_1, \ldots, \pi_n), (\pi_n, \pi_1, \ldots, \pi_{n-1}), \ldots, (\pi_2, \ldots, \pi_n, \pi_1),$$

and

$$(\pi_n, \ldots, \pi_1), (\pi_1, \pi_n, \ldots, \pi_2), \ldots, (\pi_{n-1}, \ldots, \pi_1, \pi_n)$$

composing one class are identified, meaning that we do not distinguish them when discussing circular permutations. The *circular reversals* $\rho^c(i, j)$ and *circular reversal distance* $d^c(\pi^c, \sigma^c)$ and $d^c(\pi^c)$ are defined in the same way as for linear permutations.

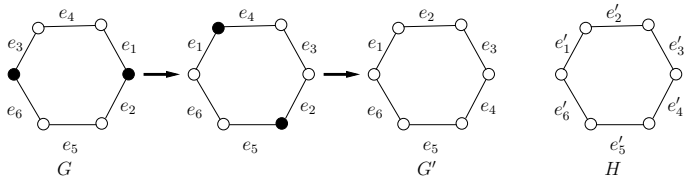


Figure 3 The construction of G' that is φ-isomorphic to H by the Whitney switches corresponding to the sorting by reversals $(3,4,1,2,5,6) \to (1,4,3,2,5,6) \to (1,2,3,4,5,6)$; $\varphi(e_i) = e'_i$ for $i \in \{1,\ldots,6\}$, the vertices of the separators for the switches are shown in black.

To see the connection between Whitney switches and circular reversals of permutations, consider a cycle G with the vertices $v_1,\ldots,v_n$ for $n \geq 4$ taken in the cycle order and the edges $e_i = v_{i-1}v_i$ for $i \in \{1,\ldots,r\}$ assuming that $v_0 = v_n$. Let $1 \leq i < j \leq n$ be such that v_i and v_j are not adjacent. Then the Whitney switch with respect to (A,B), where $A = \{v_1,\ldots,v_i\} \cup \{v_j,\ldots,v_n\}$ and $B = \{v_i,\ldots,v_j\}$ is equivalent to applying the reversal $\rho^c(i+1,j)$ to the circular permutation $(e_1,\ldots,e_n)$ of the edges of G. Moreover, let H be a cycle with n vertices and denote by $e'_1,\ldots,e'_n$ its edges in the cycle order. Notice that every bijection $\varphi\colon E(G) \to E(H)$ is a 2-isomorphism of G to H, and G and H are φ-isomorphic if and only if the circular permutation $\pi^c = (\varphi^{-1}(e'_1),\ldots,\varphi^{-1}(e'_n))$ is the same as $\sigma^c = (e_1,\ldots,e_n)$. Clearly, we can assume that π^c is a permutation of $\{1,\ldots,n\}$ and σ^c is the identity permutation. Then G can be transformed to a graph G' φ-isomorphic to H by at most k Whitney switches if and only if $d^c(\pi^c) \leq k$. An example is shown in Fig. 3.

In particular, the above observation implies the hardness of Whitney Switches, because the computing of the reversal distances is known to be NP-hard. For linear permutations, this was shown by Caprara in [7]. Then Solomon, Sutcliffe, and Lister [27] proved that it is NP-complete to decide, given a circular permutation π^c and a nonnegative integer k, whether $d^c(\pi^c) \leq k$. This brings us to the following result.

▶ **Theorem 4.** *Whitney Switches is NP-complete even when restricted to cycles.*

For our kernelization algorithm, we need some further structural results about reversals in an optimal sorting sequence.

Let $\pi = (\pi_1,\ldots,\pi_n)$ be a linear permutation. For $1 \leq i \leq j \leq n$, we say that $(\pi_i,\ldots,\pi_j)$ is an *interval* of π. An interval $(\pi_i,\ldots,\pi_j)$ is called a *block* if either $i = j$ or $i < j$ and for every $h \in \{i+1,\ldots,j\}$, $|\pi_{h-1} - \pi_h| = 1$, that is, a block is formed by consecutive integers in π in either the ascending or descending order. An inclusion maximal block is called a *strip*. In other words, a strip is an inclusion maximal interval that has no *breakpoint*, that is, a pair of elements π_{h-1},π_h with $|\pi_{h-1} - \pi_h| \geq 2$. It is said that a reversal $\rho(p,q)$ *cuts* a strip $(\pi_i,\ldots,\pi_j)$ if either $i < p \leq j$ or $i \leq q < j$, that is, the reversals separates elements that are consecutive in the identity permutation.

It is known that there are cases when every optimal sorting by reversal requires a reversal that cuts a strip. For example, as was pointed by Hannenhalli and Pevzner in [17], the permutation $(3,4,1,2)$ requires three reversals that do not cut strips, but the sorting can be done by two reversals: [1]

$$(3,4,1,2) \to (1,4,3,2) \to (1,2,3,4).$$

[1] This example can be extended for circular permutations: $(3,4,1,2,5,6) \to (1,4,3,2,5,6) \to (1,2,3,4,5,6)$.

However, it was conjectured by Kececioglu and Sankoff [20] that there is an optimal sorting that does not cut strips other than at their first or last elements. This conjecture was proved by Hannenhalli and Pevzner in [17]. More precisely, they proved that there is an optimal sorting that does not cut strips of length at least three.

It is common for bioinformatics applications, to consider *signed permutations* (see, e.g., [25]). In a signed permutation $\overrightarrow{\pi} = (\pi_1, \ldots, \pi_n)$, each element π_i has its *sign* "$-$" or "$+$". Then for $i, j \in \{1, \ldots, n\}$, the reversal reverse the sign of each element $\pi_i, \ldots, \pi_j$ besides reversing their order. We generalize this notion and define *partially signed circular permutations*, where each element has one of the sings: "$-$", "$+$" or "no sign". Formally, a *partially signed circular permutation* $\overrightarrow{\pi}^c = (\langle \pi_1, s_1 \rangle, \ldots, \langle \pi_n, s_n \rangle)$, where $(\pi_1, \ldots, \pi_n)$ is a linear permutation and $s_i \in \{-1, +1, 0\}$ for $i \in \{1, \ldots, m\}$, as the class of the linear permutations that can be obtained from $(\langle \pi_1, s_1 \rangle, \ldots, \langle \pi_n, s_n \rangle)$ by rotations and reflections such that every reflection reverse signs. For $i, j \in \{1, \ldots, n\}$, the reversal

$$\overrightarrow{\rho}^c(i,j) \circ \overrightarrow{\pi}^c = (\langle \pi_1, s_1 \rangle, \ldots, \langle \pi_{i-1}, s_{i-1} \rangle, \langle \pi_j, -s_j \rangle, \ldots, \langle \pi_i, -s_i \rangle, \langle \pi_{j+1}, s_{j+1} \rangle \ldots, \langle \pi_n, s_n \rangle)$$

if $i \leq j$, and

$$\overrightarrow{\rho}^c(i,j) \circ \overrightarrow{\pi}^c = (\langle \pi_n, -s_n \rangle, \ldots, \langle \pi_i, -s_i \rangle, \langle \pi_{j+1}, s_{j+1} \rangle, \ldots, \langle \pi_{i-1}, s_{i-1} \rangle, \langle \pi_j, -s_j \rangle \ldots, \langle \pi_1, -s_1 \rangle)$$

otherwise.

We say that $\overrightarrow{\pi}^c$ is *signed* if each s_i is either -1 or $+1$ and the *signed circular identity permutation* is $\overrightarrow{\iota}^c = (\langle 1, +1 \rangle, \ldots, \langle n, +1 \rangle)$. Also a partially signed circular permutation $\overrightarrow{\pi}^c = (\langle \pi_1, s_1 \rangle, \ldots, \langle \pi_n, s_n \rangle)$ *agrees in signs* with a signed circular permutation $\overrightarrow{\pi}'^c = (\langle \pi_1, s'_1 \rangle, \ldots, \langle \pi_n, s'_n \rangle)$ if $s_i = s'_i$ for $i \in \{1, \ldots, n\}$ such that $s_i \neq 0$, that is, the zero signs are replaced by either -1 or $+1$ in the signed permutation, and $\Sigma(\overrightarrow{\pi}^c)$ is used to denote the set of all signed circular permutations $\overrightarrow{\pi}'^c$ that agree in signs with $\overrightarrow{\pi}^c$. Then *reversal distance* $\overrightarrow{d}^c(\overrightarrow{\pi}^c, \sigma^c)$, where $\overrightarrow{\sigma}^c$ is a signed circular permutation, is the minimum number or reversal needed to obtain from $\overrightarrow{\pi}^c$ a partially signed circular permutation $\overrightarrow{\pi}'^c$ that agrees in signs with $\overrightarrow{\sigma}^c$, and $\overrightarrow{d}^c(\overrightarrow{\pi}^c) = \overrightarrow{d}^c(\overrightarrow{\pi}^c, \overrightarrow{\iota}^c)$. A a sequence of reversals of minimum length that result in a partially signed circular permutation that agrees in signs with $\overrightarrow{\iota}^c$ is an *optimal sorting* sequence.

Let $\overrightarrow{\pi}^c = (\langle \pi_1, s_1 \rangle, \ldots, \langle \pi_n, s_n \rangle)$ be a partially signed circular permutation. For $1 \leq i \leq j \leq n$, we say that $(\langle \pi_i, s_i \rangle, \ldots, \langle \pi_j, s_j \rangle)$ and $(\langle \pi_{j+1}, s_{j+1} \rangle, \ldots, \langle \pi_n, s_n \rangle, \langle \pi_1, s_1 \rangle, \ldots, \langle \pi_i, s_i \rangle)$ are *intervals* of $\overrightarrow{\pi}^c$. An interval is a *signed block* if it either has size one or for every two consecutive elements $\langle \pi_{i-1}, s_{i-1} \rangle, \langle \pi_i, s_i \rangle$, $|\pi_{i-1} - \pi_i| \leq 1$ and, moreover, if the elements of the interval are in the increasing order, then all the signs $s_i \in \{0, +1\}$, and if they are in the the decreasing order, then all the signs $s_i \in \{0, -1\}$. A *signed strip* is an inclusion maximal signed block. A reversal $\overrightarrow{\rho}^c(p, q)$ *cuts* an interval if the reversed part includes at least one element of the interval and excludes at least one element of the interval. We use the result of Hannenhalli and Pevzner [17] to show the following lemma.

▶ **Lemma 5.** *For a signed circular permutation $\overrightarrow{\pi}^c$, there is an optimal sorting sequence such that no reversal in the sequence cuts the interval formed by a signed strip of $\overrightarrow{\pi}^c$ of length at least* 5*.*

Notice that we do not claim that no reversal cuts a strip of length at least 5 that is obtained by performing the previous reversals; only the long strips of the initial permutation $\overrightarrow{\pi}^c$ are not cut by any reversal in the sorting sequence.

4 Tutte decomposition and 2-isomorphisms

In this section we provide a number of auxiliary results about 2-isomorphisms and Tutte decompositions.

We need the following folklore observation about φ-isomorphisms that we prove for completeness. For this, we extend φ on sets of edges in standard way, that is, $\varphi(A) = \{\varphi(e) \mid e \in A\}$ and $\varphi(\emptyset) = \emptyset$.

▶ **Lemma 6.** *Let G and H be n-vertex 2-connected 2-isomorphic graphs with a 2-isomorphism φ. Then G and H are φ-isomorphic if and only if there is a bijective mapping $\psi\colon V(G) \to V(H)$ such that for every $v \in V(G)$, $\varphi(E_G(v)) = E_H(\psi(v))$. Moreover, G and H are φ-isomorphic if and only if φ bijectively maps the family of the sets of edges $\{E_G(v) \mid v \in V(G)\}$ to the family $\{E_H(v) \mid v \in V(H)\}$, and this property can be checked in polynomial time.*

By Lemma 6, we can restate the task of Whitney Switches and ask whether it is possible to obtain a graph G' by performing at most k Whitney switches starting from G with the property that the extension of φ to the family of sets $\{E_{G'}(v) \mid v \in V(G')\}$ bijectively maps this family to $\{E_H(v) \mid v \in V(H)\}$.

We use Whitney's theorem [35](see also [29]).

▶ **Proposition 7** ([35]). *Let G and H be n-vertex graphs and let φ be a 2-isomorphism of G to H. Then there is a finite sequence of Whitney switches such that the graph G' obtained from G by these switches is φ-isomorphic to H.*

We also use the property of 3-connected graphs explicitly given by Truemper [29]. It also can be derived from Proposition 7.

▶ **Proposition 8** ([29]). *Let G and H be 3-connected n-vertex graphs and let φ be a 2-isomorphism of G to H. Then G and H are φ-isomorphic.*

Throughout this section we assume that G and H are n-vertex 2-connected graphs and let φ be a 2-isomorphism of G to H. Let also $\mathcal{T}^{(1)} = (T^{(1)}, \{X_t^{(1)}\}_{t\in V(T^{(1)})})$ and $\mathcal{T}^{(2)} = (T^{(2)}, \{X_t^{(2)}\}_{t\in V(T^{(2)})})$ be the Tutte decompositions of G and H, respectively, and denote by $(W_2^{(h)}, W_{\geq 3}^{(h)})$ the partition of $V(T^{(h)})$ satisfying (T4)–(T8) for $h = 1, 2$. We use Lemma 6 and Propositions 7 and 8 to show the following lemmata.

▶ **Lemma 9.** *There is an isomorphism α of $T^{(1)}$ to $T^{(2)}$ such that*

- **(i)** *for every $t \in V(T^{(1)})$, $|X_t^{(1)}| = |X_{\alpha(t)}^{(2)}|$, in particular, $t \in W_2^{(1)}$ ($t \in W_{\geq 3}^{(1)}$, respectively) if and only if $\alpha(t) \in W_2^{(2)}$ ($\alpha(t) \in W_{\geq 3}^{(2)}$, respectively),*
- **(ii)** *for every $t \in W_{\geq 3}^{(1)}$, the torso of $X_t^{(1)}$ is a 3-connected graph (a cycle, respectively) if and only if the torso of $X_{\alpha(t)}^{(2)}$ is a 3-connected graph (a cycle, respectively),*
- **(iii)** *for every $t \in V(T^{(1)})$, $\varphi(E(G[X_t^{(1)}]) = E(H[X_{\alpha(t)}^{(2)}])$.*

Let F be a 2-connected graph. Let also $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$ be the Tutte decomposition of F and let $(W_2, W_{\geq 3})$ be the partition of $V(T)$ satisfying (T4)–(T8). We denote by $\widehat{F}$ the graph obtained from F by making the vertices of X_t adjacent for every $t \in W_2$. We say that $\widehat{F}$ is the *enhancement* of F. Note that $\mathcal{T}$ is the Tutte decomposition of $\widehat{F}$ and the torso of each bag X_t is $\widehat{F}[X_t]$. Notice also that (A, B) is a Whitney separation of F if and only if (A, B) is a Whitney separation of $\widehat{F}$. We also say that F is *enhanced* if $F = \widehat{F}$.

To simplify the arguments in our proofs, it is convenient for us to switch from 2-isomorphisms of graphs to 2-isomorphisms of their enhancements. By Lemma 9, there is an isomorphism α of $T^{(1)}$ to $T^{(2)}$ satisfying conditions (i) – (ii) of the lemma. We define

the *enhanced* mapping $\widehat{\varphi}\colon E(\widehat{G}) \to E(\widehat{H})$ such that $\widehat{\varphi}(e) = \varphi(e)$ for $e \in E(G)$, and for each $e \in E(\widehat{G}) \setminus E(G)$ with its end-vertices in $X_t^{(1)}$ for some $t \in W_2^{(1)}$, we define $\widehat{\varphi}(e)$ be the edge with the end-vertices in $X_{\alpha(t)}^{(2)}$.

▶ **Lemma 10.** *The mapping $\widehat{\varphi}$ is a 2-isomorphism of $\widehat{G}$ to $\widehat{H}$. Moreover, a sequence of Whitney switches makes G φ-isomorphic to H if and only if the same sequence makes $\widehat{G}$ $\widehat{\varphi}$-isomorphic to $\widehat{H}$.*

Lemma 10 allows us to consider enhanced graph and this is useful, because we can strengthen the claim of Lemma 9.

▶ **Lemma 11.** *Let G and H be enhanced graphs. Then there is an isomorphism α of $T^{(1)}$ to $T^{(2)}$ such that conditions (i)–(iii) of Lemma 9 are fulfilled and, moreover,*

(iv) *for every $t \in V(T^{(1)})$, $G[X_t^{(1)}]$ is isomorphic to $H[X_{\alpha(t)}^{(2)}]$.*

Moreover, if $G[X_t^{(1)}]$ is 3-connected, then $G[X_t^{(1)}]$ is φ-isomorphic to $H[X_{\alpha(t)}^{(2)}]$.

For the remaining part of the sections, we assume that G and H are enhanced graphs and α is the isomorphism of $T^{(1)}$ to $T^{(2)}$ satisfying conditions (i)–(iv) of Lemmas 9 and 11.

Our next aim is to investigate properties of the sequences of Whitney switches that are used in solutions for Whitney Switches. For a sequence $\mathcal{S}$ of Whitney switches such that the graph G' obtained from G by applying this sequence is ϕ-isomorphic to H, we say that $\mathcal{S}$ is an *H-sequence*. We also say that $\mathcal{S}$ is *minimum* if $\mathcal{S}$ has minimum length.

For $t \in W_{\geq 3}^{(1)}$, we say that $X_t^{(1)}$ is *φ-good* if $G[X_t^{(1)}]$ is φ-isomorphic to $H[X_{\alpha(t)}^{(2)}]$, and $X_t^{(1)}$ is *φ-bad* otherwise. Notice that if $G[X_t^{(1)}]$ is 3-connected, then $X_t^{(1)}$ is φ-good but this not always so if $G[X_t^{(1)}]$ is a cycle.

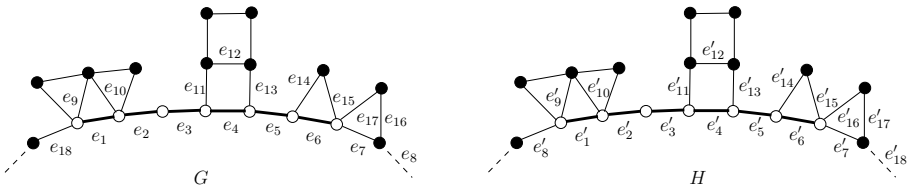


Figure 4 An example of a φ-good segment; $\varphi(e_i) = e'_i$ for $i \in \{1, \ldots, 18\}$, the vertices of the segment are white.

Let $t \in W_{\geq 3}^{(1)}$ such that $X_t^{(1)}$ is φ-bad. Clearly, $G[X_t^{(1)}]$ is a cycle. Let $\{t_1, \ldots, t_s\} = N^2_{T^{(1)}}(t)$ and denote $G_t = G[X_t^{(1)} \cup \bigcup_{i=1}^s X_{t_i}^{(1)}]$ and $H_{\alpha(t)} = H[X_{\alpha(t)}^{(2)} \cup \bigcup_{i=1}^s X_{\alpha(t_i)}^{(2)}]$. Let $P = v_0 \cdots v_r$ be a path in $G[X_t^{(1)}]$ and $e_i = v_{i-1}v_i$ for $i \in \{1, \ldots, r\}$. We say that P a *φ-good segment* of $X_t^{(1)}$ if the following holds (see Fig. 4 for an example):

(i) the length of P is at least 5,

(ii) there is a path $P' = u_0 \cdots u_r$ in $H[X_{\alpha(t)}^{(2)}]$ such that $u_{i-1}u_i = \varphi(e_i)$ for all $i \in \{1, \ldots, r\}$,

(iii) for every $i \in \{1, \ldots, r\}$ and for every $t' \in W_{\geq 3}^{(1)}$ such that $X_t^{(1)} \cap X_{t'}^{(1)} = \{v_{i-1}, v_i\}$, $X_{t'}^{(1)}$ is φ-good,

(iv) for every $i \in \{1, \ldots, r-1\}$, $\varphi(E_{G_t}(v_i)) = E_{H_{\alpha(t)}}(u_i)$.

For distinct $t_1, t_2 \in W_{\geq 3}^{(1)}$ with a common neighbor in $T^{(1)}$, we say that $X_{t_1}^{(1)}$ and $X_{t_2}^{(1)}$ are *mutually φ-good* (see Fig. 5) if they are φ-good and $G[X_{t_1}^{(1)} \cup X_{t_2}^{(1)}]$ is φ-isomorphic to $H[X_{\alpha(t_1)}^{(2)} \cup X_{\alpha(t_2)}^{(2)}]$.

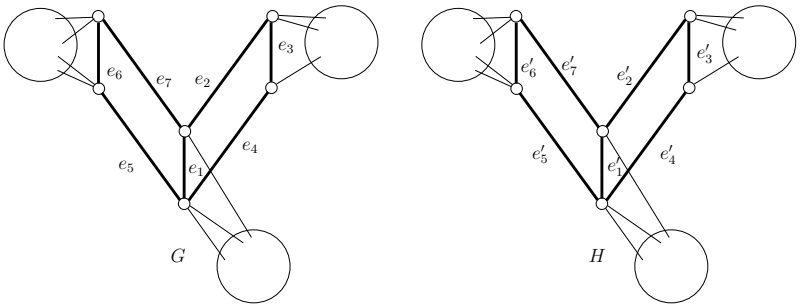


Figure 5 Mutually φ-good bags; $\varphi(e_i) = e'_i$ for $i \in \{1, \ldots, 7\}$, the vertices of the mutually φ-good bags of G and the corresponding bags of H are white.

We say that an H-sequence is *φ-good* if no Whitney switch of $\mathcal{S}$ splits (mutually) φ-good bags and segments. Formally, for every switch with respect to some Whitney separation (A, B) in $\mathcal{S}$,

- **(i)** $X_t^{(1)} \subseteq A$ or $X_t^{(1)} \subseteq B$ for every φ-good bag $X_t^{(1)}$,
- **(ii)** $V(P) \subseteq A$ or $V(P) \subseteq B$ for every φ-good segment P,
- **(iii)** $X_{t_1}^{(1)} \cup X_{t_2}^{(1)} \subseteq A$ or $X_{t_1}^{(1)} \cup X_{t_2}^{(1)} \subseteq B$ for every two distinct mutually φ-good bags $X_{t_1}^{(1)}$ and $X_{t_2}^{(1)}$.

We prove that it is sufficient to consider φ-good H-sequences. The crucial tool for our kernelization algorithm is the following lemma that is proved by using Lemmata 5 and 11. In particular, Lemma 5 is crucial for showing that we can avoid splitting φ-good segments.

▶ **Lemma 12.** *There is a minimum H-sequence of Whitney switches $\mathcal{S}$ that is φ-good.*

Let $t \in W_{\geq 3}^{(1)}$ be such that $X_t^{(1)}$ is φ-bad. Denote by $t_1, \ldots, t_s \neq t$ the nodes of $N^2_{T^{(1)}}(t)$. Let $G_t = G[X_t^{(1)} \cup \bigcup_{i=1}^s X_{t_i}^{(1)}]$ and $H_{\alpha(t)} = G[X_{\alpha(t)}^{(2)} \cup \bigcup_{i=1}^s X_{\alpha(t_i)}^{(2)}]$. In words, G_t is the subgraphs of G induced by the vertices of $X_t^{(1)}$ and the vertices of the bags at distance two in $T^{(1)}$ from t, and $H_{\alpha(t)}$ the subgraph of H induced by the vertices of the bags that are images of the bags composing G_t according to α.

We say that a vertex $v \in X_t^{(1)}$ is a *crucial breakpoint* if $\varphi(E_{G_t}(v)) \neq E_{H_{\alpha(t)}}(u)$ for every $u \in V(H_{\alpha(t)})$. We denote by $b(G)$ the total number of crucial breakpoints in the φ-bad bags and say that $b(G)$ is the *breakpoint number* of G. Recall that by our convention, G and H are enhanced graphs, but we extend this definition for the general case needed in the next section. For (not necessarily enhanced) 2-isomorphic graphs G and H, and a 2-isomorphism φ, we construct their enhancements $\widehat{G}$ and $\widehat{H}$, and consider the enhanced mapping $\widehat{\varphi}$. Then $b(G)$ is defined as $b(\widehat{G})$.

Observe that if G and H are φ-isomorphic, then $b(t) = 0$ by Lemma 6, but not the other way around.

We conclude the section by giving a lower bound for the length of an H-sequence.

▶ **Lemma 13.** *Let $\mathcal{S}$ be an H-sequence of Whitney switches. Then $b(G)/2 \leq |\mathcal{S}|$.*

5 Kernelization for Whitney Switches

In this section, we show that Whitney Switches parameterized by k admits a polynomial kernel. To do it, we obtain a more general result by proving that the problem has a polynomial kernel when parameterized by the breakpoint number of the first input graph.

▶ **Theorem 14.** *Whitney Switches has a kernel such that each graph in the obtained instance has at most* $\max\{52 \cdot b - 36, 3\}$ *vertices, where* b *is the breakpoint number of the input graph.*

Proof sketch. Let (G, H, φ, k) be an instance of Whitney Switches, where G and H are n-vertex 2-connected 2-isomorphic graphs, $\varphi\colon E(G) \to E(H)$ is a 2-isomorphism, and k is a nonnegative integer.

First, we use Proposition 3 to construct the Tutte decompositions of G and H. Denote by $\mathcal{T}^{(1)} = (T^{(1)}, \{X_t^{(1)}\}_{t \in V(T^{(1)})})$ and $\mathcal{T}^{(2)} = (T^{(2)}, \{X_t^{(2)}\}_{t \in V(T^{(2)})})$ the constructed Tutte decompositions of G and H respectively, and let $(W_2^{(h)}, W_{\geq 3}^{(h)})$ be the partition of $V(T^{(h)})$ satisfying (T4)–(T8) for $h = 1, 2$.

In the next step, we construct the isomorphism $\alpha\colon V(T^{(1)}) \to V(T^{(2)})$ satisfying conditions (i)–(iii) of Lemma 9. Recall that Lemma 9 claims that such an isomorphism always exists.

Given α, we compute the enhancements $\widehat{G}$ and $\widehat{H}$ of G and H respectively, and then define the enhanced mapping $\widehat{\varphi}\colon E(\widehat{G}) \to E(\widehat{H})$. Note that α satisfies the conditions of Lemma 11. Observe also that we can verify in polynomial time whether a bag $X_t^{(1)}$ for $t \in W_{\geq 3}^{(1)}$ is φ-good or not.

To simplify notation, let $G := \widehat{G}$, $H := \widehat{H}$ and $\varphi := \widehat{\varphi}$.

Now we apply a series of reduction rules that are applied for G, H, φ, and the Tutte decompositions of G and H.

The aim of the first rule is to decrease the total size of bags that are φ-bad (see Fig. 6 for an example).

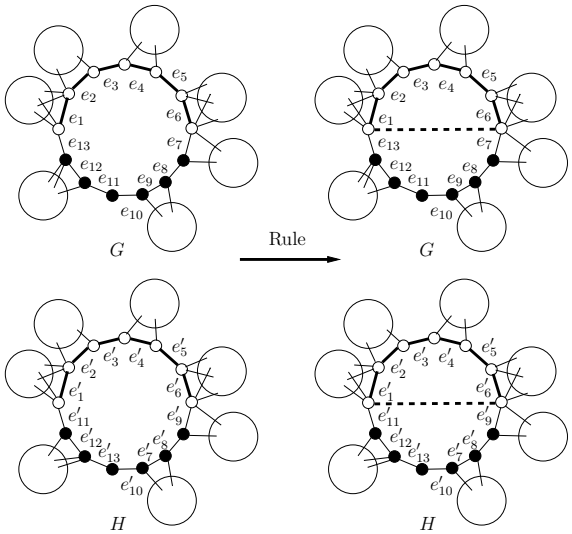


Figure 6 An example of an application of Reduction Rule 1; $\varphi(e_i) = e'_i$ for $i \in \{1, \ldots, 13\}$, the vertices of the φ-good segment in G and the corresponding segment in H are white, and the added edges are shown by dashed lines.

▶ **Reduction Rule 1.** *If for* $t \in W_{\geq 3}^{(1)}$ *such that* $X_t^{(1)}$ *is* φ*-bad, there is an inclusion maximal* φ*-good segment* $P = v_0 \cdots v_r$*, then do the following:*

- *find the path* $P' = u_0 \cdots u_r$ *in* $H[X_{\alpha(t)}^{(2)}]$ *composed by the edges* $u_{i-1}u_i = \varphi(v_{i-1}v_i)$ *for* $i \in \{1, \ldots, r\}$*,*
- *add the edge* v_0v_r *to* G *and* u_0u_r *to* H*,*
- *extend* φ *by setting* $\varphi(v_0u_r) = u_0u_r$*,*
- *recompute the Tutte decompositions of the obtained graphs and the isomorphism* α*.*

The safeness of the rule is proved by using Lemma 12. The crucial observation is that there is an optimal sequence of Whitney switches such that every φ-good segment remains in one part of every Whitney separation in the sequence, i.e., they are not split. Reduction Rule 1 is applied exhaustively while we are able to find φ-good segments. To simplify notation, we use G, H and φ to denote the obtained graphs and the obtained 2-isomorphism. We also keep the notation used for the Tutte decompositions.

Our next reduction rule is used to simplify the structure of φ-good bags by turning them into cliques (see Fig. 7 for an example).

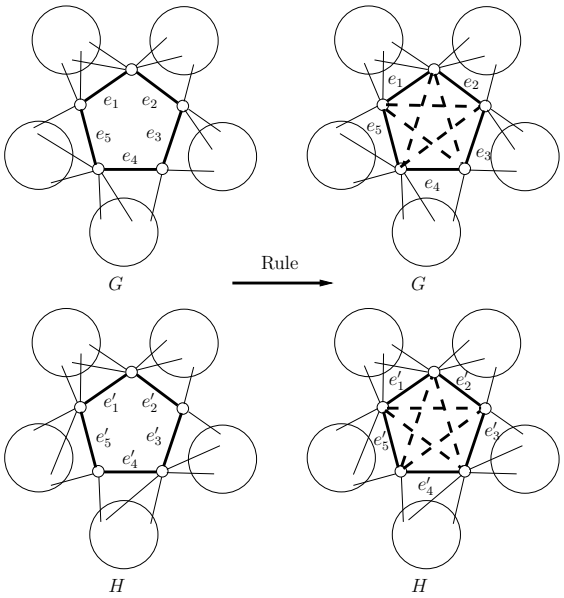


Figure 7 An example of an application of Reduction Rule 2; $\varphi(e_i) = e'_i$ for $i \in \{1, \ldots, 5\}$, the vertices of the φ-good bag of G and the corresponding bag of H are white, and the added edges are shown by dashed lines.

▶ **Reduction Rule 2.** *If for $t \in W^{(1)}_{\geq 3}$ such that $X^{(1)}_t$ is a φ-good, there are nonadjacent vertices in $X^{(1)}_t$, then compute the φ-isomorphism ψ of $G[X^{(1)}_t]$ to $H[X^{(2)}_{\alpha(t)}]$ and for every nonadjacent $u, v \in X^{(1)}_t$, do the following:*

- *add the edge uv to G and $\psi(u)\psi(v)$ to H,*
- *extend φ by setting $\varphi(uv) = \psi(u)\psi(v)$.*

The safeness of the rule follows from Lemma 12. We use that there is an optimal sequence of Whitney switches such that no Whitney separation splits φ-good bags. We apply Reduction Rule 2 for all bags of G that are not cliques. We use the same convention as for the first rule, and keep the old notation for the obtained graphs, their Tutte decompositions, and the obtained 2-isomorphism.

The next aim is to reduce the number of mutually φ-good bags by "gluing" them into cliques (see Fig. 8 for an example).

▶ **Reduction Rule 3.** *For distinct $t_1, t_2 \in W^{(1)}_{\geq 3}$ such that $X^{(1)}_{t_1}$ and $X^{(1)}_{t_2}$ are mutually φ-good,*

- *compute the φ-isomorphism ψ of $G[X^{(1)}_{t_1} \cup X^{(1)}_{t_2}]$ to $H[X^{(2)}_{\alpha(t_1)} \cup X^{(2)}_{\alpha(t_2)}]$,*
- *for every $u \in X^{(1)}_{t_1} \setminus X^{(1)}_{t_2}$ and every $v \in X^{(1)}_{t_2} \setminus X^{(1)}_{t_1}$, do the following:*
  - *add the edge uv to G and $\psi(u)\psi(v)$ to H,*
  - *extend φ by setting $\varphi(uv) = \psi(u)\psi(v)$,*
- *recompute the Tutte decompositions of the obtained graphs and the isomorphism α.*

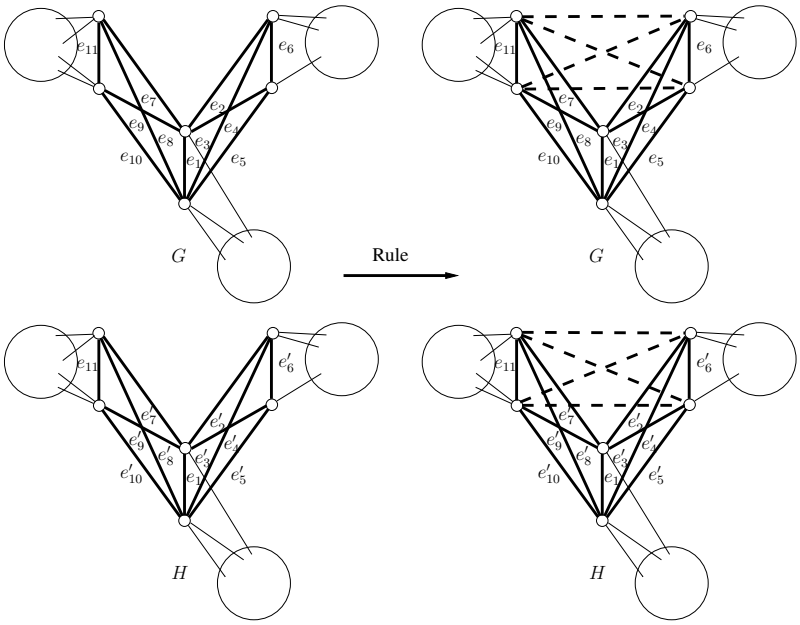


Figure 8 An example of an application of Reduction Rule 3; $\varphi(e_i) = e'_i$ for $i \in \{1, \ldots, 11\}$, the vertices of the mutually φ-good bags of G and the corresponding bags of H are white, and the added edges are shown by dashed lines.

To show the safeness, we use that, by Lemma 12, we can find an optimal sequence of Whitney switches such that the corresponding Whitney separations do not split mutually φ-good bags. Reduction Rule 3 is applied exhaustively whenever it is possible. As before, we do not change the notation for the obtained graphs, their Tutte decompositions, and the obtained 2-isomorphism.

Our next rule is used to perform the Whitney switches that are unavoidable (see Fig. 9 for an example).

▶ **Reduction Rule 4.** *If there is $t \in W_2^{(1)}$ such that $d_{T^{(1)}}(t) = 2$ and for the neighbors t_1 and t_2 of t, it holds that $X_{t_1}^{(1)}$ and $X_{t_2}^{(2)}$ are φ-good but not mutually φ-good, then do the following:*

- *find the connected components T_1 and T_2 of $T^{(1)} - t$, and construct $A = \bigcup_{t' \in V(T_1)} X_{t'}^{(1)}$ and $B = \bigcup_{t' \in V(T_2)} X_{t'}^{(1)}$,*
- *perform the Whitney switch with respect to the separation (A, B),*
- *set $k := k - 1$, and if $k < 0$, then return the trivial no-instance and stop.*

Reduction Rule 4 is applied exhaustively whenever it is possible. Note that after applying this rule, we are able to apply Reduction Rule 3 and we do it.

Suppose that the algorithm did not stop while executing Reduction Rule 4. In the same way as with previous rules, we maintain the initial notation for the obtained graphs, their Tutte decompositions, and the obtained 2-isomorphism.

Our final rule deletes simplicial vertices of degree at least 3.

▶ **Reduction Rule 5.** *If there is a simplicial vertex $v \in V(G)$ with $d_G(v) \geq 3$, then do the following:*

- *find the vertex $u \in V(H)$ such that $E_H(u) = \varphi(E_G(v))$,*
- *set $G := G - v$ and $H := H - u$,*
- *set $\varphi := \varphi|_{E(G) \setminus E_G(v)}$.*

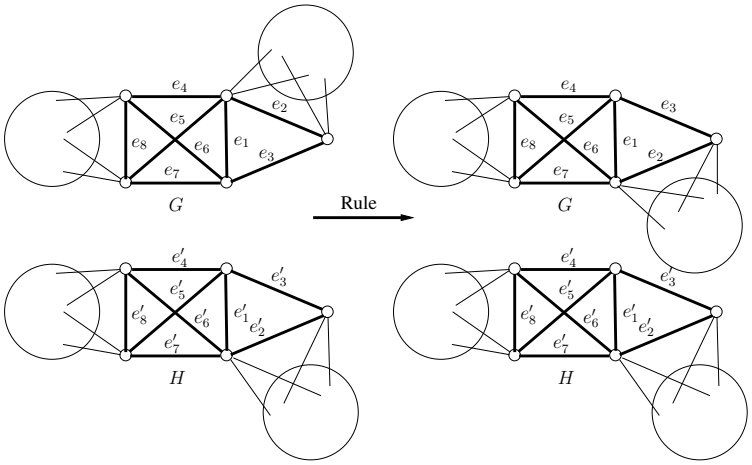


Figure 9 An example of an application of Reduction Rule 4; $\varphi(e_i) = e'_i$ for $i \in \{1, \ldots, 8\}$, the vertices of the switched φ-good bags of G and the corresponding bags of H are white.

Reduction Rule 5 is applied exhaustively. Let G, H and φ be the resulting graphs. We also keep the same notation for the Tutte decompositions of G and H and the isomorphism α following the previous convention. This completes the description of our kernelization algorithm as the graphs G and H have bounded size. We show that $|V(G)| = |V(H)| \leq \max\{52 \cdot b(G) - 36, 3\}$.

It can be shown that Reduction Rules 1–5 do not increase the breakpoint number. Therefore, for the the obtained instance (G, H, φ, k) of Whitney Switches, $|V(G)| = |V(H)| \leq \min\{52 \cdot b - 36, 3\}$, where b is the breakpoint number of the initial input graph G. ◀

Theorem 14 together with Lemma 13 imply that Whitney Switches has a polynomial kernel when parameterized by k. Thus, we can show Theorem 2 that we restate.

▶ **Theorem 2.** *Whitney Switches admits a kernel with $\mathcal{O}(k)$ vertices and is solvable in $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ time.*

In Corollary 4, we proved that Whitney Switches is NP-hard when the input graphs are constrained to be cycles. Theorem 14 indicates that it is the presence of bags in the Tutte decompositions that are cycles of length at least 4 that makes Whitney Switches difficult, because only such cycles may contain crucial breakpoint. In particular, we can derive the following straightforward corollary.

▶ **Corollary 15.** *Let (G, H, φ, k) be an instance of Whitney Switches such that $b(G) = 0$. Then Whitney Switches for this instance can be solved in polynomial time.*

For example, the condition that $b(G) = 0$ holds when G and H have no induced cycles of length at least 4, that is, when G and H are chordal graphs.

▶ **Corollary 16.** *Whitney Switches can be solved in polynomial time on chordal graphs.*

6 Conclusion

We proved that Whitney Switches admits a polynomial kernel when parameterized by the breakpoint number of the input graphs and this implies that the problem has a polynomial kernel when parameterized by k. More precisely, we obtain a kernel, where the graphs have $\mathcal{O}(k)$ vertices. Using this kernel, we can solve Whitney Switches in $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ time. It is natural to ask whether the problem can be solved in a single-exponential in k time.

Another interesting direction of research is to investigate approximability for Whitney Switches. In [3], Berman and Karpinski proved that for every $\varepsilon > 0$, it is NP-hard to approximate the reversal distance $d(\pi)$ for a linear permutation π within factor $\frac{1237}{1236} - \varepsilon$. This result can be translated for circular permutations and this allows to obtain inapproximability lower bound for Whitney Switches on cycles similarly to Corollary 4. From the positive side, the currently best 1.375-approximation for $d(\pi)$ was given by Berman, Hannenhalli, and Karpinski [2]. Due to the close relations between Whitney Switches and the sorting by reversal problem, it is interesting to check whether the same approximation ratio can be achieved for Whitney Switches.

In Whitney Switches, we are given two graphs G and H together with a 2-isomorphism and the task is to decide whether we can apply at most k Whitney switches to obtain a graph G' from G such that G' is φ-isomorphic to H. We can relax the task and ask whether we can obtain G' that is isomorphic to H, that is, we do not require an isomorphism of G to H be a φ-isomorphism. Formally, we define the following problem.

Unlabeled Whitney Switches

Input:	2-Isomorphic graphs G and H, and a nonnegative integer k.
Task:	Decide whether it is possible to obtain a graph G' from G by at most k Whitney switches such that G' is isomorphic to H.

Note that if φ is a 2-isomorphism of G to H, then the minimum number of Whitney switches needed to obtain G' that is φ-isomorphic to H gives an upper bound for the number of Whitney switches required to obtain from G a graph that isomorphic to G. However, these values can be arbitrary far apart. Consider two cycles G and H with the same number of vertices. Clearly, G and H are isomorphic but for a given 2-isomorphism φ of G to H, we may need many Whitney switches to obtain G' that is φ-isomorphic to H and the number of switches is not bounded by any constant.

Using the result of Solomon, Sutcliffe, and Lister [27], we can show that Unlabeled Whitney Switches is NP-hard for very restricted instances.

▶ **Proposition 17.** *Unlabeled Whitney Switches is NP-complete when restricted to 2-connected series-parallel graphs even if the input graphs are given together with their* 2*-isomorphism.*

Proposition 17 lead to the question about the parameterized complexity of Unlabeled Whitney Switches. In particular, does the problem admit a polynomial kernel when parameterized by k?

Notice that to deal with Unlabeled Whitney Switches, we should be able to check whether the input graphs G and H are isomorphic. If we are given a 2-isomorphism φ of G to H, then checking whether G and H are φ-isomorphic can be done in polynomial time by Lemma 6. However, checking whether G and H are isomorphic, even if a 2-isomorphism φ is given, is a complicated task. For example, it can be observed that this is at least as difficult as solving Graph Isomorphism on tournaments (recall that a *tournament* is a directed graph such that for every two distinct vertices u and v, either uv or vu is an arc). While Graph Isomorphism on tournaments may be easier than the general problem (we refer to [26, 33] for the details), still it is unknown whether this special case can be solved in polynomial time and the best known algorithm is the quasi-polynomial algorithm of Babai [1]. Given this observation, it is natural to consider Unlabeled Whitney Switches on graph classes for which Graph Isomorphism is polynomially solvable. For example, what can be said about Unlabeled Whitney Switches on planar graphs?

The relation between Whitney switches and sorting by reversals together with the reduction in the proof of Proposition 17 indicates that as the first step, it could be reasonable to investigate the following problem for sequences that generalizes Sorting by Reversals for permutations. Let $\pi = (\pi_1, \ldots, \pi_n)$ be a sequence of positive integers; note that now some elements of π may be the same. For $1 \leq i < j \leq n$, we define the *reversal* $\rho(i, j)$ in exactly the same way as for permutations. Then we can define the *reversal distance* between two n-element sequences such that the multisets of their elements are the same; we assume that the distance is $+\infty$ if the multisets of elements are distinct.

Sequence Reversal Distance

Input:	Two n-element sequences π and σ of positive integers and a nonnegative integer k.
Task:	Decide whether the reversal distance between π and σ is at most k.

By the result of Caprara in [7], this problem is NP-complete even if the input sequences are permutations. It is also known that the problem is NP–complete if the input sequences contain only two distinct elements [9]. The question, whether Sequence Reversal Distance is FPT when parameterized by k, was explicitly stated in the survey of Bulteau et. al [6] (in terms of strings) and is open and only some partial results are known [5]. We also can define the version of Sequence Reversal Distance for circular sequences and ask the same question about parameterized complexity. Using the idea behind the reduction in the proof of Proposition 17, it is easy to observe that Unlabeled Whitney Switches on 2-connected series-parallel graphs is at least as hard as the circular variant of Sequence Reversal Distance.

References

1. László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.
2. Piotr Berman, Sridhar Hannenhalli, and Marek Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, volume 2461 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2002. doi:10.1007/3-540-45749-6_21.
3. Piotr Berman and Marek Karpinski. On some tighter inapproximability results (extended abstract). In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 1999. doi:10.1007/3-540-48523-6_17.
4. Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Comput. Geom.*, 42(1):60–80, 2009. doi:10.1016/j.comgeo.2008.04.001.
5. Laurent Bulteau, Guillaume Fertin, and Christian Komusiewicz. (Prefix) reversal distance for (signed) strings with few blocks or small alphabets. *J. Discrete Algorithms*, 37:44–55, 2016. doi:10.1016/j.jda.2016.05.002.
6. Laurent Bulteau, Falk Hüffner, Christian Komusiewicz, and Rolf Niedermeier. Multivariate algorithmics for NP-hard string problems. *Bulletin of the EATCS*, 114, 2014. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/310.
7. Alberto Caprara. Sorting by reversals is difficult. In *Proceedings of the First Annual International Conference on Research in Computational Molecular Biology, RECOMB 1997, Santa Fe, NM, USA, January 20-23, 1997*, pages 75–83. ACM, 1997. doi:10.1145/267521.267531.

8 Markus Chimani, Petr Hlinený, and Petra Mutzel. Vertex insertion approximates the crossing number of apex graphs. *Eur. J. Comb.*, 33(3):326–335, 2012. doi:10.1016/j.ejc.2011.09.009.

9 David A. Christie and Robert W. Irving. Sorting strings by reversals and by transpositions. *SIAM J. Discrete Math.*, 14(2):193–206, 2001. doi:10.1137/S0895480197331995.

10 Sean Cleary and Katherine St. John. Rotation distance is fixed-parameter tractable. *Inform. Process. Lett.*, 109(16):918–922, 2009. doi:10.1016/j.ipl.2009.04.023.

11 Bruno Courcelle. The monadic second-order logic of graphs XI: hierarchical decompositions of connected graphs. *Theor. Comput. Sci.*, 224(1-2):35–58, 1999. doi:10.1016/S0304-3975(98)00306-5.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

14 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

15 Fedor V. Fomin and Petr A. Golovach. Kernelization of whitney switches. *CoRR*, abs/2006.13684, 2020. arXiv:2006.13684.

16 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization*. Cambridge University Press, Cambridge, 2019. Theory of parameterized preprocessing.

17 Sridhar Hannenhalli and Pavel A. Pevzner. To cut... or not to cut (applications of comparative physical maps in molecular evolution). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 28-30 January 1996, Atlanta, Georgia, USA*, pages 304–313. ACM/SIAM, 1996. URL: http://dl.acm.org/citation.cfm?id=313852.314077.

18 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973. doi:10.1137/0202012.

19 Iyad A. Kanj, Eric Sedgwick, and Ge Xia. Computing the flip distance between triangulations. *Discrete & Computational Geometry*, 58(2):313–344, 2017. doi:10.1007/s00454-017-9867-x.

20 John D. Kececioglu and David Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1/2):180–210, 1995. doi:10.1007/BF01188586.

21 Daniel Lokshtanov, Amer E Mouawad, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *Journal of Computer and System Sciences*, 95:122–131, 2018.

22 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Comput. Geom.*, 49:17–23, 2015. doi:10.1016/j.comgeo.2014.11.001.

23 Joan M. Lucas. An improved kernel size for rotation distance in binary trees. *Inform. Process. Lett.*, 110(12-13):481–484, 2010. doi:10.1016/j.ipl.2010.04.022.

24 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017. doi:10.1007/s00453-016-0159-2.

25 Pavel A. Pevzner. *Computational molecular biology - an algorithmic approach*. MIT Press, 2000.

26 Pascal Schweitzer. A polynomial-time randomized reduction from tournament isomorphism to tournament asymmetry. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 66:1–66:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.66.

27 Andrew Solomon, Paul J. Sutcliffe, and Raymond Lister. Sorting circular permutations by reversal. In *Algorithms and Data Structures, 8th International Workshop, WADS 2003, Ottawa, Ontario, Canada, July 30 - August 1, 2003, Proceedings*, volume 2748 of *Lecture Notes in Computer Science*, pages 319–328. Springer, 2003. doi:10.1007/978-3-540-45078-8_28.

28 Carsten Thomassen. Embeddings and minors. In *Handbook of combinatorics, Vol. 1, 2*, pages 301–349. Elsevier Sci. B. V., Amsterdam, 1995.

29 Klaus Truemper. On whitney's 2-isomorphism theorem for graphs. *Journal of Graph Theory*, 4(1):43–49, 1980. doi:10.1002/jgt.3190040106.

30 W. T. Tutte. *Connectivity in graphs*. Mathematical Expositions, No. 15. University of Toronto Press, Toronto, Ont.; Oxford University Press, London, 1966.

31 W. T. Tutte. *Graph theory*, volume 21 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2001. With a foreword by Crispin St. J. A. Nash-Williams, Reprint of the 1984 original.

32 Dirk L. Vertigan and Geoffrey P. Whittle. A 2-isomorphism theorem for hypergraphs. *J. Comb. Theory, Ser. B*, 71(2):215–230, 1997. doi:10.1006/jctb.1997.1789.

33 Fabian Wagner. Hardness results for tournament isomorphism and automorphism. In *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 572–583. Springer, 2007. doi:10.1007/978-3-540-74456-6_51.

34 Hassler Whitney. Non-separable and planar graphs. *Trans. Amer. Math. Soc.*, 34(2):339–362, 1932. doi:10.2307/1989545.

35 Hassler Whitney. 2-Isomorphic Graphs. *Amer. J. Math.*, 55(1-4):245–254, 1933. doi:10.2307/2371127.

Subexponential Parameterized Algorithms and Kernelization on Almost Chordal Graphs

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway
Fedor.Fomin@uib.no

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
Petr.Golovach@uib.no

Abstract

We study algorithmic properties of the graph class Chordal $-$ ke, that is, graphs that can be turned into a chordal graph by adding at most k edges or, equivalently, the class of graphs of fill-in at most k. We discover that a number of fundamental intractable optimization problems being parameterized by k admit *subexponential* algorithms on graphs from Chordal $-$ ke. While various parameterized algorithms on graphs for many structural parameters like vertex cover or treewidth can be found in the literature, up to the Exponential Time Hypothesis (ETH), the existence of subexponential parameterized algorithms for most of the structural parameters and optimization problems is highly unlikely. This is why we find the algorithmic behavior of the "fill-in parameterization" very unusual.

Being intrigued by this behaviour, we identify a large class of optimization problems on Chordal $-$ ke that admit algorithms with the typical running time $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$. Examples of the problems from this class are finding an independent set of maximum weight, finding a feedback vertex set or an odd cycle transversal of minimum weight, or the problem of finding a maximum induced planar subgraph. On the other hand, we show that for some fundamental optimization problems, like finding an optimal graph coloring or finding a maximum clique, are FPT on Chordal $-$ ke when parameterized by k but do not admit subexponential in k algorithms unless ETH fails.

Besides subexponential time algorithms, the class of Chordal $-$ ke graphs appears to be appealing from the perspective of kernelization (with parameter k). While it is possible to show that most of the weighted variants of optimization problems do not admit polynomial in k kernels on Chordal $-$ ke graphs, this does not exclude the existence of Turing kernelization and kernelization for unweighted graphs. In particular, we construct a polynomial Turing kernel for Weighted Clique on Chordal $-$ ke graphs. For (unweighted) Independent Set we design polynomial kernels on two interesting subclasses of Chordal $-$ ke, namely, Interval $-$ ke and Split $-$ ke graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, structural parameterization, subexponential algorithms, kernelization, chordal graphs, fill-in, independent set, clique, coloring

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.49

Related Version The full version of the paper is available at `https://arxiv.org/abs/2002.08226`.

Funding The research leading to these results has received funding from the Research Council of Norway via the project "MULTIVAL" (grant no. 263317).

Acknowledgements We thank Torstein Strømme, Daniel Lokshtanov, and Pranabendu Misra for fruitful discussions on the topic of this paper. We also grateful to Saket Saurabh for helpful suggestions that allowed us to improve our results.

1 Introduction

Many NP-hard graph optimization problems are solvable in polynomial or even linear time when the input of the problem is restricted to a special graph class. For example, the chromatic number of a perfect graph can be computed in polynomial time [34], the Feedback Vertex Set problem is solvable in polynomial time on chordal graphs [31], and Hamiltonicity on interval graphs [44]. From the perspective of parameterized complexity, the natural question here is how stable are these nice algorithmic properties of graph classes subject to some perturbations. For example, if an input n-vertex graph G is not chordal, but can be turned into a chordal graph by adding at most k edges, how fast can we solve Feedback Vertex Set on G? Can we solve the problem in polynomial time for constant k? Or maybe for $k = \log n$ or even for $k = poly(\log n)$? A word of warning is on order here. Since an algorithm for Feedback Vertex Set of running time $2^{o(n)}$ will refute the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi and Zane [37, 38], and because $k \leq \binom{n}{2}$, the existence of an algorithm of running time $2^{k^{1/2-\varepsilon}} \cdot n^{\mathcal{O}(1)}$ for some $\varepsilon > 0$ (which is polynomial for $k = (\log n)^{2/(1-2\varepsilon)}$) is unlikely. Interestingly, as we shall see, Feedback Vertex Set (and many other problems) are solvable in time $2^{k^{1/2}\log k} \cdot n^{\mathcal{O}(1)}$.

Leizhen Cai in [11] introduced a convenient notation for "perturbed" graph classes. Let $\mathcal{F}$ be a class of graphs, then $\mathcal{F} - ke$ (respectively $\mathcal{F} - ve$) is the class of those graphs that can be obtained from a member of $\mathcal{F}$ by deleting at most k edges (respectively vertices). Similarly one can define classes $\mathcal{F} + ke$ and $\mathcal{F} + ve$. Then for any class $\mathcal{F}$ and optimization problem $\mathcal{P}$ that can be solved in polynomial time on $\mathcal{F}$, the natural question is whether $\mathcal{P}$ is fixed-parameter tractable parameterized by k, the "distance" to $\mathcal{F}$.

In this paper we obtain several algorithmic results on the parameterized complexity of optimization problems on $\mathcal{F} - ke$, where $\mathcal{F}$ is the class of chordal graphs. Let us remind that a graph H is *chordal* (or *triangulated*) if every cycle of length at least four has a chord, i.e., an edge between two nonconsecutive vertices of the cycle. We denote by Chordal $- ke$ the class of graphs that can be made chordal graph by adding at most k edges. While parameterized algorithms for some optimization problems on the class of Chordal $- ke$ graphs were studied (see the section on previous work), our work introduces the first subexponential parameterized algorithms in this graph class. We prove the following.

Subexponential parameterized algorithms. We discover a large class of optimization problems on graph class Chordal $- ke$ that are solvable in time $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$. Examples of such optimization problems are: the problem of finding an induced d-colorable subgraph of maximum weight (which generalizes Weighted Independent Set for $d = 1$ and Weighted Bipartite Subgraph for $d = 2$); the problem of finding a maximum weight induced subgraph admitting a homomorphism into a fixed graph H; the problem of finding an induced d-degenerate subgraph of maximum weight and its variants like Weighted Induced Forest (or, equivalently, Weighted Feedback Vertex Set), Weighted Induced Tree, Induced Planar Graph, Weighted Induced Path (Cycle) or Weighted Induced Cycle Packing; as well as various connectivity variants of these problems like Weighted Connected Vertex Cover and Weighted Connected Feedback Vertex Set. This implies that all these problems are solvable in polynomial time for $k = (\frac{\log n}{\log\log n})^2$. On the other hand, we refute (subject to ETH) existence of a subexponential time $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ algorithms on graphs in Chordal $- ke$ for Coloring and Clique. Moreover, our lower bounds hold for way more restrictive graph class Complete $- ke$, the graphs within k edges from a complete graph. We also show that both problems are fixed-parameter tractable (FPT) (parameterized by k) on Chordal $- ke$ graphs.

Kernelization. It follows almost directly from the previous work [39, 8] that Weighted Independent Set, Weighted Vertex Cover, Weighted Bipartite Subgraph, Weighted Odd Cycle Transversal, Weighted Feedback Vertex Set and Weighted Clique do not admit a polynomial in k kernel (unless coNP $\not\subseteq$ NP/poly) on Complete $-$ ke and hence on Chordal $-$ ke. Interestingly, these lower bounds do not refute the possibility of polynomial Turing kernelization or kernelization for unweighted variants of the problems. Indeed, we show that Weighted Clique on Chordal $-$ ke parameterized by k admits a Turing kernel. For unweighted Independent Set we show that the problem admits polynomial in k kernel on graph classes Interval $-$ ke and Split $-$ ke (graphs that can be turned into an interval or split graphs, corrspondingly, by adding at most k edges).

Previous work. Chordal graphs form an important subclass of perfect graphs. These graphs were also intensively studied from the algorithmic perspective. We refer to books [9, 33, 58] for introduction to chordal graphs and their algorithmic properties.

The problem of determining whether a graph G belongs to Chordal$-ke$, that is checking whether G can be turned into a chordal graph by adding at most k edges, is known in the literature as the Minimum Fill-in problem. The name fill-in is due to the fundamental problem arising in sparse matrix computations which was studied intensively in the past [52, 55]. The survey of Heggernes [36] gives an overview of techniques and applications of minimum and minimal triangulations.

Minimum Fill-in (under the name Chordal Graph Completion) was one of the 12 open problems presented at the end of the first edition of Garey and Johnson's book [30] and it was proved to be NP-complete by Yannakakis [60]. Kaplan et al. proved that Minimum Fill-in is fixed parameter tractable by giving an algorithm of running time $16^k \cdot n^{\mathcal{O}(1)}$ in [43]. There was a chain of algorithmic improvements resulting in decreasing the constant in the base of the exponents [42, 10, 7] resulting with a subexponential algorithm of running time $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$ [28]. A significant amount of work in parameterized algorithms is devoted to recognition problems of classes $\mathcal{F} - ke$, $\mathcal{F} + ke$, $\mathcal{F} - kv$, and $\mathcal{F} + kv$ for chordal graphs and various subclasses of chordal graphs [1, 2, 4, 5, 14, 12, 13, 26, 40, 48, 59].

Parameterized algorithms, mostly for graph coloring problems, were studied on perturbed chordal graphs and subclasses of this graph class [11, 56]. Among other results, Cai [11] proved that Coloring (the problem of computing the chromatic number of a graph) is FPT (parameterized by k) on Split $-$ ke graphs. Marx [47] proved that Coloring is FPT on Chordal $+$ ke and Interval $+$ ke graphs but is W[1]-hard on Chordal $+$ kv and Interval $+$ kv graphs. Jansen and Kratsch [41] proved that for every fixed integer d, the problems d-Coloring and d-List Coloring admit polynomial kernels on the parameterized graph classes Split $+$ kv, Cochordal $+$ kv, and Cograph $+$ kv.

Liedloff, Montealegre, and Todinca [46] gave a general theorem establishing fixed-parameter tractability for a large class of optimization problems. Let $\mathcal{C}_{poly}$ be a class of graphs having at most $poly(n)$ minimal separators. (Since every chordal graph has at most n minimal separators, the class of chordal graphs is a subclass of $\mathcal{C}_{poly}$.) Let φ be a Counting Monadic Second Order Logic (CMSO) formula, G be a graph, and $t \geq 0$ be an integer. Liedloff, Montealegre, and Todinca proved that on graph class $\mathcal{C}_{poly} + kv$, the generic problem, whose task is to maximize $|X|$ subject to the following constraints: (i) there is a set $F \subseteq V(G)$ such that $X \subseteq F$, (ii) the treewidth of $G[F]$ is at most t, and (iii) $(G[F], X) \models \varphi$, is solvable in time $\mathcal{O}(n^{\mathcal{O}(t)} \cdot f(t, \varphi, k))$, and thus is fixed-parameter tractable parameterized by k. The problem generalizes many classical algorithmic problems like Independent Set, Longest Induced Path, Induced Forest, and different packing problems, see [27].

Since the class $\mathcal{C}_{poly} + kv$ contains Chordal $- ke$, the work of Liedloff et al. [46] yields that all these problems are fixed-parameter tractable on Chordal $- ke$ graphs parameterized by $k + t + |\varphi|$. However, the theorem of Liedloff et al. cannot be used to derive our results. First, this theorem provides FPT algorithm only for problems of finding an induced subgraph of constant treewidth, which is not the case in our situation. Second, even for graphs of treewidth 0, their technique does not derive parameterized algorithms with subexponential running times.

Organization of the paper. In Section 2, we introduce basic notation. In Section 3, we discuss subexponential algorithms on Chordal $- ke$. Section 4 contains conditional lower bounds (assuming ETH) for Coloring and Clique on Chordal$-ke$. Section 5 is devoted to kernelization. We give lower bounds and construct a polynomial Turing kernel for Weighted Clique on Chordal $- ke$, and construct polynomial kernels for Independent Set on Interval $- ke$ and Split $- ke$. We conclude in Section 6 with some open problems. Due to space constraints, we either omit or just sketch the proofs. The details can be found in the full version of the paper [22].

2 Preliminaries

Graphs. All graphs considered in this paper are assumed to be simple, that is, finite undirected graphs without loops or multiple edges. We follow the standard graph theoretic notation and terminology (see, e.g., [19]). For each of the graph problems considered in this paper, we let $n = |V(G)|$ and $m = |E(G)|$ denote the number of vertices and edges, respectively, of the input graph G if it does not create confusion.

A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$, called a bag, such that the following three conditions hold: (i) $\bigcup_{t \in V(T)} X_t = V(G)$, (ii) for every $uv \in E(G)$, there exists a node t of T such that bag X_t contains both u and v, and (iii) for every $u \in V(G)$, the set $T_u = \{t \in V(T) | u \in X_t\}$, i.e., the set of nodes whose corresponding bags contain u, induces a connected subtree of T.

A graph G is *chordal* (or *triangulated*) if it does not contain an induced cycle of length at least four. The intersection graph of a family of intervals of the real line is called an *interval* graph; it is also said that G is an interval graph if there is a family of intervals (called *interval model* or *representation*) such that G is isomorphic to the intersection graph of this family. A graph G is said to be *split* if its vertex set can be partition into independent set and a clique. We refer to [9, 33] for detailed introduction to these graph classes. Notice that interval and split graphs are chordal.

A *triangulation* (or a *chordal complementation*) of a graph G is a chordal supergraph H with $V(H) = V(G)$. The *size* of the triangulation is $|E(H)| - |E(G)|$. The *fill-in* of a graph G, denoted $\mathsf{fill\text{-}in}(G)$, is the minimum integer k such that $G \in$ Chordal $- ke$ or, in other words, fill-in is the minimum number of edges whose addition makes the graph chordal. An *interval complementation* of a graph G is an interval supergraph H with $V(H) = V(G)$. Similarly, a *split complementation* of G is a split supergraph H and a *clique complementation* is a complete supergraph with $V(H) = V(G)$. The *size of interval (split, clique) completion* is $|E(H)|-|E(G)|$ and we denote the minimum size of an interval (split, clique) complementation by $\mathsf{int\text{-}comp}(G)$ ($\mathsf{split\text{-}comp}(G)$, $\mathsf{c\text{-}comp}(G)$ respectively). Clearly, G has an interval (split, clique) complementation of size at most k if and only if $G \in$ Interval $- ke$ (Split $- ke$, Complete $- ke$). It is easy to see that $\mathsf{c\text{-}comp}(G) = \binom{|V(G)|}{2} - |E(G)|$, and it is known that it is NP-hard to compute $\mathsf{fill\text{-}in}(G)$ [60] and $\mathsf{int\text{-}comp}(G)$ [30] and the same holds for $\mathsf{split\text{-}comp}(G)$ [50]. We will make use of the following observation.

▶ **Observation 1.** *For every graph G, c-comp$(G) \geq$ int-comp$(G) \geq$ fill-in(G) and c-comp$(G) \geq$ split-comp$(G) \geq$ fill-in(G).*

In particular, this observation implies that complexity lower bounds obtained for graph problems parameterized by the clique completion size hold for the same problems when they are parmeterized by the interval or split completion or by the fill-in, and the hardness for the interval or split completion parameterization implies the hardness for the fill-in parameterization.

Parameterized Complexity and Kernelization. We refer to the books [16, 20, 25] for the detailed introduction to the field. In the Parameterized Complexity theorey, the computational complexity is measured as a function of the input size n of a problem and an integer *parameter* k associated with the input. A parameterized problem is said to be *fixed parameter tractable* (or FPT) if it can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some function f. Parameterized complexity theory also provides tools for obtaining complexity lower bounds. Here we use lower bounds based on *Exponential Time Hypothesis (ETH)* formulated by Impagliazzo, Paturi and Zane [37, 38]. In particular, ETH implies that k-Satisfiability with n variables cannot be solved in time $2^{o(n)}n^{\mathcal{O}(1)}$.

A *compression* of a parameterized problem Π_1 into a (non-parameterized) problem Π_2 is a polynomial algorithm that maps each instance (I, k) of Π_1 with the input I and the parameter k to an instance I' of Π_2 such that (i) (I, k) is a yes-instance of Π_1 if and only if I' is a yes-instance of Π_2, and (ii) $|I'|$ is bounded by $f(k)$ for a computable function f. The output I' is also called a *compression*. The function f is said to be the *size* of the compression. A compression is *polynomial* if f is polynomial. A *kernelization* algorithm for a parameterized problem Π is a polynomial algorithm that maps each instance (I, k) of Π to an instance (I', k') of Π such that (i) (I, k) is a yes-instance of Π if and only if (I', k') is a yes-instance of Π, and (ii) $|I'| + k'$ is bounded by $f(k)$ for a computable function f. Respectively, (I', k') is a *kernel* and f is its *size*. A kernel is *polynomial* if f is polynomial.

Even if a paramterized problem admits no polynomial kernel up to some complexity conjectures, sometimes we can reduce it to solving of a polynomial number of instances of the same problem such that the size of each instance is bounded by a polynomial of the parameter. Let Π be a parameterized problem and let $f\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ be a computable function. A *Turing kernelization* or *Turing kernel* for Π of size f is an algorithm that decides whether an instance (I, k) of Π is a yes-instance in time polynomial in $|I| + k$, when given access to an oracle that decides whether (I', k') is a yes-instance of Π in a single step if $|I'| + k \leq f(k)$.

3 Subexponential algorithms for induced d-colorable subgraphs

To construct subexponential algorithms on Chordal $-$ ke, we consider tree decompositions such that each bag is "almost" a clique.

▶ **Definition 2.** *Let k be a nonnegative integer. A tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of a graph G is* k-almost chordal *if for every $t \in V(T)$, c-comp$(G[X_t]) \leq k$, that is, every bag can be converted to a clique by adding at most k edges.*

Note that every chordal graph has 0-almost chordal tree decomposition. Given a k-almost chordal tree decomposition, we are able to construct dynamic programming algorithms that are subexponential in k for various problems. The crucial property of the graphs in Chordal $-$ ke is that we are able to construct k-almost chordal tree decompositions for them in subexponential in k time by making use of the following result of Fomin and Villanger [28].

▶ **Proposition 3** ([28]). *Deciding whether graph G is in $\textsc{Chordal} - ke$ can be done in time $2^{\mathcal{O}(\sqrt{k}\log k)} + \mathcal{O}(k^2nm)$. Moreover, if $G \in \textsc{Chordal} - ke$, then the corresponding triangulation can be found in time $2^{\mathcal{O}(\sqrt{k}\log k)} + \mathcal{O}(k^2nm)$.*

Using Proposition 3 we obtain the following lemma.

▶ **Lemma 4.** *A k-almost chordal decomposition of a graph $G \in \textsc{Chordal} - ke$ with at most n bags can be constructed in time $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$.*

The crux of our subexponential algorithms is in the following combinatorial lemma.

▶ **Lemma 5.** *Let $d \geq 1$ be an integer. Let G be a graph and let $\mathcal{F}$ be a set of induced d-colorable subgraphs of G. Let $U \subseteq V(G)$ be a set of vertices of G such that $\mathsf{c\text{-}comp}(G[U]) \leq k$, that is, U can be made a clique by adding at most k edges. Then*

- *for every $F \in \mathcal{F}$,*

$$|U \cap V(F)| \leq \frac{3d + \sqrt{d^2 + 8dk}}{2},$$

and

- *the size of the projection of $\mathcal{F}$ on U, that is, the size of the family*

$$\mathcal{S} = \{S \mid S = U \cap V(F) \text{ for some } F \in \mathcal{F}\}$$

is at most $(1 + 2^{(\sqrt{1+8k}-1)/2} \cdot |U|)^d$.

Moreover, there is an algorithm that in time $2^{\mathcal{O}(d\sqrt{k})} \cdot n^{\mathcal{O}(d)}$ outputs a family of sets $\mathcal{S}' \supseteq \mathcal{S}$ such that each set from $\mathcal{S}'$ has at most $\frac{3d+\sqrt{d^2+8dk}}{2}$ vertices, the number of sets in $\mathcal{S}'$ is $(1 + 2^{(\sqrt{1+8k}-1)/2} \cdot n)^d$ and $G[S]$ is d-colorable for $S \in \mathcal{S}'$.

Proof sketch. We partition U into sets X and Y as follows. Let X be the vertices of U that have at least one non-neighbor in U. In other words, for every $v \in X$ there is $u \in U$ that is not adjacent to v. Two observations about set X will be useful. First, because U, and hence X, can be turned into a clique by adding at most k edges, we have that $|X| \leq 2k$. Second, the remaining vertices of U, namely, $Y = U \setminus X$, form a clique. For every set $S \in \mathcal{S}$, we define $S_X = X \cap S$ and $S_Y = Y \cap S$. Note that $S = S_X \cup S_Y$.

Because Y is a clique in G, no d-colorable subgraph from $\mathcal{F}$ can contain more than d vertices from Y. Hence, $|S_Y| \leq d$.

Let $x = |S_X|$. Because $G[S_X]$ is an induced subgraph of some d-colorable graph $F \in \mathcal{F}$, we have that $G[S_X]$ is d-colorable. On the other hand, since $\mathsf{c\text{-}comp}(G[U]) \leq k$, $G[S_X]$ can be turned into complete graph by adding at most k edges. These two conditions are used to estimate x. Let us remind that *Turán graph* is the complete d-partite graph on x vertices whose partition sets differ in size by at most 1. According to Turán's theorem, see e.g. [19], Turán graph has the maximum possible number of edges among all d-colorable graphs. The number of edges in Turán's graph is at most $\frac{1}{2}x^2\frac{d-1}{d}$. Thus,

$$\binom{x}{2} - k \leq |E(G[S_X])| \leq \frac{1}{2}x^2\frac{d-1}{d} \text{ and } k \geq \binom{x}{2} - \frac{1}{2}x^2\frac{d-1}{d} = \frac{x^2 - dx}{2d}.$$

Therefore, $x \leq \frac{d+\sqrt{d^2+8dk}}{2}$. We obtain that $|S| = |S_X| + |S_Y| \leq x + d \leq \frac{3d+\sqrt{d^2+8dk}}{2}$, which implies the first claim of the lemma.

To prove the second claim, let $H = G[U]$. Observe that the complement $\overline{H}$ has at most k edges. Consider $Z \subseteq V(H)$. If $|Z| \leq \frac{\sqrt{1+8k}+1}{2}$, then the minimum degree $\delta(\overline{H}[Z]) \leq \frac{\sqrt{1+8k}-1}{2}$. If $|Z| > \frac{\sqrt{1+8k}+1}{2}$, then

$$\delta(\overline{H}[Z]) \leq \frac{2|E(\overline{H}[Z])|}{|Z|} \leq \frac{4k}{\sqrt{1+8k}+1} = \frac{\sqrt{8k+1}-1}{2},$$

that is, the minimum degree of every induced subgraph of $\overline{H}$ is at most $\frac{\sqrt{8k+1}-1}{2}$. Therefore, $\overline{H}$ is $\frac{\sqrt{1+8k}-1}{2}$-degenerate. This implies that U has at most $2^{(\sqrt{1+8k}-1)/2} \cdot |U|$ independent in G subsets. Therefore, U has at most $(1+2^{(\sqrt{1+8k}-1)/2} \cdot |U|)^d$ subsets inducing d-colorable subgraphs. The same observations also allow to construct $\mathcal{S}'$ in $2^{\mathcal{O}(d\sqrt{k})} \cdot n^{\mathcal{O}(d)}$ time. ◀

Let G be a graph and let F be an induced d-colorable subgraph of G. Informally, Lemma 5 says that for a given a k-almost chordal tree decomposition, every bag of this tree decomposition contains roughly $\mathcal{O}(d+\sqrt{dk})$ vertices of F. This statement combined with dynamic programming over the tree decomposition could easily bring us to the algorithm computing a maximum d-colored subgraph of G in time $n^{\mathcal{O}(d+\sqrt{dk})}$. However, this is not what we are shooting for; such an algorithm is not fixed-parameter tractable with parameter k. This is where the second part of the lemma becomes extremely helpful. Let us look at the family of all d-colorable induced subgraphs of G. Then the number of different intersections of the graphs from this family with a single bag of the tree decomposition is bounded by $2^{\mathcal{O}(d\sqrt{k})}n^{\mathcal{O}(d)}$. This allows us to bound the number of "partial solutions" in the dynamic programming, which in turn brings us to a parameterized subexponential algorithm. As an example of the applicability of Lemma 5, we give an algorithm for the following generic problem.

Weighted d-colorable Subgraph

Input:	Graph G with weight function $w\colon V(G) \to \mathbb{R}$.
Task:	Find a properly d-colorable induced subgraph H of G of maximum weight $\sum_{v \in V(H)} w(v)$.

For $d = 1$, this is the problem of finding an independent set of maximum weight, the Weighted Independent Set problem. For $d = 2$, this is the problem of finding an induced bipartite subgraph of maximum weight, Weighted Bipartite Subgraph.

▶ **Theorem 6.** *Let $d \geq 1$ be an integer. For a given graph G with a nice k-almost chordal tree decomposition with $n^{\mathcal{O}(1)}$ bags, the Weighted d-colorable Subgraph problem is solvable in time $2^{\mathcal{O}(\sqrt{k} \cdot d \log d)} \cdot n^{\mathcal{O}(d)}$.*

Proof sketch. Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a k-almost chordal tree decomposition of G with $|V(T)| = n^{\mathcal{O}(1)}$. We perform dynamic programming over $\mathcal{T}$. Let us note that the width of the decomposition can be of order of n. As it is standard, we assume that T is rooted at some node r. For a node t of T, let V_t be the union of all the bags present in the subtree of T rooted at t, including X_t. For vertex sets $X \subset X'$ of graph G, we say that a coloring c of $G[X]$ is *extendible* to a coloring c' of $G[X']$, if for every $x \in X$, $c(v) = c'(v)$.

For every node t, every $S \subseteq X_t$ such that $G[S]$ is d-colorable, every mapping $c\colon S \to \{1, \ldots, d\}$ of $G[S]$, we define the following value:

$$\begin{aligned} \mathrm{cost}[t, S, c] \quad = \quad & \text{maximum possible weight of a set } \widehat{S} \text{ such that} \\ & S \subseteq \widehat{S} \subseteq V_t,\ \widehat{S} \cap X_t = S, \text{ and } c \text{ is a proper coloring of } G[S] \text{ extendible} \\ & \text{to a proper } d\text{-coloring of } G[\widehat{S}]. \end{aligned} \tag{1}$$

If c is not a proper coloring of $G[S]$ or if no such set $\widehat{S}$ exists, then we put $\mathrm{cost}[t, S, c] = -\infty$. We also put $\mathrm{cost}[t, \emptyset, c]$ be the maximum possible weight of a set $\widehat{S}$ such that $\widehat{S} \subseteq V_t$, $\widehat{S} \cap X_t = \emptyset$, and $G[\widehat{S}]$ is d-colorable. Our algorithm computes the tables of values of $\mathrm{cost}[t, S, c]$ bottom-up for $t \in V(T)$ from the leaves of T. Given the table for the root, it is straightforward to compute the maximum weight of a d-colorable induced subgraph of G. The corresponding optimal subgraph can be found by the standard backtracking arguments.

The proof of the correctness for this dynamic programming is very similar to the one provided normally for graphs of bounded treewidth. However, the running time analysis is based on Lemma 5. The crucial observation that allows us to obtain a subexponential running time is that the running time of our dynamic programming algorithm, up to some polynomial multiplicative factor, is dominated by the number of triples $[t, S, c]$. The number t is in $n^{\mathcal{O}(1)}$. Every set S should induce a d-colorable subgraph, so we can restrict our attention only to sets of the form $X_t \cap V(F)$ for some d-colorable graph F. By Lemma 5, each of these sets is of size at most $d + \frac{d+\sqrt{d^2+8dk}}{2}$ and the total number of such sets S for each bag X_t is is $2^{\mathcal{O}(d\sqrt{k})} \cdot n^{\mathcal{O}(d)}$ and they can be listed in time $2^{\mathcal{O}(d\sqrt{k})} \cdot n^{\mathcal{O}(d)}$. Thus, the number of d-colorings c of each of the sets S is $d^{\mathcal{O}(|S|)} = d^{\mathcal{O}(d+\sqrt{dk})}$. Hence the total running time of the dynamic programming is $2^{\mathcal{O}(\sqrt{k} \cdot d \log d)} \cdot n^{\mathcal{O}(d)}$. ◀

Combining Lemma 4 and Theorem 6, we immediately obtain the following corollary. We say that $A \subseteq \binom{V(G)}{2} \setminus E(G)$ is a *chordal modulator* if the graph obtained from G by adding the edges A is chordal.

▶ **Corollary 7.** *Weighted d-colorable Subgraph on a graph $G \in$ Chordal $- ke$ is solvable in time $2^{\mathcal{O}(\sqrt{k}(\log k + d \log d))} \cdot n^{\mathcal{O}(d)}$. Moreover, the problem can be solved in $2^{\mathcal{O}(\sqrt{k} \cdot d \log d)} \cdot n^{\mathcal{O}(d)}$ time if a chordal modulator of size at most k is given.*

In particular, we derive the following corollary for Weighted Independent Set and Weighted Bipartite Subgraph and the dual minimization problems. In the Weighted Vertex Cover, we are given a weighted graph G and the task is to find a vertex cover of minimum weight, that is, a set of vertices X such that every edge of G has at least one endpoint in G. Similarly, in the Weighted Odd Cycle Transversal, we are asked to find a set of vertices of minimum weight such that every cycle of odd length contains at least one vertex from the set. Clearly the complement of every independent set is a vertex cover, and the complement of every induced bipartite subgraph is an odd cycle transversal.

▶ **Corollary 8.** *Weighted Independent Set (Weighted Vertex Cover) and Weighted Bipartite Subgraph (Weighted Odd Cycle Transversal) on $G \in$ Chordal $- ke$ are solvable in time $2^{\mathcal{O}(\sqrt{k} \log k)} \cdot n^{\mathcal{O}(1)}$. Moreover, the problems can be solved in $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$ time if a chordal modulator of size at most k is given.*

The technique developed to prove Theorem 6 could be used to obtain subexponential algorithms for other problems beyond Weighted d-colorable Subgraph. These algorithms are very similar to the one from Theorem 6 and we mention here only few problems.

A *homomorphism* $G \to H$ from a graph G to a graph H is a mapping from the vertex set of G to that of H such that the image of every edge of G is an edge of H. In other words, a homomorphism $G \to H$ exists if and only if there is a mapping $g : V(G) \to V(H)$, such that for every edge $uv \in E(G)$, we have $g(u)g(v) \in E(H)$. Since there is a homomorphism from G to a complete graph K_d on d vertices if and only if G is d-colorable, the deciding whether there is a homomorphism from G to H is often referred as the H-coloring. Note that if G admits an H-coloring, then G is $|V(H)|$-colorable. The only difference between solving Weighted H-colorable Subgraph, the problem of finding the maximum weight induced subgraph admitting a homomorphism to H, with Theorem 6 is that the value $\mathrm{cost}[t, S, c]$ in (1) should be redefined by setting c be a homomorphism to H.

Similar running times could be derived for the variants of Weighted d-colorable Subgraph where some additional constrains on the properties of the d-colorable induced subgraph of minimum weight are imposed by some property $\mathcal{C}$. For example, property $\mathcal{C}$ could be that the required subgraph is connected, acyclic, regular, degenerate, etc. As far as the

information of the partial solution required for property $\mathcal{C}$ is characterized by set $S \subseteq V_t$ and all possible subsets of S or all permutations of S, we can solve the corresponding problem in time $2^{\mathcal{O}((d\sqrt{k})\log(dk))} \cdot n^{\mathcal{O}(d)}$. We summarize these observations within the following theorem.

▶ **Theorem 9.** *Let $d \geq 1$ be an integer and G be a graph from* Chordal $-$ ke. *Then*

- *Weighted H-colorable Subgraph can be solved in $2^{\mathcal{O}(\sqrt{k}(\log k + |V(H)| \log |V(H)|))} \cdot n^{\mathcal{O}(|V(H)|)}$ time,*
- *Weighted d-degenerate Subgraph is solvable in time $2^{\mathcal{O}((d\sqrt{k})\log(dk))} \cdot n^{\mathcal{O}(d)}$,*
- *Weighted Induced Forest (Weighted Feedback Vertex Set), Weighted Induced Tree, Weighted Induced Path (Cycle), and Weighted Induced Cycle Packing are solvable in $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$ time.*

In some cases, we can obtain a better running time if a chordal modulator of size at most k is given. For Weighted H-colorable Subgraph, this is done in the same way as for Weighted d-colorable Subgraph. For some other problems, like Weighted Induced Forest (Weighted Feedback Vertex Set), this would demand using recent techniques for dynamic programming on graphs of bounded treewidth for problems with connectivity contstraints (see [18, 6, 23, 53]) but this goes beyond the scope of our paper.

Another extension of Theorem 6 can be derived from the very recent results of Baste, Sau and Thilikos [3] about the $\mathcal{F}$-Minor Deletion problem on graphs of bounded treewidth. Recall that a graph F is a *minor* of G if a graph isomorphic to F can be obtained from G by vertex and edge deletions and edge contractions. Respectively, G is said to be *F-minor free* if G does not contain F as a minor. For a family of graphs $\mathcal{F}$, G is $\mathcal{F}$-minor free if G is F-minor free for every $F \in \mathcal{F}$. For a family $\mathcal{F}$, the task of $\mathcal{F}$-Minor Deletion is, given a graph G, to find a minimum set of vertices X such that $G - X$ is $\mathcal{F}$-minor free. Then $\mathcal{F}$-Minor Deletion is equivalent to $\mathcal{F}$-Minor Free Induced Subgraph, whose task is to find a maximum $\mathcal{F}$-minor free induced subgraph of G. A family of graphs $\mathcal{F}$ is *connected* if every $F \in \mathcal{F}$ is a connected graph. Baste et al. [3] obtained, in particular, the following result.

▶ **Proposition 10** ([3]). *Let $\mathcal{F}$ be a finite connected family of graphs. Then $\mathcal{F}$-Minor Deletion can be solved in time $2^{\mathcal{O}(w \log w)} \cdot n^{\mathcal{O}(1)}$ on graphs of treewidth at most w.*[1]

It is well-known (see, e.g., the book [51] for the inclusion relations between the classes of sparse graphs) that if $\mathcal{F}$ is a finite family, then there is a positive integer d such that every $\mathcal{F}$-minor free graph is d-degenerate. This means that for a finite family $\mathcal{F}$, $\mathcal{F}$-minor free graphs are d-colorable for some constant d that depends on $\mathcal{F}$ only. This allows us to use Lemma 5 and then combine our approach from Theorem 6 with the techniques of Baste et al. [3]. Using Lemma 10, we obtain the following theorem.

▶ **Theorem 11.** *Let $\mathcal{F}$ be a finite connected family of graphs. Let also G be a from* Chordal$-$ ke. *Then $\mathcal{F}$-Minor Deletion (or, equivalently, $\mathcal{F}$-Minor Free Induced Subgraph) can be solved in time $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$.*

For example, this framework encompasses such problems as Induced Planar Subgraph or Induced Outerplanar Subgraph whose task is to find a subgraph of maximum size that is planar or outerplanar, respectively.

With a small adjustment the dynamic programming could be applied to the problems with specific requirements on the complement of the maximum induced d-colored subgraph. For example, consider the following problem. A set of vertices $S \subseteq V(G)$ is a *connected vertex*

[1] the constants hidden in the big-$\mathcal{O}$ notation depend on $\mathcal{F}$.

cover if S is a vertex cover and $G[S]$ is connected. Then in the Weighted Connected Vertex Cover problem, we are given a graph G with a weight function $w\colon V(G) \to \mathbb{Z}^+$ and the task is to find a connected vertex cover in G of minimum weight. Similarly, Weighted Connected Feedback Vertex Set is the problem of finding a connected feedback vertex set of minimum weight. The complement of every vertex cover is an independent set, that is a 1-colorable subgraph, and the complement of every feedback vertex set is a forest, hence 2-colorable subgraph. While now the connectivity constraints are not on the maximum induced subgraph but on its complement our previous arguments can be adapted to handle these problems.

▶ **Theorem 12.** *Weighted Connected Vertex Cover and Weighted Connected Feedback Vertex Set are solvable in time* $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$ *on* Chordal $- ke$.

In this section, we discussed optimization problems but, in many cases, similar dynamic programming can be applied for counting problems. For example, we can compute the number of (inclusion) maximal independent sets, maximal bipartite subgraphs, minimal (connected) feedback vertex sets, minimal connected vertex covers in time $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$ on Chordal $- ke$.

4 Beyond induced d-colorable subgraphs

In Section 3, among other algorithms, we gave a subexponential (in k) algorithm on Chordal $- ke$ graphs that finds a maximum d-colorable subgraph. In particular, this also implies that for every fixed d, deciding whether a graph from Chordal $- ke$ is d-colorable, can be done in time subexponential in k. In this section we show that two fundamental problems, namely, Coloring and Clique, while still being FPT, are unlikely to be solvable in subexponential parameterized time.

First, we consider the Coloring problem, where the task is for a given graph G and positive integer ℓ, to decide whether the chromatic number of G is at most ℓ, that is, if G is ℓ-colorable. Note that ℓ here is not a fixed constant as in Section 3. Cai [11] proved that Coloring is FPT (parameterized by k) on Split $- ke$ graphs. The following theorem generalizes his result by showing that Coloring is FPT on a larger class Chordal $- ke$. Our approach is based on the dynamic programming which is similar to the one we used in Section 3.

▶ **Theorem 13.** *Coloring can be solved in time* $2^{\mathcal{O}(k\log k)} \cdot n^{\mathcal{O}(1)}$ *on* Chordal $- ke$ *graphs.*

On the other hand, it is unlikely that Coloring can be solved in subexponential in k time. For this, we show the complexity lower bound based on ETH. In fact, we prove a stronger claim.

▶ **Theorem 14.** *Coloring cannot be solved in time* $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ *on* Complete $- ke$ *graphs unless ETH fails.*

Next, we consider the Clique problem that asks, given a graph G and a positive integer ℓ, whether G has a clique of size at least ℓ. We show that Clique is FPT on Chordal $- ke$ when parameterized by k even for the weighted variant of the problem in Section 5 by demonstrating that the problem admits a Turing kernel. Here, we give a lower bound.

▶ **Theorem 15.** *Clique cannot be solved in time* $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ *on graphs in* Complete $- ke$ *unless ETH fails.*

We established that Coloring and Clique do not admit subexponential algorithms on Complete $-$ ke, when parameterized by k, unless ETH fails. By Observation 1, this yields that these problems do not admit subexponential algorithms on Chordal $-$ ke as well.

5 Kernelization on Chordal-ke

In this section we discuss kernelization of the problems considered in the previous section.

Jansen and Bodlaender in [39] and Bodlaender, Jansen and Kratsch in [8] proved that Weighted Independent Set, Weighted Vertex Cover, Weighted Bipartite Subgraph, Weighted Odd Cycle Transversal, Weighted Feedback Vertex Set and Clique do not admit a polynomial kernel parameterized by the size of the minimum vertex cover of a graph unless coNP $\subseteq$ NP/ poly. It is easy to observe that if G has a vertex cover of size at most k, then $\mathsf{split\text{-}comp}(G) \leq \binom{k}{2}$. Thus, by the results of [8, 39], we obtain the following proposition.

▶ **Proposition 16.** *Weighted Independent Set, Weighted Vertex Cover, Weighted Bipartite Subgraph, Weighted Odd Cycle Transversal, Weighted Feedback Vertex Set and Clique do not admit a polynomial in k kernel on Split $-$ ke graphs unless* coNP $\subseteq$ NP/ poly.

By Observation 1, these problems parameterized by k have no polynomial kernel on Chordal $-$ ke as well unless coNP $\subseteq$ NP/ poly.

These results do not refute the existence of polynomial Turing kernels. To demonstrate this, we show that Weighted Clique has such a kernel. The input of Weighted Clique contains a graph G together with a weight function $w\colon V(G) \to \mathbb{Z}^+$ and a nonnegative integer W, and the task is to decide whether G has a clique C of weight at least W.

▶ **Theorem 17.** *Weighted Clique on Chordal $-$ ke parameterized by k admits a Turing kernel with at most $16k^2$ vertices with size $\mathcal{O}(k^8)$.*

Proof sketch. Let (G, w, W) be an instance of Weighted Clique. We use the result of Natanzon, Shamir and Sharan [49] that fill-in admits a polyopt approximation. The approximation algorithm either correctly reports that $\mathsf{fill\text{-}in}(G) > k$ or returns a set of nonedges $A \subseteq \binom{V(G)}{2} \setminus E(G)$ of size at most $8k^2$ such that the graph G' obtained by adding the edges of A is a chordal graph. In the first case, we have that $G \notin$ Chordal $-$ ke. Assume that this is not the case. Then we use the well-known property of chordal graphs (see [32, 57]) that G' has at most n inclusion-maximal cliques $C_1, \ldots, C_r$ and they can be listed in linear time. Now we observe that K is a maximal clique of G if and only if K is a clique of $G[C_i]$ for some $i \in \{1, \ldots, r\}$. Moreover, every such K contains all the vertices of C_i that are not vertices of the pairs $\{u, v\} \in A$ with $u, v \in C_i$. Then the problem is reduced to finding solutions for $G[C'_i]$ for $i \in \{1, \ldots, r\}$, where each C'_i is the subset of C_i containing the vertices of pairs $\{u, v\} \in A$ with $u, v \in C_i$. Since $|C'_i| \leq 2|A| \leq 16k^2$, we obtain the upper bound on the number of vertices. To compress the weights and obtain the upper bound on the size, we use the the technique of Frank and Tardos [29], see also [21] for applications of this technique for kernelization. ◀

While Proposition 16 rules out the existence of a polynomial kernel for Weighted Independent Set on Chordal $-$ ke graphs, for unweighted Independent Set the existence of a polynomial kernel remains open. In what follows, we obtain polynomial kernels for Independent Set on two interesting subclasses of Chordal $-$ ke, namely Interval $-$ ke and Split $-$ ke. Let us note that again, by Proposition 16, the Weighted Independent Set problem admits no polynomial kernel on Split $-$ ke.

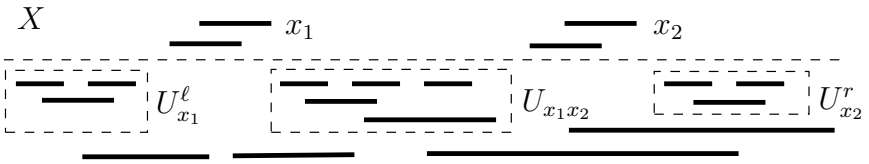


Figure 1 Structure of a maximum independent set in G.

We start with the kernel on Interval $-$ ke graphs. This kernel is the most technical part of the paper. In order to obtain the required kernel, we show that Independent Set parameterized by the size of interval completion of the input graph admits a polynomial compression into the Weighted Independent Set problem. (We state Weighted Independent Set as a decision problem, whose input contains a graph G with a weight function $w\colon V(G) \to \mathbb{Z}^+$ and a nonnegative integer W, and the task is to decide whether G has an independent set S with $w(S) \geq W$.) Then the standard arguments about polynomial compression of NP-complete problems, see e.g. [24, Theorem 1.6], yield the polynomial kernel for Independent Set on Interval $-$ ke graphs.

▶ **Theorem 18.** *Independent Set on $G \in$ Interval $-$ ke admits a compression of size $\mathcal{O}(k^{56})$ into Weighted Independent Set.*

Proof sketch. The proof is long and here we only sketch briefly the main ideas behind the algorithm. Let G be a graph and let $A \subseteq \binom{V(G)}{2} \setminus E(G)$ be a set of pairs of nonadjacent vertices such that the graph G' obtained from G by adding the edges from A becomes interval. Denote by X the set of end-vertices of the edges of A in G'.

Consider an interval model of G'. For each vertex $v \in V(G')$, let ℓ_v and r_v be, respectively, the left and right endpoint of the interval representing v. For each $v \in V(G')$, denote by G^{ℓ}_v and G^r_v the subgraphs of G' induced by the sets of vertices $U^{\ell}_v = \{u \in V(G') \mid r_u < \ell_v\}$ and $U^r_v = \{u \in V(G') \mid r_v < \ell_u\}$ respectively, and for every two distinct $u, v \in V(G')$, let G_{uv} be the subgraph induced by $U_{uv} = \{w \in V(G') \mid r_u < \ell_w \leq r_w < \ell_v\}$ (see Figure 1). For a graph H, denote by $I(H)$ a maximum independent set of H. Suppose that I is a maximum independent set of G and let $I \cap X = \{x_1, \ldots, x_s\}$ with $r_{v_{i-1}} < \ell_{v_i}$ for $i \in 2, \ldots, s$. Then it is possible to prove that

$$I' = I(G^{\ell}_{x_1}) \cup \Big(\bigcup_{i=2}^{s} I(G_{x_{i-1}x_i})\Big) \cup I(G^r_{x_s})$$

is a maximum independent set of G.

This allows us to create the following compression of the initial problem to an instance of Weighted Independent Set. Let $\mathcal{F}$ be the set of all induced subgraphs G^{ℓ}_v, G^r_v and G_{uv} for all $u, v \in X$. Consider the graph $\mathcal{G}$ with the set of vertices $X \cup \mathcal{F}$ with the following adjacencies: for distinct $u, v \in V(\mathcal{G})$, u and v are adjacent if and only if one of the following holds:

- $u, v \in X$ and $xy \in E(G)$,
- $u \in X$, $v \in \mathcal{F}$ and u is adjacent to a vertex of v in G,
- $u, v \in \mathcal{F}$ and the subgraphs u and v have either common or adjacent vertices in G.

We define the weight $w(v)$ for $v \in V(\mathcal{G})$ be one if $v \in X$ and set $w(v) = |I(v)|$ for $v \in \mathcal{F}$. It can be shown that G has an independent set of size at least W if and only if $\mathcal{G}$ has an independent set of weight at least W.

Unfortunately, the above arguments *do not work* for the following reason. We based our construction on the assumption that we know the resulting interval model. But computing such a model is an NP-hard task. Of course it would suffice even if we had a poly(OPT)

approximation algorithm for interval completion. That is, an algorithm producing in polynomial time an edge set A of polynomial in k size whose addition turns the input graph G into an interval graph. However the existence of such an approximation is a long-standing open problem. The best known result is the $\mathcal{O}(\log n)$ approximation algorithm of Rao and Richa [54] for the minimum number of edges of an interval supergraph of an n-vertex graphs. While at the end we were able to implement the above idea and obtain the required compression, the absence of a good approximation makes the proof way more difficult.

Given a graph G, we construct a vertex set X and a set of induced subgraphs $\mathcal{F}$ of $G - X$ such that the graph $\mathcal{G}$ defined above have the desired property: G has an independent set of size at least W if and only if $\mathcal{G}$ has an independent set of weight at least W. We start the construction of X using the algorithm of Natanzon, Shamir and Sharan [49] to approximate $\mathsf{fill\text{-}in}(G) \leq \mathsf{int\text{-}comp}(G)$. Initially, we set X be the set of vertices that are in the pairs of nonadjacent vertices returned by the algorithm. Then we apply a series of reduction rules that either solve the problem, or enhance X by adding vertices, or delete vertices of the graph. The reduction rules are based on the forbidden induced subgraph characterization of interval graphs given by Lekkerkerker and Boland [45]. This way we construct X of size $\mathcal{O}(k^3)$. Then we construct $\mathcal{F}$ of size $\mathcal{O}(k^{14})$ and define $\mathcal{G}$. Here again we use the technique of Frank and Tardos [29] to compress the weights. ◀

Since Weighted Independent Set is in NP and, consecutively, has a polynomial reduction to Independent Set that is NP-complete [30], by applying a standard trick, see e.g. [24, Theorem 1.6], we obtain the following corollary.

▶ **Corollary 19.** *Independent Set on $G \in$ Interval $-$ ke admits a polynomial kernel when parameterized by k.*

Finally, we show that Independent Set admits a polynomial kernel when parameterized by the split completion size. For this result, we exploit the result of Hammer and Simeone in [35] that Split Editing can be solved in polynomial time.

▶ **Theorem 20.** *Independent Set on Split $-$ ke admits a polynomial kernel with at most $2k^2(k+2)$ vertices when parameterized by k.*

6 Conclusion

In this paper, we initiated the study of parameterized subexponential and kernelization algorithms on Chordal $-$ ke graphs. The existence of such algorithms makes this graph class a very interesting object for studies. For other structural parameters, like treewidth or vertex cover, we have quite good understanding about the complexity of various optimization problems derived from general meta-theorems like Courcelle's or Pilipczuk's theorems [15, 53] and advanced algorithmic techniques [18, 17, 23]. We believe that further exploration of the complexity landscape of fill-in parameterization is an interesting research direction. If an optimization problem is NP-complete on chordal graphs, like Dominating Set, then on Chordal $-$ ke this problem is in Para-NP. On the other hand, even if a problem is solvable in polynomial time on chordal graphs, in theory, there is nothing preventing it from being Para-NP on Chordal $-$ ke. Is there a natural graph problem with this property? For many problems that are solvable in polynomial time on chordal graphs, we also established FPT algorithms on Chordal $-$ ke class. This does not exclude a possibility that there are problems that are not FPT parameterized by k but solvable in polynomial time for every fixed k. We do not know any such problem (in other words, the problem in class XP) yet. It will

be interesting to see, if there is any natural graph problem of such complexity. In addition, we proved that there are problems that are FPT on Chordal $-$ ke when parameterized by k and which cannot be solved in subexponential time unless ETH fails. We believe it would be exciting to obtain a logical characterization of problems that can be solved in subexponential time on Chordal $-$ ke when parameterized by k, similar to the classical Courcelle's theorem [15].

Some concrete open problems. Observe that for our subexponential dynamic programming algorithms, we only need a k-almost chordal tree decomposition of the input graph, that is, a decomposition where each bag can be made a clique by adding at most k edges. (Recall Definition 2.) The maximum of numbers $\mathsf{c\text{-}comp}(G[X_t]) \leq k$ can be significantly smaller than the minimum fill-in of a graph. For graphs in Chordal $-$ ke, we can find fill-in in a subexponential in k time by the algorithm of Fomin and Villanger [28]. However, we do not know if it is FPT in k to decide, whether a graph admits a k-almost chordal tree decomposition. And if yes, can it be done in subexponential time?

By the results of Natanzon, Shamir and Sharan [49], $\mathsf{fill\text{-}in}(G)$ can be approximated in polynomial time within a polyopt factor $8 \cdot \mathsf{fill\text{-}in}(G)$. Deciding whether $\mathsf{fill\text{-}in}(G) \leq k$ can be done in time $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot n^{\mathcal{O}(1)}$ by the results of Fomin and Villanger [28] (Proposition 3). Is there a constant-factor approximation FPT algorithm with running time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$? The existence of such an algorithm would speed-up our algorithms for several problems. For example, we would be able to solve Weighted Independent Set in $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$ time on Chordal-ke.

Finally, we proved that Independent Set on Interval $-$ ke and Split $-$ ke admit polynomial kernels when parameterized by k. We leave open the question whether or not this problem has a polynomial (Turing) kernel on Chordal $-$ ke.

References

1. Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Feedback vertex set inspired kernel for chordal vertex deletion. *ACM Trans. Algorithms*, 15(1):11:1–11:28, 2019. doi:10.1145/3284356.
2. Akanksha Agrawal, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Interval vertex deletion admits a polynomial kernel. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1711–1730. SIAM, 2019. doi:10.1137/1.9781611975482.103.
3. Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 951–970. SIAM, 2020.
4. Stephane Bessy and Anthony Perez. Polynomial kernels for Proper Interval Completion and related problems. *Information and Computation*, 231(0):89–108, 2013. doi:10.1016/j.ic.2013.08.006.
5. Ivan Bliznets, Fedor V. Fomin, Marcin Pilipczuk, and Michal Pilipczuk. Subexponential parameterized algorithm for interval completion. *ACM Trans. Algorithms*, 14(3):35:1–35:62, 2018. doi:10.1145/3186896.
6. Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.
7. Hans L. Bodlaender, Pinar Heggernes, and Yngve Villanger. Faster parameterized algorithms for minimum fill-in. *Algorithmica*, 61(4):817–838, 2011. doi:10.1007/s00453-010-9421-1.
8. Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.

9 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999. doi:10.1137/1.9780898719796.

10 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

11 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003. doi:10.1016/S0166-218X(02)00242-1.

12 Yixin Cao. Linear recognition of almost interval graphs. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1096–1115. SIAM, 2016.

13 Yixin Cao. Unit interval editing is fixed-parameter tractable. *Inf. Comput.*, 253:109–126, 2017. doi:10.1016/j.ic.2017.01.008.

14 Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(3):21:1–21:35, 2015. doi:10.1145/2629595.

15 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

16 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

17 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 301–310. ACM, 2013.

18 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 150–159. IEEE, 2011.

19 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

20 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

21 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. *J. Comput. Syst. Sci.*, 84:1–10, 2017. doi:10.1016/j.jcss.2016.06.004.

22 Fedor V. Fomin and Petr A. Golovach. Subexponential parameterized algorithms and kernelization on almost chordal graphs. *CoRR*, abs/2002.08226, 2020. arXiv:2002.08226.

23 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

24 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of Parameterized Preprocessing*. Cambridge University Press, 2018.

25 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization*. Cambridge University Press, Cambridge, 2019. Theory of parameterized preprocessing.

26 Fedor V. Fomin, Saket Saurabh, and Yngve Villanger. A polynomial kernel for proper interval vertex deletion. *SIAM J. Discrete Math.*, 27(4):1964–1976, 2013. doi:10.1137/12089051X.

27 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015. doi:10.1137/140964801.

28 Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM J. Computing*, 42(6):2197–2216, 2013. doi:10.1137/11085390X.

29 András Frank and Éva Tardos. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

30 Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

31 Fanica Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1(2):180–187, 1972. doi:10.1137/0201013.

32 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combinatorial Theory Ser. B*, 16:47–56, 1974. doi:10.1016/0095-8956(74)90094-x.

33 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

34 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, second edition, 1993. doi:10.1007/978-3-642-78240-4.

35 Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981. doi:10.1007/BF02579333.

36 Pinar Heggernes. Minimal triangulations of graphs: a survey. *Discrete Math.*, 306(3):297–317, 2006. doi:10.1016/j.disc.2005.12.003.

37 Russell Impagliazzo and Ramamohan Paturi. Complexity of k-sat. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*, pages 237–240. IEEE Computer Society, 1999. doi:10.1109/CCC.1999.766282.

38 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity. *J. Computer and System Sciences*, 63(4):512–530, 2001.

39 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.

40 Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. *SIAM J. Discrete Math.*, 32(3):2258–2301, 2018. doi:10.1137/17M112035X.

41 Bart MP Jansen and Stefan Kratsch. Data reduction for graph coloring problems. *Information and Computation*, 231:70–88, 2013.

42 Haim Kaplan, Ron Shamir, and Robert E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28:1906–1922, May 1999. doi:10.1137/S0097539796303044.

43 Haim Kaplan, Ron Shamir, and Robert Endre Tarjan. Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and physical mapping. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 780–791. IEEE, 1994.

44 J. Mark Keil. Finding hamiltonian circuits in interval graphs. *Inf. Process. Lett.*, 20(4):201–206, 1985. doi:10.1016/0020-0190(85)90050-X.

45 C. G. Lekkerkerker and J. Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fund. Math.*, 51:45–64, 1962/1963. doi:10.4064/fm-51-1-45-64.

46 Mathieu Liedloff, Pedro Montealegre, and Ioan Todinca. Beyond classes of graphs with "few" minimal separators: FPT results through potential maximal cliques. *Algorithmica*, 81(3):986–1005, 2019. doi:10.1007/s00453-018-0453-2.

47 Dániel Marx. Parameterized coloring problems on chordal graphs. *Theoretical Computer Science*, 351(3):407–424, 2006.

48 Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010. doi:10.1007/s00453-008-9233-8.

49 Assaf Natanzon, Ron Shamir, and Roded Sharan. A polynomial approximation algorithm for the minimum fill-in problem. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 41–47. ACM, 1998. doi:10.1145/276698.276710.

50 Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001. doi:10.1016/S0166-218X(00)00391-7.

51 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

52 S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3:119–130, 1961.

53 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: a logical approach. *CoRR*, abs/1104.3057, 2011. arXiv:1104.3057.

54 Satish Rao and Andréa W. Richa. New approximation techniques for some linear ordering problems. *SIAM J. Comput.*, 34(2):388–404, 2004. doi:10.1137/S0097539702413197.

55 D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1972.

56 Yasuhiko Takenaga and Kenichi Higashide. Vertex coloring of comparability $+ke$ and- $-ke$ graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 102–112. Springer, 2006.

57 Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984. doi:10.1137/0213035.

58 Lieven Vandenberghe, Martin S Andersen, et al. Chordal graphs and semidefinite optimization. *Foundations and Trends® in Optimization*, 1(4):241–433, 2015.

59 Yngve Villanger, Pinar Heggernes, Christophe Paul, and Jan Arne Telle. Interval completion is fixed parameter tractable. *SIAM J. Comput.*, 38(5):2007–2020, 2009. doi:10.1137/070710913.

60 M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.

On the Complexity of Recovering Incidence Matrices

Fedor V. Fomin
University of Bergen, Norway
fomin@ii.uib.no

Petr Golovach
University of Bergen, Norway
Petr.Golovach@ii.uib.no

Pranabendu Misra
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
pmisra@mpi-inf.mpg.de

M. S. Ramanujan
University of Warwick, Coventry, UK
R.Maadapuzhi-Sridharan@warwick.ac.uk

Abstract

The incidence matrix of a graph is a fundamental object naturally appearing in many applications, involving graphs such as social networks, communication networks, or transportation networks. Often, the data collected about the incidence relations can have some slight noise. In this paper, we initiate the study of the computational complexity of recovering incidence matrices of graphs from a binary matrix: given a binary matrix M which can be written as the superposition of two binary matrices L and S, where S is the incidence matrix of a graph from a specified graph class, and L is a matrix (i) of small rank or, (ii) of small (Hamming) weight. Further, identify all those graphs whose incidence matrices form part of such a superposition. Here, L represents the noise in the input matrix M. Another motivation for this problem comes from the Matroid Minors project of Geelen, Gerards and Whittle, where perturbed graphic and co-graphic matroids play a prominent role. There, it is expected that a perturbed binary matroid (or its dual) is presented as $L + S$ where L is a low rank matrix and S is the incidence matrix of a graph. Here, we address the complexity of constructing such a decomposition.

When L is of small rank, we show that the problem is NP-complete, but it can be decided in time $(mn)^{O(r)}$, where m, n are dimensions of M and r is an upper-bound on the rank of L. When L is of small weight, then the problem is solvable in polynomial time $(mn)^{O(1)}$. Furthermore, in many applications it is desirable to have the list of all possible solutions for further analysis. We show that our algorithms naturally extend to enumeration algorithms for the above two problems with delay $(mn)^{O(r)}$ and $(mn)^{O(1)}$, respectively, between consecutive outputs.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Enumeration

Keywords and phrases Graph Incidence Matrix, Matrix Recovery, Enumeration Algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.50

1 Introduction

Suppose that we are given a large binary data matrix M, and we know that this matrix is of the form $M = L + S$, where S is the incidence matrix of an undirected graph and L is a sparse binary matrix. We assume that the sums are taken over GF(2) and thus $1 + 1 = 0$. See Section 2 for a formal definition of incidence matrices. Now, consider the following two computational questions.

Question 1: How efficiently can we recover *some* incidence matrix S such that $M = L + S$ as described above?

Question 2: How efficiently can we recover *all* possible graphs whose incidence matrices form part of such a superposition $M = L + S$?

Our main motivation for studying these questions comes from the study of perturbed graphic matroids, which naturally arise in many settings. A prominent example is the emerging Matroid Minors Project of Geelen, Gerards, and Whittle [5], mirroring the Graph Minors project. Let us recall that binary matroids are represented by a matrix over $\mathsf{GF}(2)$, and graphic matroids are a sub-class of binary matroids defined by the incidence matrices of graphs. Then, for each proper minor-closed class $\mathcal{M}$ of binary matroids, there exists a non-negative integer r such that every well-connected matroid $M \in \mathcal{M}$ is either a perturbation of a graphic matroid or a co-graphic matroid [5]. In other words, either M or $M^\star$ (the dual matroid) can be decomposed as $L + S$ where S is the incidence matrix of a graph and L is a matrix of rank at most r. Matroid Minors and perturbed matroids are expected to have important applications in matroid algorithms, similar to the applications of Graph Minors and H-minor free decompositions in graph algorithms. In the applications of perturbed binary matroids [6, 4], it is expected that a decomposition of the perturbed matroid M into $L + S$ is given as a part of the input. A natural computational question that arises here is the construction of such a decomposition for a given matroid.

Such a problem can be seen as the problem of the recovery of incidence matrices, which is a fundamental representation of graphs. Often the data collected about the network incidence relations contains some slight noise or errors. Our objective is to decompose the data matrix M into two sparse matrices L and S, where L is a low weight matrix, that is matrix with a small number of ones, representing the error or noise, and S is the incidence matrix of some graph from a specified graph class.

Besides applications in matroid minors theory, the problem of superposing binary matrix M in $L+S$ has strong connections to robust Principal Component Analysis (PCA), a popular approach to robust subspace learning by the decomposition of the data matrix into low rank and sparse components. Here we have as data a matrix M, which is the superposition of a low rank component L and a sparse component S. In particular, this approach to robust PCA was popularized by Candès et al. [1], Wright et al. [11], and Chandrasekaran et al. [2]. Thus our problem can be seen as a variant of robust PCA for binary matrices, with additional constraint on sparse component S of being incidence matrix of some graph. Other variants of robust PCA when the structure of the sparse matrix S is imposed from the structure of some graph were studied in [10, 12].

In this paper, we initiate the study of the computational complexity of graph recovering problems and identify several settings which imply efficient algorithms for the questions stated above. Moreover, we also provide complexity theoretic lower bounds for certain other settings that preclude polynomial-time solvability of either question. Let us formally describe our contributions and state our results.

Our contributions. Our first contribution is the formulation of a pair of generic Graph Recovering problems dealing with two common notions of sparsity for L. Let $\mathcal{C}$ be a fixed class of simple graphs.

(MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY
Input: An $n \times m$ binary matrix M and $r \in \mathbb{N}$.
Question: Decide whether $M = L + S$, where L is a binary matrix of $\mathsf{GF}(2)$-rank at most r, S is the incidence matrix of a graph in $\mathcal{C}$ and the sums are taken over $\mathsf{GF}(2)$.

Let $\|L\|$ denote the *(Hamming) weight* of a matrix L, i.e. the number of non-zero entries in L.

(MIN-WEIGHT, $\mathcal{C}$)-GRAPH RECOVERY
Input: An $n \times m$ binary matrix M and $r \in \mathbb{N}$.
Question: Decide whether $M = L + S$, where L is a binary matrix of weight at most r, S is the incidence matrix of a graph in $\mathcal{C}$ and the sums are taken over $\mathsf{GF}(2)$.

Our second contribution is establishing the computational complexity of (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY and (MIN-WEIGHT, $\mathcal{C}$)-GRAPH RECOVERY for some fundamental graph classes and obtain the following upper and lower bounds.

(a) We show that if $\mathcal{C}$ is one of {simple graphs, acyclic graphs, trees, arboricity-d graphs, connected graphs}, then (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY can be solved in time $(mn)^{\mathcal{O}(r)}$.
(b) We show that if $\mathcal{C}$ is one of {simple graphs, acyclic graphs, trees, connected graphs}, then (MIN-WEIGHT, $\mathcal{C}$)-GRAPH RECOVERY is polynomial-time solvable, i.e., in time $(mn)^{\mathcal{O}(1)}$.
(c) We show that the running time in Result (a) cannot be improved to a purely polynomial dependence on m and n (as in Result (b)) unless $\mathsf{P}{=}\mathsf{NP}$. Specifically, we show that if $\mathcal{C}$ is any of {simple graphs, acyclic graphs, trees, arboricity-d graphs, connected graphs}, then the (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY problem is NP-complete.

A key feature of the methodology we introduce to obtain our algorithms is that it not only answers Question 1 (i.e., the decision problem) for the settings above, it also naturally extends to an algorithm that addresses the significantly harder Question 2 (i.e., the enumeration of all solutions) in these settings. Specifically, we show that in the case of Result (a) above, we can also *enumerate* all possible solution matrices S (if any exist) with a delay of time $(mn)^{\mathcal{O}(r)}$ between consecutive outputs. Similarly, we show that in the case of Result (b), we can also enumerate all possible solution matrices S (and equivalently, the corresponding graphs) with polynomial delay, i.e., a delay of time $(mn)^{\mathcal{O}(1)}$ between consecutive outputs. The importance of enumeration lies in the fact that a solution $M = S + L$ may not be unique, and further analysis of the list of all solutions is required.

Roadmap of the paper

Following the introduction of the requisite notation, we first describe the methodology behind our enumeration algorithms, where we reduce the problem of enumerating all solutions to one of *deciding* whether a solution exists for appropriate annotated variants of (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY and (MIN-WEIGHT, $\mathcal{C}$)-GRAPH RECOVERY. We then present our decision algorithms for the aforementioned annotated problems. The precise formulation of these problems involves some notation and we do not go into more details here. The algorithms for the annotated variant of (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY are centred around a novel application of Matroid Intersection involving carefully chosen matroids. On the other hand, the algorithms for (MIN-WEIGHT, $\mathcal{C}$)-GRAPH RECOVERY are designed through an

intricate analysis of the structure of bridges (or cut edges) in graphs and appropriate repeated reassignments of these to columns of our input matrix in a manner reminiscent of the "augmenting path" step in maximum matching algorithms. Finally, we give our NP-completeness result for (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY.

2 Preliminaries

Matrices and Linear Algebra

For $\ell \in \mathbb{N}$, we use $[\ell]$ to denote the set $\{1, \ldots, \ell\}$. For a matrix M, we denote the set of rows of M by $\text{rows}(M)$ and the set of column of M by $\text{cols}(M)$. For a set $P \subseteq \text{cols}(M)$, we denote by $M[P]$ the submatrix of M induced by the columns in P. Consider the set $V = \{v_1, v_2, \ldots, v_k\}$ of vectors over $\mathbb{F}$. The vectors in V are said to be *linearly dependent* if there exist elements $a_1, a_2, \ldots, a_k \in \mathbb{F}$, not all zero, such that $\sum_{i=1}^{k} a_i v_i = 0$. Otherwise these vectors are said to be *linearly independent*. The *rank* of a matrix is the cardinality of a maximum sized set of columns which are linearly independent. The vector space *spanned by* V is the set of all linear combinations of vectors in V and is denoted by $\text{span}(V)$. The vector space spanned by the set of columns of a matrix M over the field $\mathbb{F}$ is defined as $\text{span}(\text{cols}(M))$ and is denoted by $\text{col-span}(M)$. We say a matrix A has dimension $n \times m$ (or is an $n \times m$ matrix) if A has n rows and m columns. In this paper we always view the elements of a binary matrix as elements of $\mathsf{GF}(2)$, the Galois field of two elements. Then the $\mathsf{GF}(2)$-rank of a binary $n \times m$ matrix A is the minimum r such that $A = U \cdot V$, where U and V are $n \times r$ and $r \times m$ binary matrices respectively, and arithmetic operations are over $\mathsf{GF}(2)$.

Graphs

For standard graph theoretic terminology and notation, we refer to [3]. For an undirected graph G, we denote by $\text{inc}(G)$, the *incidence matrix* of G which is the $|V(G)| \times |E(G)|$ binary matrix M with a row for each vertex and a column for each edge such that for every $v \in V(G)$ and $e \in E(G)$, $M[v, e] = 1$ if and only if v is an endpoint of e. When considering the incidence matrix M of an unspecified graph, we denote by $\text{inc}^{-1}(M)$ the graph G where $V(G) = \text{rows}(M)$, $E(G) = \text{cols}(M)$ and whose incidence matrix is precisely M. For $\ell \in \mathbb{N}$, we denote by K_ℓ the complete graph on ℓ vertices. We assume without loss of generality that the vertices of K_ℓ are labelled 1 to ℓ and unless otherwise specified, the columns in $\text{inc}(K_n)$ are assumed to be arranged in lexicographically increasing order based on the endpoints of the corresponding edges.

Matroids

We recall relevant definitions related to matroids. For a broader overview on matroids, we refer to [9].

▶ **Definition 1.** *A matroid M is a pair $(E, \mathcal{I})$, where E is a set called the universe or ground set, and $\mathcal{I}$ is a family of subsets of E, called independent sets, with the following three properties : (i) $\emptyset \in \mathcal{I}$, (ii) if $I \in \mathcal{I}$ and $I' \subseteq I$, then $I' \in \mathcal{I}$, and (iii) if $I, J \in \mathcal{I}$ and $|I| < |J|$, then there is $e \in J \setminus I$ such that $I \cup \{e\} \in \mathcal{I}$.*

Any set $F \subseteq E$, $F \notin \mathcal{I}$, is called a *dependent set* and an inclusion-wise maximal set B such that $B \in \mathcal{I}$ is called a *basis*. The cardinality of a basis in a matroid M is called the *rank* of M and is denoted by $\mathsf{rank}(M)$. The rank function of M, denoted by $\text{rank}_M()$ (with the reference to M omitted when clear from the context), is a function from 2^E to $\mathbb{N} \cup \{0\}$ and is defined as, $\text{rank}(S) = \max_{S' \subseteq S, S' \in \mathcal{I}} |S'|$ for every $S \subseteq E$.

▶ **Definition 2.** *Let A be a matrix over a field $\mathbb{F}$ and E be the set of columns of A. The pair $(E, \mathcal{I})$, where $\mathcal{I}$ defined as follows, is a matroid. For every $X \subseteq E$, $X \in \mathcal{I}$ if and only if the columns of A corresponding to X are linearly independent over $\mathbb{F}$. Such matroids are called* linear matroids. *If a matroid M can be defined by a matrix A over a field $\mathbb{F}$, then we say that the matroid is* representable over $\mathbb{F}$ *and A is a linear representation of M.*

▶ **Definition 3** (Elongation of matroids)**.** *Let $M = (E, \mathcal{I})$ be a matroid and $k \in \mathbb{N}$. Suppose that $\text{rank}(M) \leq k \leq |E|$. A k-elongation matroid M_k of M is a matroid with the universe as E and $S \subseteq E$ is a basis of M_k if and only if, it contains a basis of M and $|S| = k$.*

Observe that the rank of the matroid M_k in the definition above is k.

▶ **Definition 4** (Direct-sum of matroids)**.** *Consider a set of ℓ matroids $\{M_i = (E_i, \mathcal{I}_i) \mid i \in [\ell]\}$, where $E_i \cap E_j = \emptyset$ for every $i \neq j$. The* direct-sum *of these matroids is the matroid $M = (\bigcup_{i \in [\ell]} E_i, \mathcal{I})$ where $I \in \mathcal{I}$ if and only if $I = \bigcup_{i \in [\ell]} I_i$ where $I_i \in \mathcal{I}_i$ for every $i \in [\ell]$.*

Given the representations of ℓ linear matroids over $\mathbb{F}$, it is straightforward to obtain a representation for their direct-sum over $\mathbb{F}$ and this can be done in polynomial time.

For a finite field $\mathbb{F}$, $\mathbb{F}[X]$ denotes the *ring of polynomials in X over $\mathbb{F}$* and $\mathbb{F}(X)$ denotes the *field of fractions* of $\mathbb{F}[X]$. A vector v over a field $\mathbb{F}$ is a tuple of elements from $\mathbb{F}$.

The next two propositions follow from [8].

▶ **Proposition 5** ([8])**.** *Let M be a linear matroid of rank r, over a ground set of size n, which is representable over a field $\mathbb{F}$. Given $k \geq r$, we can compute a representation of the k-elongation of M, over the field $\mathbb{F}(X)$ in $\mathcal{O}(nrk)$ field operations over $\mathbb{F}$.*

▶ **Proposition 6** ([8])**.** *Given a linear representation of the k-elongation of M over the field $\mathbb{F}(X)$ and a set of columns in the representation matrix, one can test for linear dependence of this set in polynomial time.*

In the Weighted Matroid Intersection problem, the input is a pair of matroids $M_1 = (E, \mathcal{I}_1), M_2 = (E, \mathcal{I}_2)$ and a weight function $w : E \to \mathbb{N} \cup \{0\}$ and the objective is to find a maximum-weight common independent set in the two matroids. That is, the goal is to compute $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ such that $\Sigma_{e \in I} w(e)$ is maximized. We will use the polynomial-time algorithm for this problem stated in Proposition 7.

▶ **Proposition 7** ([7])**.** *Given two general matroids M_1 and M_2 over the element set E, one can solve Weighted Matroid Intersection in time $O(\tau W n r^{1.5})$, where W is the largest weight assigned to an element, $n = |E|$, $r = min\{\text{rank}(M_1), \text{rank}(M_2)\}$ and τ is the time required to test whether a given subset of E is independent in M_1 and M_2.*

Graphic Matroids. These are matroids that arise from graphs in the following way. The graphic matroid $M(G)$ of an undirected graph G has universe $E(G)$ and a set $S \subseteq E(G)$ is independent if the subgraph with vertex set $V(G)$ and edge set S is acyclic. Graphic matroids are representable over every field and a representation of $M(G)$ over $\mathsf{GF}(2)$ can be computed in time polynomial in the size of G [9]. Observe that testing whether a set is independent in the matroid $M(G)$ can be done in polynomial time if one is given either the linear representation of the matroid or the graph G.

Transversal Matroids. For a bipartite graph $G = (X, Y, E)$, we can define a matroid M with universe X, where a set $S \subseteq X$ is independent if there exists a matching in G such that every vertex in S is an endpoint of a matching edge. We denote this matroid by $\text{Tr}(G, X)$.

Observe that if G is given, then one can use a *Maximum Matching* algorithm as a subroutine to determine in polynomial time (in the size of G) whether a given set is independent in $\mathrm{Tr}(G, X)$.

Gammoids. Let D be a digraph and $S, T \subseteq V(D)$. Then a *gammoid* with respect to D and S on ground set T is a matroid $(T, \mathcal{I})$, where $\mathcal{I}$ is defined as follows. For any $T' \subseteq T$, $T' \in \mathcal{I}$, if and only if there are $|T'|$ vertex disjoint paths which originate in S and end in T'. Observe that if D, S are given, then one can use a *Maximum Flow* algorithm as a subroutine to determine in polynomial time (in the size of D) whether a given set is independent in this gammoid.

▶ **Definition 8.** *A pair of matroids $M_1 = (E_1, \mathcal{I}_1), M_2 = (E_2, \mathcal{I}_2)$ are said to be* isomorphic *if there is a bijection $\phi : E_1 \to E_2$ such that for every $S \subseteq E_1$, $S \in \mathcal{I}_1$ if and only if $\phi(S) \in \mathcal{I}_2$ where the function ϕ is extended in the natural way to subsets of E_1. Equivalently, we say that M_1 and M_2 are isomorphic under the bijection ϕ.*

3 The enumeration algorithms for (min-rank, $\mathcal{C}$)-Graph Recovery and (min-weight, $\mathcal{C}$)-Graph Recovery

This section is devoted to our enumeration algorithms. We begin by reducing the enumeration task to one of deciding an annotated version of the problem at hand.

3.1 Reducing enumeration to decision

Our enumeration strategy is based on first designing an algorithm for the decision version of an "annotated" version of these problems. In this version, certain columns of the input matrix M already have their corresponding columns in the hypothetical solution matrix S (equivalently, in the matrix L) identified and the goal is to check whether this partial mapping can be extended to a full solution. We now formally define the annotated versions of these problems.

In the Extended (min-rank, $\mathcal{C}$)-Graph Recovery problem, we are given an $n \times m$ binary matrix M, number $r \in \mathbb{N}$, a set $C_M \subseteq \mathrm{cols}(M)$ and an injective mapping $\tau : C_M \to \mathsf{GF}(2)^n$ and the task is to *decide whether there exist* binary matrices L and S such that $M = L + S$, S is an incidence matrix of a simple graph belonging to the class $\mathcal{C}$, $\mathrm{rank}(L) \leq r$ and moreover, for every $x \in C_M$, $S[\{x\}] = \tau(x)$. Similarly in the Extended (min-weight, $\mathcal{C}$)-Graph Recovery problem, the input is the same and the the goal is to decide whether there exist binary matrices L and S such that $M = L + S$, S is an incidence matrix of a simple graph belonging to the class $\mathcal{C}$, $\|L\| \leq r$ and for every $x \in C_M$, $S[\{x\}] = \tau(x)$.

We remark that if, in an instance (M, r, C_M, τ) of either problem, it holds that $C_M = \emptyset$, then the corresponding mapping $\tau : C_M \to \mathsf{GF}(2)^n$ is denoted by $\tau_\emptyset$. Moreover, notice that by setting $C_M = \emptyset$ and $\tau = \tau_\emptyset$, we have that (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY ((MIN-WEIGHT, $\mathcal{C}$)-GRAPH RECOVERY) is a special case of Extended (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY (respectively, Extended (MIN-WEIGHT, $\mathcal{C}$)-GRAPH RECOVERY).

Our enumeration algorithms are based on the following pair of algorithms for Extended (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY and Extended (MIN-WEIGHT, $\mathcal{C}$)-GRAPH RECOVERY.

▶ **Lemma 9.** *For every $\mathcal{C} \in \{$simple graph, forest, connected graph, arboricity-d graph$\}$, there is an algorithm $\mathcal{C}$-$\mathcal{D}$ that runs in time $(mn)^{\mathcal{O}(r)}$ and correctly decides whether or not the given input (M, r, C_M, τ) is a yes-instance of Extended (min-rank, $\mathcal{C}$)-Graph Recovery.*

Algorithm 1 Algorithm $\mathcal{C}$-$\mathcal{E}$.

```
if C_M = [m] then
    Define the matrix S as S[{x}] = τ(x) for every x ∈ [m].
    if inc^{-1}(S) ∈ C and rank(M + S) ≤ r then
        Output S
    end
end
return
x ← min_{p∈[m]}{p ∉ C_M}
for each y ∈ cols(inc(K_n)) such that τ^{-1}(y) is undefined do
    if C-D((M, r, C_M ∪ {x}, τ ∪ {x ⟼ y})) returns Yes then
        C-E((M, r, C_M ∪ {x}, τ ∪ {x ⟼ y}))
    end
end
return
```

▶ **Lemma 10.** *For every $\mathcal{C} \in \{$simple graph, forest, connected graph$\}$, there is an algorithm $\mathcal{C}$-$\mathcal{D}'$ that runs in time $(mn)^{\mathcal{O}(1)}$ and correctly decides whether or not the given input (M, r, C_M, τ) is a yes-instance of Extended $(\mathsf{min\text{-}weight}, \mathcal{C})$-Graph Recovery.*

We first assume Lemma 9 and Lemma 10, and present our enumeration algorithms (Theorem 11 and Theorem 12).

▶ **Theorem 11.** *For every $\mathcal{C} \in \{$simple graph, forest, connected graph, arboricity-d graph$\}$, there is an algorithm $\mathcal{C}$-$\mathcal{E}$ that, on input (M, r, C_M, τ), outputs precisely those matrices S such that $M = S + L$ for some binary matrix L of $\mathsf{GF}(2)$-rank at most r, S is an incidence matrix of a simple graph in $\mathcal{C}$ such that for every $x \in C_M$, $S[\{x\}] = \tau(x)$. Moreover, the delay between successive outputs is $(mn)^{O(r)}$ and each such matrix S is output exactly once.*

Proof. The algorithm $\mathcal{C}$-$\mathcal{E}$ is described in Algorithm 1. The **for** loop is executed at most $\binom{n}{2}$ times in any single call to the algorithm and the recursive call to $\mathcal{C}$-$\mathcal{E}$ is made precisely when there is at least one solution to be output for a specific extension of C_M and τ. This is witnessed by the positive answer returned by the execution of the decision algorithm $\mathcal{C}$-$\mathcal{D}$. Finally, since the depth of the recursion is bounded by m, the first part of the lemma follows. The fact that this algorithm outputs precisely the required matrices (exactly once each) with delay bounded by $(mn)^{O(r)}$ follows from the correctness and running time bound of the algorithm $\mathcal{C}$-$\mathcal{D}$ in Lemma 9. ◀

The enumeration algorithm for (MIN-WEIGHT, $\mathcal{C}$)-GRAPH RECOVERY is analogous to that for (MIN-RANK, $\mathcal{C}$)-GRAPH RECOVERY (building on Lemma 10 instead of Lemma 9) and so we only state the theorem, omitting the proof.

▶ **Theorem 12.** *For every $\mathcal{C} \in \{$simple graph, forest, connected graph, arboricity-d graph$\}$, there is an algorithm $\mathcal{C}$-$\mathcal{E}'$ that, on input (M, r, C_M, τ), outputs precisely those matrices S such that $M = S + L$ for some binary matrix L such that $\|L\| \leq r$, S is an incidence matrix of a simple graph in $\mathcal{C}$ such that for every $x \in C_M$, $S[\{x\}] = \tau(x)$. Moreover, the delay between successive outputs is $(mn)^{O(1)}$ and each such matrix S is output exactly once.*

Notice that in order to enumerate all possible solutions for an instance (M, r) of (min-rank, $\mathcal{C}$)-Graph Recovery ((min-weight, $\mathcal{C}$)-Graph Recovery), it is sufficient to execute the enumeration algorithm $\mathcal{C}$-$\mathcal{E}$ (respectively, $\mathcal{C}$-$\mathcal{E}'$) on input $(M, r, \emptyset, \tau_\emptyset)$. We next prove Lemma 9.

3.2 Decision algorithms for Extended (min-rank, $\mathcal{C}$)-Graph Recovery

The goal of this subsection is to prove Lemma 9. We always assume that without loss of generality, the input M satisfies: $m, n \geq r$. Note that since we require that for every $x \in C_M$, $S[\{x\}] = \tau(x)$, the range of τ can be assumed to be $\text{cols}(\text{inc}(K_n))$.

Our algorithms for deciding the Extended (min-rank, $\mathcal{C}$)-Graph Recovery problem comprise the following two steps, the first of which only relies on S being an incidence matrix and L being a matrix of rank at most r. The second step depends on the class $\mathcal{C}$ under consideration and we will describe it in detail for each application separately.

1. We show that given M, one can enumerate in time $(mn)^{\mathcal{O}(r)}$, a set $\mathcal{Q}_M^r$ of sufficiently few sets, each of which contains at most r n-dimensional binary vectors such that for every L, S such that $M = L + S$, $\text{rank}(L) \leq r$ and S is an incidence matrix, there is a $Q \in \mathcal{Q}_M^r$ whose elements form a basis for col-span(L).
2. We then show that for each class $\mathcal{C}$ that we consider, if one is *given* a basis for the vector space spanned by the columns of the hypothetical solution matrix L of rank $\leq r$, then it is possible to determine the existence of the required matrix S in polynomial time (in the size of M) using an appropriate Weighted Matroid Intersection algorithm as a subroutine.

We begin by formalizing Step 1 of our algorithms.

▶ **Lemma 13.** *Given M and r, one can enumerate in time $(mn)^{O(r)}$, a set $\mathcal{Q}_M^r = \{Q_1, \ldots, Q_\ell\}$ satisfying the following properties: (a) $\ell = (mn)^{O(r)}$, (b) each Q_i is a set of r n-dimensional vectors over $\mathsf{GF}(2)$, (c) for every pair of matrices L, S such that $M = L + S$, S is an incidence matrix and $\text{rank}(L) \leq r$, there is an $i \in [\ell]$ such that Q_i is a basis for col-span(L).*

In the rest of this section, for given r and M, we denote by $\mathcal{Q}_M^r$ the set described in Lemma 13.

▶ **Definition 14.** *For an $n \times m$ binary matrix M and a set $Q \in \binom{\mathsf{GF}(2)^n}{\leq r}$, we denote by G_M^Q the bipartite graph $(X_M^Q = [m], Y_M^Q = \text{cols}(\text{inc}(K_n)), E_M^Q)$ where the edge set E_M^Q is defined as all those pairs (x, y) such that $x \in [m], y \in \text{cols}(\text{inc}(K_n))$ and satisfying $\exists q \in \text{span}(Q)$ such that $M[\{x\}] = y + q$.*

Note that in the graph G_M^Q, the set Y_M^Q contains all columns of the incidence matrix of *every* simple graph on n vertices.

▶ **Observation 15.** *Given M, r and Q, the graph G_M^Q can be computed in time $2^r \cdot (m+n)^{O(1)}$ and has $m + \binom{n}{2}$ vertices.*

The time bound in the above observation comes from the fact that for each $x \in X_M^Q$ and $y \in Y_M^Q$, deciding whether $(x, y) \in E_M^Q$ can be done by iterating over all the at most 2^r vectors in $\text{span}(Q)$.

▶ **Definition 16.** *Consider an $n \times m$ binary matrix M, $Q \in \binom{\mathsf{GF}(2)^n}{\leq r}$, $C_M \subseteq \text{cols}(M)$ and an injective mapping $\tau : C_M \to \mathsf{GF}(2)^n$. If $Z = \{(x, \tau(x)) \mid x \in C_M\}$ is a matching in the graph G_M^Q, then we denote by $\widetilde{G}_M^Q$ the graph obtained from G_M^Q by deleting all edges incident*

on the set $V(Z)$ except the edges in Z. Otherwise, we denote by $\widetilde{G}^Q_M$ the graph obtained from G^Q_M by deleting every edge. Then, $\mathrm{Tr}(G^Q_M, Y^Q_M, \langle C_M, \tau\rangle)$ denotes the transversal matroid $\mathrm{Tr}(\widetilde{G}^Q_M, Y^Q_M)$.

▶ **Definition 17.** *Let $\phi : Y^Q_M \to E(K_n)$ be the trivial bijection which maps $y \in Y^Q_M$ to the corresponding edge of K_n (recall that Y^Q_M is precisely the set of columns of the incidence matrix of K_n). We denote by $\mathcal{M}(K_n, Y^Q_M)$ the matroid over the element set Y^Q_M which is isomorphic to the graphic matroid $M(K_n)$ under the bijection ϕ. We assume that ϕ is extended in the natural way to subsets of vertices. That is, for every $Y \subseteq Y^Q_M$, $\phi(Y) = \bigcup_{y\in Y} \phi(y)$.*

▶ **Observation 18.** *Let ϕ and $\mathcal{M}(K_n, Y^Q_M)$ be as in Definition 17. For every $n \times m$ incidence matrix S, the graph $\mathrm{inc}^{-1}(S)$ is isomorphic to the subgraph of K_n with edge set $\phi(\mathrm{cols}(S))$.*

We are now ready to give our algorithms for each choice of $\mathcal{C}$.

3.2.1 Recovering simple graphs

▶ **Lemma 19.** *(M, r, C_M, τ) is a yes-instance of Extended-(min-rank, simple)-Graph Recovery if and only if there is a $Q \in \mathcal{Q}^r_M$ such that the graph G^Q_M has a matching saturating X^Q_M and containing the edges $\{(x, \tau(x)) \mid x \in C_M\}$.*

Proof. In the forward direction, suppose that $M = S + L$ where $\mathrm{inc}^{-1}(S)$ is a simple graph, $\mathrm{rank}(L) \leq r$ and for every $x \in C_M$, $S[\{x\}] = \tau(x)$. By Lemma 13, we know that some $Q \in \mathcal{Q}^r_M$ is a basis for col-span(L). Consequently, by Definition 14, for each $x \in [m]$, there is an edge in G^Q_M between x and $S[\{x\}]$. Since no 2 columns in S are identical ($\mathrm{inc}^{-1}(S)$ is a simple graph), this implies a matching saturating X^Q_M in G^Q_M and extending the set $\{(x, \tau(x)) \mid x \in C_M\}$ as required.

Conversely, suppose that there is a matching C saturating X^Q_M in G^Q_M and extending the set $\{(x, \tau(x)) \mid x \in C_M\}$. For each $x \in X^Q_M = [m]$, we denote by x_C the partner of x in C. By definition, for every $x \in C_M$, $x_C = \tau(x)$. We define S and L as follows. For each $x \in [m]$, set $S[\{x\}] = x_C$ and $L[\{x\}] = M[\{x\}] + x_C$. Since S contains only distinct columns and these are all contained in $\mathrm{cols}(\mathrm{inc}(K_n))$, it follows that S is the incidence matrix of a simple graph. On the other hand, Definition 14 implies that for every $x \in [m]$, since $(x, x_C) \in E^Q_M$, it must be the case that $L[\{x\}] = M[\{x\}] + x_C \in \mathrm{span}(Q)$. Consequently we have that col-span$(L) \subseteq \mathrm{span}(Q)$ and since $|Q| \leq r$, the lemma follows. ◀

Lemma 13 and Lemma 19 imply our algorithm for Extended-(min-rank, simple)-Graph Recovery.

▶ **Lemma 20.** *Extended-(min-rank, simple)-Graph Recovery can be solved in time $(mn)^{O(r)}$.*

Observe that based on Lemma 19, we may conclude that if the given instance is positive, then there is a solution $M = S + L$ where the columns of S form an independent set of size m in the transversal matroid $\mathrm{Tr}(G^Q_M, Y^Q_M)$ for some $Q \in \mathcal{Q}^r_M$. Moreover, there must be such an independent set saturated by a matching extending $\{(x, \tau(x)) \mid x \in C_M\}$. Consequently, we have the following observation which forms a critical part of all our algorithms.

▶ **Observation 21.** *(M, r, C_M, τ) is a yes-instance of Extended-(min-rank, simple)-Graph Recovery if and only if there is a $Q \in \mathcal{Q}^r_M$ such that there is an independent set of size m in the transversal matroid $\mathrm{Tr}(G^Q_M, Y^Q_M, \langle C_M, \tau\rangle)$.*

3.2.2 Recovering acyclic graphs

▶ **Lemma 22.** *(M, r, C_M, τ) is a yes-instance of* Extended (min-rank, acyclic)-Graph Recovery *if and only if there is a $Q \in \mathcal{Q}_M^r$ such that there is a common independent set of size m in the transversal matroid $\mathrm{Tr}(G_M^Q, Y_M^Q, \langle C_M, \tau\rangle)$ and the matroid $\mathcal{M}(K_n, Y_M^Q)$.*

Proof. In the forward direction, suppose that $M = S + L$ where $\mathrm{inc}^{-1}(S)$ is a forest, for every $x \in C_M$, $S[\{x\}] = \tau(x)$ and $\mathrm{rank}(L) \leq r$. By Lemma 13, we know that some $Q \in \mathcal{Q}_M^r$ is a basis for col-span(L). Moreover, we know that the columns of S form an independent set in $\mathrm{Tr}(G_M^Q, Y_M^Q, \langle C_M, \tau\rangle)$ (by Observation 21). Hence, it is sufficient to show that the set of columns of S form an independent set of the matroid $\mathcal{M}(K_n, Y_M^Q)$. But this follows from the fact that $\mathrm{inc}^{-1}(S)$ is a forest and $\mathcal{M}(K_n, Y_M^Q)$ is isomorphic to $M(K_n)$ under ϕ (see Definition 17 and Observation 18).

In the converse direction, suppose that for some $Q \in \mathcal{Q}_M^r$, there is a common independent set I of size m in the transversal matroid $\mathrm{Tr}(G_M^Q, Y_M^Q, \langle C_M, \tau\rangle)$ and the matroid $\mathcal{M}(K_n, Y_M^Q)$. We construct S and L as follows. Pick a matching C in $\widetilde{G}_M^Q$ (see Definition 16) saturating I and X_M^Q. Since $|I| = m$, such a matching exists. For each $x \in X_M^Q = [m]$, we denote by x_C the partner of x in C. Notice that by the definition of $\widetilde{G}_M^Q$, every vertex $x \in C_M$ has a unique neighbor, which must be $\tau(x)$.

We now define S and L as follows. For each $x \in [m]$, set $S[\{x\}] = x_C$ and $L[\{x\}] = M[\{x\}] + x_C$. Since S contains only distinct columns and these are all contained in $\mathrm{cols}(\mathrm{inc}(K_n))$, it follows that S is the incidence matrix of a simple graph. In addition, for every $x \in C_M$, $S[\{x\}] = \tau(x)$ by the definition of S. Moreover, from Observation 18 and the fact that the set $I = \{x_C | x \in [m]\}$ is an independent set in $\mathcal{M}(K_n, Y_M^Q)$, we have that $\mathrm{inc}^{-1}(S)$ is isomorphic to the subgraph of K_n induced by the edge set $\phi(I)$, which is a forest.

Finally, by the definition of G_M^Q, we have that for every $x \in [m]$, $M[\{x\}] + x_C \in \mathrm{span}(Q)$. Consequently we have that col-span$(L) \subseteq$ span(Q) and since $|Q| \leq r$, and the lemma follows. ◀

▶ **Lemma 23.** *Extended* (min-rank, acyclic)-Graph Recovery *can be solved in time $(mn)^{O(r)}$.*

3.2.3 Recovering Graphs of Fixed Arboricity

In the Arboricity-d Graph Recovery problem, the input is the pair (M, r) and the goal is to decide whether $M = S + L$, where $H = \mathrm{inc}^{-1}(S)$ has arboricity at most d and $\mathrm{rank}(L) \leq r$. Observe that Extended Arboricity-d Graph Recovery is a generalization of Extended Acyclic Graph Recovery (set $d = 1$).

▶ **Definition 24.** *For every $d \in \mathbb{N}$, we define the gammoid $\mathrm{Gam}^d(G_M^Q, Y_M^Q, \langle C_M, \tau\rangle)$ as follows. Consider the digraph D_1 obtained from $\widetilde{G}_M^Q$ by orienting all edges from X_M^Q to Y_M^Q. For each $y \in Y_M^Q$, construct a directed path $D_2^y = (y, 1), \ldots, (y, d)$. We define the digraph D_3 as the graph obtained by identifying each $y \in Y_M^Q$ in D_1 with $(y, 1)$ in D_2^y. The vertex set of D_3 is now $X_M^Q \cup (Y_M^Q \times [d])$. Then, $\mathrm{Gam}^d(G_M^Q, Y_M^Q, \langle C_M, \tau\rangle)$ is defined as the gammoid with respect to D_3 and X_M^Q on ground set $Y_M^Q \times [d]$.*

▶ **Definition 25.** *For each $i \in [d]$, define the matroid $\mathcal{J}^i(K_n, Y_M^Q)$ as the matroid over element set $Y_M^Q \times \{i\}$ which is isomorphic to $\mathcal{M}(K_n, Y_M^Q)$ under the bijection ψ_i where $\psi_i(y, i) = y$, for each $y \in Y_M^Q$. We denote by $\mathcal{M}^d(K_n, Y_M^Q)$ the direct-sum of the d matroids $\{\mathcal{J}^i(K_n, Y_M^Q) \mid i \in [d]\}$. We also extend each ψ_i to sets in the following way: for every $Z_1 \subseteq Y_M^Q$, and $Z_2 = Z_1 \times \{i\}$, $\psi_i(Z_2) = Z_1$.*

The following lemma generalizes Lemma 22.

▶ **Lemma 26.** *(M, r, C_M, τ) is a yes-instance of* Extended Arboricity-d Graph Recovery *if and only if there is a $Q \in \mathcal{Q}_M^r$ such that there is a common independent set of size m in the gammoid $\mathrm{Gam}^d(G_M^Q, Y_M^Q, \langle C_M, \tau\rangle)$ and the matroid $\mathcal{M}^d(K_n, Y_M^Q)$.*

▶ **Lemma 27.** *For every fixed constant d,* Extended (min-rank, d-arboricity)-Graph Recovery *can be solved in time $(mn)^{O(r)}$ on input (M, r, C_M, τ).*

Proof. (Sketch) The crux of this lemma is a proof that (M, r, C_M, τ) is a yes-instance of Extended (min-rank, d-arboricity)-Graph Recovery if and only if there is a $Q \in \mathcal{Q}_M^r$ such that there is a common independent set of size m in an appropriate defined gammoid and the matroid $\mathcal{M}^d(K_n, Y_M^Q)$. ◀

3.2.4 Recovering connected graphs

Here, we prove an analogous version of Lemma 23 for Extended (min-rank, connected)-Graph Recovery.

▶ **Lemma 28.** *Extended (min-rank, connected)-Graph Recovery can be solved in time $(mn)^{O(r)}$.*

Proof. (Sketch) The crux of this lemma is a proof that (M, r, C_M, τ) is a yes-instance of Extended (min-rank, connected)-Graph Recovery if and only if there is a $Q \in \mathcal{Q}_M^r$ such that there is a common independent set of size m in the transversal matroid $\mathrm{Tr}(G_M^Q, Y_M^Q, \langle C_M, \tau\rangle)$ and the m-elongation of the matroid $\mathcal{M}(K_n, Y_M^Q)$ (assuming that $m \geq n-1$, otherwise we have a no-instance). The details of this argument follow along the same lines as the proof of Lemma 22. The crucial part of this lemma is the choice of the m-elongation of $\mathcal{M}(K_n, Y_M^Q)$ as the matroid that we intersect with $\mathrm{Tr}(G_M^Q, Y_M^Q, \langle C_M, \tau\rangle)$. ◀

Lemma 9 is now a straightforward consequence of Lemma 20, Lemma 23, Lemma 28, and Lemma 27.

3.3 Decision algorithms for Extended (min-weight, $\mathcal{C}$)-Graph Recovery

The goal of this subsection is to prove Lemma 10. The proof is involved and has several stages. In the following, we provide a high level overview of our proof strategy.

Recall that an instance of (min-weight, $\mathcal{C}$)-Graph Recovery is of the form (M, k, C_M, τ) where $C_M \subseteq \mathrm{cols}(M)$ and τ fixes the mapping of the columns in C_M in the solution. When we just want G to be a simple graph (i.e., $\mathcal{C}$ is the class of all simple graphs), a solution $M = L + S$ minimizing the cost, $\|L\|$, can be computed in polynomial time using an algorithm for Minimum Weight Bipartite Matching, in an auxiliary bipartite graph which represents the costs of mapping each column of M to a specific edge. The design of this bipartite graph is similar in spirit to that of the graph in Definition 14. We refer to a solution $M = L + S$ minimizing the cost $\|L\|$, as a *minimum cost solution*.

To obtain connected graphs and forests, we build upon this algorithm. We first describe an algorithm that computes a minimum cost solution $M = L_0 + S_0$ along with the graph G_0 and try to reassign columns of M associated with non-bridge edges in G_0 to reduce the number of connected components without increasing the cost of the solution at hand. That is, we move from one solution to another without increasing the cost but while decreasing the number of connected components in the graph corresponding to the incidence matrix in the solution. Note that, we may need to perform a number of reassignments in a sequence

before the number of connected components is reduced. To determine this sequence of reassignments, we associate them with paths in an auxiliary digraph, where the nodes are the edges and non-edges of the current graph, and directed edges indicate feasible reassignments, or reassignments that convert a bridge in the current graph to a non-bridge. We show that the current solution minimizes the number of connected components if and only if the auxiliary digraph has no paths starting from a non-bridge edge and ending at a non-edge. This requires an intricate analysis of the structure of a solution and how reassignments affect them. This leads to an iterative algorithm that "remaps" the edges in the graph associated with the current solution using paths in the auxiliary digraph. When this algorithm terminates, we obtain a minimum cost solution that minimizes the number of connected components in the corresponding graph. We then use this algorithm as the starting point for the algorithms to recover connected graphs and forests. To recover a connected graph, we observe that any further reassignments to reduce the number of connected components must increase the cost of the solution. Therefore, we design an iterative algorithm that reassigns non-bridge edges until every edge is a bridge, or the graph becomes connected. This algorithm can be used to recover a connected graph, or a forest satisfying the required properties.

3.4 NP-completeness of (min-rank, $\mathcal{C}$)-Graph Recovery

▶ **Theorem 29.** (min-rank, $\mathcal{C}$)-*Graph Recovery is NP-complete for* $\mathcal{C} \in \{$*simple graphs, acyclic graphs, connected graphs, arboricity-d graphs*$\}$.

4 Concluding remarks and future work

In this paper, we have initiated the study of a family of graph recovery problems. In these problems, the aim is to recover the incidence matrix of a graph (contained in a specific graph class) from a given binary matrix. The input matrix is assumed to be a perturbation of an incidence matrix by either a small rank or low weight matrix. We have demonstrated the rich combinatorial structure possessed by these problems by designing decision and enumeration algorithms using classic concepts such as matchings and matroids. We leave open the following concrete questions.

1. Is (min-rank, $\mathcal{C}$)-Graph Recovery fixed-parameter tractable parameterized by r for the classes we consider? That is, can it be solved in time $f(r)(mn)^{\mathcal{O}(1)}$ for some computable function f?
2. Identify classes $\mathcal{C}$ for which (min-rank, $\mathcal{C}$)-Graph Recovery is polynomial-time solvable.
3. Characterize those $\mathcal{C}$ for which (min-weight, $\mathcal{C}$)-Graph Recovery is polynomial-time solvable.

References

1 Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *J. ACM*, 58(3):11:1–11:37, 2011. doi:10.1145/1970392.1970395.

2 Venkat Chandrasekaran, Sujay Sanghavi, Pablo A. Parrilo, and Alan S. Willsky. Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, 21(2):572–596, 2011. doi:10.1137/090761793.

3 Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 3rd edition, 2005.

4 Fedor V Fomin, Petr A Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Covering vectors by spaces in perturbed graphic matroids and their duals. *arXiv preprint*, 2019. arXiv:1902.06957.

5 Jim Geelen, Bert Gerards, and Geoff Whittle. On rota's conjecture and excluded minors containing large projective geometries. *Journal of Combinatorial Theory, Series B*, 96(3):405–425, 2006.

6 Jim Geelen and Rohan Kapadia. Computing girth and cogirth in perturbed graphic matroids. *Combinatorica*, 38(1):167–191, 2018.

7 Chien-Chung Huang, Naonori Kakimura, and Naoyuki Kamiyama. Exact and approximation algorithms for weighted matroid intersection. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 430–444, 2016. doi:10.1137/1.9781611974331.ch32.

8 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Trans. Algorithms*, 14(2):14:1–14:20, 2018. doi:10.1145/3170444.

9 James G. Oxley. *Matroid theory*, volume 21 of *Oxford Graduate Texts in Mathematics*. Oxford University Press, 2nd edition, 2010.

10 Nauman Shahid, Nathanael Perraudin, Vassilis Kalofolias, Gilles Puy, and Pierre Vandergheynst. Fast robust pca on graphs. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):740–756, 2016.

11 John Wright, Arvind Ganesh, Shankar R. Rao, YiGang Peng, and Yi Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In *Proceedings of 23rd Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2080–2088. Curran Associates, Inc., 2009. URL: http://papers.nips.cc/paper/3704-robust-principal-component-analysis-exact-recovery-of-corrupted-low-rank-matrices-via-convex-optimization.

12 Mengnan Zhao, M Devrim Kaba, René Vidal, Daniel P Robinson, and Enrique Mallada. Sparse recovery over graph incidence matrices. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 364–371. IEEE, 2018.

An Algorithmic Meta-Theorem for Graph Modification to Planarity and FOL

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway
fedor.fomin@ii.uib.no

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
petr.golovach@ii.uib.no

Giannos Stamoulis
Department of Informatics and Telecommunications,
National and Kapodistrian University of Athens, Greece
Inter-university Postgraduate Programme "Algorithms, Logic, and Discrete Mathematics" (ALMA), Athens, Greece
giannos95@gmail.com

Dimitrios M. Thilikos
LIRMM, Univ. Montpellier, CNRS, Montpellier, France
sedthilk@thilikos.info

Abstract

In general, a *graph modification problem* is defined by a graph modification operation $\boxtimes$ and a target graph property $\mathcal{P}$. Typically, the modification operation $\boxtimes$ may be vertex removal, edge removal, edge contraction, or edge addition and the question is, given a graph G and an integer k, whether it is possible to transform G to a graph in $\mathcal{P}$ after applying k times the operation $\boxtimes$ on G. This problem has been extensively studied for particilar instantiations of $\boxtimes$ and $\mathcal{P}$. In this paper we consider the general property $\mathcal{P}_\phi$ of being planar and, moreover, being a model of some First-Order Logic sentence ϕ (an FOL-sentence). We call the corresponding meta-problem Graph $\boxtimes$-Modification to Planarity and ϕ and prove the following algorithmic meta-theorem: there exists a function $f : \mathbb{N}^2 \to \mathbb{N}$ such that, for every $\boxtimes$ and every FOL sentence ϕ, the Graph $\boxtimes$-Modification to Planarity and ϕ is solvable in $f(k, |\phi|) \cdot n^2$ time. The proof constitutes a hybrid of two different classic techniques in graph algorithms. The first is the *irrelevant vertex technique* that is typically used in the context of Graph Minors and deals with properties such as planarity or surface-embeddability (that are *not* FOL-expressible) and the second is the use of *Gaifman's Locality Theorem* that is the theoretical base for the meta-algorithmic study of FOL-expressible problems.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph modification Problems, Algorithmic meta-theorems, First Order Logic, Irrelevant vertex technique, Planar graphs, Surface embeddable graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.51

Funding The research has been supported by the Research Council of Norway and the French Ministry of Europe and Foreign Affairs via the Franco-Norwegian project PHC AURORA 2019.
Fedor V. Fomin: Research Council of Norway via the project "MULTIVAL".
Petr A. Golovach: Research Council of Norway via the project "MULTIVAL".
Dimitrios M. Thilikos: DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010).

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders; Article No. 51; pp. 51:1–51:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The term *algorithmic meta-theorems* was coined by Grohe in his seminal exposition in [20] in order to describe results providing general conditions, typically of logical and/or combinatorial nature, that automatically guarantee the existence of certain types of algorithms for wide families of problems. Algorithmic meta-theorems reveal deep relations between logic and combinatorial structures, which is a fundamental issue of computational complexity. Such theorems not only yield a better understanding of the scope of general algorithmic techniques and the limits of tractability but often provide (or induce) a variety of new algorithmic results. The archetype of algorithmic meta-theorems is Courcelle's theorem [5,6] stating that all graph properties expressible in Monadic Second-Order Logic (in short, *MSOL-expressible properties*) are fixed-parameter tractable when parameterized by the size of the sentence and the treewidth of the graph.

Our meta-theorem belongs to the intersection of two algorithmic research directions: Deciding First-Order Logic properties of sparse graphs and graph planarization algorithms.

FOL-expressible properties on sparse graphs. For graph properties expressible in first-order logic (in short *FOL-expressible properties*), a rich family of algorithmic meta-theorems, were developed within the last decades. Each of these meta-theorems can be stated in the following form: for a graph class $\mathcal{C}$, deciding FOL-expressible properties is fixed-parameter tractable on $\mathcal{C}$, i.e. there is an algorithm running in $f(|\phi|) \cdot n^{\mathcal{O}(1)}$ time where $|\phi|$ is the size of the the input FOL-sentence ϕ and n is the number of vertices of the input graph. The starting point in the chain of such meta-theorems is the work of Seese [33] for $\mathcal{C}$ being the class of graphs of bounded degree [33]. The first significant extension of Seese's theorem was obtained by Frick and Grohe [16] for the class $\mathcal{C}$ of graphs of bounded local treewidth [16]. The class of graphs of bounded local treewidth contains graphs of bounded degree, planar graphs, graphs of bounded genus, and apex-minor-free graphs. The next step was done by Flum and Grohe [12], who extended these results up to graph classes excluding some minor. Dawar, Grohe, and Kreutzer [9] pushed the tractability border up to graphs locally excluding a minor. Further extension was due to Dvořák, Král, and Thomas, who proved tractability for the class $\mathcal{C}$ of being locally bounded expansion [11]. Finally, Grohe, Kreutzer, and Siebertz [22] established fixed-parameter tractability for classes that are effectively nowhere dense. In some sense, the result of Grohe et al. is the culmination of this long line of meta-theorems, because for somewhere dense graph classes closed under taking subgraphs deciding first-order properties is unlikely to be fixed-parameter tractable [11, 26].

Notice that the above line of results also sheds some light on graph modification problems. In particular, since many modification operations are FOL-expressible, in some situations when the target property $\mathcal{P}$ is FOL-expressible, the above meta-algorithmic results can be extended to graph modification problems. As a concrete example, consider the problem of removing at most k vertices to obtain a graph of degree at most 3. All vertices of the input graph of degree at least $4 + k$ should be deleted, so we delete them and adapt the parameter k accordingly. In the remaining graph all vertices are of degree at most $3 + k$ and the property of removing at most k vertices from such a graph to obtain a graph of degree at most 3 is FOL-expressible. Hence the Seese's theorem implies that there is an algorithm of running time $f(k) \cdot n^{\mathcal{O}(1)}$ solving this problem. However these theories are not applicable with instantiations of $\mathcal{P}$, like planarity, that are not FOL-expressible.

Another island of tractability for graph modification problems is provided by Courcelle's theorem and similar theorems on graphs of bounded widths. For example, graph modification problems are fixed-parameter tractable in cases where the target property $\mathcal{P}$ is MSOL-expressible under the additional assumption that the graphs in $\mathcal{P}$ have fixed treewidth (or bounded rankwidth, for MSOL_1-properties, see e.g., [7]).

To conclude, according to the current state of the art, all known algorithmic meta-theorems concerning fixed-parameter tractability of graph modification problems are attainable either when the target property $\mathcal{P}$ is FOL-expressible and the structure is sparse or when $\mathcal{P}$ is $\mathrm{MSOL}/\mathrm{MSO}_1$-expressible and the structure has bounded tree/rank-width. Interestingly, *planarity* is the typical property that escapes the above pattern: it is *not* FOL-expressible and it has *unbounded* treewidth.

Graph planarization. The Planar Vertex Deletion problem is a generalization of planarity testing. For a given graph G the goal is to find a vertex set of size at most k whose removal makes the resulting graph planar. Planarity is a nontrivial and hereditary graph property, hence by the result of Lewis and Yannakakis [27], the decision version of Planar Vertex Deletion is NP-complete. The parameterized complexity of this problem has been extensively studied.

The non-uniform fixed-parameter tractability of Planar Vertex Deletion (parameterized by k) follows from the deep result of Robertson and Seymour in Graph Minors theory [32], that every minor-closed graph class can be recognized in polynomial time. Since the class of graphs that can be made planar by removing at most k vertices is minor-closed, the result of Robertson and Seymour implies that for Planar Vertex Deletion, for each k, there exists a (non-uniform) algorithm that in time $\mathcal{O}(n^3)$ solves Planar Vertex Deletion. Significant amount of work was involved to improve the enormous constants hidden in the big-Oh and the polynomial dependence in n. Marx and Schlotter [29] gave an algorithm that solves the problem in time $f(k) \cdot n^2$, where f is some function of k only. Kawarabayashi [24] obtained the first linear time algorithm of running time $f(k) \cdot n$ and Jansen, Lokshtanov, and Saurabh [23] obtained an algorithm of running time $\mathcal{O}(2^{\mathcal{O}(k \log k)} \cdot n)$. For the related problem of contracting at most k edges to obtain a planar graph, Planar Edge Contraction, an $f(k) \cdot n^{\mathcal{O}(1)}$ time algorithm was obtained by Golovach, van 't Hof and Paulusma [19]. Approximation algorithms for Planar Vertex Deletion and for Planar Edge Deletion were studied in [2–4].

Our results. Let $\boxtimes$ be one of the following operations on graphs: Vertex removal, edge removal, edge contraction, or edge addition. We are interested whether, for a given graph G and an FOL-sentence ϕ, it is possible to transform G by applying at most k $\boxtimes$-operations, into a planar graph with the property defined by ϕ. We refer to this problem as the Graph $\boxtimes$-Modification to Planarity and ϕ problem. For example, when $\boxtimes$ is the vertex removal operation and ϕ is a tautology, then the problem is Planar Vertex Deletion. Similarly, Graph $\boxtimes$-Modification to Planarity and ϕ generalizes Planar Edge Deletion and Planar Edge Contraction. On the other hand, for the special case of $k = 0$ this is the problem of deciding FOL-expressible properties on planar graphs.

Examples of first-order expressible properties are deciding whether the input graph G contains a fixed graph H as a subgraph (H-Subgraph Isomorphism), deciding whether there is a homomorphism from a fixed graph H to G to (H-Homomorphism), satisfying degree constraints (the degree of every vertex of the graph should be between a and b for some constants a and b), excluding a subgraph of constant size or having a dominating

set of constant size. Thus Graph $\boxtimes$-Modification to Planarity and ϕ encompasses the variety of graph modification problems to planar graphs with specific properties. For example, can we delete k vertices (or edges) such that the obtained graph is planar and each vertex belongs to a triangle? Reversely, can we delete at most k vertices (or edges) from a graph such that the resulting graph is a triangle-free planar graph? Can we add (or contract) at most k edges to such that the resulting graph is 4-regular and planar? Or can we delete at most k edges resulting in a square-free or claw-free planar graph?

Informally, our main result can be stated as follows.

▶ **Theorem** (Informal). *Graph $\boxtimes$-Modification to Planarity and ϕ is solvable in time $f(k,\phi)\cdot n^2$, for some function f depending on k and ϕ only. Thus the problem is fixed-parameter tractable, when parameterized by $k+|\phi|$.*

Our theorem not only implies that Planar Vertex Deletion is fixed-parameter tractable parameterized by k (proved in [23, 29]) and that deciding whether a planar graph has a first-order logic property ϕ is fixed-parameter tractable parameterized by $|\phi|$ (that follows from [9, 11, 16, 22]). It also implies a variety of new algorithmic results about graph modification problems to planar graphs with some specific properties that cannot be obtained by applying the known results directly. Of course, for some formulas ϕ, Graph $\boxtimes$-Modification to Planarity and ϕ can be solved by more simple techniques. For example, if ϕ defines a hereditary property characterized by a finite family of forbidden induced subgraphs $\mathcal{F}$, then deciding, whether it is possible to delete at most k vertices to obtain a planar $\mathcal{F}$-free graph, can be done by combining the straightforward branching algorithm and, say, the algorithm of Jansen, Lokshtanov, and Saurabh [23] for Planar Vertex Deletion. For this, we iteratively find a copy of each $F\in\mathcal{F}$ and if such a copy exists we branch on all the possibilities to destroy this copy of F by deleting a vertex. By this procedure, we obtain a search tree of depth at most k, whose leaves are all $\mathcal{F}$-free induced subgraphs of the input graph that could be obtained by at most k vertex deletions. Then for each leaf, we use the planarization algorithm limited by the remaining budget. However, this does not work for edge modifications, because deleting an edge in order to ensure planarity may result in creating a copy of a forbidden subgraph. For such type of problems, even for very "simple" ones, like deleting k edges to obtain a claw-free planar graph, or planar graph without induced cycles of length 4, our theorem establishes the first fixed-parameter algorithms. Also our theorem is applicable to the situation when ϕ defines a hereditary property that requires an infinite family of forbidden subgraphs for its characterization and for non-hereditary properties expressible in FOL.

In our paper, we show the result for Graph $\boxtimes$-Modification to Planarity and ϕ, but further we argue that it can be extended for modification problems to graphs embeddable to a surface of a given Euler genus.

The price we pay for such generality is the running time. While the polynomial factor in the running time of our algorithm is comparable with the running time of the algorithm of Marx and Schlotter [29] for Planar Vertex Deletion, it is worse than the more advanced algorithms of Kawarabayashi [24] and Jansen et al. [23]. Similarly, the algorithms for deciding first-order logic properties on graph classes [11, 16, 22] are faster than our algorithm.

The proof of the main theorem is based on a non-trivial combination of the *irrelevant vertex* technique of Robertson and Seymour [30,31] with the *Gaifman's Locality Theorem* [17]. While both techniques were widely used, see [1, 8, 19, 21, 23, 28] and [9, 12, 16], the combination of the two techniques requires novel ideas. Following the popular trend in Theoretical Computer Science, an alternative title for our paper could be *"Robertson and Seymour meet Gaifman"*.

Organization of the paper. In Section 2 we give the formal definition of the general Graph $\boxtimes$-Modification to Planarity and ϕ problem, present the theoretical background around Gaifman's Locality Theorem, and give some preliminary definitions and results. In Section 3 we highlight the main ideas behind the proof explain how our arguments can be extended in cases where the target property is having bounded Euler genus and being a model of an FOL-sentence ϕ. Finally, in Section 4 we provide some directions for further research.

2 Problem definition and preliminaries

Before we explain our techniques, we give some necessary definitions. We denote by $\mathbb{N}$ the set of all non-negative integers. Given an $n \in \mathbb{N}$, we denote by $\mathbb{N}_{\geq n}$ the set containing all integers equal or greater than n. Given two integers x and y we define by $[x, y] = \{x, x+1, \ldots, y-1, y\}$. Given an $n \in \mathbb{N}_{\geq 1}$, we also define $[n] = [1, n]$.

All graphs in this paper are undirected, finite, and they do not have loops or multiple edges. Given a graph G, we denote by $V(G)$ and $E(G)$ the set of its vertices and edges, respectively. If $S \subseteq V(G)$, then we denote by $G \setminus S$ the graph obtained by G after removing from it all vertices in S, together with their incident edges. Also, we denote by $G \setminus v$ the graph $G \setminus \{v\}$, for some $v \in V(G)$. We also denote by $G[S]$ the graph $G \setminus (V(G) \setminus S)$.

2.1 Modifications on graphs

We define $\mathsf{OP} := \{\mathsf{vr}, \mathsf{er}, \mathsf{ec}, \mathsf{ea}\}$, that is the set of graph operations of removing a vertex, removing an edge, contracting an edge, and adding an edge, respectively. Given an operation $\boxtimes \in \mathsf{OP}$, a graph G, and a vertex set $R \subseteq V(G)$, we define the *application domain* of the operation $\boxtimes$ as

$$\boxtimes\langle G, R\rangle = \begin{cases} R, & \text{if } \boxtimes = \mathsf{vr}, \\ E(G) \cap \binom{R}{2}, & \text{if } \boxtimes = \mathsf{er}, \mathsf{ec}, \text{and} \\ \binom{R}{2} \setminus E(G), & \text{if } \boxtimes = \mathsf{ea}. \end{cases}$$

Notice that $\boxtimes\langle G, R\rangle$ is either a vertex set or a set of subsets of vertices each of size two.

Given a set $S \subseteq \boxtimes\langle G, R\rangle$, we define $G \boxtimes S$ as the graph obtained after applying the operation $\boxtimes$ on the elements of S. The vertices of G that are *affected* by the modification of G to $G \boxtimes S$, denoted by $A(S)$, are the vertices in S, in case $\boxtimes = \mathsf{vr}$ or the endpoints of the edges of S, in case $\boxtimes \in \{\mathsf{er}, \mathsf{ec}, \mathsf{ea}\}$.

Given an FOL-sentence ϕ and some $\boxtimes \in \mathsf{OP}$, we define the following meta-problem:

Graph $\boxtimes$-Modification to Planarity and ϕ (In short: G$\boxtimes$MPϕ)
Input: A graph G and a non-negative integer k.
Question: Is there a set $S \subseteq \boxtimes\langle G, V(G)\rangle$ of size k such that $G \boxtimes S$ is a planar graph and $G \boxtimes S \models \phi$?

Let $(x_1, \ldots, x_\ell) \in \mathbb{N}^\ell$ and $f, g : \mathbb{N} \to \mathbb{N}$. We use notation $f(n) = \mathcal{O}_{x_1,\ldots,x_\ell}(g(n))$ to denote that there exists a computable function $h : \mathbb{N}^\ell \to \mathbb{N}$ such that $f(n) = h(x_1, \ldots, x_\ell) \cdot g(n)$. We are ready to give the formal statement of the main theorem of this paper.

▶ **Theorem 1.** *There exists a function $f : \mathbb{N}^2 \to \mathbb{N}$ such that, for every FOL-sentence ϕ and for every $\boxtimes \in \mathsf{OP}$, G$\boxtimes$MP$\phi$ is solvable in $\mathcal{O}_{k,|\phi|}(n^2)$ time.*

2.2 Gaifman's theorem

For vertices u, v of graph G, we use $d_G(u, v)$ to denote the distance between u and v in G. We also use $N_G^{(\leq r)}(v)$ to denote the set of vertices of G at distance at most r from v.

Formulas. In this paper we deal with logic formulas on graphs. In particular we deal with formulas of first-order logic (FOL) and monadic second-order logic (MSO_2). The syntax of FOL-formulas includes the logical connectives $\vee, \wedge, \neg$, a set of variables for vertices, the quantifiers $\forall, \exists$ that are applied to these variables, the predicate $u \sim v$, where u and v are vertex variables and whose interpretation is that u and v are adjacent, and the equality of variables representing vertices. An MSO_2-formula, in addition to the variables for vertices of FOL-formulas, may also contain variables for subsets of vertices or subsets of edges. The syntax of MSO_2-formulas is obtained after enhancing the syntax of FOL-formulas so to further allow quantification on subsets of vertices or subsets of edges and introduce the predicates $v \in S$ (resp. $e \in F$) whose interpretation is that the vertex v belongs in the vertex set S (resp. the edge e belongs in the edge set F).

An FOL-formula ϕ is in *prenex normal form* if it is written as $\phi = Q_1 x_1 \dots Q_n x_n \psi$ such that for every $i \in [n]$, $Q_i \in \{\forall, \exists\}$ and ψ is a quantifier-free formula on the variables $x_1, \dots, x_n$. Then $Q_1 x_1 \dots Q_n x_n$ is referred as the *prefix* of ϕ. For the rest of the paper, when we mention the term "FOL-formula", we mean an FOL-formula on graphs that is in prenex normal form. Given an FOL-formula ϕ, we say that a variable x is a *free variable* in ϕ if it does not occur in the prefix of ϕ. We write $\phi(x_1, \dots, x_r)$ to denote that ϕ is a formula with free variables $x_1, \dots, x_r$. We call a formula without free variables a *sentence*. For a sentence ϕ and a graph G, we write $G \models \phi$ to denote that ϕ evaluates to *true* on G. Also, for a sentence ϕ we denote its length by $|\phi|$.

Gaifman sentences. Given an FOL-formula $\psi(x)$ with one free variable x, we say that $\psi(x)$ is *r-local* if the validity of $\psi(x)$ depends only on the r-neighborhood of x, that is for every graph G and $v \in V(G)$ we have

$$G \models \psi(v) \iff N_G^{(\leq r)}(v) \models \psi(v).$$

Observe that there exists an FOL-formula $\delta_r(x, y)$ such that for every graph G and $v, u \in V(G)$, we have $d_G(u, v) \leq r \iff G \models \delta_r(v, u)$ (see [13, Lemma 12.26]).

We say that an FOL-sentence ϕ is a *Gaifman sentence* when it is a Boolean combination of sentences $\phi_1, \dots, \phi_m$ such that, for every $h \in [m]$,

$$\phi_h = \exists x_1 \dots \exists x_{\ell_h} \Big(\bigwedge_{1 \leq i < j \leq \ell_h} d(x_i, x_j) > 2r_h \wedge \bigwedge_{i \in [\ell_h]} \psi_h(x_i) \Big), \tag{1}$$

where $\ell_h, r_h \geq 1$ and ψ_h is an r_h-local formula with one free variable. We refer to the variables $x_1, \dots, x_{\ell_h}$ for each $h \in [m]$ as the *basic variables* of ϕ. Moreover, for every $h \in [m]$ we call ϕ_h a *basic local sentence* of ϕ and the formula ψ_h a *local formula* of ϕ.

▶ **Proposition 2** (Gaifman's Theorem [17]). *Every FOL-sentence ϕ is equivalent to a Gaifman sentence ϕ'. Furthermore, ϕ' can be computed effectively.*

2.3 Equivalent formulations

Given a Gaifman sentence ϕ combined from sentences $\phi_1, \ldots, \phi_m$ and a unary relation symbol R, we define $\phi\|_R$ as the sentence that is the same Boolean combination of sentences $\phi_1\|_R, \ldots, \phi_m\|_R$ such that, for every $h \in [m]$,

$$\phi_h\|_R = \exists x_1 \ldots \exists x_{\ell_h} \Big(\bigwedge_{i\in[\ell_h]} x_i \in R \wedge \bigwedge_{1\leq i<j\leq \ell_h} d(x_i,x_j) > 2r_h \wedge \bigwedge_{i\in[\ell_h]} \psi_h(x_i)\Big), \tag{2}$$

where $\ell_h, r_h \geq 1$ and ψ_h is an r_h-local formula with one free variable.

Let (G,k) be an instance of the G$\boxtimes$MPϕ problem. We may assume, because of Proposition 2, that ϕ is a Gaifman sentence. We consider an enhanced version of the G$\boxtimes$MPϕ problem as follows. Let (G,R,k) be a triple, where G is a graph, $R \subseteq V(G)$, and $k \in \mathbb{N}$. We say that (G,R,k) is a $(\phi,\boxtimes)$-*triple* if there exists a set $S \subseteq \boxtimes\langle G,R\rangle$ such that $|S| \leq k$, $G \boxtimes S$ is a planar graph, and $G \boxtimes S \models \phi\|_R$. Also, we say that a set $S \subseteq \boxtimes\langle G, V(G)\rangle$ is a $\boxtimes$-*planarizer* of G if $G \boxtimes S$ is planar. It is easy to observe that the property that (G,R,k) is a $(\phi,\boxtimes)$-triple can be expressed in MSO_2. This is easy in case $\boxtimes \in \{\mathsf{vr},\mathsf{er},\mathsf{ec}\}$. In the case where $\boxtimes = \mathsf{ea}$, we observe the following:

▶ **Observation 3.** *Let $\boxtimes = \mathsf{ea}$, G be a graph, and $S \subseteq \langle G, V(G)\rangle$ where $S = \{\{v_1,u_1\},\ldots,\{v_r,u_r\}\}$. Then there exists an MSO_2-formula $\phi_{\mathcal{P},\mathcal{S}}$ on structures of the type $(G,v_1,u_1,\ldots,v_r,u_r)$ such that*

$$G \boxtimes S \text{ is a planar graph} \iff (G,v_1,u_1,\ldots,v_r,u_r) \models \phi_{\mathcal{P},S}.$$

Treewidth. A *tree decomposition* of a graph G is a pair (T,χ) where T is a tree and $\chi : V(T) \to 2^{V(G)}$ such that

1. $\bigcup_{t\in V(T)} \chi(t) = V(G)$;
2. for every edge e of G there is a $t \in V(T)$ such that $\chi(t)$ contains both endpoints of e and
3. for every $v \in V(G)$, the subgraph of T induced by $\{t \in V(T) \mid v \in \chi(t)\}$ is connected.

The *width* of (T,χ) is defined as $\mathbf{w}(T,\chi) := \max\big\{\, |\chi(t)| - 1 \;\big|\; t \in V(T)\big\}$. The *treewidth of* G is defined as

$$\mathbf{tw}(G) := \min\big\{\, \mathbf{w}(T,\chi) \;\big|\; (T,\chi) \text{ is a tree decomposition of } G\big\}.$$

Theorem 1 is a consequence of the following lemma.

▶ **Lemma 4.** *Given a Gaifman sentence ϕ and a $\boxtimes \in \mathsf{OP}$, there exists a function $f_1 : \mathbb{N}^2 \to \mathbb{N}$, and an algorithm with the following specifications:*

Reduce_Instance(k,G,S,R)

Input: an integer $k \in \mathbb{N}$, a graph G, a set $R \subseteq V(G)$, and a set $S \subseteq R$ that is a vr*-planarizer of G of size at most k.*

Output: One of the following:

1. - *if $\boxtimes \in \{\mathsf{er},\mathsf{ec},\mathsf{ea}\}$: a report that (G,k) is a* no*-instance of* G$\boxtimes$MPϕ.
   - *if $\boxtimes = \mathsf{vr}$: a vertex $u \in S$ such that $S \setminus \{u\}$ is a* vr*-planarizer of $G \setminus u$ of size at most $k-1$ and (G,k) and $(G \setminus u, k-1)$ are equivalent instances of* G$\boxtimes$MPϕ.
2. *a vertex set $X \subseteq V(G)$ and a vertex $v \in X$ such that $S \subseteq R \setminus X$ and (G,R,k) is a $(\phi,\boxtimes)$-triple iff $(G \setminus v, R \setminus X, k)$ is a $(\phi,\boxtimes)$-triple.*
3. *a tree decomposition of G of width at most $f_1(k,|\phi|)$.*

Moreover, this algorithm runs in $\mathcal{O}_{k,|\phi|}(n)$ steps.

Given Lemma 4, Theorem 1 can be proved as follows.

Proof of Theorem 1. Let ϕ be an FOL-formula. By Proposition 2, ϕ is equivalent to a Gaifman sentence ϕ'. Using the planarization algorithm from [23], we compute, in $\mathcal{O}_k(n)$ steps, a vr-planarizer S of G of size at most k. If $\boxtimes = \mathsf{ea}$, then $S := \emptyset$, while if $\boxtimes \in \{\mathsf{vr}, \mathsf{er}, \mathsf{ec}\}$, then if such a set does not exist the we safely return a negative answer (for the case of $\boxtimes = \mathsf{er}, \mathsf{ec}$, this is due to the fact that if there exists a ec- or an er-planarizer of G of size at most k then also a vr-planarizer of G of size at most k exists (see [19, Lemma 1])). We are now in position to apply recursively the algorithm **Reduce_Instance**(k, G, S, R) of Lemma 4 until either an answer or the third case appears. In the first case, we either return a negative answer, if $\boxtimes \in \{\mathsf{er}, \mathsf{ec}, \mathsf{ea}\}$, or set $(k, G, S, R) := (k-1, G \setminus v, S \setminus v, R)$ if $\boxtimes = \mathsf{vr}$, while in the second case we set $(k, G, S, R) := (k, G \setminus v, S, R \setminus X)$. In the third case we have that $\mathbf{tw}(G) \leq f_1(k, |\phi'|)$. Recall that the property that (G, R, k) is a $(\phi, \boxtimes)$-triple can be expressed in MSO_2, thus the status of the final equivalent instance (G, R, k) can be evaluated in $\mathcal{O}_{k,|\phi|}(n)$ steps by applying Courcelle's theorem. As the recursion takes at most n steps, we obtain the claimed running time. ◀

3 The algorithm

3.1 Two main lemmata

We now give two lemmata, whose combination gives the proof of Lemma 4. Before we state them, we give a series of definitions.

Let $\boxtimes \in \mathsf{OP}$, G be a graph, $k \in \mathbb{N}$, and let S be a $\boxtimes$-planarizer of G. We say that S is an *inclusion-minimal* $\boxtimes$*-planarizer* of G if none of its proper subsets is a $\boxtimes$-planarizer of G. Notice that, in the special case where $\boxtimes = \mathsf{ea}$, the unique inclusion-minimal $\boxtimes$-planarizer of G is the empty set of edges. We say that a set $Q \subseteq V(G)$ is $\boxtimes$*-planarization irrelevant* if for every inclusion-minimal $\boxtimes$-planarizer S of G that has size at most k, it holds that $A(S) \cap Q = \emptyset$.

Partially disk-embedded graphs. We define a *closed disk* Δ to be a subset of the plane homeomorphic to the set $\{(x, y) \mid x^2 + y^2 \leq 1\}$ and we use $\mathbf{bor}(\Delta)$ to denote its boundary. We say that a graph G is *partially disk-embedded in some closed disk* Δ, if there is some subgraph K of G that is embedded in Δ such that $\mathbf{bor}(\Delta)$ is a cycle of K and no vertex in $\Delta \setminus \mathbf{bor}(\Delta)$ is adjacent to a vertex not in Δ. We use the term *partially* Δ*-embedded graph* G to denote that a graph G is partially disk-embedded in some closed disk Δ. We also call the graph K *compass* of the partially Δ-embedded graph G and we always assume that we accompany a partially Δ-embedded graph G together with an embedding of its compass in Δ that is the set $G \cap \Delta$.

Grids and walls. Let $k, r \in \mathbb{N}$. The $(k \times r)$*-grid* is the Cartesian product of two paths on k and r vertices respectively. An *elementary* r*-wall*, for some odd $r \geq 3$, is the graph obtained from a $(2r \times r)$-grid with vertices (x, y), $x \in [2r] \times [r]$, after the removal of the "vertical" edges $\{(x, y), (x, y+1)\}$ for odd $x + y$, and then the removal of all vertices of degree one. Notice that, as $r \geq 3$, an elementary r-wall is a planar graph that has a unique (up to topological isomorphism) embedding in the plane such that all its finite faces are incident to exactly six edges. The *perimeter* of an elementary r-wall is the cycle bounding its infinite face, while the cycles bounding its finite faces are called *bricks*. Given an elementary wall $\overline{W}$, a *vertical path* of $\overline{W}$ is one whose vertices, in ordering of appearance, are $(i, 1), (i, 2), (i+1, 2), (i+1, 3), (i, 3), (i, 4), (i+1, 4), (i+1, 5), (i, 5), \ldots, (i, r-2), (i, r-1), (i+1, r-1), (i+1, r)$, for

some $i \in \{1, 3, \ldots, 2r-1\}$. Also an *horizontal path* of $\overline{W}$ is the one whose vertices, in ordering of appearance, are $(1, j), (2, j), \ldots, (2r, j)$, for some $j \in [2, r-1]$, or $(1, 1), (2, 1), \ldots, (2r-1, 1)$ or $(2, r), (2, r), \ldots, (2r, r)$.

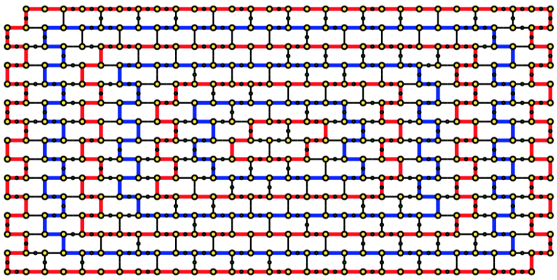


Figure 1 An 15-wall and its 7 layers.

An *r-wall* is any graph W obtained from an elementary r-wall $\overline{W}$ after subdividing edges (see Figure 1). We call the vertices that where added after the subdivision operations *subdivision vertices*, while we call the rest of the vertices (i.e., those of $\overline{W}$) *branch vertices*. The *perimeter* of W, denoted by $\mathsf{perim}(W)$, is the cycle of W whose non-subdivision vertices are the vertices of the perimeter of $\overline{W}$. Also, a vertical (resp. horizontal) path of W is a subdivided vertical (resp. horizontal) path of $\overline{W}$.

A subgraph W of a graph G is called a *wall* of G if W is an r-wall for some odd $r \geq 3$ and we refer to r as the *height* of the wall W.

Let W be a wall of a graph G and K' be the connected component of $G \setminus \mathsf{perim}(W)$ that contains $W \setminus \mathsf{perim}(W)$. The *compass* of W, denoted by $\mathsf{comp}(W)$, is the graph $G[V(K') \cup V(\mathsf{perim}(W))]$. Observe that W is a subgraph of $\mathsf{comp}(W)$ and $\mathsf{comp}(W)$ is connected.

The *layers* of an r-wall W are recursively defined as follows. The first layer of W is its perimeter. For $i = 2, \ldots, (r-1)/2$, the i-th layer of W is the $(i-1)$-th layer of the subwall W' obtained from W after removing from W its perimeter and all occurring vertices of degree one. Notice that each $(2r+1)$-wall has r layers (see Figure 1). The *central vertices* of W, denoted by $\mathsf{center}(W)$, are the two branch vertices of W that do not belong to any of its layers.

We are now in position to state the following two lemmata.

▶ **Lemma 5.** *Given a $\boxtimes \in \mathsf{OP}$, there exist two functions $f_1, f_2 : \mathbb{N}^2 \to \mathbb{N}$, and an algorithm with the following specifications:*

Find_Area(k, q, G, S)

Input: a $k \in \mathbb{N}$, an odd $q \in \mathbb{N}_{\geq 1}$, a graph G, and a set $S \subseteq V(G)$ that is a vr-planarizer of G of size at most k.

Output: One of the following:

1. *if $\boxtimes \in \{\mathsf{er}, \mathsf{ec}, \mathsf{ea}\}$: a report that (G, k) is a no-instance of $\mathrm{G}\boxtimes\mathrm{MP}\phi$.*
   - *if $\boxtimes = \mathsf{vr}$: a vertex $u \in S$ such that $S \setminus u$ is a vr-planarizer of $G \setminus u$ of size at most $k-1$ and (G, k) and $(G \setminus u, k-1)$ are equivalent instances of $\mathrm{G}\boxtimes\mathrm{MP}\phi$.*
2. *a q-wall W of G and a closed disk Δ such that*
   - *the compass of W has treewidth at most $f_2(k, q)$,*
   - *G is partially Δ-embedded, where $G \cap \Delta = \mathsf{comp}(W)$, $\mathsf{bor}(\Delta) = \mathsf{perim}(W)$,*

- $V(\mathsf{comp}(W))$ *is* $\boxtimes$*-planarization irrelevant, and*
- $N_G(S) \cap V(\mathsf{comp}(W)) = \emptyset$*, or*

3. *a tree decomposition of* G *of width at most* $f_1(k, q)$*.*

Moreover, this algorithm runs in $\mathcal{O}_{k,q}(n)$ *steps.*

By $N_G(S)$ we denote the vertices not in S adjacent in G with vertices in S. In the first possible output of the algorithm of Lemma 5 we have either a negative answer to the G$\boxtimes$MPϕ problem or an equivalent instance of G$\boxtimes$MPϕ with reduced value of k.

The main steps of the proof of Lemma 5 are the following. In case, $\boxtimes = \mathsf{ea}$ we first check whether G is planar. If not, we report a negative answer, otherwise we find a wall W in G whose size is a "big-enough" function of k and whose compass has "small-enough" treewidth using [18, Lemma 4.2]. This wall contains an (also "big-enough") subwall of W whose compass is not affected by S. In case $\boxtimes \in \{\mathsf{vr}, \mathsf{er}, \mathsf{ec}\}$, we consider the neighbors of S in the planar graph G', that is the set $N := N_G(S)$. Moreover, we consider a big enough triangulated grid Γ as a contraction of G' (using [14, Theorem 3]) and the set N_Γ of the "contraction-heirs" of the vertices of N in Γ. If $|N_\Gamma|$ is "big-enough", then we prove, using the main technical result of [10], that some of the vertices of S should be affected by every possible solution, in case $\boxtimes = \mathsf{vr}$, or that we have a no-instance, in case $\boxtimes \in \{\mathsf{er}, \mathsf{ec}\}$. If $|N_\Gamma|$ is "small-enough", then we can find a "big-enough" wall W in G whose compass is not affected by S (again using the previously mentioned result of [18]). The proof is completed by proving that this wall contains some "big-enough" subwall that is not affected by any inclusion-minimal $\boxtimes$-planarizer.

The next lemma deals with the second possible output of the algorithm of Lemma 5 and contains the "core arguments" of this paper.

▶ **Lemma 6.** *Given a Gaifman sentence* ϕ *and a* $\boxtimes \in \mathsf{OP}$*, there exists a function* $f_3 : \mathbb{N}^2 \to \mathbb{N}$ *and an algorithm with the following specifications:*

Find_Vertex$(k, \Delta, G, R, \tilde{W})$

Input: a $k \in \mathbb{N}$*, a partially* Δ*-embedded graph* G*, a set of annotated vertices* $R \subseteq V(G)$*, and a* q*-wall* $\tilde{W}$ *of* G *such that*

- $q = f_3(k, |\phi|)$*.*
- *the compass of* $\tilde{W}$ *has treewidth at most* $f_2(k, q)$*,*
- $G \cap \Delta = \mathsf{comp}(\tilde{W})$*,* $\mathsf{bor}(\Delta) = \mathsf{perim}(\tilde{W})$*,*
- $V(\mathsf{comp}(\tilde{W}))$ *is* $\boxtimes$*-planarization irrelevant, and*

Output: a vertex set $X \subsetneq V(\mathsf{comp}(\tilde{W}))$ *and a vertex* $v \in X$ *such that* (G, R, k) *is a* $(\phi, \boxtimes)$*-triple iff* $(G \setminus v, R \setminus X, k)$ *is a* $(\phi, \boxtimes)$*-triple.*

Moreover, this algorithm runs in $\mathcal{O}_{k,|\phi|}(n)$ *steps.*

Notice that the above algorithm produces a $(\phi, \boxtimes)$-triple where both R and G are reduced. To see why Lemma 4 follows from Lemma 5 and Lemma 6, observe that in the second possible output of the algorithm **Find_Area**(k, q, G, S) we can call the algorithm **Find_Vertex**$(k, \Delta, G, R, \tilde{W})$, where $\tilde{W} := W$, which outputs a vertex set $X \subsetneq V(\mathsf{comp}(\tilde{W}))$ and a vertex $v \in X$ such that (G, R, k) is a $(\phi, \boxtimes)$-triple iff $(G \setminus v, R \setminus X, k)$ is a $(\phi, \boxtimes)$-triple. Observe that since $N_G(S) \cap V(\mathsf{comp}(\tilde{W})) = \emptyset$, then $S \subseteq R \setminus X$. We insist that the algorithm **Find_Vertex**$(k, \Delta, G, R, \tilde{W})$ does not use the fact that $N_G(S) \cap V(\mathsf{comp}(\tilde{W})) = \emptyset$ but we use the latter to guarantee that $S \subseteq R \setminus X$. For the running time of Lemma 4, recall that the two algorithms of Lemma 5 and Lemma 6 run in $\mathcal{O}_{k,|\phi|}(n)$ steps.

3.2 Sketch of the proof of Lemma 6

In order to prove Lemma 6, we first find a collection $\mathcal{W}$ of "sufficiently many" subwalls of $\tilde{W}$ each with ρ layers (where ρ is "big-enough"), whose compasses are pairwise vertex-disjoint.

The key idea is to define a "characteristic" of each wall $W \in \mathcal{W}$ that encodes all possible ways that a $\boxtimes$-planarizer S of G affects $\mathsf{comp}(W)$ along with the ways that the fact that $G \boxtimes S \models \phi$ is certified by a vertex assignment to the basic variables of the Gaifman formula ϕ in $\mathsf{comp}(W)$. Recall that $\phi\|_R$ is a Boolean combination of sentences $\phi_1\|_R, \ldots, \phi_m\|_R$ so that for every $h \in [m]$,

$$\phi_h\|_R = \exists x_1 \ldots \exists x_{\ell_h} \Big(\bigwedge_{i\in[\ell_h]} x_i \in R \wedge \bigwedge_{1\le i<j\le \ell_h} d(x_i,x_j) > 2r_h \wedge \bigwedge_{i\in[\ell_h]} \psi_h(x_i)\Big),$$

where $\ell_h, r_h \geq 1$ and ψ_h is an r_h-local formula with one free variable. Notice that $\phi\|_R$ is evaluated on annotated graphs of the form (G, R). Clearly, $\phi\|_R$ is a sentence in Monadic Second Order Logic, in short, a MSO_2-sentence.

As a first step, for every $h \in [m]$, $W \in \mathcal{W}$, $S \subseteq \boxtimes\langle G, R\rangle$ of size at most k, $I_h \subseteq [\ell_h]$, and $t \in [\rho]$, we define:

$$\mathsf{sig}^{(S,I_h,t)}_{\phi_h,\boxtimes}(W) := \begin{cases} 1, & \text{if } \exists \tilde{X} = \{x_i \mid i \in I_h\} \subseteq V(\mathsf{comp}(W^{(t)}) \boxtimes S) \cap R \text{ such that } \tilde{X} \\ & \text{is } (|I_h|, r_h)\text{-scattered in } \mathsf{comp}(W^{(t)}) \boxtimes S \text{ and } G \boxtimes S \models \bigwedge_{x\in\tilde{X}} \psi_h(x), \\ 0, & \text{otherwise.} \end{cases}$$

In the above definition, $W^{(t)}$ is the subwall of W that has t layers (which are the last t layers of W) and the same center as W. Also, a set X of vertices is (α, β)-*scattered*, if $|X| = \alpha$ and there are no two vertices in X within distance $\leq 2\beta$. Intuitively, $\mathsf{sig}^{(S,I_h,t)}_{\phi_h,\boxtimes}(W) = 1$ if the application of the operation $\boxtimes$ on G as defined by S gives rise to the existence of a scattered set $\tilde{X}$ in the compass of $W^{(t)}$ so that when the vertices of $\tilde{X}$ are assigned to the basic variables of ϕ_h corresponding to I_h, the local formula ψ_h is satisfied for each $x_i \in \tilde{X}$ in the modified graph.

Next, for every $W \in \mathcal{W}$ and every $S \subseteq \boxtimes\langle G, R\rangle$ of size at most k we define:

$$\mathsf{msig}^{(S)}_{\phi,\boxtimes}(W) = \Big((\mathsf{sig}^{(S,I_1,t)}_{\phi_1,\boxtimes}(W), \ldots, \mathsf{sig}^{(S,I_m,t)}_{\phi_m,\boxtimes}(W)) \mid (I_1, \ldots, I_m, t) \in 2^{[\ell_1]} \times \cdots \times 2^{[\ell_m]} \times [\rho] \Big).$$

Clearly, $\mathsf{msig}^{(S)}_{\phi,\boxtimes}(W)$ can be seen as a $(2^\ell \cdot \rho)$-tuple of binary m-tuples, given that $\ell := \sum_{h\in[m]} \ell_h$. Let SIG be the set of all such tuples and notice that $|\mathsf{SIG}|$ is bounded by some function of k and $|\phi|$ and $\Big\{ \mathsf{msig}^{(S)}_{\phi,\boxtimes}(W) \;\Big|\; W \in \mathcal{W}, S \subseteq \boxtimes\langle G, R\rangle \text{ of size at most } k \Big\} \subseteq \mathsf{SIG}$.

It is now time to define the characteristic of a wall $W \in \mathcal{W}$. We set $r := \max_{h\in[m]}\{r_h\}$ and $d := 2(r + (\ell+1)r + r)$. We define the $(\phi, \boxtimes)$-*characteristic* of W as follows:

$$\begin{aligned} (\phi,\boxtimes)\text{-}\mathsf{char}(W) \quad = \quad \{(s,\sigma,t) \in [0,k] \times \mathsf{SIG} \times [d+1,\rho] \mid \exists S \subseteq \boxtimes\langle G,R\rangle,& \\ |S| = s,& \\ A(S) \subseteq V(\mathsf{comp}(W^{(t-d)})) \cap R,& \\ \mathsf{comp}(W) \boxtimes S \text{ is planar, and}& \\ \mathsf{msig}^{(S)}_{\phi,\boxtimes}(W) = \sigma\}.& \end{aligned}$$

Notice that all queries in the definition of $(\phi, \boxtimes)$-char(W) can be expressed in MSO_2. Indeed, this is easy to see when $\boxtimes \in \{\mathsf{vr}, \mathsf{er}, \mathsf{ec}\}$, as in this case the query "$\mathsf{comp}(W) \boxtimes S$ is planar" is trivially true, since $V(\mathsf{comp}(\tilde{W}))$ is $\boxtimes$-planarization irrelevant. In the case where $\boxtimes = \mathsf{ea}$, the MSO_2 expressibility follows from Theorem 3. As each $W \in \mathcal{W}$ has treewidth bounded by a function of k and $|\phi|$, it follows by the theorem of Courcelle that $(\phi, \boxtimes)$-char(W) can be computed in $\mathcal{O}_{k,|\phi|}(n)$ time.

We say that two walls are $(\phi, \boxtimes)$-*equivalent* if they have the same $(\phi, \boxtimes)$-characteristic. Since the collection $\mathcal{W}$ contains "sufficiently many" walls, then we can find a collection $\mathcal{W}' \subseteq \mathcal{W}$ of also "sufficiently many" walls that are pairwise equivalent. We fix a wall $W_1 \in \mathcal{W}'$ and we set $X := \mathsf{comp}(W_1^{(r)})$, where $r = \max_{h\in[m]}\{r_h\}$, and $v \in \mathsf{center}(W_1)$.

In what follows, we highlight the ideas of the proof of the fact that if (G, R, k) is a $(\phi, \boxtimes)$-triple, then $(G \setminus v, R \setminus X, k)$ is a $(\phi, \boxtimes)$-triple. We first consider a set $S \subseteq \boxtimes\langle G, R\rangle$ of size at most k that certifies that (G, R, k) is a $(\phi, \boxtimes)$-triple. Then, we pick a wall $W_2 \in \mathcal{W}' \setminus \{W_1\}$ whose compass is not affected by S. We are allowed to pick this wall since there are "sufficiently many" walls equivalent to W_1 in $\mathcal{W}'$. Our strategy is to use the fact that W_1 and W_2 are $(\phi, \boxtimes)$-equivalent in order to state a "replacement argument": we can find a $t \in [\rho]$, such that the subset S_{in} of S that affects $\mathsf{comp}(W_1^{(t)})$ and the set X of vertices of $\mathsf{comp}(W_1^{(t)})$ that are assigned to the basic variables of ϕ in order to certify that $G \boxtimes S \models \phi$, can be replaced by their "equivalent" sets $\tilde{S}$ and $\tilde{X}$ in $\mathsf{comp}(W_2^{(t)})$. As a consequence of this, for every possible solution S and vertex assignment to the basic variables of ϕ, we can find both a new solution and a new vertex assignment that "avoid" the "inner part" of W_1. This implies that the validity of any local formula of ϕ does not depend on the central vertices of W_1. Thus, we can declare one of them "irrelevant" and safely remove it from G, while storing (by reducing R to $R \setminus X$) the fact that every possible solution S and vertex assignment to the basic variables of ϕ can "avoid" the "inner part" of W_1.

To further inspect how this "replacement" is achieved, we need to dive deeper into the technicalities of the proof (through an intuitive perspective). Given a wall W, we refer to a *wall-annulus* of W as the subgraph of W that is obtained from W after removing from W all its layers, except a fixed number of consecutive layers. We think of every wall $W \in \mathcal{W}$ as divided in consecutive wall-annuli of fixed size. Since ρ is "big-enough", then we can find also "many enough" such wall-annuli. We denote each one of them by $A_i(W)$. Given a $W \in \mathcal{W}$, every wall-annulus $A_i(W)$ is divided in some regions as depicted in Figure 2.

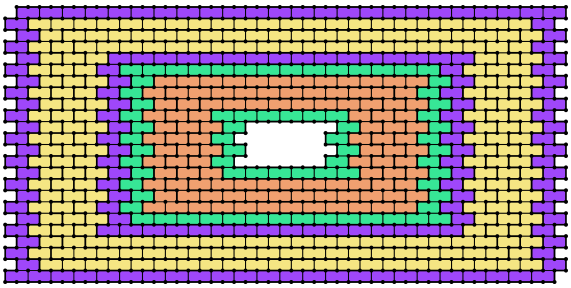


Figure 2 An example of a wall-annulus $A_i(W)$ of a wall $W \in \mathcal{W}$, together with its regions refered in the proof of Lemma 6.

The regions depicted in purple and green are consisting of r layers of the wall W (recall that $r = \max_{h\in[m]}\{r_h\}$). The regions depicted in yellow and orange are "big-enough" so as to be able to find an also "big-enough" wall-annulus that "avoids" a given vertex assignment to the basic variables of ϕ.

Since ρ is "big-enough", then we can find a wall-annulus $A_i(W_1)$ that is not affected by S. This allows us to partition S in two sets, S_{in} and S_{out} in the obvious way. The fact that W_1 and W_2 are $(\phi, \boxtimes)$-equivalent implies the existence of a set $\tilde{S}$ in W_2 certifying that $(\phi, \boxtimes)\text{-}\mathsf{char}(W_2) = (\phi, \boxtimes)\text{-}\mathsf{char}(W_1)$. Thus, by setting $S' := \tilde{S} \cup S_{\mathsf{out}}$, we have that $S' \subseteq \boxtimes\langle G, R'\rangle$, $|S'| = |S|$, and $G \boxtimes S'$ is planar. The latter is guaranteed by the fact that $V(\mathsf{comp}(\tilde{W}))$ is $\boxtimes$-planarization irrelevant, in the case $\boxtimes \in \{\mathsf{vr}, \mathsf{er}, \mathsf{ec}\}$, while in the case that

$\boxtimes = \mathsf{ea}$, the existence of the outer purple buffer of $A_i(W_1)$ (resp. $A_i(W_2)$) allows us to treat S_{in} (resp. $\tilde{S}$) and S_{out} separately, while not spoiling planarity. The last part of the proof requires to prove that $G \boxtimes S \models \phi\|_R \iff G \boxtimes S' \models \phi\|_{R'}$.

For simplicity, here we only argue why $G \boxtimes S \models \phi_h\|_R \implies G \boxtimes S' \models \phi_h\|_{R'}$ holds, as the arguments in the proof of the inverse direction are completely symmetrical. Therefore, given an (ℓ_h, r_h)-scattered set X such that ϕ_h is satisfied if the vertices of X are assigned to the basic variables of ϕ_h, we aim to find a $t \in [\rho]$ in order to "replace" the vertices in $X \cap V(\mathsf{comp}(W_1^{(t)}))$ with a set $\tilde{X}$ of vertices in $\mathsf{comp}(W_2^{(t)})$ such that the resulting vertex set X' is (ℓ_h, r_h)-scattered and ϕ_h is satisfied if the vertices of X' are assigned to the basic variables of ϕ_h. Notice that for every $h \in [m]$ such that $G \boxtimes S \models \phi_h\|_R$, these "replacement arguments" are pairwise independent.

We first deal with the possibility that the given scattered set X intersects some "inner part" of $\mathsf{comp}(W_2)$. Thus, in order to "clean" the "inner part" of $\mathsf{comp}(W_2)$, we find a wall $W_3 \in \mathcal{W}' \setminus \{W_1, W_2\}$ that "avoids" both S and X (for different $h \in [m]$, the choice of W_3 may coincide). Also, we consider a $\tilde{t} \in [\rho]$ corresponding to a layer in the yellow region of the wall-annulus $A_i(W_2)$ such that the annulus of the wall-annulus of $A_i(W_2)$ bounded by the $(t-r+1)$-th and t-th layer of W_2 is not intersected by X. Then, we "replace" the vertices of X in $\mathsf{comp}(W_2^{(\tilde{t})})$, call it X_{in} with an "equivalent" vertex set $\tilde{X}$ in $\mathsf{comp}(W_3^{(\tilde{t})})$ (notice that this is achieved by arguing for $S := \emptyset$ in the notion of $(\phi, \boxtimes)$-characteristic). This results to an (ℓ_h, r_h)-scattered set Y such that Y does not intersect $\mathsf{comp}(W_2^{(\tilde{t})})$ and $G \boxtimes S \models \bigwedge_{x \in Y} \psi_h(x)$ (see Figure 3).

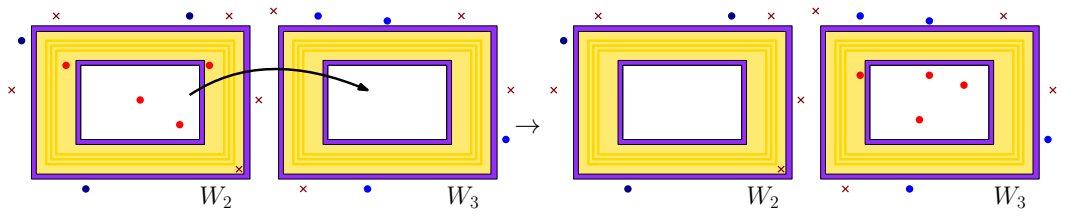


Figure 3 The "cleaning" of the "inner part" of $\mathsf{comp}(W_2)$. Left: The set $A(S)$ is depicted in cross vertices, the set $X \setminus X_{\mathsf{in}}$ is depicted in blue, and the set X_{in} is depicted in red. Right: The set $A(S)$ is depicted in cross vertices, the set $Y \setminus X_{\mathsf{in}}$ is depicted in blue, and the set $\tilde{X}$ is depicted in red.

Now, we are allowed to pick a $t \in [\rho]$ corresponding to an "orange" layer of $A_i(W_1)$ such that the annulus of the wall-annulus of $A_i(W_1)$ bounded by the $(t'-r)$-th and t'-th layer of W_1 is not intersected by X. If we set Y_{in} to be the set of vertices of Y in $\mathsf{comp}(W_1^{(t')})$, then since $\mathsf{msig}^{(S_{\mathsf{in}})}_{\phi,\boxtimes}(W_1) = \mathsf{msig}^{(\tilde{S})}_{\phi,\boxtimes}(W_2)$, then there exists a set $\tilde{Y}$ in $\mathsf{comp}(W_2^{(t')})$ that is "equivalent" to Y_{in} (see Figure 4).

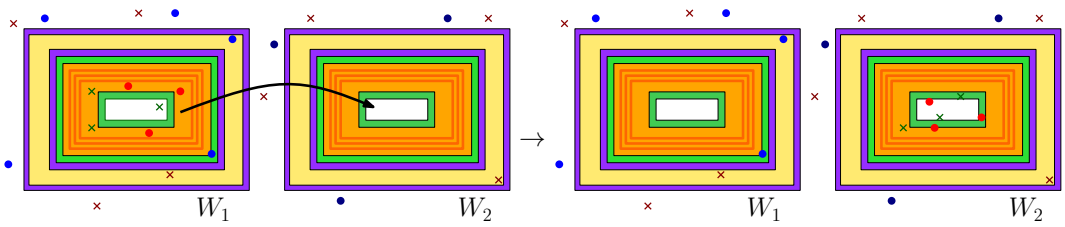


Figure 4 The last part of the proof. Left: The set $A(S_{\mathsf{out}})$ is depicted in red cross vertices, the set $A(S_{\mathsf{in}})$ is depicted in green cross vertices, the set $Y \setminus Y_{\mathsf{in}}$ is depicted in blue, and the set Y_{in} is depicted in red. Right: The set $A(S_{\mathsf{out}})$ is depicted in red cross vertices, the set $A(\tilde{S})$ is depicted in green cross vertices, the set $Y \setminus Y_{\mathsf{in}}$ is depicted in blue, and the set $\tilde{Y}$ is depicted in red.

Therefore, since $\tilde{Y}$ is in the orange region of $\mathsf{comp}(W_2)$ and Y is "avoiding" $\mathsf{comp}(W_2^{(\tilde{t})})$, then we can derive that Y and $\tilde{Y}$ are "separated" by a green and a purple region of $A_i(W_2)$. Thus, $X' := (Y \setminus Y_{\mathsf{in}}) \cup \tilde{Y}$ is an (ℓ_h, r_h)-scattered set of $G \boxtimes S'$ that "avoids" $\mathsf{comp}(W_1^{(r)})$. Moreover, ϕ_h is satisfied given that the vertices of X' of $G \boxtimes S'$ are assigned to the basic variables of ϕ_h. The proof is concluded.

3.3 Extension on graphs of bounded genus

The immediate question is whether our results can be extended to target properties that are more general than planarity (and still not FOL-expressible). The first candidate is the $\boxtimes$-Modification to g-Euler Genus and ϕ, where we ask for a set $S \subseteq \boxtimes\langle G, V(G)\rangle$ of size k such that $G \boxtimes S$ has Euler genus at most g. Notice that the property of having Euler genus at most g is not FOL-expressible. On the positive side, this property is MSO_2-expressible as there is a set $\mathcal{B}_g$ of graphs such that G has Euler genus at most g iff none of the graphs in $\mathcal{B}_g$ is a minor of G and minor containment is MSO_2-expressible. We next argue about how to adapt the techniques of this paper in order to prove that this problem can be solved in $\mathcal{O}_{k,|\phi|,g}(n^2)$ when $\boxtimes \in \{\mathsf{vr}, \mathsf{er}, \mathsf{ec}\}$. For this we first straightforwardly extend the notions of $\boxtimes$-planarization irrelevant vertex set and $\boxtimes$-planarizer to the respective notions of $\boxtimes$-*g-Euler Genus irrelevant* vertex set $\boxtimes$-*g-euler genus enforcer*. Our aim is to prove a more general version of Lemma 4 where $\boxtimes$-planarizer is replaced by $\boxtimes$-g-euler genus enforcer. The $\mathcal{O}_{k,|\phi|,g}(n^2)$ time algorithm for $\boxtimes$-Modification to g-Euler Genus and ϕ follows directly from this extended version of Lemma 4 with the same arguments as its planarization counterpart. The extended version of Lemma 4 in turn is a consequence of the generalized versions of Lemma 5 and Lemma 6 where $\boxtimes$-planarizer is replaced by $\boxtimes$-g-euler genus enforcer and $\boxtimes$-planarization irrelevant is replaced by $\boxtimes$-g-Euler Genus irrelevant. The generalized version of Lemma 5 follows as the same arguments also hold on bounded-genus graphs: the result we use from [18] has a bounded-genus analogue, the results from [14] and [10] hold for the more general graph class of apex-minor-free graphs. Also the fact that the "big-enough" q-wall that we find is $\boxtimes$-g-Euler Genus irrelevant can be proven using arguments from [25]. Having the extended version of Lemma 5, the proof of the extended version of Lemma 6 is almost identical as we still work inside a disk Δ where G is partially embedded, so that local modifications should locally respect planarity. To be precise, the main difference is that in the definition of d, we now demand that d is also lower bounded by some big-enough function of the genus which guarantees that local modifications in the disk Δ do not alter the genus of the whole graph.

4 Further research directions

In this paper we provide an algorithmic-meta theorem for the graph modifiction problem where the modification operation is in $\{\mathsf{vr}, \mathsf{er}, \mathsf{ec}, \mathsf{ea}\}$ and the target property is planarity plus being a model of some FOL-sentence ϕ. We also argued how to extend this result for modification operations in $\{\mathsf{vr}, \mathsf{er}, \mathsf{ec}\}$ for the case where instead of planarity we consider the class of graphs embeddable in a surface of Euler genus g, for fixed g. The two general challenges that we distinguish are the following.

- Pick a (non-empty) subset $\mathfrak{O}$ of $\{\mathsf{vr}, \mathsf{er}, \mathsf{ec}, \mathsf{ea}\}$ and define Graph $\mathfrak{O}$-Modification to Planarity and ϕ in the obvious way, by permitting any modification operation from $\mathfrak{O}$. It is possible (however more technical) to adapt our results for this problem in the case where $\mathsf{ea} \notin \mathfrak{O}$. However, in the case where $\mathsf{ea} \in \mathfrak{O}$ (while $|\mathfrak{O}| > 1$) the

problem becomes considerably more complicated as parts of the graph may be relocated during the modification operation (in fact, from a more general perspective, the same issue appears for the EA-MODIFICATION TO g-EULER GENUS AND ϕ problem that we avoided to consider in Subsection 3.3). We believe that this issue can be tackled using the techniques of [15]. However, the technical details of such an enterprise seem to be quite involved.

- Consider other target properties, alternative to planarity, that are not FOL-expressible. A natural challenge in this direction is to consider some finite set of graphs $\mathcal{H}$ and define the $\boxtimes$-MODIFICATION TO EXCLUDING $\mathcal{H}$-MINORS AND ϕ problem where the target property, apart from being a model of ϕ, is to exclude every graph in $\mathcal{H}$ as a minor. Notice that if $\mathcal{H}$ contains some planar graph, then the yes-instance of the problem has bounded treewidth, therefore the problem is fixed-parameter tractable due to Courcelle's Theorem. The result of this paper can be seen as $\boxtimes$-MODIFICATION TO EXCLUDING $\{K_5, K_{3,3}\}$-MINORS AND ϕ that is the simplest, however essential, version of the general problem. We conjecture that the same results can be achieved for every $\mathcal{H}$ and we believe that the techniques introduced in this paper can be the starting point of such a project.

References

1. Isolde Adler, Stavros G. Kolliopoulos, Philipp Klaus Krause, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Tight bounds for linkages in planar graphs. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2011. doi:10.1007/978-3-642-22006-7_10.
2. Chandra Chekuri and Anastasios Sidiropoulos. Approximation algorithms for euler genus and related problems. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 167–176. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.26.
3. Julia Chuzhoy. An algorithm for the graph crossing number problem. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 303–312. ACM, 2011. doi:10.1145/1993636.1993678.
4. Julia Chuzhoy, Yury Makarychev, and Anastasios Sidiropoulos. On graph crossing number and edge planarization. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1050–1069. SIAM, 2011. doi:10.1137/1.9781611973082.80.
5. Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.
6. Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *RAIRO Theor. Informatics Appl.*, 26:257–286, 1992. doi:10.1051/ita/1992260302571.
7. Bruno Courcelle and Sang-il Oum. Vertex-minors, monadic second-order logic, and a conjecture by seese. *J. Comb. Theory, Ser. B*, 97(1):91–126, 2007. doi:10.1016/j.jctb.2006.04.003.
8. Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 197–206. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.29.
9. Anuj Dawar, Martin Grohe, and Stephan Kreutzer. Locally excluding a minor. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 270–279. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.31.

10 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Bidimensional parameters and local treewidth. *SIAM J. Discret. Math.*, 18(3):501–511, 2004. doi:10.1137/S0895480103433410.

11 Zdenek Dvorák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.

12 Jörg Flum and Martin Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comput.*, 31(1):113–145, 2001. doi:10.1137/S0097539799360768.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

14 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Contraction obstructions for treewidth. *J. Comb. Theory, Ser. B*, 101(5):302–314, 2011. doi:10.1016/j.jctb.2011.02.008.

15 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Modification to planarity is fixed parameter tractable. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.28.

16 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001. doi:10.1145/504794.504798.

17 Haim Gaifman. On local and non-local properties. In J. Stern, editor, *Proceedings of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. Elsevier, 1982. doi:10.1016/S0049-237X(08)71879-2.

18 Petr A. Golovach, Marcin Kaminski, Spyridon Maniatis, and Dimitrios M. Thilikos. The parameterized complexity of graph cyclability. *SIAM J. Discret. Math.*, 31(1):511–541, 2017. doi:10.1137/141000014.

19 Petr A. Golovach, Pim van 't Hof, and Daniël Paulusma. Obtaining planarity by contracting few edges. *Theor. Comput. Sci.*, 476:38–46, 2013. doi:10.1016/j.tcs.2012.12.041.

20 Martin Grohe. Logic, graphs, and algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2008.

21 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 479–488. ACM, 2011. doi:10.1145/1993636.1993700.

22 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

23 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811. SIAM, 2014. doi:10.1137/1.9781611973402.130.

24 Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 639–648. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.45.

25 Tomasz Kociumaka and Marcin Pilipczuk. Deleting vertices to graphs of bounded genus. *Algorithmica*, 81(9):3655–3691, 2019. doi:10.1007/s00453-019-00592-7.

26 Stephan Kreutzer. Algorithmic meta-theorems. In Javier Esparza, Christian Michaux, and Charles Steinhorn, editors, *Finite and Algorithmic Model Theory*, volume 379 of *London Mathematical Society Lecture Note Series*, pages 177–270. Cambridge University Press, 2011.

27 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

28 Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.

29 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.

30 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

31 Neil Robertson and Paul D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

32 Neil Robertson and Paul D. Seymour. Graph minors. XX. wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

33 Detlef Seese. Linear time computable problems and first-order descriptions. *Math. Struct. Comput. Sci.*, 6(6):505–526, 1996.

A Constant-Factor Approximation for Directed Latency in Quasi-Polynomial Time

Zachary Friggstad
Department of Computer Science, University of Alberta, Edmonton, Canada
zacharyf@ualberta.ca

Chaitanya Swamy
Department of Combinatorics and Optimization, University of Waterloo, Canada
cswamy@uwaterloo.ca

Abstract

We consider the *directed minimum latency problem* (DirLat), wherein we seek a path P visiting all points (or clients) in a given asymmetric metric starting at a given root node r, so as to minimize the sum of the client waiting times, where the waiting time of a client v is the length of the r-v portion of P. We give the *first constant-factor approximation* guarantee for DirLat, but in quasi-polynomial time. Previously, a polynomial-time $O(\log n)$-approximation was known [12], and no better approximation guarantees were known even in quasi-polynomial time.

A key ingredient of our result, and our chief technical contribution, is an extension of a recent result of [17] showing that the integrality gap of the natural Held-Karp relaxation for *asymmetric TSP-Path* (ATSPP) is at most a constant, which itself builds on the breakthrough similar result established for *asymmetric TSP* (ATSP) by Svensson et al. [25]. We show that the integrality gap of the Held-Karp relaxation for ATSPP is bounded by a constant even if the cut requirements of the LP relaxation are relaxed from $x(\delta^{\text{in}}(S)) \geq 1$ to $x(\delta^{\text{in}}(S)) \geq \rho$ for some constant $1/2 < \rho \leq 1$.

We also give a better approximation guarantee for the *minimum total-regret problem*, where the goal is to find a path P that minimizes the total time that nodes spend in excess of their shortest-path distances from r, which can be cast as a special case of DirLat involving so-called regret metrics.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation Algorithms, Directed Latency, TSP

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.52

Related Version A full version of the paper is available at `https://arxiv.org/abs/1912.06198`.

Funding *Zachary Friggstad*: Supported by the Canada Research Chairs program and an NSERC Discovery grant.
Chaitanya Swamy: Supported in part by NSERC grant 327620-09 and an NSERC Discovery Accelerator Supplement Award.

1 Introduction

Vehicle-routing problems form a rich class of combinatorial-optimization problems that find applications in a wide variety of settings, and have been extensively studied in the Operations Research and Computer Science communities (see, e.g., [26]). These problems typically involve designing routes for vehicles to service a given underlying set of clients in the most time- and/or cost-effective fashion. A fundamental problem in this genre is the *minimum latency problem* (MLP), also known as the *traveling repairman problem* or the *delivery man problem* [1, 19, 10, 5], wherein, adopting a client-oriented perspective, we seek a route starting at a given root node and visiting all client nodes that minimizes the sum of client waiting times (or equivalently, the average client waiting time).[1]

[1] In contrast, the path-version of TSP can be seen as minimizing the *maximum* client waiting time.

We investigate *directed* MLP (DirLat), i.e., MLP in directed (or asymmetric) metrics. Formally, we are given an asymmetric metric space $(V \cup \{r\}, c)$, where V is a set of client nodes, r is a root or depot node, and $c = \{c_{u,v}\}_{u,v \in V \cup \{r\}}$ specifies the asymmetric metric: in particular, for any $u, v, w \in V \cup \{r\}$, we have $c_{u,u} = 0$, $c_{u,v} \geq 0$, and $c_{u,v} \leq c_{u,w} + c_{w,v}$. The goal is to find a Hamiltonian path P starting at the depot r to minimize $\sum_{v \in V} c_P(v)$, where $c_P(v)$ is the cost of the $r \rightsquigarrow v$ subpath of P and is interpreted as the waiting time or *latency* of node v. Throughout, we let n denote $|V|$.

Whereas we have a reasonably good understanding of MLP in undirected (i.e., symmetric) metrics – a constant-factor approximation is known (see [9] and the references therein) and recent work has also led to LP-based approaches [7, 22] for the problem – there are significant gaps in our understanding of directed MLP: the approximation factor has remained stagnant at $O(\log n)$ [12] for close to a decade, and it is not known if $\log n$ (or any super-constant function of n) constitutes a real inapproximability barrier for the problem.

2 Our contributions

Our main contribution is to provide the *first constant-factor approximation* guarantee for DirLat, albeit in quasi-polynomial time, i.e., $O\left(n^{O(\log n)}\right)$ time. This provides the first concrete indication that $\log n$ is unlikely to be an inapproximability barrier for DirLat (unless $NP \subseteq DTIME\left[n^{O(\log n)}\right]$).

▶ **Theorem 1.** *There is an $O(1)$-approximation for* DirLat *running in $O\left(n^{O(\log n)}\right)$ time.*

Our algorithm is based on a natural time-indexed linear programming (LP) relaxation that is similar to, and inspired by, the approach taken in [22] for undirected MLP. Roughly speaking, our LP (LP-Lat) utilizes variables for (v, t) pairs where $v \in V$ is a node to be visited and t is the time they should be visited, and other variables indicating the edges present on various prefixes of the optimal path. As is typical for minimum-latency problems, we utilize the LP to find rooted paths of geometrically increasing lengths and stitch them together. However, with asymmetric metrics, *both* steps present significant challenges. A key technical contribution underlying our result is a procedure for achieving the former step, namely a way of rounding the fractional prefix of the optimal path of length t to obtain a rooted path of length $O(t)$. We achieve this by generalizing some recent work by [17] on the integrality gap for *asymmetric s-t TSP-path* (ATSPP). We show that the integrality gap of a weakening of the standard LP-relaxation, where we require non-s-t cuts to only be covered to some extent strictly larger than 1 still remains a constant (see Theorem 2 below).

An interesting special case of DirLat, involves the notion of *regret* of a client: the regret of a client v lying on a rooted path P is defined as $c_P(v) - c_{r,v}$; that is, regret measures the time that node v spends waiting *in excess* of its its least possible waiting time. The notion of regret can be seen as a nuanced and better way of measuring the (dis)satisfaction of a client than the standard measure of simply considering the waiting time of a client. The latter does not differentiate between clients located at different distances from the depot and their varying expectations, and fails to take into account that a client closer to the depot that incurs a larger delay than further-away clients may face a greater level of dissatisfaction. A natural problem that arises is to find a path that minimizes the total regret of clients, i.e., to minimize $\sum_{v \in V}(c_P(v) - c_{r,v})$ (or equivalently, minimize average client regret). [2] This can be cast as a special case of DirLat by defining the regret distances,

[2] Minimizing total regret is harder than MLP in the metric (V, c). While an optimal MLP-solution for the metric (V, c) clearly yields an optimal solution to the minimum total-regret problem, this translation does not apply to near-optimal MLP-solutions. However, it is easy to see that an α-approximate solution to the minimum-total-regret problem is also an α-approximate MLP solution for the metric (V, c).

$c^{\mathsf{reg}}_{u,v} := c_{r,u} + c_{u,v} - c_{r,v}$, which form an asymmetric metric that we call the *regret metric* of (V, c). We have not attempted to optimize the constant in Theorem 1, but it is rather large; we provide a substantially improved (and explicit) guarantee for the minimum-total-regret problem (Theorem 5) when (V, c) is a *symmetric* metric (i.e. $c_{u,v} = c_{v,u}$).

Our techniques. As noted above, our algorithm utilizes a natural time-indexed LP-relaxation for DirLat. Using standard scaling techniques, one may assume all $c_{u,v}$ distances are integers that are bounded by a polynomial in n (see Appendix A). Let $T = n \cdot \max_{u,v} c_{uv}$ and notice that T is bounded by a polynomial in n. Any Hamiltonian path in the metric $(V \cup \{r\}, c)$ has length at most T, so all nodes in the optimum solution are visited by time T.

For a directed graph $G = (N, E)$, and set $S \subseteq N$, let $\delta^{\text{in}}_G(S) := \{(u, v) \in E : u \in N - S, v \in S\}$ and $\delta^{\text{out}}_G(S) := \{(u, v) \in E : u \in S, v \in N - S\}$ denote respectively the edges entering and leaving S. Define $\delta_G(S) := \delta^{\text{in}}_G(S) \cup \delta^{\text{out}}_G(S)$. If the graph is clear from the context, we may omit the subscript G. We often identify an asymmetric metric $(V \cup \{r\}, c)$ with the complete directed graph over nodes $V \cup \{r\}$, with edge costs $c_{u,v}$ for distinct $u, v \in V \cup \{r\}$. For a path P and a node v on P, recall that $c_P(v)$ is the cost of the $r \rightsquigarrow v$ subpath of P. We begin with essentially the same time-indexed LP relaxation used in [22] for the undirected MLP, specifically (LP3) in their work. For $v \in V \cup \{r\}$ and $t \in [T]$, let $x_{v,t}$ be a variable indicating that we visit v at time *exactly* t, and let $z_{uv,t}$ indicate that we finish traversing edge (u, v) at time *exactly* t. Define $[T] := \{0, 1, \ldots, T\}$.

$$\textbf{minimize}: \quad \sum_{v \in V, t \in [T]} t \cdot x_{v,t} \qquad \text{(LP-Lat)}$$

$$\textbf{subject to}: \quad \sum_{t \in [T]} x_{v,t} = 1 \qquad \forall\, v \in V \qquad (1)$$

$$\sum_{e \in \delta^{\text{in}}(S)} \sum_{t' \le t} z_{e,t'} \ge \sum_{t' \le t} x_{v,t'} \qquad \forall\, v \in V,\ \{v\} \subseteq S \subseteq V, t \in [T] \qquad (2)$$

$$x_{v,t} = \sum_{e \in \delta^{\text{in}}(v)} z_{e,t} \ge \sum_{e \in \delta^{\text{out}}(v)} z_{e,t+c_e} \qquad \forall\, v \in V, t \in [T] \qquad (3)$$

$$x, z \ge 0.$$

(We remark that the $z_{uv,t}$ variables above have a slightly different meaning from [22], wherein $z_{uv,t}$ indicated that t was traversed *by* time t. Also, we omit constraints (14) from [22], which encode that the length-t prefix of the optimal path has length at most t, as one can easily show they are implied by our slightly different approach.)

It is easy to check that an optimal solution P^* naturally corresponds to an integral solution to (LP-Lat) with the same cost as the latency of P^*. Constraints (2) admit an efficient separation oracle simply by checking for each $v \in V$ and $t \in T$ if the minimum $r - v$ cut has capacity at least $\sum_{t' \le t} x_{v,t'}$ when using a capacity of $\sum_{t' \le t} z_{e,t'}$ for each edge e.

Our proof of Theorem 1 proceeds by bucketing clients based on their fractional latencies, finding low-cost paths for these buckets, and stitching these paths together to form our final path. Our advantage over [12] comes from the fact that we guess the $O(\log T) = O(\log n)$ nodes v^*_i appearing at distances roughly 2^i along the optimum path P^*, plus their exact visiting times, ℓ^*_i, along P^*. We add constraints to (LP-Lat) to reflect these guesses. For each v^*_i, consider the nodes v that are at least, say, 2/3-visited before v^*_i but not 2/3-visited before v^*_{i-1} is visited: call this the *bucket* B_i for v^*_i. With a bit of modification, the restriction of (LP-Lat) to the times before ℓ^*_i is visited induces an LP solution with cost $O(2^i)$ for the natural ATSPP LP relaxation that covers all $v \in B_i$ to an extent of at least 2/3. That is, we get a solution to the following LP relaxation for ATSPP for $\rho = 2/3$.

$$\textbf{minimize}: \quad \sum_{u,v} c_{u,v} \cdot x_{u,v} \qquad \text{(LP-ATSPP}_\rho\text{)}$$

$$\textbf{subject to}: \quad x(\delta^{\text{out}}(v)) - x(\delta^{\text{in}}(v)) = \begin{cases} +1 & v = s \\ -1 & v = t \\ 0 & v \neq s,t \end{cases} \qquad \forall\, v \in V$$

$$x(\delta(U)) \geq 2 \cdot \rho \qquad \forall\, \emptyset \subsetneq U \subseteq V - \{s,t\}$$

$$x \geq 0.$$

The integrality gap when $\rho = 1$ was shown to be constant in [17]. We prove the following more-general result establishing a constant integrality gap for (LP-ATSPP$_\rho$) for all $1/2 < \rho \leq 1$, which is one of our chief technical results. By an *LP-relative α-approximation algorithm for* (LP-ATSPP$_\rho$) (or simply LP-relative approximation algorithm), we mean a polytime algorithm that returns an ATSPP solution of cost at most $\alpha \cdot OPT_{\text{LP-ATSPP}_\rho}$.

▶ **Theorem 2.** *There is an LP-relative $\frac{\psi}{2\rho-1}$-approximation algorithm for* (LP-ATSPP$_\rho$)*, where ψ is some absolute constant (i.e., independent of the instance).*

We do not compute the exact value of ψ, or attempt to optimize it (favoring simplicity of presentation instead). It's precise value depends on the integrality gap for ATSP, which is known to be bounded by a constant [25, 28].

Using Theorem 2, we can obtain a path P_i for each bucket B_i, of cost $O(2^i)$ spanning the nodes of $\{r\} \cup B_i$. Our final path Q will be the concatenation of these P_i paths. To obtain Theorem 1, it suffices to show that the latency under Q of each node in B_i is $O(2^i)$. For the latter, while $c(P_i) = O(2^i)$, we also need a bound of $O(2^i)$ on the cost of stitching the last node of P_{i-1} to the first node after r on P_i. This is where guessing plays the most prominent role: we show that strengthening the LP with our guess ultimately implies the new edge used to stitch P_{i-1} to P_i also has cost $O(2^i)$, as required.

As an aside, complementing Theorem 2, we show that the dependence of the integrality gap on ρ stated in Theorem 2 is asymptotically correct, and this holds even if we strengthen (LP-ATSPP$_\rho$) to require an in-flow of 1 for each $v \in V - \{s,t\}$ (but still have the relaxed cut constraints). This generalizes a similar result in [12] showing that the integrality gap of (LP-ATSPP$_\rho$) is unbounded when $\rho = 1/2$.

▶ **Theorem 3.** *The integrality gap of* (LP-ATSPP$_\rho$) *is at least $\frac{1}{2\rho-1}$, for every $1/2 < \rho \leq 1$, and this holds even if we strengthen the LP with the constraints $x(\delta^{\text{in}}(v)) = 1$ for each $v \in V - \{s,t\}$.*

Our final result pertains to the minimum total-regret problem, for which we obtain a much-improved approximation guarantee (compared to Theorem 1). Recall that this is the special case of DirLat, where the metric is the regret metric of an *undirected* metric; in the sequel, we refer to this simply as a regret metric. Our improvement stems from the following improved and explicit integrality gap for (LP-ATSPP$_\rho$) in regret metrics.

▶ **Theorem 4.** *There is an LP-relative α_ρ^{reg}-approximation algorithm for* (LP-ATSPP$_\rho$) *in regret metrics, where $\alpha_\rho^{\text{reg}} := \frac{300}{42-12\sqrt{6}} \cdot \frac{1}{2\rho-1} \approx \frac{23.8}{2\rho-1}$.*

The proof of the above result is quite different from that of Theorem 2. It exploits the structure of regret metrics, and leverages and builds upon the insights and machinery developed in [13, 14] for this class of metrics. Theorem 4 leads to the following explicit approximation factor for DirLat in regret metrics.

▶ **Theorem 5.** *There is a quasi-polynomial time 397-approximation for* DirLat *in regret metrics.*

2.1 Related Work

Nagarajan and Ravi first studied DirLat and obtained an approximation guarantee of $n^{1/2+\epsilon}$ in time $n^{O(1/\epsilon)}$ for any constant $\epsilon > 0$ [20], which extends easily to an $O(\alpha' \cdot \log^{O(1)}(n))$-approximation in quasi-polynomial time where (roughly speaking) α' is an upper bound on the integrality gap of the natural Held-Karp LP relaxation for ATSPP. They also showed α' is bounded by $O(\sqrt{n})$. Friggstad, Salavatipour, and Svitkina improved the approximation guarantee for DirLat and the upper bound on the integrality gap for ATSPP to $O(\log n)$ [12]. This is currently the best polynomial-time approximation for DirLat and no better quasi-polynomial time approximation was known before our work. If the metric is symmetric, constant-factor approximations are know. The first was given by Blum et al. [5], the best guarantee so far is a 3.59-approximation by Chaudhuri et al. [9].

Chakrabarty and Swamy [7], and Post and Swamy [22] studied LP relaxations for the undirected minimum latency problem. Using time-indexed LP relaxations, [22] obtain improved approximations for the multi-depot variant and also recover the 3.59-approximation for the single-vehicle version using an LP relaxation. Our work builds upon the ideas behind one of their LP relaxations.

The integrality-gap upper bound for ATSPP has seen various improvements since [12], which have followed analogous improvements on the integrality gap, denoted α_{ATSP}, of the Held-Karp relaxation for ATSP, its more well-studied cousin. Friggstad et al. [11] show that the integrality gap is $O(\log n/\log\log n)$ by building upon ideas introduced in [3] who proved a similar bound for α_{ATSP}. Recently, [17] shows the integrality gap is in fact $O(1)$. Specifically, they show the gap is at most $4 \cdot \alpha_{\mathsf{ATSP}} - 3$; combined with a breakthrough result of Svensson, Tarnawski, and Vegh [25], who showed $\alpha_{\mathsf{ATSP}} = O(1)$, this yields an $O(1)$ upper bound on the integrality gap for ATSPP. An even more recent development by Traub and Vygen shows that $\alpha_{\mathsf{ATSP}} \leq 22$ [28], and Traub [27] has shown the integrality gap for ATSPP is at most 43. The best lower bound known on α_{ATSP} is 2 [8].

The notion of regret has been proposed in the vehicle-routing literature (see, e.g., [24, 21]) as a more refined way of measuring client dissatisfaction than simply considering its waiting time. The underlying motivation is that since the shortest-path distance of a client from the depot is an inherent lower bound on its waiting time, it is more meaningful to measure the waiting time of a client *relative* to this lower bound. In symmetric metrics, two main regret-related problems have been investigated: finding a path (or a fixed number of paths) that minimizes maximum client regret; and finding the fewest number of bounded-regret paths to visit all clients. Constant-factor approximation algorithms are known for both problems (see [13] and the references therein). To our knowledge there is no prior work on finding provably near-optimal solutions for the total-regret (or equivalently average-regret) objective.

Outline of the paper. Section 3 presents the proofs of Theorems 1 and 5, assuming the LP-relative approximation algorithms provided by Theorems 2 and 4. Section 4 proves Theorem 2 and is concluded with the proof of Theorem 3. Finally, the proof of Theorem 4 is presented in Section 5.

3 An $O(1)$-Approximation in Quasi-Polynomial Time

In this section, we assume Theorems 2 and 4 and use them to prove Theorems 1 and 5. By scaling (see Theorem 26 in Appendix A), we may assume distances are integers bounded by a polynomial in n and that $c_{u,v} \geq 1$ for distinct nodes u, v. We also let $T = n \cdot \max_{u,v \in V \cup \{r\}} c_{u,v}$,

which is an upper bound on the cost of any Hamiltonian path. We focus on a fixed optimal path P^*. Our algorithm starts by guessing the last node v_i^* visited by P^* at some time in the interval $[2^i, 2^{i+1})$ (if any) and its exact distance $\ell_i^* \in [T]$ for each $0 \le i \le \log_2 T = O(\log n)$. Let $v_i^* = \perp$ if no such node exists for this interval. For any i, we then know that no node is visited at any time in $[2^i, 2^{i+1})$ if $v_i^* = \perp$ and, if $v_i^* \neq \perp$, we also know no node is visited at a time in the interval $(\ell_i^*, 2^{i+1})$ so we mark these times as **forbidden**. Let $A = \{i : v_i^* \neq \perp\}$ be **admissible** buckets corresponding to intervals where the optimum visits at least one node. Let $1/2 < \rho \le 1$ be a parameter we optimize later.

Algorithm 1 Directed Latency: $\boldsymbol{O(1)}$-approximation in $\boldsymbol{n^{O(\log n)}}$ time.

Input: asymmetric metric $(V \cup \{r\}, c)$ with integer distances at most T/n; parameter $\rho \in (1/2, 1]$; an LP-relative α_ρ-approximation algorithm Alg for (LP-ATSPP$_\rho$).
Output: an r-rooted path P

D1. For every choice (guess) of $v_i^* \in V \cup \{\perp\}$ for each $0 \le i \le \log_2 T$ and $\ell_i^* \in [T]$ for each such i where $v_i^* \neq \perp$, perform the following steps. Let $F = \{t \in [T] : t \in [2^i, 2^{i+1})$ where $v_i^* = \perp$ or $t \in (\ell_i^*, 2^{i+1})$ where $v_i^* \neq \perp\}$ be the forbidden times for this guess (v^*, ℓ^*) and $A = \{i \in [0, \log_2 T] : v_i^* \neq \perp\}$ the admissible buckets.

- **D1.1.** Obtain an optimal extreme point solution (x, z) to (LP-Lat) strengthened with the following additional constraints: 1) $x_{v_i^*, \ell_i^*} = 1$ for each $i \in A$ and 2) $x_{v,t} = 0$ for each $v \in V$ and $t \in F$. If the LP is infeasible, abort this guess of (v^*, ℓ^*).
- **D1.2.** For each $v \in V$, let $t(v) = t_\rho(v)$ be the minimum time such that $\sum_{t \le t(v)} x_{v,t} \ge \rho$. For $i \in A$, let $B_i = \{v \in V : t(v) \in [2^i, 2^{i+1})\}$.
- **D1.3.** For each $i \in A$, use algorithm Alg to obtain an $r - v_i^*$ path P_i spanning $\{r\} \cup B_i$.
- **D1.4.** Let P^{v^*, ℓ^*} be the path obtained by concatenating the paths $\{P_i\}_{i \in A}$ in increasing order of i, and shortcutting past repeat occurrences of r.

D2. Return the best path P^{v^*, ℓ^*} found over all guesses where the strengthening of (LP-Lat) was feasible.

Let P^* be an optimum solution and consider the iteration where (v^*, ℓ^*) is consistent with P^*. Let (x, z) be an optimum LP solution for the strengthening of (LP-Lat) by the constraints in Step (11). Clearly this strengthened LP is feasible and the value of the solution (x, z) is at most OPT, the latency of P^*.

For each $v \in V$, note that $t(v)$ is well-defined by Constraints (1). Ultimately, we will show the path P^{v^*, ℓ^*} visits each $v \in V$ by time $O(t(v))$. We begin by showing this suffices to get a constant-factor approximation.

▶ **Lemma 6.** *Let P be a path and $c \ge 1$ be such that $c_P(v) \le c \cdot t(v)$ for each $v \in V$. Then the latency of P is at most $\frac{c}{1-\rho} \cdot OPT$.*

Proof. Fix some $v \in V$. By definition of $t(v)$, we have $\sum_{t(v) \le t \le T} x_{v,t} \ge 1 - \rho$ which yields $t(v) \le \frac{1}{1-\rho} \cdot \sum_{t(v) \le t \le T} t(v) \cdot x_{v,t} \le \frac{1}{1-\rho} \cdot \sum_{t \in [T]} t \cdot x_{v,t}$. It follows that $\sum_v c_P(v) \le \frac{c}{1-\rho} \cdot OPT$. ◀

3.1 Bounding the Latency of P^{v^*, ℓ^*}

In the remainder of the proof it is convenient to view a "time-expanded" graph G_T. The nodes are pairs (v, t) with $v \in V \cup \{r\}$ and $t \in [T]$ and an edge connects (u, t) to (v, t') if $c_{u,v} = t' - t$. Observe G_T is acyclic. We can then view $z_{e,t}$ as assigning values to edges of G_T: the edge $(u, t - c_{u,v}), (v, t)$ has value $z_{(u,v),t}$ and cost $c_{u,v}$.

We begin with some observations. The constraints of (LP-Lat) mean z constitutes one unit of $(r,0)$-preflow[3] in G_T (i.e. a preflow with source vertex $(r,0)$). Namely, Constraints (3) ensure preflow is satisfied at every vertex (v,t) of G_T apart from the "source" vertex $(r,0)$. Let i' be the greatest index in A. Considering the LP constraints added in Step (11), we see $x_{v^*_{i'},\ell^*_{i'}} = 1$ and $x_{v,t} = 0$ for all $t > \ell^*_{i'}$. Thus, z must be a flow with value 1 in G_T ending at $(v^*_{i'}, \ell^*_{i'})$. Since the support of the flow z is acyclic in G_T and since one unit of flow passes through *every* (v^*_i, ℓ^*_i) node in G_T for each $i \in A$, no flow skips past node (v^*_i, ℓ^*_i). That is, no edge $((u,t),(v,t'))$ in G_T supports any z-flow if $t < \ell^*_i < t'$ for some $i \in A$, nor does any edge $((u,t),(v,t'))$ support any z-flow if $t = \ell^*_i$ yet $u \neq v^*_i$ or $t' = \ell^*_i$ yet $v \neq v^*_i$ for some $i \in A$.

Next, we recall a famous splitting-off result by Mader. The following is a slight specialization of one such result.

▶ **Theorem 7** (Mader [18]). *Let $D = (V \cup \{s\}, A)$ be a directed, Eulerian multigraph such that the $u - v$ connectivity for every $u, v \in V$ is at least k. Then for every $(u,s) \in A$ there is some $(s,v) \in A$ such that in the graph $D' = (V \cup \{s\}, A - \{(u,s),(s,v)\} \cup \{(u,v)\})$, the $u - v$ connectivity for every $u, v \in V$ is still at least k.*

Using this, we show how to compute low-cost paths covering each bucket. Roughly speaking, we show that (LP-ATSPP$_\rho$) restricted to $\{r\} \cup B_i$ with start node r and end node v^*_i has cost at most 2^{i+1}. Thus, step 13 would find a path starting at r and covering all B_i with cost at most $\alpha_\rho \cdot 2^{i+1}$ where we recall α_ρ denotes the approximation factor of the LP-relative approximation Alg.

▶ **Lemma 8.** *For each $i \in A$, we can compute a Hamiltonian $r - v^*_i$ path P_i in $G[\{r\} \cup B_i]$ with cost $\alpha_\rho \cdot 2^{i+1}$ in polynomial time.*

Proof. Let x' be a vector over edges of the metric given by $x'_{u,v} = \sum_{t<2^{i+1}} z_{(u,v),t}$ for $u, v \in V \cup \{r\}$. From the observations above, the restriction of z to edges $((u,t),(v,t'))$ where $t < 2^{i+1}$ constitutes one unit of flow from $(r,0)$ to (v^*_i, ℓ^*_i) in G_T, so x'_{uv} is then one unit of $r - v^*_i$ flow in the metric. Further, since the cost of an edge $((u, t - c_{u,v}),(v,t))$ is $c_{u,v}$ in G_T, the cost of this flow x' is exactly ℓ^*_i, which is at most 2^{i+1}.

Next we verify $x'(\delta(S)) \geq 2 \cdot \rho$ for each $S \subseteq V - \{v^*_i\}$ with $S \cap B_i \neq \emptyset$. Consider some $v \in S \cap B_i$. Constraint (2), the fact that $v \in B_i$, and the fact that $x_{v,t} = 0$ for $\ell^*_i < t < 2^{i+1}$ shows $x'(\delta^{\text{in}}(S)) = \sum_{e \in \delta(S)} \sum_{t<2^{i+1}} z_{e,t} \geq \rho$. Since x' is an $r - v^*_i$ flow and $r, v^*_i \notin S$, then flow conservation shows $x'(\delta(S)) \geq 2 \cdot \rho$.

Much like in [2] for the Prize-Collecting TSP-Path problem, one can use Theorem 7 to shortcut x' past nodes not in $B_i \cup \{r\}$ to get solution for (LP-ATSPP$_\rho$) for in the graph $G[\{r\} \cup B_i]$ (with start node $s = r$ and end node $t = v^*_i$), also with cost at most 2^{i+1}. That is, we may assume x' is rational as z is a rational vector since it is part of an extreme point of an LP with rational coefficients. Let Δ be an integer such that the vector $\Delta \cdot x'$ is integral. Consider the graph G' with nodes $V \cup \{r\} \cup \{r'\}$ where r' is a new node. The edges of G' consist of $\Delta \cdot x'_{uv}$ copies of edge uv for each $u, v \in V \cup \{r\}$, and Δ edges from v^*_i to r' and also from r' to r (each having cost 0). Note the $r - u$ connectivity for each $u \in V$ is at least $\Delta \cdot \rho$. Note, the cost of all edges in G' is at most $\Delta \cdot 2^{i+1}$.

For each $v \in V - B_i$, we iteratively perform the splitting off procedure from Theorem 7 for $s = v$. The total cost of the edges does not increase by the triangle inequality (note the edges that are removed and added all lie in the metric over $V \cup \{r\}$), and the $r - u$

[3] An s-preflow in a digraph (V,E) where $s \in V$ is an assignment $f : E \to \mathbb{R}_{\geq 0}$ such that $f(\delta^{in}(v)) \geq f(\delta^{out}(v))$ for each $v \in V - \{s\}$. The value of the preflow f is $f(\delta^{out}(s)) - f(\delta^{in}(s))$.

connectivity remains at least $\Delta \cdot \rho$ for each $u \in B_i$. After doing this for each $v \in V - B_i$, we are left with a multigraph of total edge cost cost no more than the total cost of all edges in G'. Further, if we remove all $v_i^* r'$ and $r' r$ edges, we still get the connectivity from r to any other $v \in B_i$ is at least $\Delta \cdot \rho$. If $k_{u,v}$ denotes the number of copies of uv in this new graph, setting $x''_{u,v} = k_{u,v}/\Delta$ for each $(u,v) \in G[\{r\} \cup B_i]$ yields a feasible LP solution for (LP-ATSPP$_\rho$) in the metric graph over $B_i \cup \{r\}$ (with start node r and end node v_i^*) with cost at most 2^{i+1}.

From this, the optimal solution to (LP-ATSPP$_\rho$) in $G[\{r\} \cup B_i]$ (starting at r and ending at v_i^*) has value at most 2^{i+1}. So Alg returns a Hamiltonian $r - v_i^*$ path P_i in $G[\{r\} \cup B_i]$ with cost at most $\alpha_\rho \cdot 2^{i+1}$. ◀

Next we bound the cost of stitching together the paths for the admissible buckets.

▶ **Lemma 9.** *Let P_i and $P_{i'}$ be two paths constructed in Step* (13) *for consecutive indices $i, i' \in A$. Let $u_{i'}$ be the first node on $P_{i'}$ after r and recall v_i^* is the last node of P_i. Then $c_{v_i^*, u_{i'}} \le 2^{i'+1}$.*

Proof. Note that $u_{i'} \in B_{i'}$ means $t(u_{i'}) \in [2^{i'}, 2^{i'+1})$. Also, $x_{u_{i'}, t(u_{i'})} > 0$ by definition of $t(u_{i'})$. All units of z-flow in the acyclic graph G_T pass through (v_i^*, ℓ_i^*) and also through $(v_{i'}^*, \ell_{i'}^*)$. So the restriction of z to edges $((u,t),(v,t'))$ in G_T with $\ell_i^* \le t \le t' \le \ell_{i'}^*$ constitutes one unit of $(v_i^*, \ell_i^*) - (v_{i'}^*, \ell_{i'}^*)$ flow that supports $(u_{i'}, t(u_{i'}))$. Therefore, a path decomposition of this restriction of z includes $(u_{i'}, t(u_{i'}))$ on some path. Any such path has cost exactly $\ell_{i'}^* - \ell_i^* \le 2^{i'+1}$. By the triangle inequality, $c_{v_i^*, u_{i'}} + c_{u_{i'}, v_{i'}^*} \le 2^{i'+1}$. ◀

Next, we bound the latency of each $v \in V$ along the final path P^{v^*, ℓ^*} obtained by concatenating the P_i paths for increasing indices $i \in A$ and shortcutting past all but the first occurrence of r.

▶ **Lemma 10.** $c_{P^{v^*,\ell^*}}(v) \le 4(\alpha_\rho + 1) \cdot t(v)$ *for any $v \in V$.*

Proof. Consider any $v \in V$ and say it lies on P_i. To reach v along P^{v^*,ℓ^*}, we traverse paths $P_{i'}$ for $i' < i$ plus the "stitching" edges $v_{i'}^* u_{i''}^*$ for consecutive indices $i', i'' \in A$, $i'' \le i$. By Lemma 8 and Lemma 9, the latency of v along P^{v^*,ℓ^*} can be bounded by $\sum_{i' \in A, i' \le i} \alpha_\rho \cdot 2^{i'+1} + \sum_{i' \in A, i' \le i} 2^{i'+1} \le (\alpha_\rho + 1) \cdot \sum_{i'=0}^{i} \cdot 2^{i'+1} \le 4(\alpha_\rho + 1) \cdot 2^i \le 4(\alpha_\rho + 1) \cdot t(v)$. ◀

Proof of Theorem 1. We set $\rho = 2/3$, and note that Theorem 2 yields an LP-relative $\alpha_{2/3}$-approximation algorithm, where $\alpha_{2/3} = O(1)$. The proof of Theorem 1 then follows readily from Lemmas 6 and 10 and the fact that T is bounded by a polynomial in n. ◀

We remark that even with the improved bound of $\alpha \le 22$ from [28], our approach yields an approximation ratio in the thousands. As noted earlier, we obtain a much-better guarantee for the special case of regret metrics, i.e., the minimum-total-regret problem.

Proof of Theorem 5. First, we note that a worse approximation ratio follows by choosing $\rho = 0.75$: this yields an approximation ratio α_ρ of at most 47.6 for the LP-relative algorithm in Theorem 4 for regret metrics, which combined with Lemmas 10 and 6 (and choosing ϵ sufficiently small in Theorem 26) yields a 778-approximation.

The better guarantee stated in the theorem follows by choosing the best ρ *tailored for the given instance.* (Note that there are only polynomially many combinatorially-distinct choices of ρ, and we can simply try all of these to pick the best ρ.) We analyze this by choosing a random ρ and bounding the expected latency incurred; this is similar to the use of random α-points in scheduling algorithms (see, e.g., [23]).

For $v \in V$, recall that $t_\rho(v)$ is the minimum time for $\sum_{t \le t(v)} x_{v,t} \ge \rho$. Define $\mathsf{LP}_v := \sum_{t \in [T]} t x_{v,t}$. The key is to realize that $\int_0^1 t_\rho(v) d\rho = \mathsf{LP}_v$, and leverage this in place of the coarse bound $t_\rho(v) \le \frac{\mathsf{LP}_v}{1-\rho}$ used earlier (in Lemma 6). The approximation factor α_ρ given by Theorem 4 is of the form $\frac{c}{2\rho - 1}$, where $c = 23.8$. We choose a random ρ from $(1/2, 1]$ according to the density function $8(x - 1/2)$. The expected latency incurred by a node v is then at most

$$\int_{1/2}^1 4\Big(\frac{c}{2\rho-1}+1\Big) t_\rho(v) \cdot 8(\rho - 1/2) d\rho \le 16c \cdot \int_{1/2}^1 t_\rho(v) d\rho + 16 \int_{1/2}^1 t_\rho(v) d\rho \le 16(c+1) \cdot \mathsf{LP}_v.$$

Thus, the expected total latency is at most $16(23.8 + 1) \cdot OPT \le 397 \cdot OPT$. ◀

4 Bounding the Integrality Gap of (LP-ATSPP$_\rho$)

Consider nodes V with two distinguished $s, t \in V$ ($s \ne t$) and asymmetric metric distances $c_{u,v}$ between points of V. We consider (LP-ATSPP$_\rho$) for the Asymmetric TSP Path problem where the goal is to find the cheapest Hamiltonian $s - t$ path. As mentioned earlier, the integrality gap is unbounded if $\rho \le 1/2$ [12], so we focus on the case $1/2 < \rho \le 1$. As in [17], we start with the dual of (LP-ATSPP$_\rho$).

$$\begin{aligned} \textbf{maximize}: \quad & z_t - z_s + \sum_U 2\rho \cdot y_U && && (\text{DUAL}_\rho) \\ \textbf{subject to}: \quad & z_v - z_u + \sum_{U : uv \in \delta(U)} y_U \;\le\; c_{u,v} && \forall\, u, v \\ & y \;\ge\; 0. \end{aligned}$$

Naturally, our proof borrows many steps from Köhne, Traub, and Vygen [17] but there are additional challenges we have to work through in this more general setting.

For a vector x over the edges E of the directed metric (when viewed as a complete, directed graph), let $\text{supp}(x) = \{uv \in E : x_{u,v} > 0\}$. Similarly, for a vector y over cuts of the metric let $\text{supp}(y) = \{\emptyset \subsetneq S \subseteq V - \{s,t\} : y_S > 0\}$. From now on, we focus on the graph $G = (V, \text{supp}(x))$. The proofs of Propositions 11, 12, and 14 are very similar to proofs in [17] and are omitted or just sketched in this paper.

▶ **Proposition 11.** *Given any optimal dual solution (y, z), one can find an optimal dual solution (y', z) with $\text{supp}(y')$ being laminar in polynomial time.*

In other words, we can modify y to be laminar without changing z using efficient uncrossing techniques. The proof is exactly the same as the proof in [17] essentially because the set of feasible solutions to (DUAL$_\rho$) does not change if we select different values for ρ.

The next proposition is almost identical to one in [17], but we omit the case $U = V$ in the statement. In fact, the result may not be true for this case $U = V$, we handle that separately below.

▶ **Proposition 12.** *Let x be an optimum primal solution and let and $G = (V, \text{supp}(x))$. For any $U \subseteq V - \{s,t\}$ with $x(\delta(U)) = 2\rho$, any topological ordering $U_1, \ldots, U_\ell$ of the strongly connected components of $G[U]$ satisfies:*

- $\delta^{\text{in}}(U_1) = \delta^{\text{in}}(U)$,
- $\delta^{\text{out}}(U_\ell) = \delta^{\text{out}}(U)$, *and*
- $x(\delta^{\text{out}}(U_i)) = x(\delta^{\text{in}}(U_{i+1}))$ *for any* $1 \le i < \ell$.

We sketch the proof of Proposition 12 so the reader is assured it holds, though the proof is essentially the same.

Proof sketch. Because U is a tight set, $x(\delta^{\text{in}}(U)) = \rho$. Further, $x(\delta^{\text{in}}(U_1)) \geq \rho$. All edges in $\text{supp}(x)$ entering $\delta(U_1)$ must lie in $\delta^{\text{in}}(U)$ because U_1 is the first node in the topological ordering. Thus, $\rho = x(\delta^{\text{in}}(U)) \geq x(\delta^{\text{in}}(U_1)) \geq \rho$, so equality must hold throughout and $\delta^{\text{in}}(U) = \delta^{\text{in}}(U_1)$ as we are working in the support of x. A similar statement shows $\delta^{\text{out}}(U_\ell) = \delta^{\text{out}}(U)$.

For $i > 1$ we note $\delta^{\text{in}}(U_i) \subseteq \delta^{\text{in}}(U) \cup \bigcup_{j<i} \delta^{\text{out}}(U_j)$ simply because the U_j are topologically ordered. Inductively, we have $x(\delta^{\text{out}}(U_{i-1})) = \rho$ and each edge in $\delta^{\text{in}}(U) \cup \bigcup_{j<i-1} \delta^{\text{out}}(U_j)$ is already proven to lie in $\delta^{\text{in}}(U_{j'})$ for some $j' < i$. So we see $\delta^{\text{in}}(U_i) \subseteq \delta^{\text{out}}(U_{i-1})$ and, thus,

$$\rho = x(\delta^{\text{in}}(U_{i-1})) = x(\delta^{\text{out}}(U_{i-1})) \geq x(\delta^{\text{in}}(U_i)) \geq \rho.$$

So, again, equality must hold throughout. ◀

We use a different observation to address the case $U = V$ that was omitted from Proposition 12. Intuitively, we show that it is still possible to buy a cheap set of edges to chain the strongly-connected components of G in sequence but the cost of these edges does increase relative to OPT_{LP} as $\rho \to 1/2$.

▶ **Proposition 13.** *In any topological ordering $U_1, \ldots, U_\ell$ of the strongly connected components of G, for each $1 \leq i < \ell$ there is some edge $(u, v) \in \delta^{\text{out}}(U_i) \cap \delta^{\text{in}}(U_{i+1})$ with $c_{u,v} \leq \frac{1}{2\rho-1} \cdot \sum_{u',v' \in \delta^{\text{out}}(U_i) \cap \delta^{\text{in}}(U_{i+1})} c_{u',v'} x_{u',v'}$.*

Proof. This is easy for $i = 1$ and $i = \ell - 1$. For example, we have $x(\delta^{\text{in}}(U_2) \geq \rho$ and all edges from $\delta^{\text{in}}(U_2)$ lie in $\delta^{\text{out}}(U_1)$. Thus, $x(\delta^{\text{out}}(U_1) \cap \delta^{\text{in}}(U_2)) \geq \rho$ so the cheapest edge in $\delta^{\text{out}}(U_1) \cap \delta^{\text{in}}(U_2)$ has cost at most $\frac{1}{\rho} \cdot \sum_{uv \in \delta^{\text{out}}(U_i) \cap \delta^{\text{in}}(U_{i+1}))} c_{u,v} x_{u,v}$. We finish by observing $1/\rho \leq 1/(2\rho - 1)$ as $\rho \leq 1$. A similar argument works for $i = \ell - 1$, so we now assume $1 < i < \ell - 1$.

We quickly introduce notation. For an index $1 \leq j \leq \ell$ let $U_{\leq j} = \cup_{1 \leq j' \leq j} U_{j'}$ and $U_{\geq j} = \cup_{j \leq j' \leq \ell} U_{j'}$. Let $\delta(X; Y)$ denote $\{uv \in \text{supp}(x) : u \in X, v \in Y\}$ for $X, Y \subseteq V$. With this notation, let $a = x(\delta(U_i; U_{i+1})), b = x(\delta(U_i; U_{\geq i+2})), c = x(\delta(U_{\leq i-1}; U_{i+1}))$, and $d = x(\delta(U_{\leq i-1}; U_{\geq i+1}))$. We have $a + b + c + d = x(\delta^{\text{out}}(U_{\leq i})) = 1$ as $\delta^{\text{out}}(U_{\leq i})$ is the disjoint union of the sets defining a, b, c, d. On the other hand, $\rho \leq x(\delta^{\text{out}}(U_i)) = a + b$ and $\rho \leq x(\delta^{\text{in}}(U_{i+1})) = a + c$. Therefore, $2\rho - 1 \leq (a + b) + (a + c) - (a + b + c + d) \leq a$ so $x(\delta^{\text{out}}(U_i) \cap x(\delta^{\text{in}}(U_i)) \geq 2\rho - 1$. So the cheapest edge $(u, v) \in \delta^{\text{out}}(U_i) \cap \delta^{\text{in}}(U_{i+1})$ has $c_{u,v} \leq \frac{1}{2\rho-1} \cdot \sum_{(u',v') \in \delta^{\text{out}}(U_i) \cap \delta^{\text{in}}(U_{i+1}))} c_{u',v'} x_{u',v'}$. ◀

▶ **Proposition 14.** *Let G be the support graph of an optimum solution x to* (LP-ATSPP$_\rho$) *and (y, z) an optimum dual with* $\text{supp}(y)$ *laminar. For any $U \in \text{supp}(y) \cup \{V\}$ and any $u, w \in U$ with w being reachable from u in $G[U]$, there is a $v - w$ path in $G[U]$ that crosses each set $U' \in \text{supp}(y)$ at most twice for $U' \subsetneq U$.*

Again, the proof is the same as that in [17] which only relies on Proposition 12 for $U \in \text{supp}(y)$ (i.e. not on the case $U = V$ that we omitted from the proposition in our setting). We sketch the argument briefly to ensure the reader this still holds with the omission of $U = V$ from Proposition 12.

Proof. Consider any $u - w$ path P contained in $G[U]$. Suppose $U' \in \text{supp}(y)$ is maximal among all such sets where P re-enters U' after it exits U'. Let a be the first node of P in U' and b the last node of P in U' (it could be $a = u$ or $b = v$). Inductively, replace the $a - b$ portion of P with an $a - b$ path in $G[U']$ that enters and leaves every set $U'' \in \text{supp}(y)$ at most once for $U'' \subsetneq U'$. Repeat for all such maximal $U' \in \text{supp}(y)$. ◀

4.1 Constructing the Path

Let OPT_{LP} denote the optimum solution value to (LP-ATSPP$_\rho$). Recall we let α denote an upper bound on the integrality gap of the standard Held-Karp relaxation for ATSP. We will prove the following lemma later.

▶ **Lemma 15.** *An optimal dual solution (y, z) with $\text{supp}(y)$ being laminar and $z_s - z_t \leq \frac{1}{2\rho-1} \cdot OPT_{LP}$ can be computed in polynomial time.*

Using this, we now turn to the main result of this section. Note, we are choosing simplicity in presentation over optimizing the constants in the guarantee.

Proof of Theorem 2. Complementary slackness ensures $x(\delta(U)) = 2\rho$ for each $U \in \text{supp}(y)$ Consider the edge support graph $G = (V, \text{supp}(x))$. Modify G to get an ATSP instance H by adding a new node $\overline{v}$ and edges $(t, \overline{v})$ with cost OPT_{LP} and $(\overline{v}, s)$ with cost 0.

It is easy to check that setting

$$x'_{u,v} = \begin{cases} \frac{1}{\rho} & \text{if } (u,v) \in \{(t,\overline{v}), (\overline{v}, s)\} \\ \frac{x_{u,v}}{\rho} & \text{otherwise} \end{cases}$$

yields a feasible solution for the ATSP-Circuit relaxation from [25] in instance H with cost $\frac{2}{\rho} OPT_{LP}$. Using [25], we can find a circuit W spanning all nodes in H with cost at most $\frac{2\alpha}{\rho} OPT_{LP}$ in polynomial time. This circuit must use the $(t, \overline{v})$ edge at least once as it visits $\overline{v}$. By deleting occurrences of $(t, \overline{v})$ and $(\overline{v}, s)$, we get $s - t$ walks $W_1, \ldots, W_k$ in G that collectively span all nodes in V with $\sum_j c(W_j) \leq \frac{2\alpha}{\rho} \cdot OPT_{LP} \leq 4\alpha \cdot OPT_{LP}$. We also point out $k \leq 4\alpha$ because in removing the k edges incident to $\overline{v}$ to get the walks W_i, we removed a total edge cost of $k \cdot OPT_{LP}$ from a circuit whose cost is at most $4\alpha \cdot OPT_{LP}$, so $k \leq 4\alpha$. The walks $W_1, \ldots, W_\ell$ are depicted in the top of Figure 1.

Let $U_1, \ldots, U_\ell$ be the strongly connected components of the support graph G. For each U_i, let $\mathcal{W}_i = \{j : W_j \text{ visits a node in } U_i\}$ and note $|\mathcal{W}_i| \leq k$. Unlike the case $\rho = 1$ in [17], it could be that $j \notin \mathcal{W}_i$ for some U_i and W_j. For each $1 \leq i \leq \ell$ and each $j \in \mathcal{W}_i$, let $R_{i,j}$ denote the restriction of W_j to U_i. Now, if some W_j enters U_i, then once it leaves it cannot re-enter because U_i is a strongly connected component of G. So $R_{i,j}$ is a single walk for each $j \in \mathcal{W}_i$. For such (i, j), let u^i_j and v^i_j be the first and last nodes of W_j in U_i.

Order $\mathcal{W}_i$ as $j_1 < j_2 < \ldots < j_{|\mathcal{W}_i|}$. By Proposition 14 and the fact each U_i is a strongly connected component, we can find paths P_{i,j_m} for $j_m \in \mathcal{W}_i$ from $v^i_{j_m}$ to $u^i_{j_{m+1}}$ (or u^i_1 if $m = |\mathcal{W}_i|$) where $P_{i,j}$ enters and exits each $U' \in \text{supp}(y)$ with $U' \subsetneq U_i$ at most once and does not cross any other set in $\text{supp}(y)$. Then, for each i we get a circuit C_i spanning all nodes of U_i by adding the paths $P_{i,j}$ for $j \in \mathcal{W}_i$ to the walks $R_{i,j}$.

By Proposition 13, for each $1 \leq i < \ell$ there are edges $u'_i v'_{i+1} \in \delta^{\text{out}}(U_i) \cap \delta^{\text{in}}(U_{i+1})$ with cost at most $\frac{1}{2\rho-1}$ times the fractional cost of edges in $\delta^{\text{out}}(U_i) \cap \delta^{\text{in}}(U_{i+1})$. Also, say $v'_1 = s$ and $u'_\ell = t$. By fully traversing each C_i starting at v'_i and then continuing to follow it again to reach u'_i, we get $v'_i - u'_i$ walks W'_i spanning U_i. The final path P we output is the concatenation of the walks $W'_1, W'_2, \ldots, W'_\ell$. Let $S = \{v'_i u'_{i+1} : 1 \leq i < \ell\}$ be the edges used to "stitch" these walks W'_i together. The bottom of Figure 1 depicts the final construction of P.

To bound the cost of P, first observe $c(S) \leq \frac{1}{2\rho-1} OPT_{LP}$ as the sets $\delta^{\text{out}}(U_i) \cap \delta^{\text{in}}(U_{i+1})$ are disjoint for $1 \leq i < \ell$. To bound the cost of the cycles C_i, we define a modified cost $c^y_{uv} = \sum_{U : uv \in \delta(U)}$ and observe $c(Q) = z_v - z_u + c^y(Q)$ for any $u - v$ path Q (the z-values for internal nodes of Q cancel).

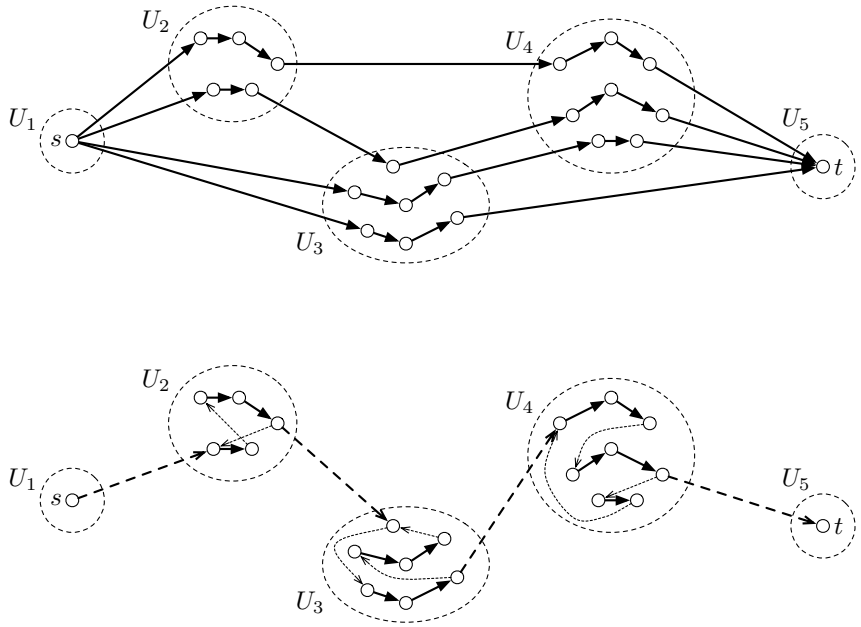


Figure 1

Top: A depiction of $s-t$ walks $W_1, \ldots, W_k$ with $k = 4$ and the strongly-connected components $U_1, \ldots, U_\ell$ with $\ell = 5$.

Bottom: The solid edges are the restrictions of the walks W_i to the strongly-connected components: these are the $R_{i,j}$ walks. The thin dashed edges in each component are the paths $P_{i,j}$ that stitch these $R_{i,j}$ to form a circuit C_i over the strongly connected component U_i. Finally, the dashed edges between components are the edges in S obtained from Proposition 13. The final path P is obtained by visiting the U_i consecutively using these dashed edges, where each visit traverses C_i fully and then travels to the start of the edge exiting U_i.

By complementary slackness, $c_{u,v} = z_v - z_u + c^y_{uv}$ for each $uv \in \text{supp}(x)$. Each C_i was formed by stitching together endpoints of $R_{i,j}$ using paths $P_{i,j}$. Each $P_{i,j}$ crosses each $U' \in \text{supp}(y), U \subsetneq U_i$ at most twice and does not cross any set in $\text{supp}(y)$ not contained in U_i. Further, no two $P_{i,j}, P_{i',j'}$ paths for $i \neq i'$ can cross the same $U' \in \text{supp}(y)$ because the two paths are contained in different components of G.

Each $U' \in \text{supp}(y)$ is crossed by at most k paths of the form $P_{i,j}$ meaning $\sum_{i,j} c^y(P_{i,j}) \leq \sum_{i,j} z_{v^i_j} - z_{u^i_j} + 2k \cdot \sum_U y_U$. We also have $c^y(R_{i,j}) = z_{u^i_j} - z_{v^i_j} + c(R_{i,j})$. Therefore, $\sum_i c^y(C_i) = \sum_i \sum_{j \in \mathcal{W}_{i,j}} c^y(P_{i,j}) + c^y(R_{i,j}) \leq 2k \sum_U y_U + \sum_{i,j \in \mathcal{W}_i} c(R_{i,j}) \leq 2k \sum_U y_U + \sum_j c(W_j)$ (the z terms for the endpoints of the $R_{i,j}$ cancel out in the first inequality).

But $c(C) = c^y(C)$ for any cycle C because, again, the z-terms cancel out. So

$$\begin{array}{rcl} c(P) \leq c(S) + 2 \cdot \sum_i c(C_i) & \leq & \frac{OPT_{LP}}{2\rho - 1} + 2 \sum_{i=1}^k c(W_i) + 2k \sum_U y_U \\ \leq \frac{OPT_{LP}}{2\rho-1} + 4\alpha \cdot OPT_{LP} + 2k \sum_U y_U & \leq & O(1) \cdot \frac{1}{2\rho-1} \cdot OPT_{LP} + \frac{k}{\rho} \left(OPT_{LP} + z_s - z_t \right) \\ \leq \frac{O(1)}{2\rho-1} \cdot OPT_{LP} + \frac{k}{\rho} \cdot (z_s - z_t). & & \end{array}$$

Here, $O(1)$ refers to some constant that is independent of ρ where we also recall k is bounded by a constant as well. Using Lemma 15 to bound $z_s - z_t$ finishes the proof. ◀

4.2 Bounding $z_s - z_t$: Proof of Lemma 15

We prove Lemma 15 to finish the proof of Theorem 2. Our approach is more direct than [17], they used an argument that shifts LP weight around to show that $y_U > 0$ implies U is not an $s-t$ separator in the support graph $G = (V, \text{supp}(x))$. We establish this fact using complementary slackness applied to the LP used to find the optimal solution to DUAL_ρ with minimum possible $z_s - z_t$. We comment that their proof could also be adapted to show what we want, we are presenting this alternative proof because we feel it is more naturally motivated: we already want to minimize $z_s - z_t$ among all optimal duals so it is natural to ask what complementary slackness gives for $y_U > 0$.

Let x be an optimal primal solution to LP-ATSPP$_\rho$. Note that if we restricted the variables of (LP-ATSPP$_\rho$) and the constraints of (DUAL_ρ) to supp(x) then x and (y,z) remains optimal. For any feasible solution (y,z) to (DUAL_ρ), we know $z_t - z_s \le OPT_{LP}$ because $y \ge 0$. So the following LP is bounded. Note, we first solved (LP-ATSPP$_\rho$) to compute OPT_{LP} which is then a fixed value (not a variable) in DUAL_ρ-Z below.

$$\textbf{maximize}: \quad z_t - z_s \qquad (\text{DUAL}_\rho\text{-Z})$$

$$\textbf{subject to}: \quad z_t - z_s + \sum_{\emptyset \subsetneq U \subseteq V - \{s,t\}} 2\rho \cdot y_U \ \ge \ OPT_{LP} \qquad (4)$$

$$z_v - z_u + \sum_{U : uv \in \delta(U)} y_U \ \le \ c_{u,v} \qquad \forall\, u,v \in \text{supp}(x) \qquad (5)$$

$$y \ \ge \ 0.$$

The second constraint asserts (y,z) is a feasible solution for (DUAL_ρ), so the first constraint then asserts it is an optimal solution for DUAL_ρ In fact, in any feasible solution the first constraint must hold with equality. We prove $z_s - z_t \le \frac{1}{2\rho - 1} \cdot OPT_{LP}$ for an optimal solution (y,z) to (DUAL_ρ-Z). With this, we finish the proof of Lemma 15 by simply noting that Proposition 11 shows we can uncross the support of y while leaving z unchanged.

The LP that is dual to (DUAL_ρ-Z) has a variable κ for Constraint (4) of (DUAL_ρ-Z) and new variables x'_{uv} for each instance uv of Constraint (5).

$$\textbf{minimize}: \quad \sum_{uv \in \text{supp}(x)} c_{u,v} \cdot x'_{uv} - OPT_{LP} \cdot \kappa$$

$$\textbf{subject to}: \quad x'(\delta^{\text{out}}(v)) - x'(\delta^{\text{in}}(v)) \ = \ \begin{cases} 1+\kappa & v = s \\ -1-\kappa & v = t \\ 0 & v \ne s,t \end{cases} \qquad \forall\, v \in V$$

$$x'(\delta(U)) \ \ge \ 2\rho \cdot \kappa \qquad \forall\, \emptyset \subsetneq U \subseteq V - \{s,t\}$$

$$x', \kappa \ \ge \ 0.$$

▶ **Lemma 16.** *In an optimal solution (y,z) to $DUAL_\rho$-Z, if $y_U > 0$ then there is an $s-t$ path in the graph $G[V-U]$.*

Proof. Let x' be an optimal solution to the dual of (DUAL_ρ-Z). Then $y_U > 0$ implies $x'(\delta(U)) = 2\rho \cdot \kappa$ so, by flow conservation, $x'(\delta^{\text{in}}(U)) = \rho \cdot \kappa$.

On the other hand, x' constitutes an $s-t$ flow of value $1+\kappa$. Consider a decomposition of x' into paths and cycles. The total weight of paths that do not enter U is at least $1 + \kappa - \rho \cdot \kappa = 1 + (1-\rho) \cdot \kappa > 0$. Thus, there is an $s-t$ path in G that does not pass through U. ◀

Continuing as in [17], let $U_1, \ldots, U_k$ be the maximal sets in $\text{supp}(y)$. In the graph G' obtained by contracting each U_i, we have by Lemma 16 that for each contracted node U_i there is an $s-t$ path in G' that avoids U_i. By a variant of Menger's Theorem (Lemma 9 in [17]), there are node-disjoint $s-t$ paths P_1, P_2 in G'. Consider the edges of P_1 and P_2 in G. For any U_i, at most one of P_1 or P_2 enters (and exits) U_i. Suppose it is the case that one of them $\overline{P} \in \{P_1, P_2\}$ enters U_i. Let u, v be the first and last nodes of $\overline{P}$ as it passes through U_i. By Proposition 14, we can find a $u-v$ path in $G[U_i]$ that crosses each $U' \in \text{supp}(y)$ contained in U at most twice, and does not cross any other set in $\text{supp}(y)$. Add these edges to $\overline{P}$.

Do this for each U_i that is entered by some $\overline{P} \in \{P_1, P_2\}$. We get paths P_1', P_2' using only edges in $\text{supp}(x)$ that, collectively, cross each set in $\text{supp}(y)$ at most twice. Thus, $0 \leq c(P_1) + c(P_2) = c^y(P_1) + c^y(P_2) + 2 \cdot (z_t - z_s) \leq 2 \cdot \sum_{U \in \text{supp}(y)} y_U + 2 \cdot (z_t - z_s)$. Multiplying the terms in this bound by ρ and then subtracting $(2\rho - 1) \cdot (z_t - z_s)$ from both sides, we see $(2\rho - 1) \cdot (z_s - z_t) \leq \sum_{U \in \text{supp}(y)} 2\rho \cdot y_U + z_t - z_s = OPT_{LP}$. ◀

4.3 A Bad Example for (LP-ATSPP$_\rho$)

We show that the dependence on the factor $\frac{1}{2\rho-1}$ in our analysis of the integrality gap of (LP-ATSPP$_\rho$) is asymptotically tight.

Proof of Theorem 3. Consider the following metric depicted in Figure (2), which is inspired from the example showing the integrality gap is unbounded if $\rho = 1/2$ from [12]. The solid edges have cost 0 and the dashed edges have cost 1. The cost of all other edges not depicted is the shortest path distance in this graph (using a cost of 1 if there is no path in this graph). The number beside each edge uv indicates the value of $x_{u,v}$. It can be easily check that this is a feasible solution for (LP-ATSPP$_\rho$) even if we added the constraints $x(\delta^{\text{in}}(v)) = 1$ for each $v \in V - \{s, t\}$. An optimal integral solution must use an edge with cost 1, yet this LP solution only has cost $2\rho - 1$ so the integrality gap of (LP-ATSPP$_\rho$) is at least $\frac{1}{2\rho-1}$. ◀

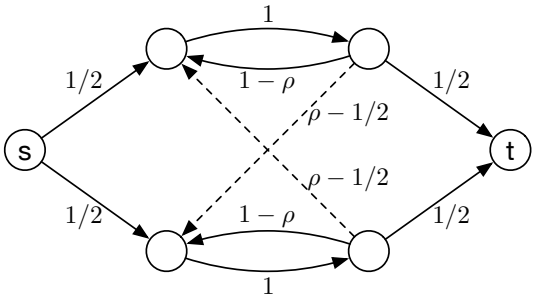


Figure 2 The bad integrality gap example for LP-ATSPP$_\rho$.

5 An Improved Integrality Gap Bound for (LP-ATSPP$_\rho$) in Regret Metrics

Let V be nodes and $s, t \in V$ be the start and end points. Let c be *symmetric* metric distances $c_{u,v} \geq 0$. For each $u, v \in V$, let $c^{\text{reg}}_{u,v} = c_{r,u} + c_{u,v} - c_{r,v}$ be the *regret* metric induced by c. It is convenient to consider a complete directed graph over V where for distinct $u, v \in V$ we have $c_{u,v} = c_{v,u}$ yet (u, v) and (v, u) are themselves distinct edges: the *bidirected* variant of the natural undirected graph associated with (V, c). The following observations about regret metrics can be found in [13].

▶ **Observation 17.** *If c is a metric (asymmetric or symmetric) then c^{reg} is an asymmetric metric. For any $u, v \in V$ and any $u - v$ path P, $c(P) = c^{reg}(P) + c_{u,v}$. For any cycle C, $c(C) = c^{reg}(C)$.*

We consider integrality gap bounds for (LP-ATSPP$_\rho$) when the metric is a regret metric. In [14], it was shown the integrality gap bound is 2 in the standard case $\rho = 1$ and that this is tight. For the purpose of getting better approximations for DirLat in regret metrics (i.e. the problem of minimizing the average time a node v waits *in excess* of their shortest path distance $c_{r,v}$ from the depot), we give explicit integrality gap bounds for the more general case $1/2 < \rho \leq 1$.

Note, in the case $\rho = 1$ that the analysis from [14] produces a stronger result. But the analysis does not extend in any clear way to the case $\rho < 1$. We begin by recalling the following structural result by Bang-Jensen et al about decomposing preflows into branchings [4], which was made efficient by Gabow [15] (see also [22]).

▶ **Theorem 18** (Bang Jensen et al. [4], Gabow [15], Post and Swamy [22])**.** *Let $D = (\{r\} \cup V, A)$ be a directed graph and $x \in \mathbb{Q}^A_{\geq 0}$ be a preflow. Let $\lambda_v := \min_{\{v\} \subseteq S \subseteq V} x(\delta^{\text{in}}(S))$ be the $r - v$ connectivity in D under capacities $\{x_a\}_{a \in A}$. Let $K > 0$ be rational. We can obtain out-branchings $B_1, \ldots, B_q$ rooted at r, and rational weights $\gamma_1, \ldots, \gamma_q \geq 0$ such that $\sum_{i=1}^q \gamma_i = K, \sum_{i: q \in B_i} \gamma_i \leq x_a$ for all $a \in A$, and $\sum_{i: v \in B_i} \geq \min\{K, \lambda_v\}$ for all $v \in V$. Moreover, such a decomposition can be computed in time that is polynomial in $|V|$ and the bit complexity of K and x.*

We require a definition and results from [13], some of which are adaptations of concepts in [6].

▶ **Definition 19.** *Let P be a path starting at s. For each $uv \in P$, say uv is **red** on P if there are nodes x, y on the $s - u$ portion of P_i and $v - t$ portion of i, respectively, such that $c_{r,x} \geq c_{r,y}$. For each $v \in P$, let red(v, P) be the maximal subset of red edges of the subpath of P containing v. Note, red(v, P) could be empty if v is not incident to a red edge. The **red intervals** of P are the maximal subpaths of its red edges.*

Intuitively, the red edges are part of intervals of P that do not make progress toward reaching t. Their total c^{reg}-costs can be shown to be comparable to their total c-costs, which is formalized as follows.

▶ **Lemma 20** (Blum et al [6])**.** *For any $s - t$ path P, $\sum_{uv \text{ red on } P} c_{u,v} \leq \frac{3}{2} c^{reg}(P)$.*

Further, if we were to keep at most one node from each maximal red interval of edges and shortcut past the other nodes, the resulting path $s = v_0, v_1, \ldots, v_k = t$ has $c_{r,v_i} < c_{r,v_{i+1}}$. So the union of any collection of paths that are shortcut in such a way forms an acyclic graph.

Now, a solution to (LP-ATSPP$_\rho$) can be viewed as a preflow of value 1 rooted at s with $\lambda_v \geq \rho$ for each $v \in V - t$ and $\lambda_t = 1$. From this observation, we round a solution using techniques from [13]. The full description is in Algorithm 2. Here, $1/2 < \delta < \rho$ is some parameter we set later to optimize the performance of the algorithm.

▶ **Lemma 21.** *The paths P_i from Step 2 satisfy $\sum_i \gamma_i \cdot c^{\text{reg}}(P_i) \leq 2 \cdot OPT_{LP}$.*

Proof. In [13], it is observed for any $s - t$ path P that $c^{\text{reg}}(P) = c(P) - c_{s,t}$ and that $c(C) = c^{\text{reg}}(C)$ for any cycle C. Thus, as x is an $s - t$ flow with value 1 we have $OPT_{LP} = \sum_{uv} c^{\text{reg}}_{u,v} x_{u,v} = (\sum_{uv} c_{u,v} x_{u,v}) - c_{s,t}$. This can be seen by, say, comparing the c^{reg}-cost with the c-cost of paths and cycles in a path/cycle decomposition of x.

Algorithm 2 Rounding (LP-ATSPP$_\rho$) in regret metrics.

Input: asymmetric metric $(V \cup \{r\}, c^{\text{reg}})$ obtained from symmetric distances c.
Output: an Hamiltonian $s-t$-rooted path P.

R1. Solve (LP-ATSPP$_\rho$) to get an optimal extreme point solution x with value OPT_{LP}.

R2. Use Theorem 18 to find a convex combination of out-branchings $B_1, \ldots, B_q$ rooted at s and weights $\gamma_1, \ldots, \gamma_q \geq 0$ summing to 1 such that t lies on each B_i and each $v \in V - \{s,t\}$ lies on at least a ρ-fraction of these branchings. Turn each B_i into a $s-t$ path P_i by adding the reverse (v,u) of each arc $(u,v) \in B_i$ that does not appear on the unique $s-t$ path in B_i and shortcutting the resulting Eulerian $s-t$ walk past repeated nodes.

R3. Define a cut requirement function $f : 2^V \to \{0,1\}$ where $f(S) = 1$ if $\sum_{i:\text{red}(v,P_i) \subseteq S} \gamma_i < \delta$ for all $v \in S$. Observe f is downward-monotone: $f(S) \geq f(T)$ for sets $\emptyset \subsetneq S \subseteq T$. Use the LP-based 2-approximation in [16] to find a forest of undirected edges F such that $|\delta(S) \cap F| \geq f(S)$. Let $\mathcal{C}$ be the components of F and let $C_1, \ldots, C_{|C|}$ be cycles on each component of F obtained by doubling and shortcutting each tree in F. For each cycle C_j of $\mathcal{C}$, let $w \in C_i$ be some *witness node* such that $\sum_{i:\text{red}(w,P_i) \subseteq V} \gamma_i \geq \delta$. Let W be the set of all witness over all C_j (note, it could be $W \cap \{s,t\} \neq \emptyset$). View each C_j as being traversed in some arbitrary direction.

R4. For each P_i, let P_i^W be the set of all nodes in $W \cap P_i$ such that all nodes of $\text{red}(w, P_i)$ are contained in the nodes of a single cycle C_j. Shortcut P_i past nodes not in $P_i^W \cup \{s,t\}$ and call this path P_i'. Note the nodes of P_i' lie in $W \cup \{s,t\}$.

R5. View P_i' with associated weights γ_i/δ as the path decomposition of an acyclic $s-t$ flow z with value $1/\delta$ with $z(\delta(w)) \geq 1$ for each $w \in W$. Further, $z(\delta^{\text{out}}(s)) = 1/\delta < 2$. By integrality of flows with upper- and lower-bounds on each node, we may decompose z as a convex combination of integral flows satisfying these bounds such that each flow supported consists of either 1 or 2 paths. Let P be the cheapest path among the flows with only one path in this decomposition. Note that P is an $s-t$ path spanning all of W.

R6. Complete P into a Hamiltonian $s-t$ path by adding all edges of the cycles C_i and shortcutting the resulting Eulerian walk.

Each P_i is obtained by adding the reverse of each edge uv of B_i not on the $s-t$ path in B_i (and then shortcutting the resulting Eulerian walk). Thus, $c(P_i) \leq 2 \cdot c(B_i) - c_{s,t}$ so $c^{\text{reg}}(P_i) \leq 2 \cdot (c(B_i) - c_{s,t})$. Thus, $\sum_i \gamma_i \cdot c^{\text{reg}}(P_i) \leq 2 \cdot \sum_i \gamma_i \cdot (c(B_i) - c_{s,t}) = (2 \cdot \sum_i \gamma_i \cdot c(B_i)) - 2 \cdot c_{s,t}$. Now, the convex combination of the B_i is dominated by x, so $\sum_i \gamma_i \cdot c(B_i) \leq \sum_e x_e \cdot c_e$. Finally, as x constitutes one unit of $s-t$ flow, the c-cost of x differs from the c^{reg}-cost of x exactly by $c_{s,t}$, so we finally see $\sum_i \gamma_i \cdot c^{\text{reg}}(P_i) \leq 2 \cdot OPT_{LP}$. ◀

The proofs of the following two lemmas proceed in a way that is very similar to related results [13] (though, their end goal was quite different). We defer their proofs to the end of this section.

▶ **Lemma 22.** *In Step 2, the function f is downward-monotone and $\sum_j c^{reg}(C_j) \leq \frac{6}{\rho - \delta} OPT_{LP}$.*

▶ **Lemma 23.** *The graph over V with edges $\cup_{i=1}^q P_i'$ is an acyclic graph. Further, for each $w \in W$ we have $\sum_{i:w \text{ lies on } P_i'} \gamma_i \geq \delta$. Finally, $\sum_{i=1}^q c^{reg}(P_i') \leq 2 \cdot OPT_{LP}$.*

We now describe how to complete the analysis.

▶ **Lemma 24.** *In Step 5, the flow z has acyclic support, sends $1/\delta$ units of flow from s to t, and has $z(\delta^{\text{in}}(w)) \geq 1$ for each $w \in W$. The resulting path P has cost $\frac{2}{2\delta - 1} \cdot OPT_{LP}$.*

Proof. We have $\sum_i \gamma_i/\delta = 1/\delta$. As each P_i' is an $s-t$ flow, we have z given by $z_{uv} = \sum_{i:uv \in P_i} \gamma_i/\delta$ is an $s-t$ flow of value $1/\delta$. Then by Lemma 23, the support of z is acyclic, $z(\delta^{\text{in}}(w)) \geq 1$ for each $w \in W$, and $\sum_{uv} c^{\text{reg}}_{u,v} z_{uv} \leq \frac{2}{\delta} \cdot OPT_{LP}$.

By integrality of flows with integral lower- and upper-bounds on the flow through each vertex, z may be decomposed into a convex-combination of integral flows f satisfying the lower-bound $f(\delta^{\text{in}}(w)) \geq 1$ for each $w \in W$ and $1 \leq f(\delta^{\text{out}}(s)) \leq 2$. Furthermore, the fraction of these flows f with $f(\delta^{\text{out}}(s)) = 1$ is exactly $2 - 1/\delta$, so the c^{reg}-cost of one such flow is at most $\frac{1}{2-1/\delta}\frac{2}{\delta} \cdot OPT_{LP} = \frac{2}{2\delta-1} \cdot OPT_{LP}$. Such a flow f has no cycles because the support of z is acyclic, so the edges supported by f form an $s-t$ path spanning all $w \in W$. ◀

The final path is formed from grafting the cycles $C_1, \ldots, C_{|\mathcal{C}|}$ into P, so the above results yield the following.

▶ **Theorem 25.** *The final path computed in Step 6 is a Hamiltonian $s-t$ path with c^{reg}-cost at most $\left(\frac{6}{\rho-\delta} + \frac{2}{2\delta-1}\right) \cdot OPT_{LP}$.*

Proof. By Lemma 24, the path P is an $s-t$ path spanning W with c^{reg}-cost at most $\frac{2}{2\delta-1} \cdot OPT_{LP}$. Each cycle C_j over a component in $\mathcal{C}$ contains precisely one node in W, so the graph $P \cup_{j=1}^{|\mathcal{C}|} C_j$ has an Eulerian $s-t$ walk that visits all nodes. By Lemma 22, the total c^{reg}-cost of all cycles is at most $\frac{6}{\rho-\delta} \cdot OPT_{LP}$. The result follows because shortcutting this Eulerian walk to get a Hamiltonian path does not increase the cost of the walk, by the triangle inequality. ◀

By setting $\delta = \frac{(2\sqrt{6}-1)\cdot\rho+6-\sqrt{6}}{10}$ (which optimizes the parameter), we get our main result showing the integrality gap is at most $\frac{300}{42-12\sqrt{6}} \cdot \frac{1}{2\rho-1} \approx \frac{23.8}{2\rho-1}$.

Proof of Lemma 22. That f is downward monotone is direct from the definition. We construct a vector x' over edges the undirected complete graph with nodes V with edge costs c. That is, for each undirected edge uv let $x'_{uv} = \frac{1}{\rho-\delta} \sum_{\substack{i: uv \text{ or } vu \\ \text{is red on } P_i}} \gamma_i$. We first claim $x'(\delta(S)) \geq f(S)$ for each $\emptyset \subsetneq S \subseteq V$. That is, suppose S is such that $f(S) = 1$ and let v satisfy $\sum_{i:\text{red}(v,P_i)\subseteq V} \gamma_i < \delta$. Since v lies on a ρ-fraction of paths in total, this means a $(\rho-\delta)$-fraction of paths P_i have some edge of $\text{red}(v, P_i)$ crossing S, as required.

From Lemma 20, the total c-cost of all red edges on P_i is at most $\frac{3}{2}c^{\text{reg}}(P_i)$. Thus, $\sum_{uv} c_{u,v}x'_{uv} \leq \frac{3}{2}\frac{1}{\rho-\delta}OPT_{LP}$. From using the LP-based 2-approximation in [16], the c-cost of the result forest is then at most $\frac{3}{\rho-\delta}OPT_{LP}$. By doubling the edges to get the cycles C_j, $\sum_j c(C_j) \leq \frac{6}{\rho-\delta}OPT_{LP}$. Finally, we chose an arbitrary direction for traversing each C_j but the c^{reg}-cost of a directed cycle is the same as its c-cost, so the result follows. ◀

Proof of Lemma 23. We claim that we do not keep two nodes from any red interval for each P_i when we form P'_i. But this is immediate from the fact that no cycle C_j contains two nodes of W.

By the definition of red intervals, any path P' obtained from a path P by shortcutting past all but one node in each red interval yields has its nodes appearing in strictly distance-increasing order. So, the P'_i paths all start at the same location, all end at the same location, and their internal nodes strictly increase in distance from s. So the union of all P'_i is an acyclic graph.

Now, consider some $w \in W$ and say it lies on cycle C_j. At least a δ-fraction of paths P_i spanning w satisfy $\text{red}(w, P_i) \subseteq C_j$ because $f(V(C_j)) = 0$, so each $w \in W$ lies on at least a δ-fraction of paths P'_i.

Since P'_i are obtained by shortcutting nodes from P_i, $\sum_{i=1}^{q} c^{\text{reg}}(P'_i) \leq \sum_{i=1}^{q} c^{\text{reg}}(P_i) \leq 2 \cdot OPT_{LP}$ by Lemma 21. ◀

References

1 F. Afrati, S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the traveling repairman problem. *Informatique Theorique et Applications*, 20(1):79–87, 1986.

2 H. C. An, R. Kleinberg, and D. B. Shmoys. Improving Christofides algorithm for the s-t path TSP. *Journal of the ACM*, 62(5):34, 2015.

3 A. Asadpour, M. X. Goemans, A. Madry, S. Oveis Gharan, and A. Saberi. An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem. In *In Proceedings of SODA*, pages 379–389, 2010.

4 J. Bang-Jensen, A. Frank, and B. Jackson. Preserving and increasing local edge-connectivity in mixed graphs. *SIAM J. Discrete Math.*, 8(2):155–178, 1995.

5 A. Blum, P. Chalasani, D. Coppersmith, W. R. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *In Proceedings of STOC*, pages 163–171, 1994.

6 A. Blum, S. Chawla, D. R. Karger, T. Lane, and A. Meyerson. Approximation algorithms for orienteering and discount-reward TSP. *SIAM J. Comput.*, 37(2):653–670, 2007.

7 D. Chakrabarty and C. Swamy. Facility location with client latencies: linear-programming based techniques for minimum latency problems. *Math. of Operations Research*, 41(3):865–883, 2016.

8 M. Charikar, M. X. Goemans, and H. J. Karloff. On the integrality ratio for the asymmetric traveling salesman problem. *Math. of Operations Research*, 31(2):245–252, 2006.

9 K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *In Proceedings of FOCS*, pages 36–45, 2003.

10 M. Fischetti, G. Laporte, and S. Martello. The delivery man problem and cumulative matroids. *Operations Research*, 41:1065–1064, 1993.

11 Z. Friggstad, A. Gupta, and M. Singh. An improved integrality gap for asymmetric TSP paths. *Math. of Operations Research*, 41(3):745–757, 2016.

12 Z. Friggstad, M. R. Salavatipour, and Z. Svitkina. Asymmetric traveling salesman path and directed latency problems. *SIAM J. Comput*, 42(4):1596–1619, 2013.

13 Z. Friggstad and C. Swamy. Approximation algorithms for regret-bounded vehicle routing and applications to distance-constrained vehicle routing. In *In Proceedings of STOC*, pages 744–753, 2014.

14 Z. Friggstad and C. Swamy. Compact, provably-good LPs for orienteering and regret-bounded vehicle routing. In *In Proceedings of IPCO*, pages 199–211, 2017.

15 H. Gabow. Perfect arborescence packing in preflow mincut graphs. In *In Proceedings of SODA*, pages 528–538, 1996.

16 M. X. Goemans and D. P. Williamson. Approximating minimum-cost graph problems with spanning tree edges. *Operations Research Letters*, 16:183–189, 1994.

17 A. Köhne, V. Traub, , and J. Vygen. The asymmetric traveling salesman path LP has constant integrality ratio. In *In Proceedings of IPCO*, pages 288–298, 2019.

18 W. Mader. Konstruktion aller n-fach kantenzusammenhängenden Digraphen. *Europ. J. Combinatorics*, 3:63–67, 1982.

19 E. Minieka. The delivery man problem on a tree network. *Annals of Operations Res.*, 18:261–266, 1989.

20 V. Nagarajan and R. Ravi. The directed minimum latency problem. In *APPROX/RANDOM 2008. Proceedings*, Lecture Notes in Computer Science, pages 193–206. Springer, 2008.

21 J. Park and B. Kim. The school bus routing problem: A review. *European Journal of Operational Research*, 202(2):311–319, 2010.

22 I. Post and C. Swamy. Linear-programming based techniques for multi-vehicle minimum latency problems. In *Proceedings of SODA*, pages 512–531, 2015.

23 M. Skutella. List scheduling in order of α-points on a single machine. *Efficient Approximation and Online Algorithms*, 3484:250–291, 2006.

24 M. Spada, M. Bierlaire, and T. Liebling. Decision-aiding methodology for the school bus routing and scheduling problem. *Transportation Science*, 39(4):477–490, 2005.

25 O. Svensson, J. Tarnawski, and L. Vegh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. In *In Proceedings of STOC*, pages 204–213, 2018.

26 P. Toth and eds D. Vigo. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

27 V. Traub. *Approximation Algorithms for Traveling Salesman Problems*. PhD thesis, University of Bonn, 2020. URL: `http://hss.ulb.uni-bonn.de/2020/5834/5834.htm`.

28 V. Traub and J. Vygen. An improved approximation algorithm for ATSP. In *In Proceedings of STOC*, pages 1–13, 2020.

A Reduction to Instances with Polynomially-Bounded Integer Distances

▶ **Theorem 26.** *For any constant $\epsilon > 0$, if there is an $\alpha(n)$-approximation for instances of* DirLat *where each $c_{u,v}$ is a positive integer bounded by a polynomial in n and $1/\epsilon$ and where $c_{u,v} \geq 1$ for nodes $u \neq v$, then there is an $(\alpha(n) + \epsilon)$-approximation for general instances of* DirLat.

Proof. Compute a value ν such that $OPT \leq \nu \leq n^2 \cdot OPT$ where OPT is the optimum solution to the given DirLat instance. For example, ν could be the smallest value such that all nodes can be covered by a single walk in the graph $G_\nu = (V + r, E_\nu)$ consisting of directed edges $E_\nu = \{uv : c_{u,v} \leq \nu\}$. This can be checked, for example, by contracting the strongly-connected components of G_ν and checking if topologically sorting the resulting directed acyclic graph results in a single chain of components with the root in the first component.

Now, the case $OPT = 0$ can detected in polynomial time as this is equivalent to checking if the strongly-connected components of the graph using only distance-0 edges forms a chain. So we assume $OPT > 0$, thus $\nu > 0$. We then assume $c_{u,v} \geq \epsilon \cdot \nu/n^3$ by increasing any distance that is smaller to this amount: the distances remain metric and the latency of any node on the optimum solution increases by at most $n \cdot \nu \leq \epsilon \cdot OPT/n$, so the total latency increases by at most $\epsilon \cdot OPT$.

Next, we may assume all distances satisfy $c_{u,v} \leq (\alpha(n) + 2\epsilon) \cdot \nu$ for the following reason. Suppose we update each distance $c_{u,v} > (\alpha(n) + 2\epsilon) \cdot \nu$ with $c_{u,v} = (\alpha(n) + 2\epsilon) \cdot \nu$. It is easy to check these updated distances also form a metric. The optimum solution cost is still OPT because no edge used by the optimum solution has its length shortened (as $\nu \geq OPT$). Also, note a solution P with $c(P) \leq (\alpha(n) + \epsilon) \cdot OPT$ will only use edges uv where $c_{u,v} < (\alpha(n) + 2\epsilon) \cdot \nu$. So an $(\alpha + \epsilon)$-approximation in the metric with these truncated distances yields an $(\alpha + \epsilon)$-approximation for the original distances.

Next, for all $u, v \in V + r$ let $d''(u, v) = \left\lfloor c_{u,v} \cdot \frac{n^4}{\nu \cdot \epsilon} \right\rfloor$. Let d' be the shortest path metric using edge distances given by d''. Let OPT' denote the optimum solution to DirLat instance with distances d'. Observe

$$d'(u, v) \leq d''(u, v) \leq \frac{n^4}{\nu \cdot \epsilon} c_{u,v}.$$

Furthermore, $c_{u,v} \leq (\alpha(n) + 2\epsilon) \cdot \nu$ for each edge uv means $d'(u, v) \leq \frac{n^4}{\epsilon} \cdot (\alpha(n) + \epsilon)$. So all distances under d' are polynomially-bounded integers. We also see $OPT' \leq \frac{n^4}{\nu \cdot \epsilon} \cdot OPT$ by consider an optimum solution to the original instance, but under the new distances d'.

Now consider a solution P with $d'(P) \leq \alpha(n) \cdot OPT'$. As d' is a metric, we may assume P is a Hamiltonian path so P traverses n edges. By replacing each edge in P with its shortest path using distances d'', we obtain a walk W with $d''(W) = d'(P) \leq \alpha(n) \cdot OPT'$. For each edge uv, we have $d''(u,v) + 1 \geq c_{u,v} \cdot \frac{n^4}{\nu \cdot \epsilon}$. So the cost of W under d can be bounded as follows where sums over edges in W include as many terms of uv as its multiplicity in W.

$$\begin{aligned} c(W) &\leq \frac{\epsilon \cdot \nu}{n^4} \cdot \sum_{uv \in W} (d''(u,v) + 1) \\ &= \frac{\epsilon \cdot \nu}{n^4} \cdot (d''(W) + |W|) \\ &\leq \frac{\epsilon \cdot \nu}{n^4} \cdot (\alpha(n) \cdot OPT' + |W|) \\ &\leq \alpha(n) \cdot OPT + \frac{\epsilon \cdot \nu}{n^2} \\ &\leq (\alpha(n) + \epsilon) \cdot OPT. \end{aligned}$$

The last two bounds use $|W| \leq n \cdot |P| \leq n^2$ and $\nu \leq n^2 \cdot OPT$. ◀

On Compact RAC Drawings

Henry Förster
Wilhelm-Schickard-Institut für Informatik, University of Tübingen, Germany
foersth@informatik.uni-tuebingen.de

Michael Kaufmann
Wilhelm-Schickard-Institut für Informatik, University of Tübingen, Germany
mk@informatik.uni-tuebingen.de

Abstract

We present new bounds for the required area of Right Angle Crossing (RAC) drawings for complete graphs, i.e. drawings where any two crossing edges are perpendicular to each other. First, we improve upon results by Didimo et al. [15] and Di Giacomo et al. [12] by showing how to compute a RAC drawing with three bends per edge in cubic area. We also show that quadratic area can be achieved when allowing eight bends per edge in general or with three bends per edge for p-partite graphs. As a counterpart, we prove that in general quadratic area is not sufficient for RAC drawings with three bends per edge.

2012 ACM Subject Classification Mathematics of computing → Graphs and surfaces; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms; Human-centered computing → Graph drawings

Keywords and phrases RAC drawings, visualization of dense graphs, compact drawings

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.53

Acknowledgements We thank Patrizio Angelini for useful discussions and proofreading and the anonymous referees of an earlier version for helpful comments.

1 Introduction

Graphs that appear in real-world applications are in fact mostly nonplanar. Experiments on the human perception of graph drawings indicate that two important parameters affecting readability are angles formed by two edges at their crossing points (the larger the better) [20, 21] as well as the number of bends along an edge (the fewer the better) [24, 25]. The first theoretical drawing model that has taken these experimental results into account is the so-called *RAC* (or *right-angle-crossing*) drawing introduced in [15]. In some sense, the RAC model generalizes the popular orthogonal graph drawing model [17]. Formally, a RAC drawing is a node-link drawing of a graph, in which edges are drawn as polylines so that the angles formed at the crossing points of two edges are always equal to $\pi/2$. Since a RAC drawing is a geometric embedding, in most studies on RAC drawings the number of bends per edge has also been taken into account. In the following, we denote a RAC drawing with at most k bends per edge as a *RAC_k drawing*.

Many main research questions of graph drawing have been studied for RAC drawings. Regarding their density, already Didimo et al. [15] showed that graphs admitting straight-line RAC drawings have at most $4n-10$ edges, which is a tight bound. They also showed that the density for graphs admitting RAC_1 or RAC_2 drawings is subquadratic, whereas all graphs admit a RAC_3 drawing. Subsequently, Arikushi et al. [6] showed that graphs admitting a RAC_1 drawing can only have $6.5n-13$ edges, while graphs admitting RAC_2 drawings can have at most $74.2n$ edges; the former bound for RAC_1 drawings was recently improved by Angelini et al. [2] to $5.5n-11$. The recognition problem for graphs admitting straight-line RAC drawings is known to be NP-hard [5], even in the case where the resulting drawing must

Table 1 Overview of area bounds on RAC drawings of general graphs.

Known Results			**Our New Results**	
RAC_3:	$\mathcal{O}(n^4)$	[15]	RAC_3:	$\mathcal{O}(n^3)$
RAC_4:	$\mathcal{O}(n^3)$	[12]	RAC_3 (p-partite):	$\mathcal{O}(p^4n^2)$
$RAC_{\geq 3}$:	$\Omega(n^2)$	[15]	RAC_3:	$\omega(n^2)$
RAC_6:	$\mathcal{O}(n^{2.75})$	[26]	RAC_8:	$\mathcal{O}(n^2)$

be upward [3] or 1-planar [8]. Note that a k-planar graph is a graph which admits a drawing where every edge is crossed at most k times. While the recognition of graphs admitting RAC_3 drawings is trivial, the corresponding problem for graphs admitting RAC_1 or RAC_2 drawings is yet unsettled. Curiously, the maximally dense graphs admitting straight-line RAC drawings have been shown to be 1-planar [18]. In addition, subclasses of 1-planar graphs have been investigated: Brandenburg et al. [9] proved that all IC-planar graphs admit a straight-line RAC drawing, which has been shown to be not true for NIC-planar graphs [7]. IC-planar and NIC-planar graphs are graphs with a 1-planar drawing where the set of vertices involved in a crossing shares at most zero and one vertices with the set of vertices involved in a different crossing, respectively. Di Giacomo et al. [11] and Hong and Nagamochi [19] studied variants of RAC drawings with restricted vertex positioning in the straight-line setting. Angelini et al. [3] showed that all graphs of maximum degree three admit a RAC_1 drawing, whereas graphs of maximum degree six admit a RAC_2 drawing.

To evaluate the area of RAC drawings, vertices and bends are assumed to be located on an integer grid. The area of a drawing then is the product of the number of horizontal and vertical grid lines appearing in a bounding axis-aligned rectangle. It is known that even planar graphs may still require quadratic area in any straight-line RAC drawing [3]. Recently, Chaplick et al. [10] showed that NIC planar graphs admit RAC_1 drawings in polynomial area, whereas 1-planar graphs admit RAC_2 drawings in polynomial area. The drawing algorithm by Didimo et al. [15] achieves RAC_3 drawings in $\mathcal{O}(n^4)$ area. Subsequently, Di Giacomo et al. [12] improved the area to $\mathcal{O}(n^3)$ for RAC_4 drawings and Rahmati and Emami [26] recently achieved $\mathcal{O}(n^{2.75})$ area for RAC_6 drawings. For the closely related family of *LAC graphs* (short for *large-angle-crossing*), in which edges may cross at angles at least $\pi/2 - \varepsilon$ for some small $\varepsilon > 0$, Di Giacomo et al. [12] also showed that the complete graph on n vertices admits a drawing with one bend per edge in $\mathcal{O}(n^2(\cot \varepsilon/2)^2)$ area, which can be assumed to be $\mathcal{O}(n^2)$ area for fixed values of ε. Note however that for very small values of ε such as $\pi/180$, the multiplicative constant is very large and may therefore be infeasible in practise. For further results on LAC drawings see also [4, 16].

It is noteworthy that these drawing algorithms only place vertices and bends on an integer grid while crossings may occur on non-grid points. The positions of crossings are implicitly defined by the positions of endpoints of the intersecting segments. Since by the crossing lemma [1, 22] there are $\Omega(n^4)$ crossings in the complete graph K_n, it is impossible to achieve an area bound of $o(n^4)$ if also the crossings are required to be on the grid. In the variant where crossings are located on the grid, the algorithm by Didimo et al. [15] for RAC_3 drawings in $\mathcal{O}(n^4)$ area yields optimal solutions. We emphasize that it is not trivial to compute RAC drawings with $\mathcal{O}(n^2)$ area with additional bends as only $\mathcal{O}(1)$ bends per edge can fit in $\mathcal{O}(n^2)$ area. Finally, tradeoffs between area and planar thickness [14] as well as number of crossings per edge [13] also have been investigated.

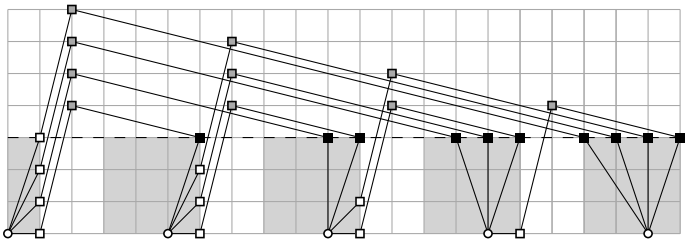


Figure 1 RAC_3 drawing of K_5 in 21×7 area.

We emphasize that we study *simple* graphs on n vertices, i.e., graphs without self-loops and parallel edges. The restriction to simple graphs is common in this line of research as each edge connecting the same vertices must be assigned distinct positions for its bends. In Section 2, we give new area upper bounds. We prove that every graph admits a RAC_3 drawing in $\mathcal{O}(n^3)$ area improving the known bound by a factor of n. Also, we show that even $\mathcal{O}(n^2)$ area can be achieved when eight bends per edge are allowed or when the input graph is p-partite. Then, in Section 3, we prove that quadratic area cannot be achieved in general for RAC_3 drawings. See also Table 1 for an overview of our new results compared to results from the literature. We conclude the paper with some open problems.

2 New Area Upper Bounds for RAC Drawings

▶ **Theorem 1.**

1. *Every n-vertex graph $G = (V, E)$ admits a RAC_3 drawing in $\mathcal{O}(n^3)$ area.*
2. *Every n-vertex graph $G = (V, E)$ admits a RAC_8 drawing in $\mathcal{O}(n^2)$ area.*
3. *Every p-partite n-vertex graph $G = (V, E)$ admits a RAC_3 drawing in $\mathcal{O}(p^4 n^2)$ area.*

Proof (of Result 1). Our algorithm is a refinement of the algorithm by Didimo et al. [15]. Note that it is easy to see that the drawings produced by the algorithm in [15] require $\Theta(n^4)$ area as two bends for each edge are located on a horizontal line.

For an example of a drawing of K_5 refer to Fig. 1. The vertices and the segments incident to vertices are drawn planar in a disjoint region of quadratic area for each vertex; see the gray regions in Fig. 1. In contrast to [15], each vertex (except for the outermost two) is incident to two types of bends (white and black squares in Fig. 1) which lead to vertices with larger and smaller indices, resp. The remaining two segments of edges use nearly horizontal or nearly vertical slopes.

We number the vertices arbitrarily from 0 to $n-1$. We place vertex v_i at position $(i \cdot n, 0)$; see white disks in Fig. 1. The three bends of edge (v_i, v_j) with $i < j$ are placed as follows: Bend $a_{i,j}$ connected to v_i is placed at $(i \cdot n + 1, j - i - 1)$; see white squares in Fig. 1. The middle bend $b_{i,j}$ is placed at $(i \cdot n + 2, n + j - i - 2)$; see gray squares in Fig. 1. Bend $c_{i,j}$ connected to v_j is placed at $(j \cdot n - j + i + 2, n - 2)$; see black squares in Fig. 1.

It remains to show that the resulting drawing is indeed RAC. Consider the start segments incident to vertex v_i, i.e., segments of types $(v_i, a_{i,j})$ for $(v_i, v_j) \in E$ and $(v_i, c_{j,i})$ for $(v_j, v_i) \in E$. Together with v_i they form a *fan* and do not intersect each other as the bend points of types $a_{i,j}$ and $c_{j,i}$ are distinct. Note that while $a_{0,n-1}$ is located at $(1, n-2)$, for other vertices v_i the bend $c_{i-1,i}$ is located at $(i \cdot n + 1, n - 2)$. These fans are drawn in disjoint *start regions* (gray shaded areas in Fig. 1); more precisely, the fan of v_i is located within a rectangle ranging from 0 to $n-2$ in y-direction and from $(i-1) \cdot n + 3$ to $i \cdot n + 1$ in x-direction. Since all segments $(b_{i,j}, c_{i,j})$ are located above the start regions and because

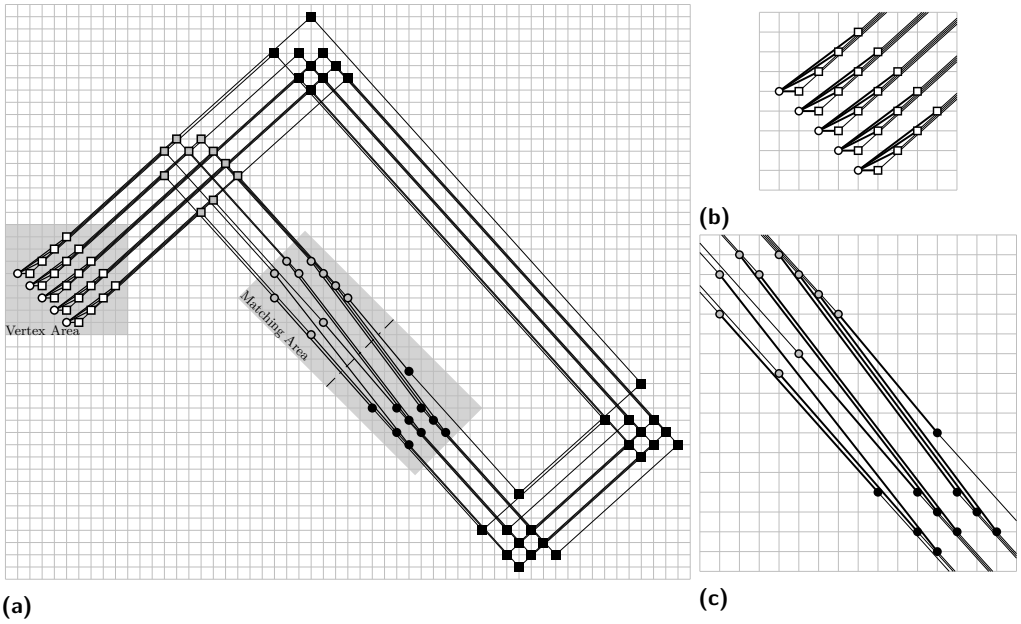


Figure 2 (a) RAC_8 drawing of K_5 in 55×47 area, (b)–(c) zoom into vertex and matching area.

segments $(a_{i,j}, b_{i,j})$ are located between x-coordinates $i \cdot n + 1$ and $i \cdot n + 2$ (i.e., they are located between two start regions), there are no crossings within start regions. As all crossings occur between segments of type $(a_{i,j}, b_{i,j})$ (which have slope $n - 1$) and $(b_{i,j}, c_{i,j})$ (which have slope $-1/(n-1)$), all proper crossings are at right angles.

It remains to prove that there are no overlaps. Recall that bend points in each start region are distinct, hence, we consider only the remaining segments. Segments of edges with a common endpoint cannot overlap. As the regions containing edges $(a_{i,j}, b_{i,j})$ and $(a_{k,l}, b_{k,l})$ for $i \neq k$ are disjoint, overlaps may only occur at segments $(b_{i,j}, c_{i,j})$ and $(b_{k,l}, c_{k,l})$ for some $i \neq k$. Since both segments have the same slope and their crossings with the horizontal at $y = n - 2$ (dashed in Fig. 1) are distinct (i.e., $c_{i,j}$ and $c_{k,l}$), they also do not overlap.

As the lowest x- and y-coordinates are both 0 whereas the largest x- and y-coordinates are $(n-1)n + 1$ and $2n - 3$, resp., the area bound follows.

Proof (of Result 2). We describe how to draw K_n for odd n in quadratic area. For even n, refer to the construction of K_{n+1}. For an example of a drawing of K_5 refer to Fig. 2. We number the vertices arbitrarily from 0 to $n - 1$. The general idea is as follows: vertices and start segments are located in the *vertex area* such that each start segment bend is connected to a segment whose slope is slightly less than 1; see Fig. 2b. Edges are treated as two half edges that are routed to the *matching area* independently with segments of slopes s and $-1/s$. One half edge is routed to the top left half of the matching area (gray bends in Fig. 2a) and the other half edge is routed to the bottom right half (black bends in Fig. 2a). In the matching area half edges are matched crossing-free realizing an edge for each pair of vertices; see Fig. 2c. We point out that in principle each half edge may be routed to either half of the matching area. When we define the matching of the bends in the matching area, we will show which half edges have to be routed to which half of the matching area.

We place vertex v_i at position $(i, -i)$; see white circles in Fig. 2b. Let e_i^j be the j-th half edge incident to vertex v_i for $0 \leq j < n - 2$. We place the bend of e_i^j that is closest to v_i at $(i + j + 1, j - i)$; see white squares in Fig. 2b. Bends incident to the same vertex v_i are

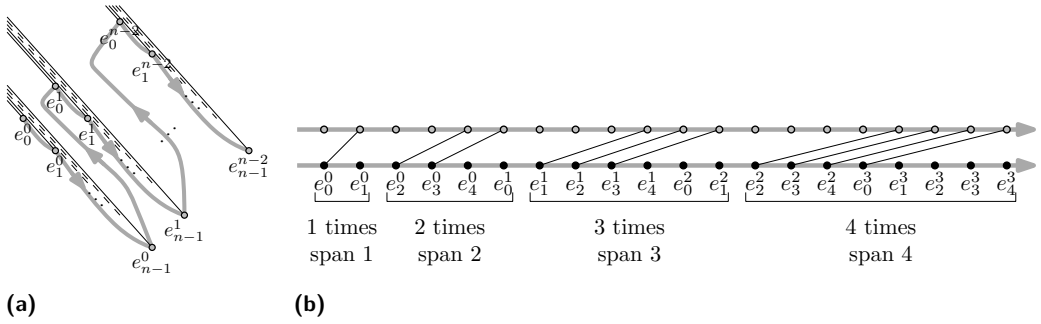


Figure 3 (a) Accessability of matching bends in the top left half of the matching area, and (b) matching assignment for $n = 5$ used for the drawing in Fig. 2.

located on a diagonal of slope 1 and all start segments of v_i are above this diagonal. Since v_i is located below the diagonal of vertex v_{i-1}, start segments do not intersect each other and all $n \cdot (n-1)$ start segment bends are within a rectangle of quadratic area tilted by $\pi/4$.

We choose $s = (2n-1)/2n$ achieving the following: If $n \cdot (n-1)$ bends are located in a rectangle R as defined by the start segment bends and a segment of slope s is added to each of these points, the next integer point used by any of those segments is located outside of R. This procedure "copies" the bends at $k \cdot 2n$ horizontal and $k \cdot (2n-1)$ vertical distance for $k \in \mathbb{Z}$. A similar property holds for segments of slope $-1/s$. Further, since s is slightly less than 1 and start segments are above the line of slope 1 through start segment bends, those additional segments of slope s do not intersect any start segment; see Fig. 2b. In the matching area all bends in the top left (bottom right, resp.) half are accessible from the bottom right (top left, resp.) without intersections; see Fig. 2c.

Next, we define the bend points of half edge e_i^j leading from vertex to matching area. Recall that the bend in the vertex area (white squares in Fig. 2a) is at $(i+j+1, j-i)$. If e_i^j is routed to the top left half of the matching area, we place one bend at $(2n+i+j+1, 2n+j-i-1)$ (gray squares in Fig. 2a) and enter the matching area with a bend at $(4n+i+j, j-i-1)$ (see gray circles in Fig. 2a). If e_i^j is routed to the bottom right half of the matching area, we instead create a sequence of three bends at $(4n+i+j+1, 4n+j-i-2)$, $(10n+i+j-2, -2n+j-i-2)$ and $(8n+i+j-2, -4n+j-i-1)$ (see black squares in Fig. 2a) and enter the matching area with a bend at $(6n+i+j-1, -2n+j-i-1)$ (see black circles in Fig. 2a). The leftmost x-coordinate is 0 (for vertex v_0), while the rightmost one is $12n-5$ (for a bend of e_{n-1}^{n-2}). Conversely, the topmost y-coordinate is $5n-4$ (for a bend of e_0^{n-2}) while the bottommost one is $-5n$ (for a bend of e_{n-1}^0). Hence, the drawing requires $(12n-5) \times (10n-3)$ area.

It remains to show that *matching segments* between matching bends are crossing-free. Consider the accessibility of matching bends from the other half of the matching area: Each matching bend in the top left half of the matching area is accessible from below the diagonal of slope -1 passing through it, since from above its incident segment of slope $-1/s$ is forming an obstacle. A similar statement is true for matching bends in the bottom right half of the matching area. We can use this observation to define an ordering of the matching bends based on their accessability. The first accessible matching bend is of half edge e_0^0; i.e., incident to v_0, the second is of e_1^0; i.e., incident to v_1. This pattern continues increasing in i until all e_i^0 are encountered; see Fig. 3a. Afterwards, all e_i^1 are encountered, in increasing order of i. This pattern repeats increasing in j, until all half edges e_i^j are encountered, in increasing order of i; see Fig. 3a.

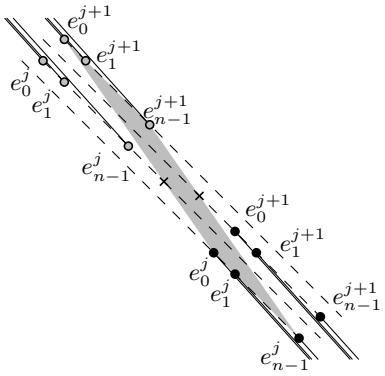


Figure 4 Area for drawing matching segments of half edges e_i^j and e_k^{j+1}.

We use the two linear orders of matching bends to define a planar matching between both halves of the matching area. Note that both linear orders are identical; see Fig. 3b. The matching that we define has two properties: First, the distance between matched bends in the linear order is bounded by $n-1$. Second, the matching ensures that every pair of vertices is connected exactly once. We now describe the specific matching: First, we connect the first matching bend of the bottom right half of the matching area with the second matching bend of the top left half of the matching area. Then, we connect the next two matching bends of the bottom right half of the matching area with the following two matching bends of the top left half of the matching area. We continue this pattern while always increasing the size of the groups of matched pairs by one which also increases the *span* of the matching segments, i.e., the distance between the connected matching bends in the linear order; see Fig. 3b. Then, there are exactly k matching segments of span k for all values $1 \leq k \leq n-1$. A matching segment of span k connects a vertex v_i whose bend is in the bottom right half of the matching area with vertex $v_{(i+k) \bmod n}$, i.e. with the neighbor whose index is k larger in the cyclic order of vertices. We prove that every pair of vertices is matched exactly once using that n is odd and that therefore the distance of vertices in the cyclic order is at most $(n-1)/2$. Due to cyclicity, segments with span $n-k > (n-1)/2$ correspond to a connection from vertex v_i in the top left half of the matching area to vertex $v_{(i-(n-k)) \bmod n} = v_{(i+k) \bmod n}$, i.e. again to the neighbor whose index is k larger in the cyclic order of vertices. Hence, for $k \leq (n-1)/2$, there are k segments of span k and $n-k$ segments of span $n-k$, which means that in total n vertices are matched to their neighbors whose indices are k larger. In order to see that all of them are distinct, we apply a recursive argument: Clearly, this is true for $k=1$. Assume that all matching segments of spans k and $n-k$ had different neighbors, then remove the matched bends that were matched with span k ($n-k$, resp.) from the left (right, resp.) end of the linear order. In total, we remove $2n$ vertices each from the ends of both linear orders, i.e., $2k$ from the left and $2(n-k)$ from the right, before finding the segments of spans $k+1$ and $n-(k+1)$. Thus, their endpoints differ.

Finally, we show that the matching segments are planar straight-line segments. First observe that since the span of segments is at most $n-1$, half edge e_i^j is matched with a half edge e_k^ℓ for $\ell \in \{j, j+1\}$. Further, notice that the diagonal on which the matching bends of the j-th half edges in the top left half of the matching area are located is halfway in between the corresponding diagonals for the matching bends of the j-th and $j+1$-th half edges in the bottom right half; see Fig. 4. We show that the intersection of the line through the bend of e_0^{j+1} in the top left half and the bend of e_0^j in the bottom right half of the matching area crosses the diagonal through the bends of e_0^j and e_{n-1}^j in the top left half of the matching

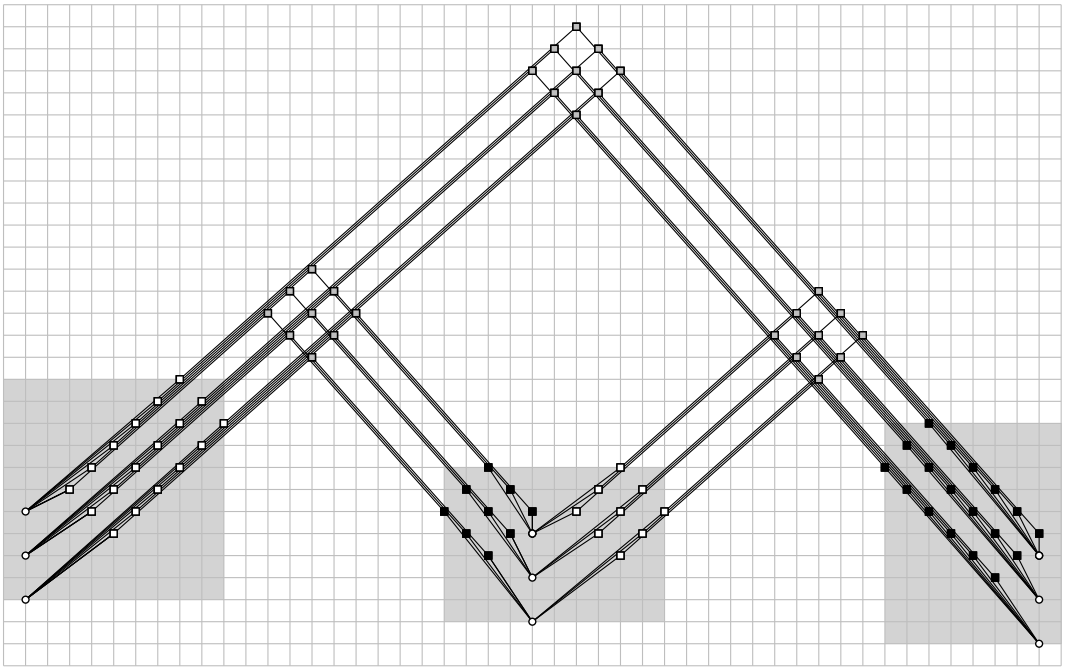


Figure 5 RAC_3 drawing of $K_{3,3,3}$ in 46×28 area.

area to the right of the bend of e^j_{n-1}; see crosses in Fig. 4. A symmetric property follows for bottom right half and consequently, segments between the j-th and $j+1$-th half edge are crossing-free. The first line goes through points $(4n+j+1, j)$ (bend of e^{j+1}_0 in the top left half of the matching area) and $(6n+j-1, -2n+j-1)$ (bend of e^j_0 in the bottom right half of the matching area) and hence has slope $-(2n+1)/(2n-2)$. The second line goes through point $(5n+j-1, -n+j)$ (bend of e^j_{n-1} in the top left half of the matching area) and has slope -1. We compute the two line equations based on the fact that we know a point on each line and the corresponding slopes:

$$y = -\frac{2n+1}{2n-2}x + \frac{8n^2+4nj+6n-j+1}{2n-2} \qquad \text{and} \qquad y = -x+4n+j+1$$

and the x-coordinate of the intersection point $x = 16/3n + j - 1/3$ that is to the right of the bend of e^j_{n-1} in the top left half of the matching area as claimed.

Proof (of Result 3). We describe how to draw $K_{(n_p)_p}$, the complete p-partite graph with n_p vertices per partition. Refer to Fig. 5 for an example drawing of $K_{3,3,3}$. If the partitions have different sizes, we augment the graph to $K_{(n_p)_p}$ where n_p is the number of vertices in the largest partition. Note that $n_p < n$ and $pn_p \geq n$. We number the partitions arbitrarily from 0 to $p-1$ and the vertices in each partition from 0 to $n_p - 1$. Let v^j_i denote the i-th vertex of the j-th partition.

We position vertex v^j_i at $(2pn_pj + 2n_pj - j, 2i - j)$; see white circles in Fig. 5. Edge (v^j_i, v^ℓ_k) with $j < \ell$ is drawn with the following three bends:

- The bend incident to vertex v^j_i is located at $(2pn_pj + n_p\ell + n_pj + k - i - j + 1, n_p\ell - n_pj + i + k - j)$; see white squares in Fig. 5.
- The middle bend is located at $(pn_p\ell + pn_pj + n_p\ell + n_pj + k - i - j + 1, pn_p\ell - pn_pj + n_p\ell - n_pj + i + k - \ell)$; see gray squares in Fig. 5.
- The bend incident to vertex v^ℓ_k is located at $(2pn_p\ell + n_p\ell + n_pj + k - i - \ell + 1, n_p\ell - n_pj + i + k - \ell)$; see black squares in Fig. 5.

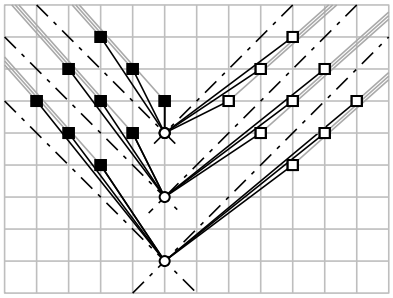


Figure 6 Detail of the vertex stars of the middle partition in Fig. 5.

The lowest x-coordinate assigned is 0 (for vertices in partition 0), while the highest x-coordinate assigned is $2p^2n_p - 2n_p - p + 1$ (for vertices in partition $p-1$). Conversely, the lowest y-coordinate is $-p+1$ (for vertex v_0^{p-1}) whereas the largest y-coordinate is $p^2n_p + n_p - p - 1$ (for the middle bend of edge $(v^0_{n_p-1}, v^{p-1}_{n_p-1})$). Since $pn_p \leq pn$, it follows, that the total area is $\mathcal{O}(p^4n^2)$.

It remains to discuss that the resulting drawing is RAC. First we have a look at the segments incident to the middle bend (i.e., the segments which are not start segments). All of these segments have slopes $(n-1)/n$ (between white and gray squares in Fig. 5) or $-n/(n-1)$ (between black and gray squares in Fig. 5). Hence, each pair of these segments is either parallel or perpendicular.

Next, we show that start segments of different partitions do not intersect; see gray shaded areas in Fig. 5. To see this, we consider the rightmost bend incident to a vertex of partition j and the leftmost incident to a vertex of partition ℓ such that $j < \ell$. The rightmost bend of partition j belongs to edge $(v_0^j, v^{p-1}_{n_p-1})$ and has x-coordinate $2pn_pj + pn_p + n_pj - j$ whereas the leftmost bend of partition ℓ belongs to edge $(v^0_{n_p-1}, v_0^\ell)$ and has x-coordinate $2pn_p\ell + n_p\ell - n_p - \ell + 2$, i.e., at least $2pn_pj + 2pn_p + n_pj - j + 1$ since $\ell \geq j+1$. Hence, the leftmost bend of partition ℓ is at least $pn_p + 1 \geq n+1$ units right of the rightmost bend of partition j. Hence, start segments from different partitions cannot intersect.

In addition, we establish that middle bends are located outside of vertex fans. To see this, consider the topmost bend of a vertex fan of partition j. This is either $y_r = pn_p + n_p - n_pj - j - 2$ (bend of edge $(v^j_{n_p-1}, v^{p-1}_{n_p-1})$) or $y_\ell = n_pj + 2n_p - j - 2$ (bend of edge $(v^0_{n_p-1}, v^j_{n_p-1})$). Since middle bends appear in groups of n_p^2 bends, we only consider the lowest of these bends which is always incident to two vertices v_0^a and v_0^b for some partitions a and b such that $b > a$. The y-coordinate of this bend is equal to $y_m = pn_p(b-a) + n_p(b-a) - b$. Clearly, $y_m \leq y_r$ is only possible if $(b-a) = 1$ and $b \geq j+2$ in which case the middle bends are between two partitions whose start segments are to the right of the start segments of partition j (and hence the middle bends are to the right as well). Also since $(b-a) \geq 1$, it holds that $y_m \geq pn_p + n_p - b$ and, since $b \leq p-1$, also $y_m \geq pn_p + n_p - p + 1$. However, even for $j = p-1$ it holds that $y_\ell = pn_p + n_p - p - 1$ and hence $y_\ell < y_m$.

Next, we show planarity for the start segments. We first observe that each vertex v_i^j is incident to two sets of bends, namely,

- those that belong to an edge (v_k^ℓ, v_i^j) for some $\ell < j$. These are located on a diagonal with slope -1 left of the vertex; see black squares in Fig. 5. We will refer to these as the *left bends* of the vertex fan.
- those that belong to an edge (v_i^j, v_k^ℓ) for some $\ell > j$. These are located on a diagonal with slope 1 right of the vertex; see white squares in Fig. 5. We will refer to these as the *right bends* of the vertex fan.

We will show that v_i^j is located on the intersection of the diagonal with slope -1 located one unit below the diagonal through its left bends and the diagonal with slope 1 located one unit above the diagonal through its right bends; see dashed diagonals in Fig. 6. Hence, the start segments do not intersect segments between its start segment bends and the corresponding middle bends. Also, since the latter "middle" segments have slope $(n-1)/n$ or $-n/(n-1)$ the start segments of vertex v_i^j do not intersect middle segments of vertices v_{i-1}^j and v_{i+1}^j (as their middle segments only intersect the diagonal defining the position of v_i^j after moving n or $n-1$ units to the left/right).

The diagonal located one unit below the left vertex fan bends passes through the point $p_\ell = (2pn_pj + n_pj + i - j + 1, n_pj + i - j - 1)$ (one unit below the bend of edge (v_0^0, v_i^j). It is easy to verify that $(2pn_pj + 2n_pj - j, 2i - j) = (1, -1) \cdot (n_pj - i - 1) + p_\ell$. Similarly, the diagonal located one unit above the right vertex fan bends passes through the point $p_r = (2pn_pj + 2n_pj + n_p - i - j + 1, n_p + i - j + 1)$ (one unit above the bend of edge (v_i^j, v_0^{j+1}). It holds that $(2pn_pj + 2n_pj - j, 2i - j) = (-1, -1) \cdot (n_p - i + 1) + p_r$ as required.

Next, we show that the start segments of partition j do not intersect the middle segments incident to a start segment bend of partition $\ell \neq j$. To do so, we show that the middle segments of partition ℓ are located above all start segments. Recall that the topmost y-coordinate of a right bend in j occurs on edge $(v_{n_p-1}^j, v_{n_p-1}^p)$ and is equal to $pn_p - n_pj + 2n_p - j - 2$, and that the topmost y-coordinate of a left bend in j occurs on edge $(v_{n_p-1}^0, v_{n_p-1}^j)$ and is equal to $n_pj + 2n_p - j - 2$. Thus, the topmost y-coordinate of any start segment bend is at most $pn_p + 2n_p - j - 2$. We consider two cases.

First, consider the case $\ell > j$. Further assume that $\ell = j + 1$ since the middle segments of partition $\ell' > \ell$ are located above those of partition ℓ. As established earlier, the horizontal distance between the start regions of partitions j and ℓ is at least $pn_p + 1$. Observe that at the leftmost x-coordinate of the start region of partition ℓ, we encounter the left bend of $(v_{n_p-1}^0), v_0^\ell)$ with y-coordinate $n_p\ell + n_p - \ell - 1 = n_pj + 2n_p - j - 2$. On the other hand, if we continue k units in x-direction, we have distance $pn_p + 1$ towards partition j and encounter a left bend with y-coordinate $n_pj + 2n_p - j - 2 - k$. Because the slope of middle segments incident to left bends is $-pn_p/(pn_p - 1)$, any such middle segment has y-coordinate at least $(pn_p+1+k)\cdot\frac{pn_p}{pn_p-1} + n_pj + 2n_p - j - 2 - k > pn_p + k + n_pj + 2n_p - j - 2 - k = pn_p + n_pj + 2n_p - j - 2$ which is larger than $pn_p + 2n_p - j - 2$. Hence, such a middle segment is above each start segment of region j.

Second, assume that $\ell < j$. Here, we can assume that $\ell = j - 1$ by a similar argument as before. Again, the horizontal distance between the start segment bends of partitions j and ℓ is at least $pn_p + 1$. At the rightmost x-coordinate that belongs to the start region of partition ℓ is the right bend of edge $(v_0^\ell, v_{n_p-1}^{p-1})$ with y-coordinate $pn_p - n_p\ell + n_p - \ell - 2 = pn_p - n_pj + 2n_p - j - 1$. If we continue k units in negative x-direction, we have a minimum distance of $pn_p + 1 + k$ towards partition j and encounter a left bend with y-coordinate $pn_p - n_pj + 2n_p - j - 1 - k$. Recall that the slope of middle segments incident to such bends is $(pn_p - 1)/pn_p$. Note that such middle segment has y-coordinate at least $(pn_p+1+k)\cdot\frac{pn_p-1}{pn_p} + pn_p - n_pj + 2n_p - j - 1 - k = 2pn_p - n_pj + 2n_p - j - 2 + -k/pn_p$ when above a start segment of region j. Since $k < jn_p$, we have that $2pn_p - n_pj + 2n_p - j - 2 + -k/pn_p > 2pn_p - n_pj + 2n_p - j - 2 + -j/p$. In addition, $j \leq (p-1)$ and we conclude that $2pn_p - n_pj + 2n_p - j - 2 + -j/p > pn_p + 3n_p - j - 2 - (p-1)(p)$ which is larger than $pn_p + 2n_p - j - 2$.

We conclude that middle segments of other partitions do not enter the start regions of other partitions. Finally, we only have to show that no two bends overlap and that no bend is located on an independent segment. First consider the start segment bends. Since middle bends are not inside vertex start regions, the only segments to consider here are the

middle segments incident to the same partition. These clearly do not overlap any of the start segment bends since the next grid points are located at least $n - 1 = pn_p - 1$ to the left/right while there are only at most $(p-1)n_p = pn_p - n_p$ bends each using consecutive x-coordinates.

Since the middle segments only shift the gridlike structure of start segment bends of the same partition, it follows, that no two bends of the same partition can overlap and that no bend is located on an independent segment from the same partition. Middle segments of two different partitions cannot overlap as well, as that would imply that one of the middle edges would pass through a start segment bend of the other partition. This is not possible since start segments of different partitions are at least $n + 1$ units horizontally apart from each other while their (consecutively pairwise) vertical distance is one. ◀

Since k-planar graphs are $\Theta(\sqrt{k})$-vertex colorable [23], we also obtain:

▶ **Corollary 2.** *Every n-vertex k-planar graph admits a (not necessarily k-planar) RAC$_3$ drawing in $\mathcal{O}(k^2n^2)$ area.*

3 An Area Lower Bound for RAC$_3$ Drawings

We show that $\mathcal{O}(n^2)$ area cannot be achieved for RAC$_3$ drawings in general. We give an outline of our proof by contradiction: First, we show there are $\Omega(n^2)$ edges that have $\Omega(n^4)$ crossings on two sets S_i and T_i of parallel segments of maximum cardinality, where segments in S_i are perpendicular to segments in T_i and may intersect. Moreover, there must exist $\Omega(n^2)$ edges with both a segment in S_i and in T_i. Then, we derive properties on the length of the segments in S_i and T_i depending on their slope. This allows us to subdivide the drawing area into a constant number of disjoint regions $\mathcal{R}$, which can contain only one endpoint of a segment from S_i or from T_i of the same edge. We then restrict the possible positions of vertices incident to such endpoints located in a region $R \in \mathcal{R}$. As a result, in Lemma 13, we obtain that the edges with both a segment from S_i and a segment from T_i induce a subgraph which is p-partite for some $p > 1$ except for a linear number of so-called complete edges. Based on this observation, in the proof of Theorem 14, we identify a complete subgraph which has too few edges with both a segment from S_i and from T_i leading to a contradiction.

▶ **Lemma 3.** *Let Γ be a RAC drawing of K_n with $\mathcal{O}(1)$ bends per edge. Then there exist two sets of parallel edge segments S_i and T_i with cardinalities $|S_i| = \Omega(n^2)$ and $|T_i| = \Omega(n^2)$ in Γ such that the segments of S_i are perpendicular to the segments of T_i.*

Proof. We use the following two properties: First, by the crossing lemma, there are $\Omega(n^4)$ crossings in any drawing of K_n. Second, all crossings appear between perpendicular edge segments. We partition the set of segments of the drawing based on their slopes. More precisely, for some $k \in \mathbb{N}$, there are $2(k+1)$ sets of edge segments $S_0, \ldots, S_k$ and $T_0, \ldots, T_k$ such that S_i and T_i are perpendicular to each other. W.l.o.g. also assume that $|S_i| \geq |T_i|$ and that $|T_i| \geq |T_{i+1}|$. Since each edge has $\mathcal{O}(1)$ bends, there are at most cn^2 segments assigned to either set S_i for a constant c. Then, $|S_0| + \sum_{i=1}^{k} |S_i| = cn^2$ or in other words $|S_0| = cn^2 - \sum_{i=1}^{k} |S_i|$. Hence, we obtain the following relation for the number of crossings:

$$\Omega(n^4) \leq cr(\Gamma) \leq |S_0||T_0| + \sum_{i=1}^{k} |S_i||T_i| = \left(cn^2 - \sum_{i=1}^{k} |S_i|\right)|T_0| + \sum_{i=1}^{k} |S_i||T_i|$$

$$= cn^2|T_0| - \sum_{i=1}^{k} (|T_0| - |T_i|)|S_i| \leq cn^2|T_0|$$

which implies that $|S_0| = \Omega(n^2)$ and $|T_0| = \Omega(n^2)$. ◀

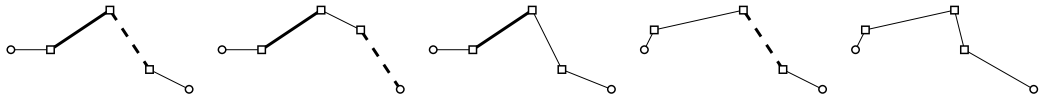


Figure 7 (a)–(b) Two edges belonging to E_i^{ST}, (c) an edge belonging to E_i^S, (d) an edge belonging to E_i^T, and, (e) an edge belonging to none of E_i^S, E_i^T and E_i^{ST}. Segments belonging to S_i are drawn bold and solid, segments belonging to T_i bold and dashed.

We show another property of edge sets contributing $\Omega(n^4)$ crossings. To this end, we consider all maximal sets of parallel edge segments that are involved in $\Omega(n^4)$ crossings and we partition these sets into two families $\mathcal{S} = \{S_1, \ldots, S_k\}$ and $\mathcal{T} = \{T_1, \ldots, T_k\}$ such that the segments in S_i and T_i are perpendicular while the segments in S_i and $S_j \cup T_j$ for $j \neq i$ are not. Observe that in contrast to the proof of Lemma 3, we now only consider segment sets S_i and T_i involved in $\Omega(n^4)$ crossings. Note that in a drawing with $\mathcal{O}(1)$ bends per edge, k is constant. In the following, we will discuss properties of pairs of sets $S_i \in \mathcal{S}$ and $T_i \in \mathcal{T}$. Let E_i^S (E_i^T, resp.) denote the set of edges with segments from S_i (T_i, resp.) but not from T_i (S_i, resp.), and E_i^{ST} the set of edges with segments from both S_i and T_i; see Fig. 7 for an illustration. In addition, let $E_{i,j}^{SX}$ denote the set of edges with segments from both S_i and from X_j where $X \in S, T$ and let $E_{i,j}^{TX}$ from both T_i and from X_j where $X \in S, T$. Note that $E_{i,i}^{ST} = E_i^{ST}$. The next lemmas show that there are S_i and T_i with $|E_i^{ST}| = \Omega(n^2)$.

▶ **Lemma 4.** *Let Γ be a RAC drawing of K_n with $\mathcal{O}(1)$ bends per edge. Then, there exists either sets $S_i \in \mathcal{S}$, $X_j \in \mathcal{S} \cup \mathcal{T}$ such that $|E_{i,j}^{SX}| = \Omega(n^2)$, or sets $T_i \in \mathcal{T}$, $X_j \in \mathcal{S} \cup \mathcal{T}$ such that $|E_{i,j}^{TX}| = \Omega(n^2)$.*

Proof. First, if $|E_{i,j}^{SX}| = \Omega(n^2)$ for some $S_i \in \mathcal{S}$ and $X_j \in \mathcal{S} \cup \mathcal{T}$ or if $|E_{i,j}^{TX}| = \Omega(n^2)$ for some $T_i \in \mathcal{T}$ and $X_j \in \mathcal{S} \cup \mathcal{T}$ with $i \neq j$; the lemma holds. Otherwise $|E_{i,j}^{SX}| = o(n^2)$ for all $S_i \in \mathcal{S}$ and $X_j \in \mathcal{S} \cup \mathcal{T}$ and $|E_{i,j}^{TX}| = o(n^2)$ for all $T_i \in \mathcal{T}$ and $X_j \in \mathcal{S} \cup \mathcal{T}$ with $i \neq j$. For a contradiction, also assume that $|E_i^{ST}| = o(n^2)$ for all $1 \leq i \leq k$. Hence, E_i^{ST} participates in $o(n^4)$ crossings. Also, assume w.l.o.g. that $|\bigcup_{i=1}^k E_i^S| \geq |\bigcup_{i=1}^k E_i^T|$. Consider the graph $G' = K_n \setminus \bigcup_{i=1}^k E_i^T$. Since G' contains $\bigcup_{i=1}^k E_i^S$, G' still has $\Omega(n^2)$ edges by Lemma 3. In Γ, there exists a valid subdrawing Γ' of G'. In Γ', by the crossing lemma, there still must be $\Omega(n^4)$ crossings between E_i^S and E_i^{ST} over all i. However, there are $o(n^4)$ crossings in Γ' from E_i^{ST} for $1 \leq i \leq k$; a contradiction for constant k. ◀

▶ **Lemma 5.** *Let Γ be a RAC$_3$ drawing of K_n. Then, $|E_{i,j}^{SX}| = o(n^2)$ for each pair of sets $S_i \in \mathcal{S}$, $X_j \in \mathcal{S} \cup \mathcal{T}$ with $i \neq j$ and $|E_{i,j}^{TX}| = o(n^2)$ for each pair of sets $T_i \in \mathcal{T}$, $X_j \in \mathcal{S} \cup \mathcal{T}$ with $i \neq j$.*

Proof. Assume w.l.o.g. that $|E_{i,j}^{ST}| = \Omega(n^2)$. Since only two start segments per vertex can belong to S_i and T_j, respectively, there are $\Omega(n^2)$ edges in $E_{i,j}^{ST}$, where the segments from S_i and T_j are not start segments. Let $\tilde{E}_{i,j}^{ST}$ denote this set of edges. Consider the start segments of $\tilde{E}_{i,j}^{ST}$ and let $\mathcal{P}_{start} = \{P_1, \ldots, P_r\}$ be a partitioning of the start segments into maximal sets of parallel segments. Since each vertex can be incident to only two start segments of the same slope, it follows that $|P_\ell| = \mathcal{O}(n)$ for all $1 \leq \ell \leq r$. Hence, there are only $\mathcal{O}(n^2)$ intersections between a perpendicular pair of start segments in $\tilde{E}_{i,j}^{ST}$. Similarly, if P_ℓ is perpendicular to S_i or T_j, it takes part in only $\mathcal{O}(n^3)$ intersections.

Consider the subgraph G' induced by $\tilde{E}_{i,j}^{ST}$. Note that G' has $\Omega(n^2)$ edges and hence by the crossing lemma it must have $\Omega(n^4)$ crossings. However, as established earlier, the subdrawing in Γ of G' only has $\mathcal{O}(n^3)$ intersections; a contradiction. ◀

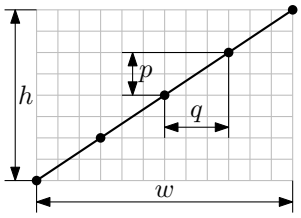


Figure 8 A fine-horizontal grid line (bold) with slope p/q, and its shared points with the coarse grid (gray lines).

The following lemma summarizes Lemmas 4 and 5.

▶ **Lemma 6.** *Let Γ be a RAC$_3$ drawing of K_n. Then, there exists a pair of sets $S_i \in \mathcal{S}$, $T_i \in \mathcal{T}$ such that $|E_i^{ST}| = \Omega(n^2)$, for some $1 \leq i \leq k$; i.e., $|E_i^{ST}| \geq c_{ST} n^2$ for an appropriate constant c_{ST} and sufficiently large n.*

Next, we investigate one pair of perpendicular segment sets $S_i \in \mathcal{S}$ and $T_i \in \mathcal{T}$. In the following analysis and all illustrations, we assume w.l.o.g. that the slope of segments in S_i is positive. First, we show that segments in S_i and T_i follow the grid lines of a finer grid that is tilted w.r.t. the coarse integer grid containing vertices and bends; see Fig. 8. We use this to show that segments in S_i and T_i are long w.r.t. the smaller side of the bounding rectangle.

▶ **Lemma 7.** *Let Γ be a RAC drawing of K_n with height h and width w and with $\mathcal{O}(1)$ bends per edge. Also, let $s = p/q$ be the slope of segments in $S_i \in \mathcal{S}$ for coprime integers p and q. Then,*

1. $\max\{p, q\} \in \Omega\left(\sqrt{n^4/(w \cdot h)}\right)$ *or* $pq \in \Omega\left(n^4/\max\{w^2, h^2\}\right)$*; and*
2. $p, q \in \mathcal{O}(\min\{w, h\})$.

Proof. Since the endpoints of each segment are grid points, the slope s_i of segments in S_i and the slope $-1/s_i$ of segments in T_i are rational numbers. Hence, the intersections between S_i and T_i are located at points with rational coordinates. By scaling the grid appropriately (i.e., by the factor of $p^2 + q^2$), we achieve integer coordinates for the intersections. In other words, all intersections are located on a *fine grid* while vertices and bends are on the integer grid which we call the *coarse grid*.

More precisely, the fine grid is defined by the *fine-horizontal* grid lines of slope s_i and by the *fine-vertical* grid lines of slope $-1/s_i$ each passing through at least two of the $h \cdot w$ vertices of the coarse grid. Note that by definition all vertices of the coarse grid are also vertices of the fine grid. Depending on the values of p and q, we observe that fine grid lines may pass through more than two points of the coarse grid; see Fig. 8. This limits how many fine grid lines exist. To see this, consider two consecutive fine-horizontal grid lines ℓ_1 and ℓ_2. Both lines ℓ_i (for $i \in \{1, 2\}$) can be expressed by a line formula of form $y = p/q \cdot x + b_i$. Since each line passes through integer points it holds that $b_i = 1/q \cdot c_i$ for some $c_i \in \mathbb{Z}$. More precisely, since ℓ_1 and ℓ_2 are consecutive, $|c_2 - c_1| = 1$ and the vertical distance between two consecutive fine-horizontal grid lines is $1/q$. In addition, we can compute the horizontal distance at the same y-coordinate by setting $p/q \cdot x_1 + b_1 = p/q \cdot x_2 + b_2$. Solving this equation yields $|x_2 - x_1| = q/p \cdot |b_2 - b_1| = 1/p \cdot |c_2 - c_1| = 1/p$ implying that the horizontal distance between two consecutive fine-horizontal grid lines is $1/p$. Analogously, the horizontal (vertical, resp.) distance between two fine-vertical grid lines is $1/q$ ($1/p$, resp.). Thus, there are at most $\Theta(\max\{wp, hq\})$ fine-horizonal and $\Theta(\max\{wq, hp\})$ fine-vertical grid lines.

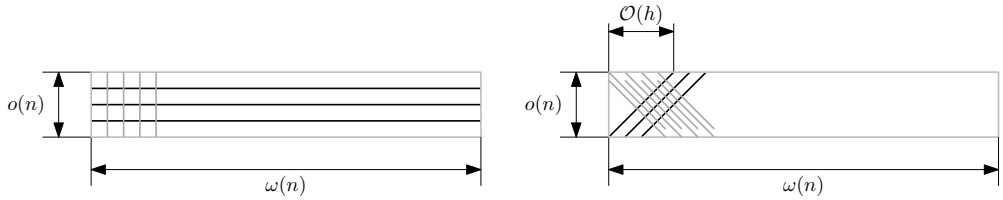


Figure 9 Proof of Lemma 8. If the area is $\omega(n) \times o(n)$, (a) there are $o(n)$ fine-horizontal grid lines (black), or, (b) fine-horizontal grid lines intersect $\mathcal{O}(h^2)$ fine-vertical grid lines (gray) each.

These two sets of grid lines intersect in $\Theta(\max\{w^2pq, whp^2, whq^2, h^2pq\})$ grid points, which must be $\Omega(n^4)$, the required number of crossings. Thus, $\max\{p,q\} \in \Omega\left(\sqrt{n^4/(w \cdot h)}\right)$ or $pq \in \Omega\left(n^4/\max\{w^2, h^2\}\right)$ which yields Property 1 of the lemma. Since the endpoints of all segments are located on the coarse grid, both $h, w \geq \max\{p,q\}$, which implies Property 2. ◀

The following lemma refines Lemma 7 for $\mathcal{O}(n^2)$ area and shows that both width and height are $\mathcal{O}(n)$ while segments in S_i and T_i have $\Omega(n)$ length.

▶ **Lemma 8.** *Let Γ be a RAC drawing of K_n with height h and width w and with $\mathcal{O}(1)$ bends per edge in $\mathcal{O}(n^2)$ area. Also, let p/q be the slope of segments in S_i such that p and q are coprime. Then,*

1. $h, w \in \Theta(n)$, *and,*
2. $\max\{p,q\} \in \Theta(n)$.

Proof. Assume for a contradiction that $h = o(n)$, i.e., $w = \omega(n)$. By Lemma 7,

1. $\max\{p,q\} \in \Omega\left(\sqrt{n^4/(w \cdot h)}\right) = \Omega(n)$ or $pq \in \Omega(n^4/\max\{w^2, h^2\}) = \Omega(h^2)$; and
2. $p, q \in \mathcal{O}(\min\{w, h\}) = \mathcal{O}(h)$

hold. By Property 2, it can only be $pq \in \Omega(h^2)$ but not $\max\{p,q\} \in \Omega(n)$. Consider the fine grid as defined in the proof of Lemma 7. First, if the fine-horizontal grid lines are in fact horizontal (i.e., $p = 0$), there can only be $\mathcal{O}(h)$ fine-horizontal grid lines since the height of the drawing is $\mathcal{O}(h)$; see Fig. 9a. Otherwise, the slope of the fine-horizontal grid lines is not horizontal. Recall that $p, q \in \mathcal{O}(h)$ since the height of the drawing is $\mathcal{O}(h)$. Since $pq = \Omega(h^2)$, $p, q \in \Theta(h)$. Hence, fine-horizontal grid lines have only length $\mathcal{O}(h)$ inside the bounding box and can only be crossed by $\mathcal{O}(h^2)$ fine-vertical grid-lines each; see Fig. 9b. Since $h = o(n)$, it is impossible to achieve $\Omega(n^4)$ crossings as in total there are only $\Theta(n^2)$ fine-horizontal grid lines. Thus, the assumption $h = o(n)$ leads to a contradiction. It follows that $h = \Theta(w)$, and hence, $w \in \Theta(n)$ and $\max\{p,q\} \in \Theta(n)$. ◀

So far, we considered properties of RAC drawings with $\mathcal{O}(1)$ bends per edge. The remaining results in this section hold specifically for RAC_3 drawings. Next, we explore connections realizable with edges in E_i^{ST} for a pair of perpendicular sets of segments $S_i \in \mathcal{S}$ and $T_i \in \mathcal{T}$. Based on slope p_i/q_i of segments in S_i for coprime integers p_i and q_i, consider a checkerboard partitioning of the drawing area into a set of square-shaped disjoint regions $\mathcal{R}_i$ of side length $\max\{p_i, q_i\}/2$ each. By Lemma 8, $\max\{p_i, q_i\} \in \Theta(n)$ and $h, w \in \Theta(n)$; and hence $|\mathcal{R}_i| = \mathcal{O}(1)$. By the choice of the slope, the length of segments in S_i have to be multiples of $\sqrt{p_i^2 + q_i^2}$. In particular, each segment in S_i has length larger than $\max\{p_i, q_i\}$. Due to the length of segments in S_i and the size of regions, we observe the following:

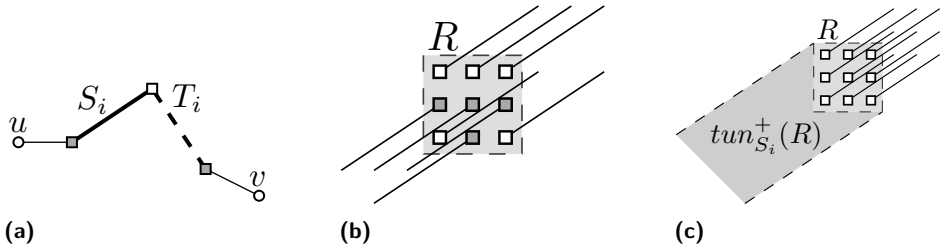


Figure 10 (a) An edge (u, v) whose middle segments (bold) belong to perpendicular set of segments $S_i \in \mathcal{S}$ and $T_i \in \mathcal{T}$. The gray bend incident to u (v, resp.) is an $S_i(T_i$, resp.)-endpoint. (b) A region R with set of bends $ep^+_{S_i}(R)$ (white squares) and set of bends $ep^-_{S_i}(R)$ (gray squares). (c) A region R with set of bends $ep^+_{S_i}(R)$ (white squares), their corresponding S_i-segments, and $tun^+_{S_i}(R)$.

▶ **Observation 9.** *At most one endpoint of a segment in S_i or T_i is in region $R \in \mathcal{R}_i$. All segments of S_i or T_i with an endpoint in R cross the boundary of R.*

As each vertex can only be endpoint of two segments in S_i and of two segments in T_i, there are only $\mathcal{O}(n)$ *start segments* in S_i and T_i , i.e., segments directly incident to a vertex. Hence, we only consider the bends of edges with both a middle segment in S_i and a middle segment in T_i where *middle segments* are segments which are not start segments. Refer to Fig. 10a for an illustration of such an edge. We refer to bends that are endpoints of a middle segment in S_i and of a start segment as *S_i-endpoints*. Analogously, we define *T_i-endpoints*.

▶ **Observation 10.** *Let $e \in E$ be an edge with two middle segments from S_i and T_i. The corresponding S_i- and T_i-endpoints are located in two disjoint regions of $\mathcal{R}_i$.*

Based on Observation 10, consider S_i- and T_i-endpoints in a region $R \in \mathcal{R}_i$ independently. Let $ep_{S_i}(R)$ ($ep_{T_i}(R)$) denote the set of S_i-endpoints (T_i-endpoints, resp.) in R. Further, $ep_{S_i}(R)$ can be subdivided into $ep^+_{S_i}(R)$, i.e. the set of S_i-endpoints that are the bottom endpoints of their corresponding S_i-segment, and $ep^-_{S_i}(R)$, i.e. the set of S_i-endpoints that are the corresponding top endpoints; see Fig. 10b. In other words, the S_i-segment incident to an endpoint in $ep^+_{S_i}(R)$ leaves R in positive y direction. Similarly, we subdivide $ep_{T_i}(R)$ into $ep^+_{T_i}(R)$ and $ep^-_{T_i}(R)$.

The segments in S_i and T_i form obstacles for possible connections of S_i- and T_i-endpoints to vertices. As a result, we will identify regions which have a visibility to many of the vertices connected to one of the sets of endpoints of region R, say $ep^+_{S_i}(R)$, to which we refer as *tunnels*. The *S_i-tunnel* $tun_{S_i}(R)$ of R is the region bounded by two lines parallel to the segments in S_i enclosing R. Further, $tun_{S_i}(R)$ is split by R into *S_i^+-tunnel* $tun^+_{S_i}(R)$ below R (see Fig. 10c) and the *S_i^--tunnel* $tun^-_{S_i}(R)$ above R. Similarly, we define T_i-tunnels for R.

Next, we define so-called *plausible* positions for all but $o(n)$ vertices connected to bends in $ep^+_{S_i}(R)$; the following analysis can be analogously adapted for bends in $ep^-_{S_i}(R)$. To realize those connections, bends have to be connected to some vertices by a start segment. Consider the set of slopes $A = \{0, 1/4, 1/2, 3/4, 1, 4/3, 2, 4, \infty\}$ and the two slopes $s_\ell \in A$ and $s_r \in A$ closest to the slope p/q of segments in S_i; see Figure 11a. Further, let α_ℓ denote the angle between slopes s_ℓ and p/q and α_r the angle between slopes s_r and p/q. Observe that $0 < \alpha_\ell, \alpha_r < \pi/4$. The choice of slopes in A is arbitrary and is simply used to discretize the slope p/q with a new slope whose nominator and denominator can be both expressed as a constant. For a bend b in $ep^+_{S_i}(R)$, we define a region of *S_i^+-plausible positions* by a wedge opposite to the attached S_i-segment delimited by two rays of slopes s_ℓ and s_r, resp.; see

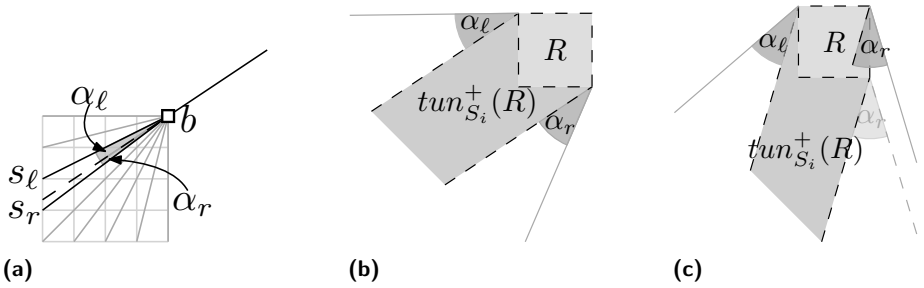


Figure 11 (a) Wedge of angle $\alpha_\ell + \alpha_r$ at bend $b \in ep^+_{S_i}(R)$ yielding S_i^+-plausible positions for b. (b)–(c) S_i^+-plausible region $plaus^+_{S_i}(R)$ of R, for different slopes of segments in S_i.

Fig. 11a. The union of the plausible positions of all bends in $ep^+_{S_i}(R)$ defines the *S_i^+-plausible region* $plaus^+_{S_i}(R)$ of R and consists of the union of R, $tun^+_{S_i}(R)$ and two attached wedges of angles α_ℓ and α_r, resp., on both sides of $tun^+_{S_i}(R)$. Observe these two wedges may be attached to two adjacent or two opposite corners of R depending on the slope of segments in S_i; see Figs. 11b and 11c. S_i^--, T_i^+- and T_i^--plausible regions are defined analogously.

▶ **Lemma 11.** *Let Γ be a RAC$_3$ drawing of K_n in $\mathcal{O}(n^2)$ area and let R be a region such that w.l.o.g. $|ep^+_{S_i}(R)| = \Omega(n^2)$. Vertices outside of $plaus^+_{S_i}(R) \cup tun^-_{S_i}(R)$ are directly connected to only $\mathcal{O}(n)$ bends in $ep^+_{S_i}(R)$ in total.*

Proof. The segments in S_i can only be crossed by segments of T_i. There are at most two start segments for each vertex that belong to T_i, hence, only two start segments of each vertex can cross segments of S_i. Those can be neglected as they are only $\mathcal{O}(n)$ segments and, in the following, we only consider start segments with different slopes. Consider an S_i-endpoint b and assume that b is connected to vertex v outside of $plaus^+_{S_i}(R) \cup tun^-_{S_i}(R)$. Assume w.l.o.g. that v is to the left of $plaus^+_{S_i}(R) \cup tun^-_{S_i}(R)$. The segment connecting b and v has a slope diverging by more than α_ℓ from slope p/q. Hence, b may be attached to a vertex v to the left of $plaus^+_{S_i}(R)$ only if a ray of slope s_ℓ with right endpoint b does not cross any segment in S_i with endpoint in $ep^+_{S_i}(R)$. This is true because the segment between b and v will intersect at least the segments that are also intersected by the ray of slope s_ℓ.

Consider the set of S_i-endpoints B for which such a crossing-free ray of slope s_ℓ exists; see Fig. 12a. Note that all rays are parallel and do not overlap. Since all possible slopes $s_\ell \in A$ can be expressed as a quotient p_ℓ/q_ℓ for $p_\ell, q_\ell \in \mathcal{O}(1)$ and since all rays hit one integer point, the minimum distance between two such parallel rays is $\Omega(1)$. Since R has size $\mathcal{O}(n) \times \mathcal{O}(n)$, it follows that there are only $\mathcal{O}(n)$ parallel rays of slope s_ℓ and hence $|B| = \mathcal{O}(n)$. ◀

In the following, we consider a region R and its set of *neighbored regions* $\mathcal{N}(R)$. A neighbored region $R' \in \mathcal{N}(R)$ is a region obtained by shifting R $c_1 q + c_2 p$ units along the x-axis and $c_1 p - c_2 q$ units along the y-axis for integers c_1, c_2; see Fig. 12b. Note that $\mathcal{N}(R)$ contains projections of $R \in \mathcal{R}_i$ that are not necessarily part of $\mathcal{R}_i$. Region R' contains all T_i-endpoints that are reached from S_i-endpoints in R by a segment in S_i with length $|c_1| \cdot \sqrt{p^2+q^2}$ followed by a segment in T_i of length $|c_2| \cdot \sqrt{p^2+q^2}$. Assume w.l.o.g. that $c_1, c_2 > 0$. Then, the edges with S- and T-endpoints in R and R', resp., will have a bend in $ep^+_{S_i}(R)$ and $ep^+_{T_i}(R')$. The symmetric cases where $c_1 < 0$ or $c_2 < 0$ are analogous. We say that a vertex v is an *R-vertex* if it is directly connected to $\Omega(n)$ bends in $ep^+_{S_i}(R)$ but to only $o(n)$ bends in $ep^+_{T_i}(R')$. Conversely, we say that v is an *R'-vertex* if it is directly connected to $\Omega(n)$ bends in $ep^+_{T_i}(R')$ but to only $o(n)$ bends in $ep^+_{S_i}(R)$. In the following, we show that except for $\mathcal{O}(n)$ edges, the edges with S- and T-endpoints in neighbored regions

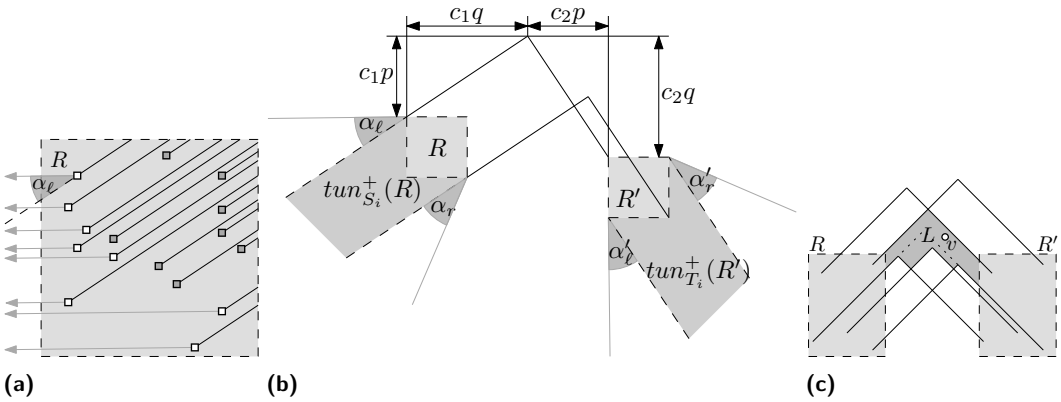


Figure 12 (a) Rays of slopes s_ℓ (arrows) attached to bends visible from outside $plaus^+_{S_i}(R) \cup tun^-_{S_i}(R)$. (b) Illustration of a region R and one of its neighbored regions R'. (c) Illustration of an L-tunnel L between region R and $R' \in \mathcal{N}(R)$ with a vertex v.

induce a bipartite subgraph between R- and R'-vertices. We refer to those exceptional edges which are either connecting two R- or R'-vertices or have an endpoint which is neither R- nor R'-vertex as *complete edges*. Intuitively speaking, a complete edge connects the set of R- and R'-vertices which are otherwise behaving like the partitions of a bipartite graph. In particular, every complete edge

(i) is either incident to a vertex which is neither R- nor R'-vertex, or
(ii) has a start segment that connects an R-vertex with a bend in $ep^+_{T_i}(R')$, or,
(iii) has a start segment that connects an R'-vertex with a bend in $ep^+_{S_i}(R)$.

We refer to the special endpoint as a *complete* endpoint. We first show that vertices in the intersection of $tun^-_{S_i}(R)$ and $tun^-_{T_i}(R')$ can be complete endpoint for only $\mathcal{O}(n)$ edges. Later, we will consider the case where vertices are not located in the intersection of $tun^-_{S_i}(R)$ and $tun^-_{T_i}(R')$. Consider a vertex v located in the intersection $tun^-_{S_i}(R) \cap tun^-_{T_i}(R')$ in a so-called *L-tunnel* between R and R'. An L-tunnel is a region bounded by edges with S- and T-endpoints in R and R', resp., that is open to both R and R' such that v can see into regions R and R'; see Fig. 12c. More precisely, an L-tunnel L is an open subregion of $tun^-_{S_i}(R) \cap tun^-_{T_i}(R')$ bounded from below by an alternating sequence of S_i and T_i segments between R and R' and from above by two segments, one from S_i and one from T_i, while it is bounded to the left by the boundary of R and to the right by the boundary of R'.

▶ **Lemma 12.** *Let Γ be a RAC$_3$ drawing of K_n in $\mathcal{O}(n^2)$ area, let R be a region such that w.l.o.g. $|ep^+_{S_i}(R)| = \Omega(n^2)$ and let $R' \in \mathcal{N}(R)$. There are $\mathcal{O}(n)$ complete edges with both a bend in $ep^+_{S_i}(R)$ and in $ep^+_{T_i}(R')$ whose complete endpoints are an in L-tunnels in $tun^-_{S_i}(R) \cap tun^-_{T_i}(R')$.*

Proof. Let v be a complete endpoint in $tun^-_{S_i}(R) \cap tun^-_{T_i}(R')$, hence, it is connected to bends in $ep^+_{S_i}(R)$ and to bends in $ep^+_{T_i}(R')$. Further, assume that v is complete endpoint for at least two edges; the complete endpoints that we disregard only contribute $\mathcal{O}(n)$ edges. Assume w.l.o.g. that the slope of segments in S_i is less than 1. Then, the slope of segments in T_i is less than -1 and v is located above R'. Let $B(v)$ denote the set of bends that v is connected to in $ep^+_{T_i}(R')$. We further divide $B(v)$ into $B^-(v)$ and $B^+(v)$, i.e., the set of bends b such that v is located in the halfplane above and below b's segment in T_i, resp. We first show, that each $b \in B^\pm(v)$ shares its y-coordinate with no other bend $b' \in B^\pm(v)$. Then, we show that for two vertices v and v' in $tun^-_{S_i}(R) \cap tun^-_{T_i}(R')$, the y-coordinates of bends in $B^\pm(v)$ and $B^\pm(v')$ differ. As a result, there are only $\mathcal{O}(n)$ bends in R', which implies that there are only $\mathcal{O}(n)$ complete edges.

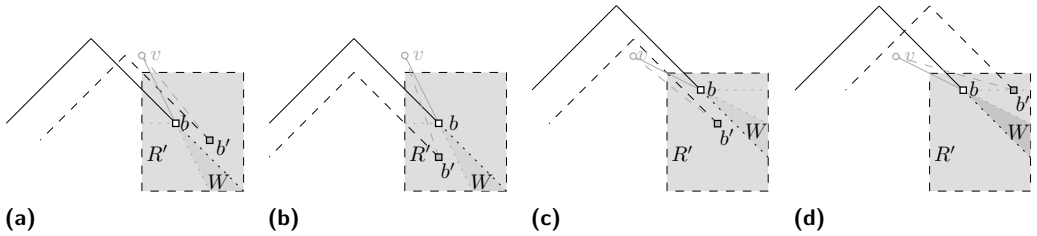


Figure 13 A bend b incident to a vertex v restricts the position of other bends b' to a wedge W.

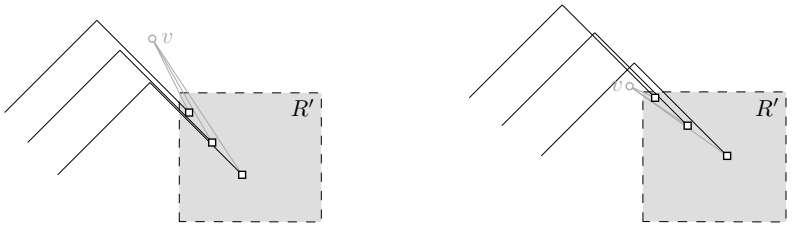


Figure 14 (a) Edges below a vertex v incident to v do not intersect while (b) edges above a vertex v incident to v pairwise intersect.

Consider a vertex v and a bend $b \in B(v)$. First, assume that $b \in B^-(v)$. Note that since v is located above R' in the intersection $tun^-_{S_i}(R) \cap tun^-_{T_i}(R')$, the segment between b and v has slope less than -1. Then v can only be incident to another bend $b' \in B^-(v)$ whose T_i-segment is below v only if b' is located in the wedge W obtained by the elongation of the T_i-segment of b and the segment between b and v. Otherwise the T_i-segment incident to b' would intersect the segment between b and v (see Fig. 13a) or the segment between b' and v would intersect the T_i-segment incident to b (see Fig. 13b). Since the angle between the two segments spanning the wedge W from above is less than π, W contains no other bend with the same y-coordinate as b. Moreover, we observe that the T_i-segments incident to such bends and consecutive S_i-segments do not cross each other (see Fig. 14a) since all S_i-(T_i-, resp.)segments between R and R' have the same length. Second, consider the case where $b \in B^+(v)$. Here, the argumentation is analogous to the previous case (see Figs. 13c and 13d). Note that in this case, the edges using bends in $B^+(v)$ pairwise intersect (see Fig. 14b). Still segments incident to v have negative slopes.

It remains to consider the dependencies of the neighborhoods of two vertices v and v' located in $tun^-_{S_i}(R) \cap tun^-_{T_i}(R')$. First, consider the positions of bends in $B^-(v)$ and of bends in $B^-(v')$. There are three possibilities for the relative positioning of v and v':

1. v and v' appear in different L-tunnels such that w.l.o.g. the L-tunnel of v' appears below the bottommost bend $b_{bot}(v) \in B^-(v)$. Then v' appears in the halfplane below the segment in T_i attached to $b_{bot}(v)$. Even more, the topmost bend $b_{top}(v') \in B^-(v')$ must be located below v'; see Fig 15a. Hence, the y-coordinates of $B^-(v)$ and $B^-(v')$ are different.
2. v and v' appear in the same L-tunnel. W.l.o.g. the topmost bend $b_{top}(v') \in B^-(v')$ will be located in a wedge W below $b_{bot}(v) \in B^-(v)$ delimited by the elongation of the T_i-segment through $b_{bot}(v)$ and a horizontal through $b_{bot}(v)$; see Fig. 15b. The horizontal segment delimits W as by the choice of the size of regions, v' will be located above R', hence it also does not belong to W. Therefore, y-coordinates of $B^-(v)$ and $B^-(v')$ differ.

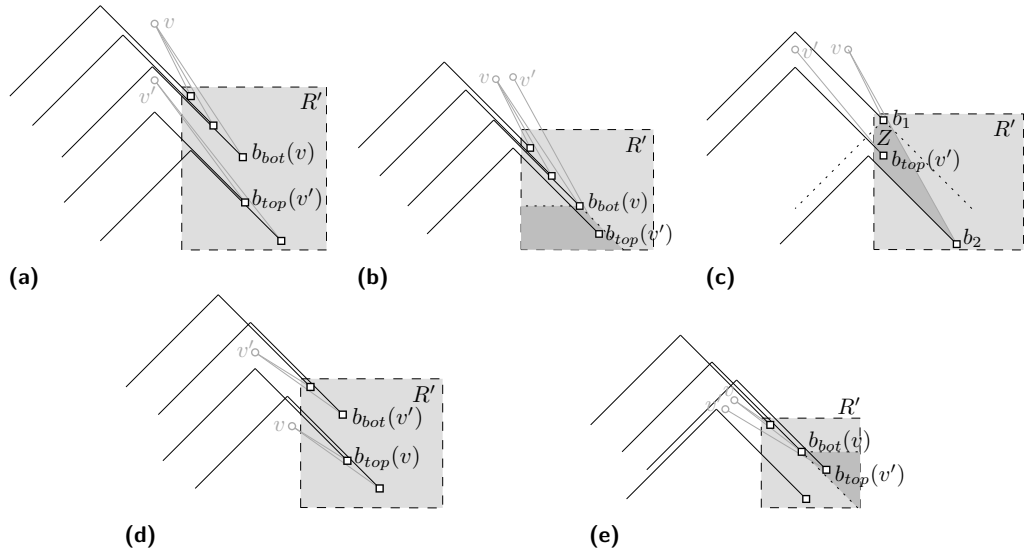


Figure 15 (a)–(c) Bends in $B^-(v)$ and bends in $B^-(v')$ have different y-coordinates. (d)–(e) Bends in $B^+(v)$ and bends in $B^+(v')$ have different y-coordinates.

3. v' is located in between the T_i-segments incident to $b_1, b_2 \in B^-(v)$. The bends in $B^-(v')$ can only be located in a region Z bounded by two lines of the slopes of S_i and T_i passing through b_1, the T_i-segment incident to b_2, the segment between v and b_2 and the boundary of region R'. All points in this region have smaller y-coordinates than b_1 and larger y-coordinates than b_2; see Fig. 15c. Note that the line parallel to segments of S_i passing through b_1 is part of the boundary as otherwise the segments incident to b_1 and a bend in $B^-(v')$, resp., would intersect preventing v' from having segments to bends in R.

Second, consider how the positions of bends in $B^+(v)$ and bends in $B^+(v')$ depend on each other. Here, there are only two possibilities for the relative positioning of v and v' which are analogous to the Cases 1 and 2 above, see Figs. 15d and 15e.

Note that a bend in $\bigcup_v B^-(v)$ and in $\bigcup_v B^+(v)$ may share a common y-coordinate. By the previous analysis, this is the only possibility for two bends in $\bigcup_v B(v)$ to share a y-coordinate. It follows that $|\bigcup_v B^-(v)| = \mathcal{O}(n)$ and $|\bigcup_v B^+(v)| = \mathcal{O}(n)$ and hence only a linear number of complete edges can be realized as claimed. ◀

Now, we summarize the partial results from Lemmas 11 and 12 to conclude that most vertices are R- or R'-vertices with only $\mathcal{O}(n)$ incident complete edges.

▶ **Lemma 13.** *Let Γ be a RAC$_3$ drawing of K_n in $\mathcal{O}(n^2)$ area. Further let R be a region such that w.l.o.g. $|ep^+_{S_i}(R)| = \Omega(n^2)$ and let $R' \in \mathcal{N}(R)$. Then there exist only $\mathcal{O}(n)$ complete edges that have a bend in $ep^+_{S_i}(R)$ and in $ep^+_{T_i}(R')$.*

Proof. Assume $tun^-_{S_i}(R)$ is delimited by the extension of two segments of S_i between R and R'. Otherwise, R can be restricted to a smaller region that only includes S_i-endpoints that are connected to T_i-endpoints located in R'. Assume that there are $\omega(n)$ complete edges with S- and T-endpoints in R and R', resp. By Lemma 12, there exist only $\mathcal{O}(n)$ complete edges with S- and T-endpoints in R and R', resp., whose complete endpoints are located in $tun^-_{S_i}(R) \cap tun^-_{T_i}(R')$. Since $tun^-_{S_i}(R)$ is bounded by two segments of S_i between R and R', this covers all complete endpoints in $tun^-_{S_i}(R) \cap tun^-_{T_i}(R')$ and other complete endpoints located in

$tun^-_{S_i}(R) \cup tun^-_{T_i}(R')$ lie outside of $plaus^+_{S_i}(R) \cup tun^-_{S_i}(R)$ or outside of $plaus^+_{T_i}(R') \cup tun^-_{T_i}(R')$. By Lemma 11, only $\mathcal{O}(1)$ vertices outside of $plaus^+_{S_i}(R) \cup tun^-_{S_i}(R)$ can each be connected to $\Omega(n)$ bends of $ep^+_{S_i}(R)$, also only $\mathcal{O}(1)$ vertices outside of $plaus^+_{T_i}(R') \cup tun^-_{T_i}(R')$ can each be connected to $\Omega(n)$ bends of $ep^+_{T_i}(R')$. Thus, there must be complete endpoints in $plaus^+_{S_i}(R) \cap plaus^+_{T_i}(R')$ contradicting $plaus^+_{S_i}(R) \cap plaus^+_{T_i}(R') = \emptyset$. ◀

Lemma 13 shows, that most edges with S- and T-endpoints in R and R', resp., define a bipartite subgraph. As a result, edges between R and $\mathcal{N}(R)$ define a p-partite subgraph for some constant $p > 1$. However, since the drawn graph is complete, all R-vertices have to define a clique. This will lead to a contradiction in the proof of the main theorem of this section.

▶ **Theorem 14.** *There is no RAC_3 drawing of K_n in $\mathcal{O}(n^2)$ area for sufficiently large n.*

Proof. Assume there is a RAC_3 drawing Γ of K_n in $\mathcal{O}(n^2)$ area. We show that there is a complete subgraph G' with $\Omega(n)$ vertices drawn in Γ with only $o(n^2)$ edges from E^{ST}_i for all pairs of perpendicular edge segments $S_i \in \mathcal{S}$ and $T_i \in \mathcal{T}$. This contradicts the property from Lemma 6 for the subdrawing of G'. Let c_{ST} denote the multiplicative constant from Lemma 6, i.e. $|E^{ST}_i| \geq c_{ST} n^2$.

We compute $G' = (V', E')$ iteratively. We initialize G' by G. We consider all pairs of sets of segments $S_i \in \mathcal{S}$ and $T_i \in \mathcal{T}$ with perpendicular slopes such that $|E^{ST}_i| \geq c_{ST} n^2$. There can be only a constant number of pairs of segment sets in $\mathcal{S}$ and $\mathcal{T}$ of size $\Omega(n^2)$. For each such pair, let $\mathcal{R}_i$ be a checkerboard partitioning of the drawing area into square-shaped disjoint regions of side length $\max\{p_i, q_i\}/2$ defined by slope p_i/q_i of S_i for coprime integers p_i and q_i. We consider all regions $R \in \mathcal{R}_i$. Due to the size of regions, there is only a constant number of regions in $\mathcal{R}_i$. Moreover, the number of neighbored regions $R' \in \mathcal{N}(R)$ is constant. Hence, there is a constant number n_{comb} of combinations of index i, region R and neighbored region R'.

We iteratively perform the following procedure while there are at least $c_{ST}|V'|^2$ edges with a bend in $ep^+_{S_i}(R)$ and a bend in $ep^{\pm}_{T_i}(R')$ for one of n_{comb} of combinations of index i, region R and neighbored region R' where $ep^{\pm}_{T_i}(R')$ is either $ep^+_{T_i}(R')$ or $ep^-_{T_i}(R')$ depending on the choice of R'. Let V_R denote the set of R- and $V_{R'}$ denote the set of R'-vertices. Assume w.l.o.g. that $|V_R| \geq |V_{R'}|$. By Lemma 13, the vertices that are neither R- nor R'-vertices are connected to $\mathcal{O}(n)$ bends in $ep^+_{S_i}(R)$, $ep^+_{T_i}(R')$ or $ep^-_{T_i}(R')$ in total. More precisely, there are $c_{comp}|V'|$ such bends for an appropriately chosen constant c_{comp}. Recall that vertices that are not incident to $\Omega(n)$ bends, say at least $c_R|V^*|$ for an appropriately chosen constant c_R, in $ep^+_{S_i}(R)$ and $ep^+_{T_i}(R')$ are either R- or R'-vertex for the resulting graph $G^* = (V^*, V^* \times V^*)$. By the prior observation, $|V_R| = \Omega(n)$, more precisely, $|V_R| \geq (|V'| - c_{comp}|V'|/c_R|V^*|)/2$. We then continue to consider the complete subgraph induced by V_R. Note that since n_{comb} is constant, by Lemma 13, this subgraph and the subgraphs in the future iterations contain only $\mathcal{O}(n)$ edges with both a segment in S_i and a segment in T_i and bends in $ep^+_{S_i}(R)$ and $ep^+_{T_i}(R')$ or $ep^-_{T_i}(R')$ for all $R' \in \mathcal{N}(R)$ for sufficiently large n. Thus, we set $G' \leftarrow (V_R, V_R \times V_R)$ and continue with the next iteration.

After performing all at most n_{comb} iterations, there are less than $c_{ST}|V'|^2$ edges with both a bend in $ep^+_{S_i}(R)$ and a bend in $ep^+_{T_i}(R')$ and $ep^-_{T_i}(R')$ for all combinations of index i, region $R \in \mathcal{R}_i$ and neighbored region $R' \in \mathcal{N}(R)$. Hence, the resulting subgraph G' is drawn with $|E^{ST}_i| < c_{ST}|V'|^2$ for each pair of sets of segments $S_i \in \mathcal{S}$ and $T_i \in \mathcal{T}$ with perpendicular slopes which contradicts Lemma 6. ◀

Our proofs explicitly use the assumption of quadratic area (Lemmas 8 to 12) and three bends per edge (Lemmas 11 to 13). Even for $\omega(n^2)$ area, our proof does not apply.

4 Open Questions

We raise the following open questions:

(i) How many bends are needed for achieving quadratic area RAC drawings? We showed that three are insufficient and that eight are enough.

(ii) Is cubic area optimal for RAC_3 drawings? Our lower bound proof might be extendable.

(iii) Is quadratic area achievable in simple RAC drawings? In *simple drawings*, every pair of edges shares at most one point (crossing or endpoint); a property our algorithms do not guarantee.

References

1 M. Ajtai, V. Chvátal, M. M. Newborn, and E. Szemerédi. Crossing-free subgraphs. In *Theory and practice of combinatorics*, volume 60 of *North-Holland Math. Stud.*, pages 9–12. North-Holland, Amsterdam, 1982.

2 Patrizio Angelini, Michael A. Bekos, Henry Förster, and Michael Kaufmann. On rac drawings of graphs with one bend per edge. *Theoretical Computer Science*, 2020. doi:10.1016/j.tcs.2020.04.018.

3 Patrizio Angelini, Luca Cittadini, Walter Didimo, Fabrizio Frati, Giuseppe Di Battista, Michael Kaufmann, and Antonios Symvonis. On the perspectives opened by right angle crossing drawings. *J. Graph Algorithms Appl.*, 15(1):53–78, 2011. doi:10.7155/jgaa.00217.

4 Patrizio Angelini, Giuseppe Di Battista, Walter Didimo, Fabrizio Frati, Seok-Hee Hong, Michael Kaufmann, Giuseppe Liotta, and Anna Lubiw. Large angle crossing drawings of planar graphs in subquadratic area. In Alberto Márquez, Pedro Ramos, and Jorge Urrutia, editors, *Computational Geometry - XIV Spanish Meeting on Computational Geometry, EGC 2011, Dedicated to Ferran Hurtado on the Occasion of His 60th Birthday*, volume 7579 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2011. doi:10.1007/978-3-642-34191-5_19.

5 Evmorfia N. Argyriou, Michael A. Bekos, and Antonios Symvonis. The straight-line RAC drawing problem is np-hard. *J. Graph Algorithms Appl.*, 16(2):569–597, 2012. doi:10.7155/jgaa.00274.

6 Karin Arikushi, Radoslav Fulek, Balázs Keszegh, Filip Moric, and Csaba D. Tóth. Graphs that admit right angle crossing drawings. *Comput. Geom.*, 45(4):169–177, 2012. doi:10.1016/j.comgeo.2011.11.008.

7 Christian Bachmaier, Franz J. Brandenburg, Kathrin Hanauer, Daniel Neuwirth, and Josef Reislhuber. NIC-planar graphs. *Discrete Applied Mathematics*, 232:23–40, 2017. doi:10.1016/j.dam.2017.08.015.

8 Michael A. Bekos, Walter Didimo, Giuseppe Liotta, Saeed Mehrabi, and Fabrizio Montecchiani. On RAC drawings of 1-planar graphs. *Theor. Comput. Sci.*, 689:48–57, 2017. doi:10.1016/j.tcs.2017.05.039.

9 Franz J. Brandenburg, Walter Didimo, William S. Evans, Philipp Kindermann, Giuseppe Liotta, and Fabrizio Montecchiani. Recognizing and drawing IC-planar graphs. *Theor. Comput. Sci.*, 636:1–16, 2016. doi:10.1016/j.tcs.2016.04.026.

10 Steven Chaplick, Fabian Lipp, Alexander Wolff, and Johannes Zink. Compact drawings of 1-planar graphs with right-angle crossings and few bends. *Comput. Geom.*, 84:50–68, 2019. doi:10.1016/j.comgeo.2019.07.006.

11 Emilio Di Giacomo, Walter Didimo, Peter Eades, and Giuseppe Liotta. 2-layer right angle crossing drawings. *Algorithmica*, 68(4):954–997, 2014. doi:10.1007/s00453-012-9706-7.

12 Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Henk Meijer. Area, curve complexity, and crossing resolution of non-planar graph drawings. *Theory Comput. Syst.*, 49(3):565–575, 2011. doi:10.1007/s00224-010-9275-6.

13 Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. Area requirement of graph drawings with few crossings per edge. *Comput. Geom.*, 46(8):909–916, 2013. doi:10.1016/j.comgeo.2013.03.001.

14 Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. Area-thickness trade-offs for straight-line drawings of planar graphs. *Comput. J.*, 60(1):135–142, 2017. doi:10.1093/comjnl/bxw075.

15 Walter Didimo, Peter Eades, and Giuseppe Liotta. Drawing graphs with right angle crossings. *Theor. Comput. Sci.*, 412(39):5156–5166, 2011. doi:10.1016/j.tcs.2011.05.025.

16 Vida Dujmović, Joachim Gudmundsson, Pat Morin, and Thomas Wolle. Notes on large angle crossing graphs. *Chicago J. Theor. Comput. Sci.*, 2011, 2011. URL: http://cjtcs.cs.uchicago.edu/articles/CATS2010/4/contents.html.

17 Christian A. Duncan and Michael T. Goodrich. Planar orthogonal and polyline drawing algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 223–246. Chapman and Hall/CRC, 2013.

18 Peter Eades and Giuseppe Liotta. Right angle crossing graphs and 1-planarity. *Discrete Applied Mathematics*, 161(7-8):961–969, 2013. doi:10.1016/j.dam.2012.11.019.

19 Seok-Hee Hong and Hiroshi Nagamochi. Testing full outer-2-planarity in linear time. In Ernst W. Mayr, editor, *Graph-Theoretic Concepts in Computer Science - 41st International Workshop, WG 2015*, volume 9224 of *Lecture Notes in Computer Science*, pages 406–421. Springer, 2015. doi:10.1007/978-3-662-53174-7_29.

20 Weidong Huang. Using eye tracking to investigate graph layout effects. In Seok-Hee Hong and Kwan-Liu Ma, editors, *APVIS 2007, 6th International Asia-Pacific Symposium on Visualization 2007*, pages 97–100. IEEE Computer Society, 2007. doi:10.1109/APVIS.2007.329282.

21 Weidong Huang, Peter Eades, and Seok-Hee Hong. Larger crossing angles make graphs easier to read. *J. Vis. Lang. Comput.*, 25(4):452–465, 2014. doi:10.1016/j.jvlc.2014.03.001.

22 Frank Thomson Leighton. *Complexity Issues in VLSI*. Foundations of Computing Series. MIT Press, Cambridge, MA, 1983.

23 János Pach and Géza Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997. doi:10.1007/BF01215922.

24 Helen C. Purchase. Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2):147–162, 2000. doi:10.1016/S0953-5438(00)00032-1.

25 Helen C. Purchase, David A. Carrington, and Jo-Anne Allder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, 2002.

26 Zahed Rahmati and Fatemeh Emami. RAC drawings in subcubic area. *Information Processing Letters*, 159-160:105945, 2020. doi:10.1016/j.ipl.2020.105945.

Fast Preprocessing for Optimal Orthogonal Range Reporting and Range Successor with Applications to Text Indexing

Younan Gao
Faculty of Computer Science, Dalhousie University, Halifax, Canada
yn803382@dal.ca

Meng He
Faculty of Computer Science, Dalhousie University, Halifax, Canada
mhe@cs.dal.ca

Yakov Nekrich
Department of Computer Science, Michigan Technological University, Houghton, MI, USA
yakov.nekrich@googlemail.com

Abstract

Under the word RAM model, we design three data structures that can be constructed in $O(n\sqrt{\lg n})$ time over n points in an $n \times n$ grid. The first data structure is an $O(n \lg^{\epsilon} n)$-word structure supporting orthogonal range reporting in $O(\lg \lg n + k)$ time, where k denotes output size and ϵ is an arbitrarily small constant. The second is an $O(n \lg \lg n)$-word structure supporting orthogonal range successor in $O(\lg \lg n)$ time, while the third is an $O(n \lg^{\epsilon} n)$-word structure supporting sorted range reporting in $O(\lg \lg n + k)$ time. The query times of these data structures are optimal when the space costs must be within $O(n \operatorname{polylog} n)$ words. Their exact space bounds match those of the best known results achieving the same query times, and the $O(n\sqrt{\lg n})$ construction time beats the previous bounds on preprocessing. Previously, among 2d range search structures, only the orthogonal range counting structure of Chan and Pătraşcu (SODA 2010) and the linear space, $O(\lg^{\epsilon} n)$ query time structure for orthogonal range successor by Belazzougui and Puglisi (SODA 2016) can be built in the same $O(n\sqrt{\lg n})$ time. Hence our work is the first that achieve the same preprocessing time for optimal orthogonal range reporting and range successor. We also apply our results to improve the construction time of text indexes.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Data structures design and analysis

Keywords and phrases orthogonal range search, geometric data structures, orthogonal range reporting, orthogonal range successor, sorted range reporting, text indexing, word RAM

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.54

Related Version A full version of the paper is available at `http://arxiv.org/abs/2006.11978`.

1 Introduction

Two dimensional orthogonal range search problems have been studied intensively in the communities of computational geometry, data structures and databases. The goal of these problems is to maintain a set, N, of points on the plane in a data structure such that one can efficiently compute aggregate information about the points contained in an axis-aligned query rectangle Q. Among these problems, *orthogonal range counting* and *orthogonal range reporting* are perhaps the most fundamental; the former counts the number of points contained in $N \cap Q$ while the latter reports them. Another well-known problem is *orthogonal range successor*, which asks for the point in $N \cap Q$ with the smallest x- or y-coordinate. Range

counting, reporting and successor have many applications including text indexing [23, 8, 6, 25], Lempel-Ziv factorization [4] and consensus trees in phylogenetics [18], to name a few. See [22] for a survey on the connection between text indexing and various range searching techniques.

Most work on orthogonal range searching [13, 17, 11, 28, 32] focuses on achieving the best tradeoffs between query time and space, and preprocessing time is often neglected. However, the preprocessing time of a data structure matters when it is used as a building block of an algorithm processing plain data, as the total running time includes that needed to build the structure. Furthermore, an orthogonal range search structures with fast construction time are preferred when preprocessing huge amounts of data, e.g., when used as components of text indexes built upon large data sets from search engines and bioinformatics applications. The work of Chan and Pătraşcu [12] is the first that breaks the $O(n \lg n)$ bound on the construction time of 2d orthogonal range counting structures; they designed an $O(n)$-word structure with $O(\lg n / \lg \lg n)$ query time that can be built in $O(n\sqrt{\lg n})$ time. Their ideas were further extended to design an $O(n \lg \sigma / \sqrt{\lg n})$-time algorithm to build a binary wavelet trees over a string of length n drawn from $[\sigma]$ [26, 2][1], which is a key data structure used in succinct text indexes. More recently, Belazzougui and Puglisi [4] showed how to construct, in $O(n\sqrt{\lg n})$ time, an $O(n)$-word data structure supporting range successor in $O(\lg^{\epsilon} n)$ time, and applied this algorithm to achieve new results on Lempel-Ziv parsing.

The previous work on constructing orthogonal range search structures in $O(n\sqrt{\lg n})$ time focuses on linear space data structures. To achieve optimal query time for 2d orthogonal range reporting and range successor using near-linear space, however, the best tradeoffs under the word RAM model requires superlinear space [11, 32]. The increased space costs are needed to encode more information, posing new challenges to fast construction. We thus investigate the problem of designing data structures with optimal query times for range reporting and range successor that can be built in $O(n\sqrt{\lg n})$ time, while matching the space costs of the best known solutions. We also consider a closely related problem called *sorted range reporting* [28] to achieve similar goals. In this problem, we report all points in $N \cap Q$ in a sorted order along either x- or y-axis. The query time should depend on the number of points actually reported even if the procedure is ended early by user.

Previous Work. The research on 2d orthogonal range reporting has a long history [30, 13, 1, 17, 27, 19, 9, 11]. Researchers have achieved three best tradeoffs between query time and space costs under the word RAM model; we follow the state of the art and assume that the input points are in rank space. The solution with optimal query time of $O(\lg \lg n + k)$ and space cost of $O(n \lg^{\epsilon} n)$ words is due to Alstrup et al. [1], while the best linear-space solution is designed by Chan et al [11] which answers a query in $O((1+k) \lg^{\epsilon} n)$ time, where k is the output size and ϵ is an arbitrarily small constant. Chan et al. also proposed an $O(\lg \lg n)$-word structure with $O((1+k) \lg \lg n)$ query time and another tradeoff matching that of Alstrup et al. [1].

The 2d orthogonal range successor problem was also studied extensively. After a series of work [21, 20, 15, 14, 31], Nekrich and Navarro [28] gave two solutions to this problem; the first uses $O(n)$ words and answers a query in $O(\lg^{\epsilon} n)$ time, while the second uses $O(n \lg \lg n)$ words to answer a query in $O((\lg \lg n)^2)$ time. Zhou [32] decreased the query time of the latter to $O(\lg \lg n)$ without increasing space costs. By definition, a solution to orthogonal range successor can be used to answer sorted range reporting queries. Furthermore, Nekrich and Navarro [28] also designed a data structure using $O(n \lg^{\epsilon} n)$ words to support sorted range

[1] In this paper, $[\sigma]$ denotes $\{0, 1, \ldots, \sigma - 1\}$.

reporting in $O(\lg\lg n + k)$ time. Hence, the best three time-space tradeoffs for the original 2d orthogonal range reporting problem has also been achieved for the sorted version. The optimality of the $O(\lg\lg n + k)$ query time for orthogonal range reporting and the $O(\lg\lg n)$ query time for orthogonal range successor when no more than $O(n \operatorname{polylog} n)$ space can be used is established by a lower bound on range emptiness [29].

Alstrup et al. [1] claimed that their structure for optimal orthogonal range reporting can be constructed in $O(n \lg n)$ expected time. Even though preprocessing times are not given in [11, 28, 32], straightforward analyses reveal that the other data structures we surveyed here can be built in $O(n \lg n)$ worst-case time (Bille and Gørtz [6] also claimed that the preprocessing time of the $O(n \lg\lg n)$-word structure of Chan et al. [11] is $O(n \lg n)$). Hence, when faster preprocessing time is needed in their solution to Lempel-Ziv decomposition, Belazzougui and Puglisi [4] had to design a new linear-space data structure for orthogonal range successor with $O(n\sqrt{\lg n})$ preprocessing time and $O(\lg^{\epsilon} n)$ query time. No attempts have been published to achieve similar preprocessing times for other tradeoffs.

Our Results. Under the word RAM model, we design the following three data structures that can be constructed in $O(n\sqrt{\lg n})$ time over n points in an $n \times n$ grid:

- An $O(n \lg^{\epsilon} n)$-word structure supporting orthogonal range reporting in $O(\lg\lg n + k)$ time, where k denotes the output size and ϵ is an arbitrarily small constant;
- An $O(n \lg\lg n)$-word structure supporting orthogonal range successor in $O(\lg\lg n)$ time;
- An $O(n \lg^{\epsilon} n)$-word structure supporting sorted range reporting in $O(\lg\lg n + k)$ time.

The query times of these structures are optimal when space costs must be within $O(n \operatorname{polylog} n)$ words. Their exact space bounds match those of the best known results achieving the same query times, and the $O(n\sqrt{\lg n})$ construction time beats the previous bounds on preprocessing. Note that even though our third result implies the first one, our data structure for the first is much simpler. In addition, our results can be used to improve the construction time of text indexes. For a text string T of length n over alphabet $[\sigma]$, we design

- A text index of $O(n \lg\sigma \lg^{\epsilon} n)$ bits that can be constructed in $O(n \lg\sigma/\sqrt{\lg n})$ time and can report the $\mathtt{occ}$ occurrences of a pattern of length p in time $O(p/\log_{\sigma} n + \log_{\sigma} n \lg\lg n + \mathtt{occ})$, where ϵ is any small positive constant. This improves one result of Munro et al. [25] who designed the first text indexes with both sublinear construction time and query time for small σ; for the same time-space tradeoff, their preprocessing time is $O(n \lg\sigma \lg^{\epsilon} n)$.
- A text index of $O(n \lg^{1+\epsilon} n)$ bits for any constant $\epsilon > 0$ built in $O(n\sqrt{\lg n})$ time that supports position-restricted substring search [23] in $O(p/\log_{\sigma} n + \lg p + \lg\lg\sigma + \mathtt{occ})$ time. Previous indexes with similar query performance require $O(n \lg n)$ construction time.

Overview of Our Approach. We first discuss why some obvious approaches will not work. The modern approach of Chan et al [11] for orthogonal range reporting is based on a problem called ball inheritance which they defined over range trees. This solution is well-known for its simplicity, and by choosing different parameters in their approach to ball inheritance, they obtain all three best known tradeoffs. One natural idea is to redesign the structures stored at range tree nodes to use bit packing to speed up construction. However, even though we have achieved construction time matching the state of the art for these structures, it is still not enough to construct the data structures for the tradeoffs of ball inheritance that we need quickly enough. Another idea is to tune the parameters in the approach of Belazzougui and Puglisi [4], hoping to obtain the tradeoffs that we aim for, as they already showed how to construct in $O(n\sqrt{\lg n})$ time a linear space, $O((k+1) \lg^{\epsilon} n)$ query

time structure for orthogonal range reporting. Their solution uses many trees grouped into $O(\lg^\epsilon n)$ levels of granularity. If we borrow ideas from [11] and set parameters to achieve different tradeoffs, we would use $O(1/\epsilon)$ or $O(\lg\lg n)$ levels of granularity. However, to return a point in the answer, their query algorithm would perform operations requiring $O(\lg\lg n)$ time at each level of granularity. Thus, at best, the former would give an $O(n\lg^\epsilon n)$-word structure with $O((k+1)\lg\lg n)$ query time and the latter an $O(n\lg\lg n)$-word structure with $O((k+1)(\lg\lg n)^2)$ query time. Either solution is inferior to best known tradeoffs. This however is fine in the original solution, as the total cost of spending $O(\lg\lg n)$ time at each of the $O(\lg^\epsilon n)$ levels is bounded by $O(\lg^{\epsilon'} n)$ for any $\epsilon' > \epsilon$.

We thus design new approaches. For optimal orthogonal range reporting, our overall strategy is to perform two levels of reductions, making it sufficient to solve ball inheritance in special cases with fast preprocessing time. More specifically, we first use a generalized wavelet tree and range minimum/maximum structures to reduce the problem in the general case to the special case in which the points are from a $2^{\sqrt{\lg n}} \times n'$ (narrow) grid, where $n' \leq n$. In this reduction, we need only support ball inheritance over a wavelet tree with high fanout. We further reduce the problem over points in a narrow grid to that over a (small) grid of size at most $2^{\sqrt{\lg n}} \times 2^{2\sqrt{\lg n}}$. This is done by grouping points and selecting representatives from each group, so that previous results with slower preprocessing time can be used over a smaller set of representatives. Finally, over the small grid, we solve ball inheritance when the coordinates of each point can be encoded in $O(\sqrt{\lg n})$ bits. The ball inheritance structures in both special cases can be built quickly by redesigning components with fast preprocessing, though the second case requires a twist to the approach of Chan et al [11]. Our solutions to optimal range successor and sorted range reporting are based on similar strategies, though we preform more levels of reductions.

In the main body of this paper, we describe our data structures for optimal range reporting and successor, while those for optimal sorted range reporting are deferred to the full version of this paper.

2 Preliminaries

In this section, we describe and sometimes extend the previous results used in this paper. The proofs omitted from this section can be found in the full version of this paper.

Notation. We adopt the word RAM model with word size $w = \Theta(\lg n)$ bits, where n denotes the size of the given data. Our complete solutions use several sets of homogeneous components. We present a lemma to bound the costs of each different type of components, which is then applied over the entire set of these components to calculate the total cost. The size, n', of the data that each component represents may be less than n which is the input size of the entire problem, so when the cost of constructing the component is bounded by a function of the form $f(n')/\operatorname{polylog}(n)$ to take advantage of the word size, we keep both n' and n in the lemma statement, as commonly done in previous work on similar topics. In this case, the construction algorithm usually uses a universal table of $o(n)$ bits, whose content solely depends on the value of n, and hence can be constructed once in $o(n)$ time and used for all data structure components of the same type. Thus unless otherwise stated, these lemmas assume the existence of such a table without stating so explicitly in the lemma statements, and we define and analyze the table in the proof. This also applies to algorithms that manipulate sequences of size n'. Occasionally the query algorithms of a data structure may need a universal table as well, and we explicitly state it if this is the case.

We say a sequence $A \in [\sigma]^n$ is in *packed* form if the bits of its elements are concatenated and stored in as few words as possible. Thus, when packed, A occupies $\lceil n\lceil \lg \sigma \rceil / w \rceil$ words.

Generalized Wavelet Trees. Given a sequence $A[0..n-1]$ drawn from alphabet $[\sigma]$, a d-ary generalized wavelet tree [24] T_d over A is a balanced tree in which each internal node has d children, where $2 \leq d \leq \sigma$. For simplicity, assume that σ is a power of d. Each node of T_d then represents a range of alphabet symbols defined as follows: At the leaf level, the i-th leaf from the left represents the integer range $[i, i]$ for each $i \in [0..\sigma - 1]$. The range represented by an internal node is the union of the ranges represented by its children. Hence the root represents $[0, \sigma - 1]$, and T_d is a complete tree having $\log_d \sigma + 1$ levels. Each node u is further associated with a subsequence, $A(u)$, of A, in which $A(u)[i]$ stores the i-th entry in A that is in the range represented by u. Thus the root is associated with the entire sequence A. To save storage, $A[u]$ is not stored explicitly in [24]. Instead, each internal node u stores a sequence $S(u)$ of integers in $[d]$, where $S(u)[i] = j$ if $A(u)[i]$ is within the range represented by the jth child of u. All the $S(u)$'s built for internal nodes occupy $O(n \lg \sigma)$ bits in total.

Generalized wavelet trees share fundamental ideas with range trees but are more suitable for compact data structures over sequences which may contain duplicate values. When we use them in this paper, we sometimes explicitly store $A(u)$ for each node u, and may even associate with u an additional array $I(u)$ in which $I(u)[i]$ stores the index of $A(u)[i]$ in the original sequence A. We call $A(u)$ the *value array* of u, and $I(u)$ the *index array*. In this paper, if we construct value and/or index arrays for each node, we explicitly state so. If not, it implies that we build a wavelet tree in which each node u is associated with $S(u)$ only. Furthermore, unless otherwise specified, we apply the standard pointer-based implementation to represent the tree structure of a wavelet tree, which is preprocessed in time linear to the number of tree nodes such that the lowest common ancestor of any two nodes can be located in $O(1)$ time [5]. We also number the levels of the tree incrementally starting from the root level, which is level 0. We have the following two lemmas on constructing wavelet trees:

▶ **Lemma 1.** *Let $A[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$ and $I[0..n'-1]$ be a packed sequence in which $I[i] = i$ for each $i \in [0..n'-1]$, where $n' \leq n$ and $\sigma \leq 2^{O(\sqrt{\lg n})}$. Given A and I as input, a d-ary wavelet tree over A with value and index arrays in packed form can be constructed in $O(n' \lg \sigma(\lg n' + \lg \sigma)/\lg n + \sigma)$ time, where d is an arbitrary power of 2 with $2 \leq d \leq \sigma$. If index arrays are not constructed, the construction time can be lowered to $O(n' \lg^2 \sigma / \lg n + \sigma)$; this bound still applies when neither value nor index arrays are built.*

▶ **Lemma 2.** *Let $A[0..n-1]$ be a sequence drawn from alphabet $[\sigma]$. A d-ary wavelet tree over A with value and index arrays can be built in $O(n \lg \sigma / \lg d)$ time where $2 \leq d \leq \sigma$.*

A sequence $A[0..n-1]$ drawn from $[\sigma]$ can be viewed as a point set $N = \{(A[i], i) | 0 \leq i \leq n-1\}$. Let T be a d-ary wavelet tree constructed over A. Then *ball inheritance* [11] can be defined over T which asks for the support of these operations: i) `point`(v, i), which returns the point $(A(v)[i], I(v)[i])$ in N for an arbitrary node v in T and an integer i; and ii) `noderange`(c, d, v), which, given a range $[c, d]$ and a node v of T, finds the range $[c_v, d_v]$ such that $I(v)[i] \in [c, d]$ iff $i \in [c_v, d_v]$. If we store the value and index arrays explicitly, it is trivial to support these operations, but the space cost is high. To save space, we only store $S(v)$ for each node v and design auxiliary structures. The following lemma presents previous results:

▶ **Lemma 3** ([11, Theorem 2.1], [10, Lemma 2.3])**.** *A generalized wavelet tree over a sequence $A[0..n-1]$ drawn from $[\sigma]$ can be augmented with ball inheritance data structure in $O(n \lg n f(\sigma))$ bits to support* `point` *in $O(g(\sigma))$ time and* `noderange` *in $O(g(\sigma) + \lg \lg n)$ time, where (a) $f(\sigma) = O(1)$ and $g(\sigma) = O(\lg^\epsilon \sigma)$; (b) $f(\sigma) = O(\lg \lg \sigma)$ and $g(\sigma) = O(\lg \lg \sigma)$; or (c) $f(\sigma) = O(\lg^\epsilon \sigma)$ and $g(\sigma) = O(1)$.*

Data Structures for `rank` and `select`. Given a sequence A drawn from alphabet $[\sigma]$, a $\texttt{rank}_c(A, i)$ operation computes the number of elements equal to c in $A[0..i]$, where $c \in [\sigma]$, while a $\texttt{select}_c(A, i)$ returns the index of the entry of A containing the i-th occurrence of c. We have the following two lemmas on building `rank`/`select` structures.

▶ **Lemma 4.** *Let $A[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$, where $n' \leq n$ and $\sigma = O(\text{polylog}\, n)$. A data structure of $n'\lceil \lg \sigma \rceil + o(n' \lg \sigma)$ bits supporting* `rank` *in $O(1)$ time can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time.*

▶ **Lemma 5** ([2, Lemma 2.1])**.** *Given a packed bit sequence $B[0..n-1]$, a systematic data structure occupying $o(n)$ extra bits can be constructed in $O(n/\lg n)$ time, which supports* `rank` *and* `select` *in constant time.*

In the above lemma, a data structure is *systematic* if it requires the input data to be stored verbatim along with the additional information for answering queries. A restricted version of `rank` is called *partial rank*; a partial rank operation, $\texttt{rank}'(A, i)$, computes the number of elements equal to $A[j]$ in $A[0..j]$. The following lemma presents a solution to supporting $\texttt{rank}'$, which is an easy extension of [3, Lemma 3.5].

▶ **Lemma 6.** *Given a sequence $A[0..n-1]$ drawn from alphabet $[\sigma]$, a data structure of $O(n \lg \sigma)$ bits can be constructed in $O(n + \sigma)$ time, which supports* $\texttt{rank}'$ *in constant time.*

Range Minimum/Maximum. Given a sequence A of n integers, a range minimum/maximum query $\texttt{rmq}(i, j)$/$\texttt{rMq}(i, j)$ with $i \leq j$ returns the position of a minimum/maximum element in the subsequence $A[i..j]$. Fischer and Heun [16] considered this problem:

▶ **Lemma 7** ([16])**.** *Given an array A of n integers, a data structure of $O(n)$ bits can be constructed in $O(n)$ time, which answers* `rmq`/`rMq` *in $O(1)$ time without accessing A.*

We further build an auxiliary structure upon a packed sequence A under the *indexing model*: after the the data structure is built, A itself need not be stored verbatim; to answer a query, it suffices to provide an operator that can retrieve any element in A.

▶ **Lemma 8.** *Let $A[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$, where $\sigma \leq 2^{\sqrt{\lg n}}$ and $n' \leq n$. There is a data structure using $O(n' \lg \lg n)$ extra bits constructed in $O(n' \lg \sigma / \lg n)$ time, which answers* `rmq`/`rMq` *in $O(1)$ time and $O(1)$ accesses to the elements of A. The query procedure uses a universal table of $o(n)$ bits.*

3 Fast Construction of $\texttt{rank}'$ Query Structures

In this section we focus on how to efficiently construct data structures for $\texttt{rank}'$ queries over a sequence $A[0..n'-1]$ drawn from alphabet $[\sigma]$, where $n' \leq n$ and $\sigma \leq 2^{\sqrt{\lg n}}$. This is needed to solve ball inheritance in a special case. Lemma 4 already solves this problem when $\sigma \leq \lg n$, so we assume $\lg n < \sigma \leq 2^{\sqrt{\lg n}}$ in the rest of this section.

In our solution, we conceptually divide sequence A into chunks of length σ. For simplicity, assume that n' is a multiple of σ. Let A_k denote the kth chunk, where $0 \leq k \leq n'/\sigma - 1$. For each $c \in [0, \sigma - 1]$, we define the following data structures:

- A bitvector $B_c = 1^{\texttt{rank}_c(A_0,\sigma)}01^{\texttt{rank}_c(A_1,\sigma)}0 \ldots 1^{\texttt{rank}_c(A_{n'/\sigma-1},\sigma)}0$, which encodes the number of occurrences of symbol c in each chunk in unary. B_c is represented using Lemma 5 to support `rank` and `select` in constant time.
- A sequence $P_c[0..n'/\sigma - 1]$, in which $P_c[i] = \texttt{rank}'(A_i, c)$ for each $i \in [0, n'/\sigma - 1]$, i.e., $P_c[i]$ stores the answer to a partial rank query performed locally within A_i at position c.

Note that we have one B_c for each alphabet symbol c, while we have one P_c for each relative position c in the chunks of A. We have the following lemma on supporting queries using these data structures, with a space analysis.

▶ **Lemma 9.** *The data structures in this section occupy $n' \lg \sigma + o(n' \lg \sigma)$ extra bits and support* `rank'` *in $O(1)$ time and $O(1)$ accesses to elements of A.*

Proof. In B_c, each 1 bit corresponds to an occurrence of symbol c in A, while each 0 corresponds to a chunk. Thus, these bit vectors have n' 1s and $n'/\sigma \times \sigma = n'$ 0s in total. Therefore, the lengths of all these bit vectors sum up to $2n'$. By Lemma 5, $o(n')$ bits are needed to augment them to support `rank` and `select`. As each chunk has σ elements, encoding an entry of each P_c requires $\lceil \lg \sigma \rceil$ bits. Thus $P_0, \ldots, P_{\sigma-1}$ occupy $n' \lceil \lg \sigma \rceil$ bits in total. The total space usage of all the data structures in this section is therefore $2n' + o(n') + n' \lceil \lg \sigma \rceil$ bits, which is $n' \lg \sigma + o(n' \lg \sigma)$ when $\sigma > \lg n$.

A query $\texttt{rank}'(A, j)$ can be answered as follows:

$$\texttt{rank}'(A, j) = select_0(B_c, t) - (t-1) + P_\tau[t], where\ \tau = j \bmod \sigma, t = \lfloor \frac{j}{\sigma} \rfloor, and\ c = A[j]$$

As the `select` query over B_c takes constant time, answering $\texttt{rank}'(A, j)$ requires $O(1)$ time and a single access to A. ◀

Next, we consider how to construct the sequences B_c's efficiently.

▶ **Lemma 10.** *Bitvectors $B_0, B_1, \ldots, B_{\sigma-1}$ can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time.*

Proof. We first construct a sequence $M[0..n' + n'/\sigma - 1]$ in which each element is encoded in $\lceil \lg \sigma \rceil + 1$ bits. In M, n' elements are *regular elements*, and the rest are *boundary elements* each of which is an integer whose binary expression simply consists of $\lceil \lg \sigma \rceil + 1$ 0-bits. M is divided into n'/σ chunks, and each chunks contains σ regular elements followed by a boundary element. The subsequence of the σ regular elements in the i-th chunk can be obtained by appending a 1-bit to the end of the binary expression of each element in A_k.

Next we show how to create M efficiently with the help of a universal table U. This table has an entry for each possible pair (D, t), where D is a sequence of length $b = \lfloor \frac{\lg n}{2\lceil \lg \sigma \rceil} \rfloor$ drawn from $[\sigma]$ and t is an integer in $[0, b]$. If $t = 0$, this entry stores a sequence of length b which is obtained by appending a 1-bit to the end of the binary expression of each element in D. Otherwise, this entry stores a sequence of length $b+1$ consisting of three sections: the first section is obtained by appending a 1-bit to the end of the binary expression of each of the first t elements in D, the second section is a boundary element, and the third section is obtained by appending a 1-bit to the end of the binary expression of each of the last $b - t$ elements in D. As there are at most $n^{1/2}$ possible sequences of length b drawn from σ and t has $b+1$ possible values, U has at most $n^{1/2}(b+1)$ entries. Since each entry is encoded in at most $(b+1)(\lceil \lg \sigma \rceil + 1) = O(\text{polylog}(n))$ bits, U uses $o(n)$ bits. With U, we can scan A and process b of its elements in constant time; whether or where a boundary element should be created when processing these b elements can be inferred by keeping track of the number of elements that we have scanned so far. Note that at most one boundary element will be created when reading b elements from A, as $b < \lg n < \sigma$. The time needed to create M is hence $O(n'/b) = O(n' \lg \sigma / \lg n)$.

From M we determine the content of $B_0, B_1, \ldots, B_{\sigma-1}$ by constructing a tree T over M similar to large extent to a binary wavelet tree and associating each node u of T with a sequence $M(u)$. At the root node r of T, we set $M(r) = M$, and we perform the following recursive procedure at any node u at level l of T where $l \in [0, \lceil \lg \sigma \rceil - 1]$: We create the left

child, u_0, and the right child, u_1, of u, and perform a linear scan of $M(u)$. During the scan, for each $i \in [0, |M(u) - 1|]$, if $M(u)[i]$ is a boundary element, it is appended to both $M(u_0)$ and $M(u_1)$. If $M(u)[i]$ is not a boundary element and its lth most significant bit is 0, $M(u)[i]$ is appended to $M(u_0)$. If its lth significant bit is 1, it is appended to $M(u_1)$. After generating the sequences $M(u_0)$ and $M(u_1)$, we discard the sequence $M(u)$. We finish recursion after we create $\lceil \lg \sigma \rceil$ levels, i.e., we only examine the first $\lceil \lg \sigma \rceil$ bits of each element of M to determine the tree structure. Thus, this tree has σ leaves, and the sequences associated with the leaves from left to right are named M_0, M_1, ..., $M_{\sigma-1}$. They form a partition of M.

To speed up this process, we use a universal table U'. Recall that $b = \lfloor \frac{\lg n}{2\lceil \lg \sigma \rceil} \rfloor$. U' has an entry for each possible pair (E, c), where E is a sequence of length b drawn from universe $[2\sigma]$ and c is an integer in $[0, \lceil \lg \sigma \rceil - 1]$. This entry stores a pair of packed sequences E_0 and E_1 defined as follows: E_0 or E_1 stores the boundary elements in E and the regular elements in E whose c-th most significant bit is 0 or 1, respectively. The elements in E_0 retain their relative order in E, and the same is true with E_1. As U' has $2^{b \times (\lceil \lg \sigma \rceil + 1)} \times \lceil \lg \sigma \rceil$ entries and each entry stores a pair of packed sequences occupying $O(b\lceil \lg \sigma \rceil)$ bits in total, U' uses $o(n)$ bits. By performing table lookups in U', we can process $M(u)$ in $O(|M(u)| \lg \sigma / \lg n + 1)$ time. Note that we assign n' regular and $2^l \times \frac{n'}{\sigma}$ boundary elements to the nodes at tree level l. Summing over all $O(\sigma)$ nodes of the tree, the total time required to construct this tree is $O(\sum_{l=0}^{\lceil \lg \sigma \rceil - 1}((n' + 2^l \times \frac{n'}{\sigma}) \lg \sigma / \lg n) + \sigma) = O(n' \lg^2 \sigma / \lg n + \sigma)$.

To construct bitvectors B_c for any $0 \le c \le \sigma - 1$, a crucial observation is that the i-th bit in B_c is the same as the least significant bits of the i-th elements of M_c. Thus it takes $O(|B_c|(\lg \sigma + 1)/ \lg n + 1)$ time to compute the content of B_c using bit packing. B_c can then be represented in $O(|B_c|/ \lg n + 1)$ time to support `rank` and `select` by Lemma 5. Summing over all σ bitvectors, the time required to construct B_0, B_1, ..., $B_{\sigma-1}$ from $M_0, M_1, \ldots, M_{\sigma-1}$ is $O(n' \lg \sigma / \lg n + \sigma)$.

Overall, given A, the construction time of these bit vectors is

$$O(n' \lg \sigma / \lg n + (n' \lg^2 \sigma / \lg n + \sigma) + (n' \lg \sigma / \lg n + \sigma)) = O(n' \lg^2 \sigma / \lg n + \sigma). \quad \blacktriangleleft$$

It remains to show how to build all sequences $P_0, P_1, \ldots, P_{\sigma-1}$ efficiently.

▶ **Lemma 11.** *Sequences $P_0, P_1, \ldots, P_{\sigma-1}$ can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time.*

Proof. The construction consists of two phases. In the first phase, we compute the set of pairs $R_k = \{(i, \texttt{rank}'(A_k, i)) | 0 \le i \le \sigma - 1\}$ for each chunk A_k. Even though $P_i[k] = \texttt{rank}'(A_k, i)$ and thus the entries of all the P_i's have been computed in this phase, the pairs themselves generated for A_k are not in any order that allows us to directly assign values from these pairs to entries of P_i's quickly enough. Thus, in the second phase, we reorganize all n' pairs computed from all the chunks, to construct $P_0, P_1, \ldots, P_{\sigma-1}$ efficiently.

We first show how to compute the pair set R_k for each A_k efficiently. Let $I[0, \sigma - 1]$ denote a packed sequence such that $I[i] = i$ for each $i \in [0, \sigma - 1]$. Note that I can be constructed once in $O(\sigma)$ time and shared with all chunks. By Lemma 1, a binary wavelet tree, in which node u is associated with $A(u)$ and $I(u)$ as defined before, over A_k could be constructed in $O(\sigma \lg^2 \sigma / \lg n + \sigma)$ time. However, the second term $O(\sigma)$, when summed over all n'/σ chunks, is too expensive to afford. Thus, we modify the structure of a wavelet tree to decrease this term. In the modified tree, when a node v satisfies $|A(v)| \le b = \lfloor \frac{\lg n}{2\lceil \lg \sigma \rceil} \rfloor$, we make v a leaf node without any descendants. With this modification, we observe the following two properties. First, if a leaf node l satisfies $|A(l)| > b$, then the tree level of l must be $\lg \sigma$ and all entries of $A(l)$ store the same symbol. Second, as there are at most $\lceil \sigma / b \rceil$ nodes at each level, the modified tree has $O(\sigma / b \times \lg \sigma) = O(\sigma \lg^2 \sigma / \lg n)$ nodes. The

$O(\sigma)$ term in construction time in Lemma 1 follows from the fact that a wavelet tree has $O(\sigma)$ leaves. With fewer leaves, the modified tree can be constructed in $O(\sigma \lg^2 \sigma / \lg n)$ time. After this tree is constructed, we only keep the sequences $A(l)$ and $I(l)$ for each leaf node l and call them *leaf sequences*. We discard the rest of the tree.

To further compute R_k using these leaf sequences, observe that, for any symbol α, there exists one leaf l such that $A(l)$ contains all the occurrences of α in A. Thus $(I(l)[i], \texttt{rank}'(A_k, I(l)[i])) = (I(l)[i], \texttt{rank}'(A(l), i))$ holds, which we can use to reduce the problem of computing the pairs in R_k to the problem of computing the answer to a partial rank query at each position of $A(l)$ for each leaf l. Hence for each leaf l, we define a packed sequence $Q(l)[0..|A(l)|-1]$ in which $Q(l)[i] = \texttt{rank}'(A(l), i)$ to store these answers. To construct $Q(l)$ efficiently, we consider two cases. When $|A(l)| \leq b$, we apply a universal table U'' to generate $Q(l)$ in constant time. U'' has an entry for each possible pair (F, x), where F is a sequence of length b drawn from universe $[\sigma]$, and x is an integer in $[0, b]$. This entry stores a packed sequence $G[0..x]$ in which $G[i] = \texttt{rank}'(F, i)$. Similar to U in the proof of Lemma 10, U'' uses $o(n)$ bits. When $|A(l)| > b$, all entries of $A(l)$ store the same symbol. Thus, we have $Q(l)[i] = i$ for each $i \in [0, |A(l)|-1]$, and hence we can create $Q(l)$ by copying the first $|A(l)|$ elements from the sequence I which we created before. In either case, $Q(l)$ can be constructed in $O(|A(l)| \lg \sigma / \lg n + 1)$ time. Let l_i denote the $(i+1)$-st leaf visited in a preorder traversal of the tree, and f the number of leaves. Since $\sum_{i=0}^{f} |Q(l_i)| = \sigma$ and $f = O(\sigma \lg^2 \sigma / \lg n)$, the total time required to build $Q(l_0), Q(l_1), \ldots, Q(l_{f-1})$ is $O(\sigma \lg^2 \sigma / \lg n)$. Then we construct the concatenated packed sequence $I_k = I(l_0)I(l_1)\ldots I(l_{f-1})$ and $Q_k = Q(l_0)Q(l_1)\ldots Q(l_{f-1})$. It requires $O(\sigma \lg^2 \sigma / \lg n)$ time to concatenate these sequences if we process $\Theta(\lg n)$ bits, i.e., $O(1)$ words, in constant time by performing bit operations. Since for any $i \in [0, \sigma-1]$, $(I_k[i], Q_k[i])$ is a distinct pair in R_k, I_k and Q_k store all the pairs in R_k. We perform the steps in this and the previous paragraphs for all the chunks in A, and the total time spent in this phase is $O(n' \lg^2 \sigma / \lg n + \sigma)$.

Next we construct $P_0, P_1, \ldots, P_{\sigma-1}$ efficiently using the pairs computed in the previous phase. We first build in $O(n' \lg^2 \sigma / \lg n)$ time two concatenated packed sequences each of length n': $I' = I_0 I_1 \ldots I_{n'/\sigma-1}$ and $Q = Q_0 Q_1 \ldots Q_{n'/\sigma-1}$. Then we construct a binary wavelet tree over I'. Each node, u, of the wavelet tree is associated with two sequences, $I'(v)$ which contains all the elements of I' whose values are within the range represented by v, retaining their relative order in I', and $Q(v)$ in which $Q(v)[i]$ is the element in Q corresponding to $I'(v)[i]$. The wavelet tree construction algorithm of Lemma 1 can be modified easily to construct this wavelet tree in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time. Let l'_i denote the $(i+1)$st leaf of this wavelet tree in preorder. Observe that all the entries in $I'(l'_i)$ store i, and $I'(l'_i)[j]$ initially came from A_j, i.e., $I'(l'_i)[j]$ corresponds to the ith position in chunk A_j. Therefore, $Q(l'_i)[j] = P_i[j]$, and we have $P_i = Q(l'_i)$. The processing time required for this phase is also $O(n' \lg^2 \sigma / \lg n + \sigma)$, which is the same as the bound for the first phase. Therefore, the total time required to construct all sequences $P_0, P_1, \ldots, P_{\sigma-1}$ is $O(n' \lg^2 \sigma / \lg n + \sigma)$. ◀

Combining Lemmas 4, 9, 10 and 11, we have the following result:

▶ **Lemma 12.** *Let $A[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$, where $n' \leq n$ and $\sigma = O(2^{O(\sqrt{\lg n})})$. With the help of a universal table of $o(n)$ bits, a data structure using $n' \lceil \lg \sigma \rceil + o(n' \lg \sigma)$ extra bits can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time to support* $\texttt{rank}'$ *queries in $O(1)$ time and $O(1)$ accesses to elements of A.*

4 Fast Construction of Data Structures for Ball Inheritance

We now solve, with fast preprocessing, ball inheritance for the special cases needed later to match the time and space bounds in parts (b) and (c) of Lemma 3. The omitted proofs are deferred to the full version of this paper. One strategy is to construct the solution of Chan et al. [11] by replacing some of their components with those we designed with faster preprocessing. This yields:

▶ **Lemma 13.** *Let $X[0, n-1]$ be a sequence drawn from alphabet $[\sigma]$ denoting the point set $N = \{(X[i], i) | 0 \le i \le n-1\}$, where $2^{\sqrt{\lg n}} \le \sigma \le n$. A $2^{\sqrt{\lg n}}$-ary wavelet tree over X occupying $O(n \lg \sigma \cdot f(\sigma) + n \lg n)$ bits can be constructed in $O(n \lg \sigma / \sqrt{\lg n})$ time to support* `point` *in $O(g(\sigma))$ time and* `noderange` *in $O(\lg \lg n + g(\sigma))$ time, where (a) $f(\sigma) = O(\lg(\lg \sigma / \sqrt{\lg n}))$ and $g(\sigma) = O(\lg(\lg \sigma / \sqrt{\lg n}))$; or (b) $f(\sigma) = O(\lg^{\epsilon} \sigma)$ and $g(\sigma) = O(1)$ for any constant $\epsilon > 0$. The* `noderange` *query requires a universal table of $o(n)$ bits.*

▶ **Lemma 14.** *Let $X[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$ and $Y[0..n'-1]$ be a packed sequence in which $Y[i] = i$ for each $i \in [0..n'-1]$, where $\sigma = O(2^{O(\sqrt{\lg n})})$ and $n' = O(\sigma^{O(1)})$. Given X and Y as input, a d-ary wavelet tree over X using $O(n' \lg \sigma \lg(\lg \sigma / \lg d) + \sigma w)$ bits of space can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma \log_d \sigma)$ time to support* `point` *in $O(\lg(\lg \sigma / \lg d))$ time and* `noderange` *in $O(\lg \lg \sigma)$ time, where d is a power of 2 upper bounded by $min(\sigma, 2^{\sqrt{\lg n}})$.*

This strategy however cannot achieve, with the preprocessing time as in Lemma 14, part (c) of Lemma 3 when the coordinates of points can be encoded in $O(\sqrt{\lg n})$ bits. For this special case, we twist the approach of Chan et al.: they only store point coordinates explicitly at the leaf level of the wavelet tree, while we take advantage of the smaller grid size to store coordinates at more levels. This allows us to build `rank'` structures at fewer levels of the tree, decreasing the preprocessing time. The details are as follows.

Recall that, when used to represent the given point set N, each node u of the d-ary wavelet tree T is conceptually associated with an ordered list, $N(u)$, of points whose x-coordinates are within the range represented by u, and these points are ordered by y-coordinate. Assume for simplicity that σ is a power of d, and that both $1/\epsilon$ and $\tau = \log_d^{\epsilon} \sigma$ are integers. We assign a color to each level of T: Level 0 is assigned color 0, while any other Level l is assigned color $\max\{c \mid \tau^c \text{ divides } l \text{ and } 0 \le c \le 1/\epsilon - 1\}$. For each node u of T at a level assigned with color $1/\epsilon - 1$, we store the coordinates of the points in $N(u)$ explicitly. For any other node v (let l be the level l of v and c the color assigned to level l), we do not store $N(v)$. Instead, for each $i \in [0, |N(v)|]$, we store a *skipping pointer* $Sp(v)[i]$, which stores, at the closest level l' satisfying $l' > l$ and l' is a multiple of τ^{c+1}, the descendant of v at level l' containing point $N(v)[i]$ in its ordered list of points. This descendant is encoded by its rank among all the descendants of v at level l' in left-to-right order. We use Lemma 12 to support $O(1)$-time `rank'` over $Sp(v)$. Then, since both $N(u)$ and $N(Sp(u)[i])$ order points by y-coordinate, a `rank'`$(Sp(u), i)$ query gives the position of the point $N(u)[i]$ in $N(Sp(u)[i])$. Thus, to compute `point`(v, i), we follow skip pointers starting from v by performing `rank'`, until we reach a level with color $1/\epsilon - 1$, where we retrieve coordinates. With this we have:

▶ **Lemma 15.** *Let $X[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$ and $Y[0..n'-1]$ be a packed sequence in which $Y[i] = i$ for each $i \in [0..n'-1]$, where $\sigma = O(2^{O(\sqrt{\lg n})})$ and $n' = O(\sigma^{O(1)})$. Given X and Y as input, a d-ary wavelet tree over X using $O(n' \lg \sigma \log_d^{\epsilon} \sigma + \sigma w)$ bits for any positive constant ϵ can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma \log_d \sigma)$ time to support* `point` *in $O(1)$ time and* `noderange` *in $O(\lg \lg \sigma)$ time, where d is a power of 2 upper bounded by $min(\sigma, 2^{\sqrt{\lg n}})$. The* `noderange` *query requires a universal table of $o(n)$ bits.*

5 Optimal Orthogonal Range Reporting with Fast Preprocessing

We now design data structures that support orthogonal range reporting in optimal time and can be constructed fast. Previously, with a solution to ball inheritance, Chan et al. [11] was able to design a relatively simple approach achieving three current best tradeoffs for orthogonal range reporting. However, we have only designed alternative solutions to ball inheritance with fast construction time in special cases. Therefore, we design a different data structure with optimal query time for orthogonal range reporting. The strategy is to use a generalized wavelet tree and our solution to range minimum/maximum (Lemma 8) to reduce the orthogonal range reporting problem in the general case to the special case in which the points are from a $2^{\sqrt{\lg n}} \times n'$ (narrow) grid. In this reduction, we need only support ball-inheritance over a wavelet tree with high fanout which is solved by part (b) of Lemma 13. We further reduce the range reporting problem over points in a narrow grid to this problem over a (small) grid of size at most $2^{\sqrt{\lg n}} \times 2^{2\sqrt{\lg n}}$, to which we can apply Lemma 15 for ball inheritance. Hence we describe our solutions over a small, narrow and general grid in this order, as the solution to the next case uses that to the previous.

5.1 Orthogonal Range Reporting in a Small Grid

▶ **Lemma 16.** *Let N be a set of δ points with distinct y-coordinates in a $2^{\sqrt{\lg n}} \times \delta$ grid where $\delta \leq 2^{2\sqrt{\lg n}}$. Given packed sequences X and Y respectively encoding the x- and y-coordinates of these points where $Y[i] = i$ for any $i \in [0, \delta - 1]$, a data structure occupying $O(\delta \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})$ bits can be constructed in $O(\delta + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ time to support orthogonal range reporting over N in $O(\lg \lg n + \mathtt{occ})$ time, where ϵ is an arbitrary positive constant and $\mathtt{occ}$ is the number of reported points.*

Proof. We build a binary wavelet tree T over X augmented with support for ball inheritance. By Lemma 15, T occupies $O(\delta \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})$ bits and can be built in $O(\delta + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ time. It also supports `point` in $O(1)$ time and `noderange` in $O(\lg \lg n)$ time. For any internal node v of T, its value array $A(v)$ is built at some point when augmenting T to solve ball inheritance, though $A(v)$ may be discarded eventually. When $A(v)$ was available, we build a data structure $M(v)$ to support range minimum and maximum queries over $A(v)$ using Lemma 8. As T has $\lceil \sqrt{\lg n} \rceil$ non-leaf levels and the total length of the value arrays of the nodes at each tree level is δ, over all internal nodes, these structures use $O(\delta \sqrt{\lg n} \lg \lg n)$ bits in total and the overall construction time is $\sum_v O(|A(v)|/\sqrt{\lg n} + 1) = O(\delta + 2^{\sqrt{\lg n}})$. These costs are subsumed in the storage and construction costs of T. Recall that $A(v)$ stores the x-coordinates of the set, $N(v)$, of points from N whose x-coordinates are within the range represented by v, and the entries of $A(v)$ are ordered by the corresponding y-coordinates of these points. Thus any entry of $A(v)$ can be retrieved by `point` in constant time. Therefore, even after $A(v)$ is discarded, $M(v)$ can still support `rmq`/`rMq` over $A(v)$ in $O(1)$ time.

Given a query range $Q = [a, b] \times [c, d]$, we first locate the lowest common ancestor u of l_a and l_b in constant time, where l_a and l_b denote the a-th and b-th leftmost leaves of T, respectively. Let u_l and u_r denote the left and right children of u, respectively, $[c_l, d_l] = \mathtt{noderange}(c, d, u_l)$ and $[c_r, d_r] = \mathtt{noderange}(c, d, u_r)$. Then $Q \cap N = (([a, +\infty) \times [c_l, d_l]) \cap N(u_l)) \cup (([0, b] \times [c_r, d_r]) \cap N(u_r))$. In this way, we reduce a 2-d 4-sided range reporting in N to 2-d 3-sided range reporting in $N(u_l)$ and $N(u_r)$. To report points in $([a, +\infty) \times [c_l, d_l]) \cap N(u_l)$, we need only report the points in $N(u_l)[c_l, d_l]$ whose x-coordinates are at least a. This can be done by performing range maximum queries over $A(u_l)$ recursively

as follows. We perform $\texttt{rMq}(c_l, d_l)$ to get the index m of the point p that has the maximum x-coordinate in $N(u_l)[c_l, d_l]$, and retrieve its coordinates $(p.x, p.y)$ by $\texttt{point}(u_l, m)$. If $p.x \geq a$, we report p and perform the same process recursively in $N(u_l)[c_l, m-1]$ and $N(u_l)[m+1, d_l]$. Otherwise we stop. The points in $([0, b] \times [c_r, d_r]) \cap N(u_r)]$ can be reported in a similar way. To analyze the query time, observe that we perform `noderange` twice in $O(\lg \lg n)$ time. The recursive procedure is called $O(\texttt{occ})$ times, and each time it is performed, it uses $O(1)$ time. All other steps require $O(1)$ time. Therefore, the overall query time is $O(\lg \lg n + \texttt{occ})$. ◀

5.2 Orthogonal Range Reporting in a Narrow Grid

Our solution for points in a $2^{\sqrt{\lg n}} \times n'$ grid for any $n' \leq n$ uses the following previous result:

▶ **Lemma 17** ([11, Section 2], [6, Lemma 5]). *Given a set, N, of n points in $[u] \times [u]$, a data structure of $O(n \lg^{1+\epsilon} n)$ bits can be constructed in $O(n \lg n)$ time, which supports orthogonal range reporting over N in $O(\lg \lg u + \texttt{occ})$ time, where $\texttt{occ}$ is the number of reported points.*

The following lemma presents our solution for a narrow grid:

▶ **Lemma 18.** *Let N be a set of n' points with distinct y-coordinates in a $2^{\sqrt{\lg n}} \times n'$ grid where $n' \leq n$. Given packed sequences X and Y respectively encoding the x- and y-coordinates of these points where $Y[i] = i$ for any $i \in [0, n'-1]$, a data structure occupying $O(n' \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}} + n'w/2^{\sqrt{\lg n}})$ bits can be constructed in $O(n' + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ time to support orthogonal range reporting over N in $O(\lg \lg n + \texttt{occ})$ time, where ϵ is an arbitrary positive constant and $\texttt{occ}$ is the number of reported points.*

Proof. Let $b = 2^{2\sqrt{\lg n}}$. We need only consider the case in which $n' > b$ as Lemma 16 applies otherwise. Assume for simplicity that n' is divisible by b. We divide N into n'/b subsets, and for each $i \in [0, n'/b - 1]$, the ith subset, N_i, contains points in N whose y-coordinates are in $[ib, (i+1)b - 1]$. Let p be a point in N_i. We call its coordinates $(p.x, p.y)$ *global coordinates*, while $(p.x', p.y') = (p.x, p.y \bmod b)$ its *local coordinates* in N_i; the conversion between global and local coordinates can be done in constant time. Hence the points in N_i with their local coordinates can be viewed as a point set in a $2^{\sqrt{\lg n}} \times 2^{2\sqrt{\lg n}}$ grid, and we apply Lemma 16 to construct an orthogonal range search structure over N_i.

We also define a point set $\hat{N}$ in a $2^{\sqrt{\lg n}} \times n'/b$ grid. For each set N_i where $i \in [0, n'/b - 1]$ and each $j \in [0, 2^{\sqrt{\lg n}} - 1]$, we store a point (j, i) in $\hat{N}$ iff there exists at least one point in N_i whose x-coordinate is j. Thus the number of points in $\hat{N}$ is at most $n'/b \times 2^{\sqrt{\lg n}} = n'/2^{\sqrt{\lg n}}$. We apply Lemma 17 to construct an orthogonal range search structure over $\hat{N}$. In addition, for each $i \in [0, n'/b - 1]$ and $j \in [0, 2^{\sqrt{\lg n}} - 1]$, we store a list $P_{i,j}$ storing the local y-coordinates of the points in N_i whose x-coordinates are equal to j.

Given a query range $Q = [x_1, x_2] \times [y_1, y_2]$, we first check if $\lfloor y_1/b \rfloor$ is equal to $\lfloor y_2/b \rfloor$. If it is, then the points in the answer to the query reside in the same subset $N_{\lfloor y_1/b \rfloor}$, and we can retrieve these points by performing an orthogonal range query in $N_{\lfloor y_1/b \rfloor}$, which requires $O(\lg \lg n + \texttt{occ})$ time by Lemma 16. Otherwise, we decompose Q into three subranges $Q_1 = [x_1, x_2] \times [y_1, b(\lfloor y_1/b \rfloor + 1) - 1]$, $Q_2 = [x_1, x_2] \times [b(\lfloor y_1/b \rfloor + 1), b\lfloor y_2/b \rfloor - 1]$ and $Q_3 = [x_1, x_2] \times [b\lfloor y_2/b \rfloor, y_2]$. The points in $N \cap Q_1$ and $N \cap Q_3$ are in $N_{\lfloor y_1/b \rfloor}$ and $N_{\lfloor y_2/b \rfloor}$, respectively, and by Lemma 16, they can be reported in $O(\lg \lg n + \texttt{occ}_1)$ and $O(\lg \lg n + \texttt{occ}_3)$ time, respectively, where $\texttt{occ}_1 = |N \cap Q_1|$ and $\texttt{occ}_3 = |N \cap Q_3|$. The points in $N \cap Q_2$ are in $N_{\lfloor y_1/b \rfloor + 1}, N_{\lfloor y_1/b \rfloor + 2}, \ldots, N_{\lfloor y_2/b \rfloor - 1}$. To retrieve them, we first perform an orthogonal range query in $\hat{N}$ with query range $\hat{Q} = [x_1, x_2] \times [\lfloor y_1/b \rfloor + 1, \lfloor y_2/b \rfloor - 1]$. Let (x, y) be a point

in $\hat{N} \cap \hat{Q}$. The existence of this point means that is at least one point in $N_y \cap Q_2$ whose x-coordinates are equal to x; the local y-coordinates of these points are stored in $P_{y,x}$ which we retrieve and convert to global coordinates. After examining all the points in $\hat{N} \cap \hat{Q}$ and retrieving their corresponding points in $N \cap Q_2$ in this way, we have computed all the points in $N \cap Q_2$ in $O(\lg\lg n + \mathtt{occ}_2)$ time where $\mathtt{occ}_2 = |N \cap Q_2|$. The overall query processing time is thus $O(\lg\lg n + \mathtt{occ})$.

To bound the storage costs, by Lemma 16, the orthogonal range reporting structure over each N_i uses $O(2^{2\sqrt{\lg n}} \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})$ bits. Thus, the range reporting structures over $N_0, N_1, \ldots, N_{n/b-1}$ occupy $O((n'/b) \times (2^{2\sqrt{\lg n}} \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})) = O(n' \lg^{1/2+\epsilon} n + n'w/2^{\sqrt{\lg n}})$. As there are at most $n'/2^{\sqrt{\lg n}}$ points in $\hat{N}$, by Lemma 17, the range reporting structure for $\hat{N}$ occupies $O(n' \lg^{1+\epsilon} n/2^{\sqrt{\lg n}}) = o(n')$ bits. There are n' points in all $P_{i,j}$'s and each of their local y-coordinates can be encoded in $\lg b = 2\sqrt{\lg n}$ bits. In addition, each $P_{i,j}$ requires a pointer to encode its memory location, so $n'/b \times 2^{\sqrt{\lg n}} = n'/2^{\sqrt{\lg n}}$ pointers are needed. Therefore, the total storage cost of all $P_{i,j}$'s is $O(n'w/2^{\sqrt{\lg n}} + n'\sqrt{\lg n})$. Thus the space costs of all structures add up to $O(n' \lg^{1/2+\epsilon} n + n'w/2^{\sqrt{\lg n}})$ bits. Note that the above analysis assumes $n' > b$. Otherwise, $O(n' \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})$ bits are needed, so we use $O(n' \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}} + n'w/2^{\sqrt{\lg n}})$ as the space bound on both cases.

Regarding construction time, when $n' > b$, observe that the point sets $N_0, N_1, \ldots, N_{n'/b-1}$ and $\hat{N}$, as well as the sequences $P[i,j]$ for $i = 0, 1, \ldots, n'/b-1$ and $j = 0, 1, \ldots, 2^{\sqrt{\lg n}} - 1$, can be computed in $O(n')$ time. By Lemma 17, The range reporting structure for $\hat{N}$ can be built in $O(n'/b \times \lg n) = o(n')$ time. Finally, the total construction time of the range reporting structures for $N_0, N_1, \ldots, N_{n/b-1}$ is $O(n'/2^{2\sqrt{\lg n}} \times (2^{2\sqrt{\lg n}} + \sqrt{\lg n} \times 2^{\sqrt{\lg n}})) = O(n')$, which dominates the total preprocessing time of all our data structures. When $n' \leq b$, the construction time is $O(n' + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ by Lemma 16, so we use $O(n' + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ as the upper bound on construction time in both cases. ◀

5.3 Orthogonal Range Reporting in an $n \times n$ Grid

We first describe a solution that is slight more general, which requires the grid to be of size $\sigma \times n$ with $2^{\sqrt{\lg n}} \leq \sigma \leq n$, as it will be needed for some applications to be described later.

▶ **Lemma 19.** *Given a sequence $X[0, n-1]$ drawn from alphabet $[\sigma]$ denoting the point set $N = \{(X[i], i) | 0 \leq i \leq n-1\}$, a data structure of $O(n \lg^{1+\epsilon} \sigma + n \lg n)$ bits for any constant $\epsilon > 0$ can be constructed in $O(n \lg \sigma / \sqrt{\lg n})$ time to support orthogonal range reporting over N in $O(\lg\lg n + \mathtt{occ})$ time, where $2^{\sqrt{\lg n}} \leq \sigma \leq n$ and $\mathtt{occ}$ is the number of reported points.*

Proof. We build a $2^{\sqrt{\lg n}}$-ary wavelet tree T upon $X[0, n-1]$ with support for ball inheritance using part (b) of Lemma 13. As in the proof of Lemma 16, for each internal node $v \in T$, we build a data structure $M(v)$ to support range minimum and maximum queries over its value array $A(v)$ in constant time using Lemma 8, even $A(v)$ is not be explicitly stored. Recall that $A(v)$ stores the x-coordinates of the ordered list, $N(v)$, of points from N whose x-coordinates are within the range represented by v, and these points are ordered by y-coordinate. Furthermore, v is associated with another sequence $S(v)$ drawn from alphabet $[2^{\sqrt{\lg n}}]$, in which $S(v)[i]$ encodes the rank of the child of v that contains $N(v)[i]$ in its ordered list. Let $\hat{S}(v)$ denote the point set $\{(S(v)[i], i) | 0 \leq i \leq |S(v)| - 1\}$, and we use Lemma 18 to build a structure supporting orthogonal range reporting over $\hat{S}(v)$.

Given a query range $Q = [a, b] \times [c, d]$, we first locate the lowest common ancestor u of l_a and l_b in constant time, where l_a and l_b denote the a-th and b-th leftmost leaves of T, respectively. Let u_i denote the ith child of u, for any $i \in [0, 2^{\sqrt{\lg n}} - 1]$. We first locate two children, $u_{a'}$ and $u_{b'}$, of u that are ancestors of l_a and l_b, respectively. They can be found in constant time by simple arithmetic as each child of u represents a range of equal size. Then the answer, $Q \cap N$, to the query can be partitioned into three point sets $A_1 = Q \cap N(v_{a'})$, $A_2 = Q \cap (N(v_{a'+1}) \cup N(v_{a'+2}) \cup \ldots N(v_{b'-1}))$ and $A_3 = Q \cap N(v_{b'})$. With $O(\lg \lg n)$-time support for `noderange` and constant-time support for `point` and `rmq`/`rMq`, we can use the algorithm in the proof of Lemma 16 to perform 3-sided range queries over $N(v'_a)$ and $N(v'_b)$ to compute $A_1 \cup A_3$ in $O(\lg \lg n + |A_1| + |A_3|)$ time. To compute A_2, observe that any entry, $\hat{S}(v)[i]$, can be obtained by replacing the x-coordinate of point $N(v)[i]$ with the rank of the child whose ordered list contains $N(v)[i]$. Hence, by performing range reporting over $\hat{S}$ to compute $S \cap ([a' + 1, b' - 1] \times [c_v, d_v])$, where $[c_v, d_v] =$ `noderange`(c, d, v), we can find the set of points in $\hat{S}(v)$ corresponding to the points in A_2. For each point returned, we use `point` to find its original coordinates in N and return it as part of A_2. This process uses $O(\lg \lg n + |A_2|)$ time. Hence we can compute $Q \cap N$ as $A_1 \cup A_2 \cup A_3$ in $O(\lg \lg n +$ `occ`$)$ time.

Now we analyze the space costs. T with support for ball inheritance uses $O(n \lg^{1+\epsilon} \sigma + n \lg n)$ bits for any positive ϵ. For each internal node v, since $w = \Theta(\lg n)$, the data structure for range reporting over $\hat{S}$ uses $O(|S(u)| \lg^{1/2+\epsilon'} n + 2^{\sqrt{\lg n}} \lg n + |S(u)| \lg n / 2^{\sqrt{\lg n}})$ bits for any positive ϵ'. This subsumes the cost of storing $M(u)$ which is $O(|S(u)| \lg \lg n)$ bits. As T has $O(\sigma / 2^{\sqrt{\lg n}})$ internal nodes, the total cost of storing these structures at all internal nodes is $\sum_u O(|S(u)| \lg^{1/2+\epsilon'} n + 2^{\sqrt{\lg n}} \lg n + |S(u)| \lg n / 2^{\sqrt{\lg n}}) = O(n \lg \sigma / \sqrt{\lg n} \times \lg^{1/2+\epsilon'} n + \sigma \lg n) = O(n \lg \sigma \lg^{\epsilon'} n + \sigma \lg n)$. As $\lg n \leq \lg^2 \sigma$ and $\sigma \leq n$, this is bounded by $O(n \lg^{1+2\epsilon'} \sigma)$. Setting $\epsilon' = \epsilon/2$, the space bound turns to be $O(n \lg^{1+\epsilon} \sigma)$ bits. Overall, the data structures occupy $O(n \lg^{1+\epsilon} \sigma + n \lg n)$ bits.

Finally, we analyze the construction time. As shown in Lemma 13, T with support for ball inheritance can be constructed in $O(n \lg \sigma / \sqrt{\lg n})$ time. For each internal node u of T, constructing $M(u)$ and the range reporting structure over $\hat{S}(v)$ requires $O(|S(u)| + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ time. As T has $O(\sigma / 2^{\sqrt{\lg n}})$ internal nodes, these structures over all internal nodes can be built in $\sum_u O(|S(u)| + \sqrt{\lg n} \times 2^{\sqrt{\lg n}}) = O(n \lg \sigma / \sqrt{\lg n} + \sigma \sqrt{\lg n}) = O(n \lg \sigma / \sqrt{\lg n})$ as $\sigma \leq n$. The preprocessing time of all data structures is hence $O(n \lg \sigma / \sqrt{\lg n})$. ◀

Our result on points over an $n \times n$ gird immediately follows.

▶ **Theorem 20.** *Given a set, N, of n points in rank space, a data structure of $O(n \lg^{1+\epsilon} n)$ bits for any constant $\epsilon > 0$ can be constructed in $O(n\sqrt{\lg n})$ time to support orthogonal range reporting in $O(\lg \lg n +$* `occ`$)$ *time, where* `occ` *is the number of reported points.*

6 Optimal Orthogonal Range Successor with Fast Preprocessing

In this section, we assume that a range successor query asks for the lowest point in the query rectangle. The following theorem presents our result on fast construction of structures for optimal range successor; we provide a proof sketch, while leaving the full proof to the full version of this paper:

▶ **Theorem 21.** *Given n points in rank space, a data structure of $O(n \lg \lg n)$ words can be constructed in $O(n\sqrt{\lg n})$ time to support orthogonal range successor in $O(\lg \lg n)$ time.*

Proof (sketch). Our approach is similar to that in Section 5, but more levels of reductions are required. Let the sequence $X[0, n-1]$ denote the point set $N = \{(X[i], i) | 0 \leq i \leq n-1\}$. We build a $2^{\sqrt{\lg n}}$-ary wavelet tree T upon $X[0, n-1]$ with support for ball inheritance using part (a) of Lemma 13. As shown in the proof of Lemma 19, a query can be answered by locating the lowest common ancestor, u, of the two leaves corresponding to the end points of the query x-range, and then performing two 3-sided queries over the point sets represented by two children of u and one 4-sided query over $S(u)$. For the 3-sided queries, Zhou [32] already designed an indexing structure, which, with our $O(\lg\lg n)$-time support for `point` and `noderange`, can answer a 3-sided query in $O(\lg\lg n)$ time. The construction time is linear, but it is fine since T has only $O(\sqrt{\lg n})$ levels. The 4-side query over $S(u)$ is a range successor query over n' points in a $2^{\sqrt{\lg n}} \times n'$ (medium narrow) grid for any $n' \leq n$.

For such a medium narrow grid, we use the sampling strategy in Lemma 18 to reduce the problem to range successor over a set of n' points in a $2^{\sqrt{\lg n}} \times n'$ grid where $n' \leq 2 \times 2^{2\sqrt{\lg n}} - 1$. The sampling is adjusted, as we need select at most $2^{\sqrt{\lg n}}$ sampled points from each subset. The grid size of $2^{\sqrt{\lg n}} \times n'$ with $n' \leq 2 \times 2^{2\sqrt{\lg n}} - 1$ is the same as that in Lemma 16, so one may be tempted to apply the same strategy of building a binary wavelet tree to reduce it to the problem of building index structures for 3-sided queries. However, we found that, to construct the structure of Zhou [32] over n' points whose coordinates are encoded in $O(\sqrt{\lg n})$ bits, $O(n' \lg\lg n/\sqrt{\lg n})$ time is required, which is a factor of $\lg\lg n$ more than the preprocessing time of the `rmq` structure needed in the proof of Lemma 16. This factor comes from rank reduction in [32], which requires us to sort packed sequences. To overcome this additional cost, we build a $\lg^{1/4} n$-ary wavelet tree over the x-coordinates, whose number of levels is a factor of $O(\lg\lg n)$ less than that of a binary wavelet tree. As discussed for the general case, this strategy reduces the current problem to orthogonal range successor over n' points in an $\lg^{1/4} n \times n'$ (small narrow) grid with $n' \leq n$.

For a small narrow grid, there are two cases. If $n' > \lg n$, we build a binary wavelet tree of height $O(\lg\lg n)$. In the query algorithm, after finding the lowest common ancestor of the two leaves corresponding to the end points of the query x-range, we do not perform 3-sided queries. Instead, we traverse the two paths leading to these two leaves. This requires us to traverse down $O(\lg\lg n)$ levels, and at each level, we perform certain `rank`/`select` operations in constant time, with the right auxiliary structures at each node. No extra support for ball inheritance is needed as we can simply go down the tree level by level to map information. Finally, if $n' < \lg n$, we use sampling to reduce it to even smaller grids of size at most $\lg^{1/4} n \times \lg^{3/4} n$, over which a query can be answered using a table lookup. ◀

7 Applications

We now apply our range search structures to the text indexing problem, in which we preprocess a text string $T \in [\sigma]^n$, where $\sigma \leq n$. Given a pattern string $P[0..p-1]$, a *counting query* computes the number of occurrences of P in T and a *listing query* reports these occurrences.

Text indexing and searching in sublinear time. When both T and P are given in packed form, a text index of Munro et al. [25] occupies $O(n \lg \sigma)$ bits, can be built in $O(n \lg \sigma/\sqrt{\lg n})$ time and supports counting queries in $O(p/\log_\sigma n + \lg n \log_\sigma n)$ time (there are other tradeoffs, but this is their main result). Thus for small alphabet size which is common in practice, they achieve both $o(n)$ construction time and $o(p)$ query time, while previous results achieve at most one of these bounds. To support listing queries, however, they need to increase space cost to $O(n \lg \sigma \lg^\epsilon n)$ bits and construction time to $O(n \lg \sigma \lg^\epsilon n)$, and then a listing query

can be answered in $O(p/\log_\sigma n + \log_\sigma n \lg\lg n + \texttt{occ})$. The increase in storage and construction costs stems from one component they used which is an orthogonal range reporting structure over $t = O(n/r)$ points in a $\sigma^{O(r)} \times t$ grid, for $r = c\log_\sigma n$ for any constant $c < 1/4$. We can apply Lemma 19 over this point set to decrease the construction time of their index for listing queries to match that for counting queries:

▶ **Theorem 22.** *Given a packed text string T of length n over an alphabet of size σ, an index of $O(n \lg\sigma \lg^\epsilon n)$ bits can be built in $O(n \lg\sigma/\sqrt{\lg n})$ time for any positive constant ϵ. Given a packed pattern string P of length p, this index supports listing queries in $O(p/\log_\sigma n + \log_\sigma n \lg\lg n + \texttt{occ})$ time where* occ *is the number of occurrences of P in T.*

Position-restricted substring search. In a position-restricted substring search [23], we are given both a pattern P and two indices $0 \le l \le r \le n-1$, and we report all occurrences of P in $T[l..r]$. Makinen and Navarro [23] solves this problem using an index for the original text indexing problem and a two-dimensional orthogonal range reporting structure. Different text indexes and range reporting structures yield different tradeoffs. The tradeoff with the fastest query time supports position-restricted substring search in $O(p + \lg\lg n + \texttt{occ})$ time, where occ is the output size, and it uses $O(n \lg^{1+\epsilon} n)$ bits and can be constructed in $O(n \lg n)$ time. Again, the construction time of the range reporting structure is the bottleneck, which can be improved by Theorem 20. We can also use a new text index by Bille et al. [7] to achieve speedup when P is given as a packed sequence. We have:

▶ **Theorem 23.** *Given a text T of length n over an alphabet of size σ, an index of $O(n \lg^{1+\epsilon} n)$ bits can be built in $O(n\sqrt{\lg n})$ time for any constant $0 < \epsilon < 1/2$. Given a packed pattern string P of length p, this index supports position-restricted substring search in $O(p/\log_\sigma n + \lg p + \lg\lg\sigma + \texttt{occ})$ time, where* occ *in the size of the output.*

References

1. Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000*, pages 198–207. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892088.
2. Maxim Babenko, Paweł Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 572–591. Society for Industrial and Applied Mathematics, 2015.
3. Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Transactions on Algorithms (TALG)*, 16(2):1–54, 2020.
4. Djamal Belazzougui and Simon J Puglisi. Range predecessor and lempel-ziv parsing. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2053–2071. Society for Industrial and Applied Mathematics, 2016.
5. Michael A Bender and Martın Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321(1):5–12, 2004.
6. Philip Bille and Inge Li Gørtz. Substring range reporting. *Algorithmica*, 69(2):384–396, 2014. doi:10.1007/s00453-012-9733-4.
7. Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, pages 6:1–6:11, 2017. doi:10.4230/LIPIcs.CPM.2017.6.
8. Prosenjit Bose, Meng He, Anil Maheshwari, and Pat Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In *11th International Symposium on Algorithms and Data Structures*, volume 5664 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2009.

9 Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. In *22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1145, 2011. doi:10.1137/1.9781611973082.85.

10 Timothy M Chan, Meng He, J Ian Munro, and Gelin Zhou. Succinct indices for path minimum, with applications. *Algorithmica*, 78(2):453–491, 2017.

11 Timothy M Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the ram, revisited. In *27th Symposium on Computational Geometry*, pages 1–10. ACM, 2011.

12 Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 161–173, 2010. doi:10.1137/1.9781611973075.15.

13 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988. doi:10.1137/0217026.

14 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, M. Sohel Rahman, German Tischler, and Tomasz Walen. Improved algorithms for the range next value problem and applications. *Theoretical Computer Science*, 434:23–34, 2012. doi:10.1016/j.tcs.2012.02.015.

15 Maxime Crochemore, Marcin Kubica, Tomasz Walen, Costas S. Iliopoulos, and M. Sohel Rahman. Finding patterns in given intervals. *Fundamenta Informaticae*, 101(3):173–186, 2010. doi:10.3233/FI-2010-283.

16 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011.

17 Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *15th International Symposium on Algorithms and Computation*, volume 3341 of *Lecture Notes in Computer Science*, pages 558–568. Springer, 2004.

18 Jesper Jansson, Zhaoxian Li, and Wing-Kin Sung. On finding the adams consensus tree. *Information and Computation*, 256:334–347, 2017. doi:10.1016/j.ic.2017.08.002.

19 Marek Karpinski and Yakov Nekrich. Space efficient multi-dimensional range reporting. In *15th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 215–224, 2009. doi:10.1007/978-3-642-02882-3_22.

20 Orgad Keller, Tsvi Kopelowitz, and Moshe Lewenstein. Range non-overlapping indexing and successive list indexing. In *10th Workshop on Algorithms and Data Structures, Proceedings*, volume 4619 of *Lecture Notes in Computer Science*, pages 625–636. Springer, 2007. doi:10.1007/978-3-540-73951-7_54.

21 Hans-Peter Lenhof and Michiel H. M. Smid. Using persistent data structures for adding range restrictions to searching problems. *Informatique Theorique et Applications*, 28(1):25–49, 1994. doi:10.1051/ita/1994280100251.

22 Moshe Lewenstein. Orthogonal range searching for text indexing. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 267–302, 2013. doi:10.1007/978-3-642-40273-9_18.

23 Veli Mäkinen and Gonzalo Navarro. Position-restricted substring searching. In *7th Latin American Symposium on Theoretical Informatics*, pages 703–714. Springer, 2006.

24 Veli Mäkinen and Gonzalo Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387(3):332–347, 2007. doi:10.1016/j.tcs.2007.07.013.

25 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Text indexing and searching in sublinear time. In *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, pages 24:1–24:15, 2020. doi:10.4230/LIPIcs.CPM.2020.24.

26 J Ian Munro, Yakov Nekrich, and Jeffrey S Vitter. Fast construction of wavelet trees. *Theoretical Computer Science*, 638:91–97, 2016.

27 Yakov Nekrich. A data structure for multi-dimensional range reporting. In *23rd ACM Symposium on Computational Geometry (SoCG)*, pages 344–353, 2007. doi:10.1145/1247069.1247130.

28 Yakov Nekrich and Gonzalo Navarro. Sorted range reporting. In *13th Scandinavian Symposium and Workshops, 2012. Proceedings*, pages 271–282, 2012. doi:10.1007/978-3-642-31155-0_24.

29 Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *38th Annual ACM Symposium on Theory of Computing, 2006*, pages 232–240. ACM, 2006. doi:10.1145/1132516.1132551.

30 Dan E. Willard. On the application of sheared retrieval to orthogonal range queries. In *2nd Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry (SoCG) 1986*, pages 80–89. ACM, 1986. doi:10.1145/10515.10524.

31 Chih-Chiang Yu, Wing-Kai Hon, and Biing-Feng Wang. Improved data structures for the orthogonal range successor problem. *Computational Geometry*, 44(3):148–159, 2011. doi:10.1016/j.comgeo.2010.09.001.

32 Gelin Zhou. Two-dimensional range successor in optimal time and almost linear space. *Information Processing Letters*, 116(2):171–174, 2016. doi:10.1016/j.ipl.2015.09.002.

Dual Half-Integrality for Uncrossable Cut Cover and Its Application to Maximum Half-Integral Flow

Naveen Garg
Indian Institute of Technology Delhi, India
naveen@cse.iitd.ac.in

Nikhil Kumar
Indian Institute of Technology Delhi, India
nikhil@cse.iitd.ac.in

Abstract

Given an edge weighted graph and a forest F, the *2-edge connectivity augmentation problem* is to pick a minimum weighted set of edges, E', such that every connected component of $E' \cup F$ is 2-edge connected. Williamson et al. gave a 2-approximation algorithm (WGMV) for this problem using the primal-dual schema. We show that when edge weights are integral, the WGMV procedure can be modified to obtain a half-integral dual. The 2-edge connectivity augmentation problem has an interesting connection to routing flow in graphs where the union of supply and demand is planar. The half-integrality of the dual leads to a tight 2-approximate max-half-integral-flow min-multicut theorem.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Combinatorial Optimization, Multicommodity Flow, Network Design

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.55

Funding *Naveen Garg*: Janaki and K.A. Iyer Chair Professor

1 Introduction

Let $G = (V, E)$ be an undirected graph with integer edge costs $c : E \to \mathbb{Z}^+$ and let $f : 2^V \to \mathbb{Z}^+$ be a requirement function on sets of vertices. We wish to find a set of edges, E' of minimum total cost such that for every set S the number of edges in E' across S is at least the requirement of S, ie. $f(S)$. This problem captures many scenarios in network design and has been the subject of much investigation. The Steiner forest problem, minimum weight maximum matching and other problems can be modeled by requirement functions which are *proper* and 0-1 (see Definition 3) and for such functions Agrawal, Klein, Ravi [1] and Goemans, Williamson [5] gave a primal-dual algorithm that is a 2-approximation. The key idea of primal-dual algorithms is to use complementary slackness to guide the construction of the dual and primal solutions which are within a factor 2 of each other.

To use this approach for the Steiner network design problem where the requirements of sets are not just 0-1, Williamson et al. [8] extend the primal dual algorithm of GW to the setting of 0-1 *uncrossable* requirement functions (see Definition 5); we call this the WGMV algorithm. The idea was to augment the connectivity of the solution in rounds with each round augmenting the requirements of unsatisfied sets by 1. The WGMV algorithm for uncrossable functions also builds a dual solution and while the primal solution constructed is integral, nothing is known of the integrality of the dual solution. In particular while for proper functions it is possible to argue that the dual solution constructed by the GW procedure is half-integral the same is not true for the WGMV procedure for uncrossable functions as is illustrated by the example in Section 4.1.

For weakly supermodular requirement functions (see Definition 4) Jain [6] gave a 2-approximation algorithm based on iterative rounding. Although this algorithm does not build a dual solution, the iterative rounding technique saw a lot of interesting applications and quickly became an integral part of tool-kit of approximation algorithms. This together with the fact that the dual solution constructed by the WGMV procedure seems useful only for certifying the approximation guarantee of the procedure, implied that there were no further results on the nature and properties of the dual solution.

In [2] the authors show that the problem of finding maximum multiflow when the union of the supply and demand edges forms a planar graph can be reduced to the problem of finding a large dual solution for a suitable cut-covering problem with uncrossable requirement function. In addition, a primal solution would correspond to a multicut and a half-integral dual solution would correspond to a half-integral multiflow. Therefore, a primal solution which is within twice a half-integral dual solution would imply a 2-approximate max-half-integral-multiflow min-multicut theorem for such graph classes. In [2] the authors also show instances where max-half-integral-multiflow min-multicut gap can be arbitrarily close to 2, implying that our result is best possible.

In this paper we show that a suitable modification to the WGMV procedure does indeed lead to a half-integral dual solution of value at least half the primal solution.

▶ **Theorem 1.** *Let $G = (V, E)$ be an undirected graph with edge costs $c : E \to \mathbb{Z}^+$ and a uncrossable requirement function $f : 2^V \to \{0, 1\}$. One can find a subset of edges F and an assignment, y, of non-negative half-integral dual variables to sets such that for all edges $e \in E$, $\sum_{S:e\in\delta(S)} y_S \leq c_e$ and $\sum_{e\in F} c_e \leq 2\sum_S f(s)y_S$.*

To achieve this we need to build an alternate stronger analysis of the 2-approximation of the WGMV algorithm and these are the main results of this paper. In Section 3 we argue that the Goemans-Williamson algorithm for proper functions leads to half-integral duals. To prove the above, we come up with a notion of parity of a node with respect to the current dual solution. The crux of our argument is to show that all nodes in an active set have the same parity. We then employ the idea of ensuring that all nodes in an active set have the same parity to modify the WGMV procedure in Section 6. However our procedure for ensuring uniform parity entails reducing some edge costs by 1/2. Since this decrease in edge costs also needs to be bounded by the dual solution we need a stronger guarantee on the total degree of the active sets in each iteration of the WGMV procedure. We develop this alternate analysis in Section 5. Finally, Section 7 shows how maximum flow in Seymour graphs corresponds to building the dual solution for a suitable uncrossable cut cover problem and lets us claim the following result which is also best possible.

▶ **Theorem 2.** *Let $G + H$ be planar. There exists a feasible half-integral flow of value F and a multicut of value C such that $C \leq 2F$. Further, such a flow and cut can be computed in polynomial time.*

2 Preliminaries

Given a graph $G = (V, E)$ with edge costs $c : E \to \mathbb{R}^+$ and a 0-1 requirement function $f : 2^V \to \{0, 1\}$ we are interested in picking a subset of edges E' of minimum total cost such that every set with requirement 1 has at least one edge of E' across it. In other words, for all $S \subseteq V$, $|\delta_{E'}(S)| \geq f(S)$, where $|\delta_{E'}(S)|$ is the number of edges in E' which have exactly one endpoint in S.

▶ **Definition 3.** *A function $f : 2^V \to \{0,1\}$ is called **proper** if $f(V) = 0, f(S) = f(V - S)$ for all $S \subseteq V$ and for any disjoint $A, B \subseteq V$, $f(A \cup B) \leq \max\{f(A), f(B)\}$.*

▶ **Definition 4.** *A function $f : 2^V \to \{0,1\}$ is called **weakly supermodular** if $f(V) = f(\phi) = 0$ and for any $A, B \subseteq V$, $f(A)+f(B) \leq \max\{f(A\cap B)+f(A\cup B), f(A\setminus B)+f(B\setminus A)\}$.*

▶ **Definition 5.** *A function $f : 2^V \to \{0,1\}$ is called **uncrossable** if $f(V) = f(\phi) = 0$ and for any $A, B \subseteq V$, if $f(A) = f(B) = 1$, then either $f(A \cap B) = f(A \cup B) = 1$ or $f(A \setminus B) = f(B \setminus A) = 1$.*

It is easy to argue that every proper function is also weakly supermodular and every weakly supermodular function is also uncrossable. In this paper we will only be interested in uncrossable requirement functions and shall refer to the problem in this setting as the *uncrossable cut cover problem* (UCC). The following integer program for UCC is well known.

$$\begin{array}{rcll} \text{minimize} & & \sum_{e\in E} c_e x_e & \\ \text{subject to} & & & \\ \sum_{e\in\delta(S)} x_e & \geq & f(S) & S \subseteq V \\ x_e & \in & \{0,1\} & e \in E \end{array}$$

We can relax the integrality constraint on x_e to $0 \leq x_e \leq 1$ to get a linear programming relaxation of the above. The dual program of the relaxation can be given as:

$$\begin{array}{rcll} \text{maximize} & & \sum_{S\subseteq V} f(S)y(S) & \\ \text{subject to} & & & \\ \sum_{S:e\in\delta(S)} y_S & \leq & c_e & e \in E \\ y_S & \geq & 0 & S \subseteq V \end{array}$$

Williamson et al. [8] gave a primal-dual 2-approximation algorithm for the above integer program for uncrossable f.

3 Half-integrality of the GW-dual for proper functions

We first argue that the Goemans-Williamson (GW) algorithm - for the case when requirement functions are proper and edge costs are integral - constructs a half-integral dual whose value is at least half the primal integral solution.

The GW algorithm proceeds by raising dual variables corresponding to sets of vertices and picking edges which are tight into the current solution. An edge e is tight when the sum of dual variables of sets containing exactly one end-point of e equals $c(e)$. The algorithm raises dual of all minimal sets S such that $f(S) = 1$ but no edge going across S has been picked in the current solution. We imagine growing the duals in a continuous manner and define a notion of time: $t = 0$ at start of the algorithm and y_S increases by δ during $[t, t + \delta]$ if S is a minimally unsatisfied set at every point of time in $[t, t + \delta]$. If f is proper, these minimal sets correspond exactly to the connected components formed by the set of tight edges. Let C be a connected component at time t. If $f(C) = 1$ then C is active while C is inactive if $f(C) = 0$. In each iteration, the GW procedure raises dual variables of all active sets simultaneously till an edge goes tight. At this point the connected components are recomputed and the algorithm continues with the next iteration unless all sets are inactive. Let F be the set of tight edges picked after the first phase. In a second phase, called the *reverse delete*, the GW algorithm considers the edges of F in the reverse order in which they were added to F. If the removal of an edge from F does not violate the requirement function of any set then the edge is removed.

We shall only be concerned with the first phase of the GW algorithm since it is in this phase that the dual variables, $y : 2^V \to \mathbb{R}_{\geq 0}$ are set. Let $\mathcal{S} = \{S : y_S > 0\}$ and note that this family of sets is laminar. For $v \in S, S \in \mathcal{S}$, we define the parity of v with respect to S as $\pi_v(S) = \left\{\sum_{T:v\in T\subseteq S} y_T\right\}$, where $\{x\}$ denotes the fractional part of x. If S is active at time t then there exists a vertex $v \in S$ which for all times in $[0, t]$ was in an active component; we call such a vertex an *active vertex* of set S.

We now argue that the GW procedure ensures that for all $S \in \mathcal{S}$, for all $u, v \in S$, $\pi_u(S) = \pi_v(S)$. We call this quantity the parity of set S, $\pi(S)$, and show that $\pi(S) \in \{0, 1/2\}$. Let S be formed by the merging of sets S_1, S_2 at time t. We induct on the iterations of the GW procedure and assume that all vertices in S_1 (respectively S_2) have the same parity with respect to S_1 (respectively S_2). If S_1 is active at time t then $\pi(S_1) = \pi_v(S_1) = \{t\}$ where v is an active vertex of set S_1. Similarly if S_2 is active at time t then $\pi(S_2) = \{t\}$. Thus if both S_1, S_2 are active at time t then $\pi(S_1) = \pi(S_2)$ and hence all vertices of S have the same parity with respect to S. Let $e = (u, v), u \in S_1, v \in S_2$ be the edge which gets tight when S_1, S_2 merge at time t. Since $l(e)$ is integral and $\sum_{T:u\in T\subseteq S_1} y_T + \sum_{T:v\in T\subseteq S_2} y_T = l(e)$, we have that $\pi(S_1) = \pi(S_2) \in \{0, 1/2\}$.

Suppose only S_1 is active at time t. By our induction hypothesis $\pi(S_2) \in \{0, 1/2\}$. Once again, since $l(e)$ is integral and $\sum_{T:u\in T\subseteq S_1} y_T + \sum_{T:v\in T\subseteq S_2} y_T = l(e)$, we have that $\pi(S_1) = \pi(S_2)$ which implies that all vertices of S have the same parity with respect to S.

Since $\pi(S), \pi(S_1) \in \{0, 1/2\}$, it must be the case that $\{y_S\} \in \{0, 1/2\}$. Since this is true for all sets $S \in \mathcal{S}$ this implies that the duals constructed by the GW procedure are half-integral.

4 The WGMV algorithm

We now give a brief description of the algorithm in [8]. Given an undirected graph $G = (V, E)$ with edge costs $c_e \geq 0$ and a uncrossable function f we wish to find a set of edges $F' \subseteq E$ such that for any $S \subseteq V, |F' \cap \delta(S)| \geq f(S)$. A set S is said to be unsatisfied if $f(S) = 1$ but no edge crosses S in the current solution.

The algorithm works in iterations. At the beginning of every iteration the algorithm computes a collection of minimally unsatisfied sets. Williamson et al.[8] show that minimally unsatisfied sets are disjoint and can be found in polynomial time (follows easily from uncrossability). Raise the dual variables corresponding to all minimally unsatisfied sets simultaneously until some edge is tight (the total dual across it equals its cost). All edges that go tight are added to a set T. The edges of T are considered in an arbitrary order and $e \in T$ is added to F if it crosses a minimally unsatisfied set. Note that whenever an edge is added to F the collection of minimally unsatisfied sets is recomputed. The *growth phase* of the WGMV procedure stops when all sets are satisfied; let F be the set of edges picked in this phase.

The edges of F are considered in the reverse order in which they were picked. An edge $e \in F$ is dropped from the solution if its removal keeps the current solution feasible.

At the end of the procedure, we have a set of edges F and a feasible dual solution y_S such that $\sum_{e\in F} c_e x_e \leq 2\sum_S f(S)y_S$. By weak duality, $\sum_{e\in F} c_e x_e \geq \sum_S f(S)y_S$ and this shows that the cost of solution picked by the algorithm is at most twice the optimal.

4.1 Duals constructed by WGMV are not half-integral

In the example in Figure 1, the red edges are not edges of the graph G. For a set $S \subseteq V$,$f(S) = 1$ iff there is exactly one red edge with exactly one end point in S. Thus this problem corresponds to picking edges so as to augment the red tree into a 2-edge connected

Algorithm 1 Primal-Dual Algorithm for uncrossable functions.

```
1: procedure WGMV( G = (V, E) with cost c_e, uncrossable function f)
2:     y ← 0, F ← φ
3:     while ∃S ⊆ V such that S is not satisfied do
4:         Compute C, the collection of minimally unsatisfied sets with respect to F.
5:         Increase y_C for all C ∈ C simultaneously until some edge e ∈ δ(C), C ∈ C is tight
   (c_e = Σ_{S:e∈δ(S)} y_S)
6:         Add all tight edges to T
7:         for all e ∈ T do
8:             if ∃C ∈ C, e ∈ δ(C) then
9:                 F ← F ∪ {e}; Recompute C
10:    for all e ∈ F do
11:        // Edges of F are considered in the reverse order in which they were added to F
12:        if F \ {e} is feasible then
13:            F ← F \ {e}
14:    return F
```

graph. It is known that f is uncrossable. In each iteration the WGMV procedure raises dual variables corresponding to all minimally unsatisfied sets. The edge (c, d) gets tight in the first iteration. At the end of the first iteration $y_{\{e\}} = 1/2$ and so in the second iteration $y_{\{e\}}$ increases to 3/4 and $y_{\{b,c,d\}}$ to 1/4 before edge (b, e) goes tight.

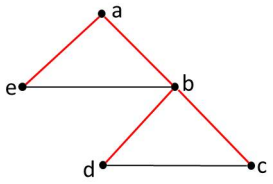


Figure 1 Example showing that the duals constructed by the WGMV procedure are not half-integral.

5 A stronger analysis of the WGMV algorithm

To analyse the algorithm, Willimason et al.[8] argue that in each iteration the total contribution of the dual variables to the primal solution is at most twice the increase in the value of the dual solution. This then, added over all iterations, implies that $\sum_{e\in F} c_e x_e \leq 2\sum_S f(S)y_S$. If in an iteration the dual values of all active sets increases by δ then the contribution of the dual variables to the primal solution equals δ times the total degree of the active sets in F. On the other hand the increase in the value of the dual solution is δ times the number of active sets and hence Williamson et al. argue that in each iteration the average degree of the active sets in F is at most 2.

Let $\mathcal{S}$ be the collection of minimally unsatisfied sets identified during a run of the algorithm. Note thst we do not claim that $y_S > 0$ for $S \in \mathcal{S}$. The uncrossability of f implies that $\mathcal{S}$ is a laminar family. Add V, the set of all vertices, to $\mathcal{S}$ and construct a tree, $\mathcal{T} = (X, Y)$, which has vertex set $X = \{v_S | S \in \mathcal{S}\}$. v_A is the parent of v_B iff A is the minimal set in $\mathcal{S}$ containing B.

Each set $S \in \mathcal{S}$ is labelled with the number of the iteration in which S became satisfied; let $l : \mathcal{S} \to [T]$ be this function. Let $\mathcal{S}^i$ be the sets with label at least i; these are the minimally unsatisfied sets encountered in iterations i or later. Similarly, each edge $e \in F$ is labeled with the number of the iteration in which it became tight. We overload notation and let $l : F \to [T]$ also denote this function. Let $F^i \subseteq F$ be edges with label at least i. We note a few properties of these labels.

1. if $B \subset A$ then $l(B) \leq l(A)$.
2. if $e \in \delta_F(S)$ then $l(e) \geq l(S)$

Let $v_{B_1}, v_{B_2}, \ldots v_{B_p}$ be the children of node v_A in $\mathcal{T}$ (see Figure 2). We number sets so that $l(B_1) \geq l(B_2) \geq \cdots \geq l(B_p)$. Let $p_i \in [p]$ be the largest index such that $l(B_{p_i}) \geq i$. Hence all sets $B_j, j \in [p_i]$ are in $\mathcal{S}^i$. Let $X^i_A = A \setminus \cup_{j \in [p_i]} B_j$ and H^i_A be a graph whose nodes correspond to sets $X^i_A, B_1, \ldots, B_{p_i}$ and edges correspond to the edges between these sets in F. Since sets $B_j, j \in [p_i]$ have label at least i, edges in H^i_A have label at least i and hence they are in F^i.

▷ Claim 6. H^i_A is a forest.

Proof. For contradiction assume H^i_A has a cycle and consider the edge of the cycle, say $e = (u, v)$, which was added last to F. We consider two cases.

$u \in B_r$ and $v \in B_s$, $r, s \in [p_i]$. When e was picked, both B_r, B_s had another edge in F across them and were therefore satisfied. Recall that $\mathcal{S}$ is the collection of all the minimally unsatisfied sets encountered during the growth phase of the algorithm. Picking e did not lead to any unsatisfied set in $\mathcal{S}$ getting satisfied and this is a contradiction.

$u \in B_r$ and $v \in X^i_A$, $r \in [p_i]$. No subset of vertices in X^i_A is unsatisfied in the i^{th} (or any subsequent) iteration. When e was picked, B_r had another edge in F across it and was therefore satisfied. Once again picking e did not lead to any unsatisfied set in $\mathcal{S}$ getting satisfied and this is a contradiction. ◁

Since H^i_A is a forest on $p_i + 1$ vertices it contains at most p_i edges.

▶ **Definition 7.** *A set $A \in \mathcal{S}$ is* critical *in iteration i if H^i_A is a tree of which the node corresponding to X^i_A, is a leaf.*

For a set $A \in \mathcal{S}^i$, let $\alpha^i(A) = \delta_F(A) \setminus \cup_{S \subset A, S \in \mathcal{S}^i} \delta_F(S)$. Thus $\alpha^i(A)$ is the set of edges of F which have one endpoint in the set $A \setminus \cup_{S \subset A, S \in \mathcal{S}^i} S$ and the other endpoint in $V \setminus A$. Equivalently $\alpha^i(A)$ is the subset of edges in $\delta_F(A)$ which are incident on vertices in X^i_A. We note the following important property of $\alpha^i(A)$.

▷ Claim 8. Let $A \in \mathcal{S}^i$. The collection of sets $\left\{\alpha^i(S) | S \in \mathcal{S}^i, S \subseteq A\right\}$ forms a partition of the set $\delta_F(A)$.

Let $\mathcal{A}^i$ be the collection of minimally unsatisfied sets whose dual is raised in iteration i of the WGMV algorithm. These are the *active sets* in iteration i. Note that

1. $\mathcal{A}^i \subseteq \mathcal{S}^i$.
2. A set $S \in \mathcal{S}$ is contained in $\mathcal{S}^i$ if and only if there exits an $A \in \mathcal{A}^i$ such that $A \subseteq S$.
3. If $A \in \mathcal{A}^i$ then no subset of A is in $\mathcal{S}^i$ which implies $\alpha^i(A) = \delta_F(A)$.

▶ **Lemma 9.** $\sum_{S \in \mathcal{S}^i} \left|\alpha^i(S)\right| \leq 2\left|\mathcal{A}^i\right| - 2 + \left|\mathcal{R}^i\right|$ *where $\mathcal{R}^i$ is the collection of critical sets in iteration i.*

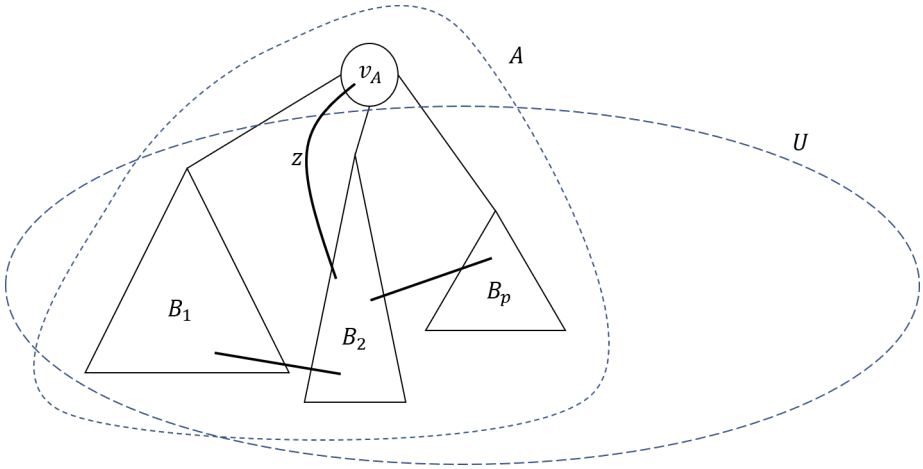


Figure 2 Illustrating the notation used. A is a critical set. The thick edges are the edges in H^i_A.

Proof. We show an argument built on redistributing tokens which help us prove the above lemma. We begin by assigning every node of tree $\mathcal{T}$ a number of tokens equal to two less than twice the number of its children in $\mathcal{S}^i$. Thus a node with 1 child in $\mathcal{S}^i$ gets no tokens. We also give every node that corresponds to a critical set in iteration i an additional token. It is easy to see that the total number of tokens distributed initially is $2\left|\mathcal{A}^i\right| - 2 + \left|\mathcal{R}^i\right|$.

v_A transfers one token to each edge in H^i_A incident on X^i_A and 2 tokens each to remaining edges in H^i_A. If v_A has p_i children in $\mathcal{S}^i$ and is critical in iteration i, it was assigned $2p_i - 1$ tokens and these are sufficient to undertake the above assignment. If v_A is not critical then it was assigned $2p_i - 2$ tokens and again this is sufficient to complete the transfer of tokens to edges in H^i_A.

For every edge in $e \in F^i$ there is a unique $A \in \mathcal{S}^i$ such that e is in H^i_A. If e has an endpoint in X^i_A it is assigned 1 token by v_A. Note that this edge contributes 1 to the sum on the left. The remaining edges of F^i are assigned 2 tokens each and this is also their contribution to the sum on the left. This establishes that the sum on the left equals the number of tokens assigned to edges which is at most the number of tokens assigned to nodes which in turn equals the quantity in the right. ◀

▶ **Lemma 10.** *If A is critical in iteration i then $\alpha^i(A) \neq \phi$.*

Proof. Since A is critical, H^i_A is a tree and X^i_A is a leaf node. Let e be the unique edge in H^i_A incident to X^i_A. Consider the step in the reverse delete phase when edge e was considered and was retained in F only because its deletion would have caused some set to become unsatisfied. Let $U \subseteq V$ be the minimal such set and note that e is the only edge in F' across U at this step.

▷ Claim 11. $\forall j \in [p_i]$, $U \cap B_j = \phi$ or $U \cap B_j = B_j$.

Proof. For a contradiction assume that for some $j \in [p_i]$, $\phi \neq U \cap B_j \subset B_j$. Since $f(B_j) = f(U) = 1$ by uncrossability either $f(B_j \cap U) = 1$ or $f(B_j \setminus U) = 1$. In either case, during the growth phase we must have added an edge, say g, to F between $B_j \setminus U$ and $B_j \cap U$ in an iteration before B_j became a minimally unsatisfied set. Thus, in the reverse delete phase when we considered e, edge g was in F and hence e was not the only edge across U. ◁

If $U \cap A$ includes some sets $B_j, j \in [p_i]$ and not the others then the number of edges across the set U will be more than 1. Thus either $\cup_{j \in p_i} B_j \subseteq A \cap U$ or $\cup_{j \in [p_i]} B_j \subseteq A \setminus U$. Since $f(A) = f(U) = 1$ by uncrossability we have either $f(A \cap U) = 1$ or $f(A \setminus U) = 1$. If $\cup_{j \in [p_i]} B_i \subseteq A \cap U$ then $f(A \setminus U) \neq 1$ as that would imply a minimal unsatisfied set in X^i_A which would be a contradiction. Similarly if $\cup_{j \in [p_i]} B_j \subseteq A \setminus U$ then $f(A \cap U) \neq 1$. Hence we need to consider only two cases

1. $\cup_{j \in [p_i]} B_j \subseteq A \cap U$ and $f(A \cap U) = f(A \cup U) = 1$: F should have an edge across the set $A \cup U$. Since the only edge across U goes to $A \setminus U$, there should be an edge across A that is incident to X^i_A.
2. $\cup_{j \in [p_i]} B_j \subseteq A \setminus U$ and $f(A \setminus U) = f(U \setminus A) = 1$: F should have an edge across the set $U \setminus A$. Since the only edge across U goes from $A \cap U$ to $A \setminus U$, there should be an edge across A that is incident to $A \cap U \subseteq X^i_A$.

Hence in both cases we conclude that there is an edge across A incident to X^i_A which implies $\alpha^i(A) \neq \phi$. ◀

▶ **Lemma 12.** *The total degree (in F) of sets in $\mathcal{A}^i$ is at most twice $\left|\mathcal{A}^i\right|$.*

Proof. A set in $\mathcal{A}^i$ cannot be critical in iteration i. Further for $S \in \mathcal{A}^i$, $\left|\alpha^i(S)\right|$ equals the degree of S in F. By Lemma 10 if A is critical in iteration i then $\alpha^i(A) \neq \phi$. Hence $\sum_{S \in \mathcal{S}^i} \left|\alpha^i(S)\right| \geq \sum_{S \in \mathcal{A}^i} |\delta_F(A)| + \left|\mathcal{R}^i\right|$ where $\mathcal{R}^i$ is the collection of critical sets. Applying Lemma 9, we obtain $\sum_{S \in \mathcal{A}^i} |\delta_F(A)| \leq 2\left|\mathcal{A}^i\right| - 2$ which proves the lemma. ◀

6 Modifying WGMV

We now modify the WGMV algorithm so that the duals obtained are half-integral while ensuring that the primal solution has cost at most twice the dual solution. In doing so we are guided by the fact that the GW algorithm constructed half-integral duals since the parity of all vertices in a set was identical. This property does not hold true for the WGMV algorithm as seen in the example in Figure 1.

As before, let $\mathcal{S}$ be the set of minimally unsatisfied sets during a run of the algorithm. Our modification to the WGMV algorithm involves reducing costs of some edges in $\delta(S), S \in \mathcal{S}$ by $1/2$. Let $\delta'(S) \subseteq \delta(S)$ denote the subset of edges whose cost was reduced by $1/2$ when considering S. We now define the *parity* of an edge e with respect to a set $S \in \mathcal{S}, e \in \delta(S)$ as

$$\pi_e(S) = \left\{ \sum_{e \in \delta(T), T \subseteq S} y_T + \tfrac{1}{2} |\{T \subseteq S | e \in \delta'(T)\}| \right\}$$

where as before $\{x\}$ denotes the fractional part of x. Our modification to the WGMV procedure is:

> Let S be a set which becomes minimally unsatisfied at time t and let $x \in S$ be an active vertex of set S. Then $\pi_x(S) = \{t\}$. For edge $e \in \delta(S)$, if $\pi_e(S) \neq \{t\}$ then decrease c_e by $1/2$ (note e gets included in $\delta'(S)$).

We decrease the costs of edges in $\delta(S)$ in the above manner only when S becomes minimally unsatisfied and need to argue that the total cost of edges in F can still be bounded by twice the sum of the dual variables. Our modification allows us the following claim.

▷ Claim 13. $\forall S \in \mathcal{S}, \forall e, f \in \delta(S), \pi_e(S) = \pi_f(S)$

When we increase dual variables of sets in $\mathcal{A}^i$ in iteration i, one or more edges go tight and these are added to a set T. Let t^i be the time at which we stop growing dual variables of sets in $\mathcal{A}^i$. The edges of T are considered in an arbitrary order and $e \in T$ is added to F if it

crosses a minimally unsatisfied set. Note that whenever an edge is added to F the collection of minimally unsatisfied sets is recomputed. Let $\mathcal{C}$ be the collection of minimally unsatisfied sets after all edges in T have been considered. For every $S \in \mathcal{C}$ and every edge $e \in \delta(S)$, if $\pi_e(S) \neq \{t\}$ then we reduce the cost of edge e by $1/2$. All edges that go tight after this step are included in T and the process repeated until no edge gets added to T. The minimally unsatisfied sets at this stage are the active sets, $\mathcal{A}^{i+1}$ for iteration $i+1$.

Algorithm 2 Modification to an iteration of the WGMV algorithm.

$\mathcal{C}$ is the collection of minimally unsatisfied sets with respect to F.
T is the set of tight edges which have not yet been included in F.
repeat
  for all $e \in T$ **do**
    if $\exists C \in \mathcal{C}, e \in \delta(C)$ **then**
      $F \leftarrow F \cup \{e\}$; compute $\mathcal{C}$
  $T \leftarrow \phi$
  for all $C \in \mathcal{C}$ **do**
    for all $e \in \delta(C)$ **do**
      if $\pi_e(C) \neq \{t\}$ **then**
        $c_e \leftarrow c_e - 1/2$
        if e is tight **then**
          $T \leftarrow T \cup \{e\}$
until $T = \phi$

Let $\mathcal{C}^i$ be the collection of sets in $\mathcal{S}$ which properly contain a set in $\mathcal{A}^i$ and are subsets of some set in $\mathcal{A}^{i+1}$. Formally, $\mathcal{C}^i = \left\{S \in \mathcal{S} | \exists A \in \mathcal{A}^i, \exists B \in \mathcal{A}^{i+1}, A \subset S \subseteq B\right\}$. Note that

1. $\mathcal{A}^{i+1} \setminus \mathcal{A}^i \subseteq \mathcal{C}^i$,
2. $\mathcal{A}^i \cap \mathcal{C}^i = \phi$,
3. any edge whose cost is reduced by $1/2$ in iteration i goes across a set in $\mathcal{C}^i$,
4. $\mathcal{C}^i \cap \mathcal{C}^{i+1} = \phi$ for $i \in [T-1]$

Before $A \in \mathcal{C}^i$ was considered in iteration i we would have considered the sets in $\mathcal{S}^i$ corresponding to children of node v_A in tree $\mathcal{T}$. Let $B_j, j \in [p_i]$ be these sets and note that they belong to $\mathcal{C}^i \cup \mathcal{A}^i$. For each $B_j, j \in [p_i]$ we would already have reduced the cost of edges $e \in \delta(B_j)$ if $\pi_e(B_j) \neq \left\{t^i\right\}$. Hence when considering A we would only be reducing the cost of edges in $\delta(A)$ which are incident to $A \setminus \cup_{j \in [p_i]} B_j = X^i_A$. Thus the edges of F whose cost is reduced when considering $A \in \mathcal{C}^i$ are subsets of $\alpha^i(A)$, let this subset be $\beta^i(A)$.

After iteration i, (reduced) cost of an edge e is $c_e - \sum_{S: e \in \delta(S)} y_S$, where y is the dual value after iteration i. Note that as the algorithm proceeds, (reduced) cost of edges decrease. To prove that the modified WGMV procedure gives a 2-approximate solution, we bound the total reduction in costs of edges in F by twice the total increase in the value of dual variables. In iteration i, the total reduction in edge costs of F due to increase of dual variables of sets A^i equals $\gamma^i \sum_{S \in \mathcal{A}^i} |\delta_F(S)| = \gamma^i \sum_{S \in \mathcal{A}^i} \left|\alpha^i(S)\right|$, where $\gamma^i = t^i - t^{i-1}$ is the increase in the dual variable of a set in $\mathcal{A}^i$. The other reduction occurs when we reduce by $1/2$ the costs of edges due to parity considerations. The total reduction in the cost of edges of F due to this reason is at most $1/2 \sum_{S \in \mathcal{C}^i} \left|\beta^i(S)\right|$.

To prove the approximation guarantee of WGMV, authors in [8] show that in every iteration the total reduction in cost of edges in F is at most twice the total increase in dual values in that iteration. To prove the approximation guarantee of modified WGMV, we need

to charge the reduction in edge costs across iterations. To do this, we introduce a procedure for marking and unmarking sets. All sets are unmarked before the first iteration of the algorithm. In the first iteration a set $A \in \mathcal{S}$ is not marked

1. if A is critical or,
2. if node v_A exhausts all its tokens and $\alpha^1(A) = \phi$

All other sets in $\mathcal{S}$ are marked in iteration 1. Let M be the number of sets which are marked.

In iteration i we unmark a set $S \in \mathcal{C}^i$ if it is critical and $\beta^i(S) \neq \phi$. Let M_i be the number of sets unmarked in iteration i. In Lemma 16 we argue that we unmark a set only if it has a mark on it.

▶ **Lemma 14.** *In any iteration $i > 1$,*

$$\gamma^i \sum_{S \in \mathcal{A}^i} \left|\alpha^i(S)\right| + (1/2) \sum_{s \in \mathcal{C}^i} \left|\beta^i(S)\right| - M_i/2 \leq 2\gamma^i(|A_i| - 1)$$

Proof. Recall $\mathcal{R}^i$ is the collection of critical sets in iteration i.

$$\gamma^i \sum_{S \in \mathcal{S}^i} \left|\alpha^i(S)\right| \geq \gamma^i \sum_{S \in \mathcal{A}^i} \left|\alpha^i(S)\right| + \gamma^i \sum_{S \in \mathcal{C}^i} \left|\alpha^i(S)\right| + \gamma^i \sum_{S \in \mathcal{S}^i \setminus \mathcal{A}^i \cup \mathcal{C}^i} \left|\alpha^i(S)\right| \tag{1}$$

By Lemma 10 we obtain

$$\gamma^i \sum_{S \in \mathcal{S}^i \setminus \mathcal{A}^i \cup \mathcal{C}^i} \left|\alpha^i(S)\right| \geq \gamma^i \left|\mathcal{R}^i \setminus \mathcal{C}^i\right| \tag{2}$$

and the unmarking procedure gives

$$\gamma^i \sum_{S \in \mathcal{C}^i} \left|\alpha^i(S)\right| + M_i/2 \geq (1/2) \sum_{S \in \mathcal{C}^i} \left|\beta^i(S)\right| + \gamma^i \left|\mathcal{R}^i \cap \mathcal{C}^i\right| \tag{3}$$

Inequality 3 holds since

1. if S is not critical it contributes $\gamma^i\left|\alpha^i(S)\right|$ to the left and $\left|\beta^i(S)\right|$ to the right and $\beta^i(S) \subseteq \alpha^i(S)$.
2. if S is critical but $\beta^i(S) = \phi$ then S contributes $\gamma^i\left|\alpha^i(S)\right|$ to the left and γ^i to the right and $\alpha^i(S) \neq \phi$.
3. if S is critical and $\beta^i(S) \neq \phi$ then S contributes $\gamma^i\left|\alpha^i(S)\right| + 1/2$ to the left and $(1/2)\left|\beta^i(S)\right| + \gamma^i$ to the right. Since $\phi \neq \beta^i(S) \subseteq \alpha^i(S)$ and $\gamma^i \geq 1/2$, the contribution to the left is more than the contribution of S to the right.

Adding inequalities 1, 2 and 3 we get

$$\gamma^i \sum_{S \in \mathcal{S}^i} \left|\alpha^i(S)\right| \geq \gamma^i \sum_{S \in \mathcal{A}^i} \left|\alpha^i(S)\right| + \gamma^i \sum_{S \in \mathcal{C}^i} \left|\beta^i(S)\right| + \gamma^i \left|\mathcal{R}^i\right| - M_i/2 \tag{4}$$

Inequality 4 when combined with the inequality in Lemma 9 and together with the fact that $\gamma^i \geq 1/2$ proves the lemma. ◀

Iteration 1 differs from other other iterations since we mark sets in this iteration. For iteration 1 we make the following claim.

▶ **Lemma 15.**

$$\gamma^1 \sum_{S \in \mathcal{A}^1} \left|\alpha^1(S)\right| + (1/2) \sum_{s \in \mathcal{C}^1} \left|\alpha^1(S)\right| + (1/2)(M - M_1) \leq 2\gamma^1(|A_1| - 1)$$

Proof. Inequalities 1 and 3 remain unchanged for iteration 1 (with 1 replacing i) while inequality 2 is modified due to the marks placed on sets. A is marked if it is not critical and $\alpha^1(A) \neq \phi$; let m be the number of such sets. This together with Lemma 10 gives

$$\gamma^1 \sum_{S \in \mathcal{S}^1 \setminus \mathcal{A}^1 \cup \mathcal{C}^1} \left|\alpha^1(S)\right| \geq \gamma^1 \left|\mathcal{R}^1 \setminus \mathcal{C}^1\right| + m/2 \tag{5}$$

Adding inequalities 1, 3 (with $i = 1$) and inequality 5 we get

$$\gamma^1 \sum_{S \in \mathcal{S}^1} \left|\alpha^1(S)\right| \geq \gamma^1 \sum_{S \in \mathcal{A}^1} \left|\alpha^1(S)\right| + \gamma^1 \sum_{S \in \mathcal{C}^1} \left|\beta^1(S)\right| + \gamma^1 \left|\mathcal{R}^1\right| + (1/2)(m - M_1) \tag{6}$$

Recall that we also mark a set A when node v_A does not exhaust all its tokens. Note that the number of such sets is $M - m$ and hence the inequality on Lemma 9 becomes

$$\sum_{S \in \mathcal{S}^1} \left|\alpha^1(S)\right| + M - m \leq 2(\left|\mathcal{A}^1\right| - 1) + \left|\mathcal{R}^1\right| \tag{7}$$

Combining inequality 6 and inequality 7 and using the fact that $\gamma^i \geq 1/2$ proves the lemma. ◀

Summing the inequality in Lemma 15 and Lemma 14 over all iterations gives us

$$\sum_{i \in [T]} \left(\gamma^i \sum_{S \in \mathcal{A}^i} \alpha^i(S) + (1/2) \sum_{s \in \mathcal{C}^i} \beta^i(S) - M_i/2 \right) + M/2 \leq \sum_{i \in [T]} 2\gamma^i(|A_i| - 1)$$

Since we unmark a set only if it has been marked in iteration 1 (Lemma 16), $M \geq \sum_{i \in [t]} M_i$. Therefore, the total reduction in the cost of the edges of F over all iterations (which is the total cost of edges in F) is at most the quantity on the left of the above inequality. Hence the cost of the solution F is at most twice the total dual raised over all iterations and this completes the proof of Theorem 1.

It remains to show that a set is unmarked only if it has been marked in iteration 1.

▶ **Lemma 16.** *If $A \in \mathcal{C}^i$ is critical in iteration i but not marked in iteration 1 then $\beta^i(A) = \phi$.*

Proof. Let $\{B_j | j \in [p]\}$ be the sets corresponding to children of v_A and $X^1_A = A \setminus \cup_{j \in [p]} B_j$. If H^1_A has a tree spanning nodes corresponding to sets X^1_A, $B_j, j \in [p]$, then edges of $\delta(A)$ would have equal parity. If A becomes a minimal unsatisfied set at time t then B_1 was active till time t. Therefore the parity of edges in $\delta(B_1)$ and hence those of all edges in $\delta(A)$ would equal $\{t\}$ which would imply $\beta^i(A) = \phi$.

Since A is unmarked either it is critical in iteration 1 or v_A exhausts all its tokens and $\alpha^1(A) = \phi$. In the former case we have a tree spanning nodes corresponding to sets X^1_A, $B_j, j \in [p]$. In the latter case if there is no such tree there would be a tree spanning nodes corresponding to sets $B_j, j \in [p]$ and no edge in $\delta_F(A)$ incident to X^1_A. Again, this implies that all edges in $\delta_F(A)$ have equal parity. ◀

7 Computing half-integral flow in Seymour graphs

In this section, we describe the connection between multicommodity flows/multicuts and connectivity augmentation problems from [2]. In particular, we will be interested in `2ECAP`, a special case of the `UCC` problem defined in [2].

▶ **Definition 17** (2-edge connectivity Augmentation Problem (2ECAP)). *Given an undirected graph (without loops but possible parallel edges) $G = (V, E \cup Y)$ and edge weights $w : E \to \mathbb{Z}_{\geq 0}$ find a minimum weight set of edges $E' \subseteq E$ such that each connected component of $(V, E' \cup Y)$ is 2-edge connected.*

For every $S \subseteq V$, let $f : S \to \{0, 1\}$ be defined as follows: $f(S) = 1$ iff exactly one edge of Y crosses the cut $(S, V \setminus S)$, otherwise it is zero. 2ECAP can be formulated equivalently as: find a minimum weight subset of edges $E' \subseteq E$ such that at least $f(S)$ edges of E' cross the cut $(S, V \setminus S)$. It is well known that f as defined above is uncrossable and hence the WGMV algorithm can be used to compute a 2-approximate solution.

Now, we define the multicommodity flow problem. Let $G = (V, E)$ be a simple undirected graph with edge capacities $c : E \to \mathbb{Z}_{\geq 0}$ (called the supply graph) and $H = (V, F)$ be a simple graph each edge of which corresponds to a commodity and the endpoints of that edge are the source/sink of that commodity (called the demand graph). Given any G and H, an instance of sum multicommodity flow asks for a feasible flow of maximum total value between the end points of demand edges. A minimum multicut is a set of edges of G of minimum total weight whose removal disconnects endpoints of all the demand edges in G. It is easy to see that the value of minimum multicut (C) is always greater than the value of the maximum flow (F). Given a class of instances, the maximum value of the ratio between C and F is known as the `flow-multicut gap` for the class. This gap is $\theta(\log k)$ for general G, H while it is $O(1)$ for planar G and arbitrary H. There is rich literature on proving `flow-multicut gaps` [3, 4, 7].

If we restrict the flow to be integral (resp. half integral), we call the flow-multicut gap as the integral (resp. half integral) `flow-multicut gap`. An instance of the multicommodity flow/multicut problem is called a Seymour instance if the union of the supply and demand graphs is planar. In [2], the authors establish a `flow-multicut gap` of at most 2 for Seymour instances by showing that the problem of computing a multicut in a Seymour instance is equivalent to solving an appropriate instance of 2ECAP in the planar dual of the supply and demand graph. Given a planar graph G, let G^* denotes its planar dual. Formally,

▶ **Lemma 18** ([2]). *C is a multicut for the instance (G, H) if and only if C^* is a feasible solution to 2ECAP for the instance $(V^*, E^* \cup F^*)$.*

The WGMV algorithm immediately gives a 2-approximation algorithm for multicuts in Seymour instances. In order to prove the `flow-multicut gap`, [2] shows that the duals constructed by the WGMV algorithm correspond to flow paths in G and that this correspondence is value preserving, ie. total flow is equal to the total value of the dual and if the duals constructed are integral (resp. half integral), then the corresponding flows are integral (resp. half integral). Formally,

▶ **Lemma 19** ([2]). *There exists a flow of value $\sum_{S \subseteq V^*} y_S$ in G.*

[2] show how to extract a half-integral flow of value at least half of any given fractional flow and an integral flow of value at least half any given half integral flow. This shows a half integral (resp. integral) flow-multicut gap of 4 (resp. 8). Using our modified WGMV algorithm, we obtain a half integral dual (and hence half integral flow) of value at least half the cost of the 2ECAP solution and hence the multicut. This gives us a 2 (resp. 4) approximate half-integral (resp. integral) `flow-multicut` theorem for Seymour instances.

▶ **Theorem 20.** *Let $G + H$ be planar. There exists a feasible integral flow of value F and a multicut of value C such that $C \leq 4F$. Further, such a flow and cut can be computed in polynomial time.*

[2] shows a class of Seymour instances such that the half-integral flow-multicut gap approaches 2 from below. This, along with our upper bound of 2 proves that Theorem 2 is tight. The best known lower bound for the integral `flow-multicut gap` is also 2 and it remains an interesting open question to determine the exact gap.

References

1. Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995. doi:10.1137/S0097539792236237.
2. Naveen Garg, Nikhil Kumar, and András Sebő. Integer plane multiflow maximisation: Flow-cut gap and one-quarter-approximation. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 144–157. Springer, 2020.
3. Naveen Garg, Vijay V Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi) cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.
4. Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
5. Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. doi:10.1137/S0097539793242618.
6. Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001. doi:10.1007/s004930170004.
7. Philip Klein, Serge A Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 682–690. ACM, 1993.
8. David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. *Combinatorica*, 15(3):435–454, 1995. doi:10.1007/BF01299747.

An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams

Martin Held
Universität Salzburg, FB Computerwissenschaften, Austria
held@cs.sbg.ac.at

Stefan de Lorenzo
Universität Salzburg, FB Computerwissenschaften, Austria
slorenzo@cs.sbg.ac.at

Abstract

We present a simple wavefront-like approach for computing multiplicatively weighted Voronoi diagrams of points and straight-line segments in the Euclidean plane. If the input sites may be assumed to be randomly weighted points then the use of a so-called overlay arrangement [Har-Peled&Raichel, Discrete Comput. Geom. 53:547–568, 2015] allows to achieve an expected runtime complexity of $\mathcal{O}(n \log^4 n)$, while still maintaining the simplicity of our approach. We implemented the full algorithm for weighted points as input sites, based on CGAL. The results of an experimental evaluation of our implementation suggest $\mathcal{O}(n \log^2 n)$ as a practical bound on the runtime. Our algorithm can be extended to handle also additive weights in addition to multiplicative weights, and it yields a truly simple $\mathcal{O}(n \log n)$ solution for solving the one-dimensional version of this problem.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Voronoi Diagram, multiplicative weight, additive weight, arc expansion, overlay arrangement, implementation, experiments, CGAL, exact arithmetic

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.56

Related Version A full version of this paper is available at [9], `https://arxiv.org/abs/2006.14298`.

Supplementary Material The source code of our implementation is available at GitHub and can be used freely under the GPL(v3) license: `https://github.com/cgalab/wevo`.

Funding Work supported by Austrian Science Fund (FWF): Grant P31013-N31.

1 Introduction

The multiplicatively weighted Voronoi diagram (MWVD) was introduced by Boots [4]. Aurenhammer and Edelsbrunner [2] present a worst-case optimal incremental algorithm for constructing the MWVD of a set of n points in $\mathcal{O}(n^2)$ time and space. They define spheres on the bisector circles (that are assumed to be situated in the xy-plane) and convert them into half-planes in $\mathbb{R}^3$ using a spherical inversion. Afterwards, these half-planes are intersected. Thus, every Voronoi region is associated with a polyhedron. Finally, the intersection of every such polyhedron with a sphere that corresponds to the xy-plane is inverted back to $\mathbb{R}^2$. We are not aware of an implementation of their algorithm, though. (And it seems difficult to implement.) In any case, the linear-time repeated searches for weighted nearest points indicate that its complexity is $\Theta(n^2)$ even if the combinatorial complexity of the resulting Voronoi diagram is $o(n^2)$. Later Aurenhammer uses divide&conquer to obtain an $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space algorithm for the one-dimensional weighted Voronoi diagram [1].

Har-Peled and Raichel [8] show that a bound of $\mathcal{O}(n \log^2 n)$ holds on the expected combinatorial complexity of a MWVD if all weights are chosen randomly. They sketch how to compute MWVDs in expected time $\mathcal{O}(n \log^3 n)$. Their approach is also difficult to implement because it uses the algorithm by Aurenhammer and Edelsbrunner [2] as a subroutine.

Vyatkina and Barequet [13] present a wavefront-based strategy to compute the MWVD of a set of n lines in the plane in $\mathcal{O}(n^2 \log n)$ time. The Voronoi nodes are computed based on edge and break-through events. An edge event takes place when an wavefront edge disappears. A break-through event happens whenever a new wavefront edge appears.

Since the pioneering work of Hoff et al. [10] it has been well known that discretized versions of Voronoi diagrams can be computed using the GPU framebuffer. More recently, Bonfiglioli et al. [3] presented a refinement of this rendering-based approach. It is obvious that their approach could also be extended to computing approximate MWVDs. However, the output of such an algorithm is just a set of discrete pixels instead of a continuous skeletal structure. Its precision is limited by the resolution of the framebuffer and by the numerical precision of the depth buffer.

2 Our Contribution

Our basic algorithm allows us to compute MWVDs in worst-case $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n^2)$ space. A refined version makes use of the result by Har-Peled and Raichel [8]: We use their overlay arrangement to keep the expected runtime complexity bounded by $\mathcal{O}(n \log^4 n)$ if the point sites are weighted randomly. Hence, for the price of a multiplicative factor of $\log n$ we get an algorithm that is easier to implement. Our experiments suggest that this bound is too pessimistic in practice and that one can expect the actual runtime to be bounded by $\mathcal{O}(n \log^2 n)$. However, our experiments also show that one may get a quadratic runtime if the weights are not chosen randomly. Our algorithm does not require the input sites to have different multiplicative weights, and it can be extended to additive weights and to (disjoint) straight-line segments as input sites. Furthermore, it yields a truly simple $\mathcal{O}(n \log n)$ solution for computing MWVDs in one dimension, where all input points lie on a line.

Our implementation is based on exact arithmetic and the Computational Geometry Algorithms Library (CGAL) [12]. It is publicly available on GitHub under `https://github.com/cgalab/wevo`. To the best of our knowledge, this is the first full implementation of an algorithm for computing MWVDs that achieves a decent expected runtime complexity.

3 Preliminaries

Let $S := \{s_1, s_2, \ldots, s_n\}$ denote a set of n distinct weighted points in $\mathbb{R}^2$ that are indexed such that $w(s_i) \leq w(s_j)$ for $1 \leq i < j \leq n$, where $w(s_i) \in \mathbb{R}^+$ is the weight associated with s_i. It is common to regard the weighted distance $d_w(p, s_i)$ from an arbitrary point p in $\mathbb{R}^2$ to s_i as the standard Euclidean distance $d(p, s_i)$ from p to s_i divided by the weight of s_i, i.e., $d_w(p, s_i) := \frac{d(p,s_i)}{w(s_i)}$. The *(weighted) Voronoi region* $\mathcal{VR}_w(s_i, S)$ of s_i relative to S is the set of all points of the plane such that no site s_j in $S \setminus \{s_i\}$ is closer to p than s_i, that is, $\mathcal{VR}_w(s_i, S) := \{p \in \mathbb{R}^2 : d_w(p, s_i) \leq d_w(p, s_j) \text{ for all } j \in \{1, 2, \ldots, n\}\}$. Then the multiplicatively weighted Voronoi diagram (MWVD), $\mathcal{VD}_w(S)$, of S is defined as $\mathcal{VD}_w(S) := \bigcup_{s_i \in S} \partial\, \mathcal{VR}_w(s_i, S)$.

A connected component of a Voronoi region is called a *face*. For two distinct sites s_i and s_j of S, the *bisector* $b_{i,j}$ of s_i and s_j models the set of points of the plane that are at the same weighted distance from s_i and s_j. Hence, a non-empty intersection of two Voronoi

regions is a subset of the bisector of the two defining sites. Following common terminology, a connected component of such a set is called a *(Voronoi) edge* of $\mathcal{VD}_w(S)$. An end-point of an edge is called a *(Voronoi) node*. It is known that the bisector between two unequally weighted sites forms a circle[1]. An example of a MWVD is shown in Figure 1.

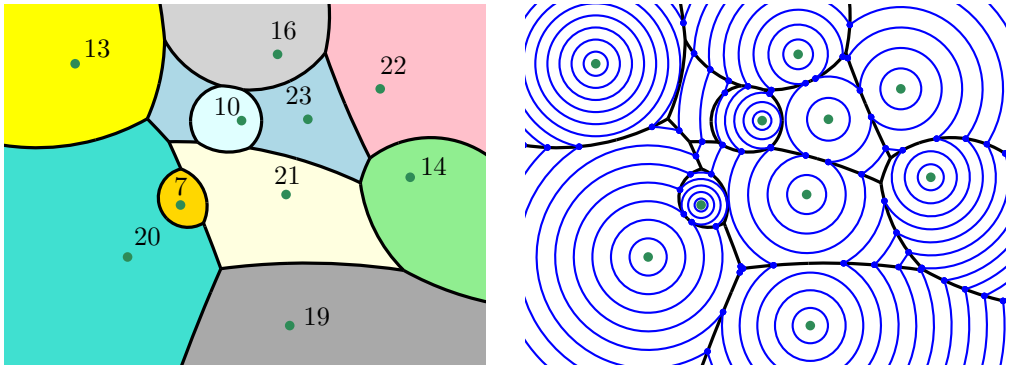


Figure 1 Left: The numbers next to the points indicate their weights and the corresponding MWVD is shown. Right: Wavefronts (in blue) for equally-spaced points in time.

The wavefront $\mathcal{WF}(S,t)$ emanated by S at time $t \geq 0$ is the set of all points p of the plane whose minimal weighted distance from S equals t. More formally,

$$\mathcal{WF}(S,t) := \left\{ p \in \mathbb{R}^2 : \min_{s_i \in S} d_w(p, s_i) = t \right\}.$$

The wavefront consists of circular arcs which we call *wavefront arcs*. A common end-point of two consecutive wavefront arcs is called *wavefront vertex*; see the blue dots in Figure 1.

4 Offset Circles

For the sake of descriptional simplicity, we start with assuming that no point in the plane has the same weighted distance to more than three sites of S. For $t \geq 0$, the *offset circle* $c_i(t)$ of the i-th site s_i is given by a circle centered at s_i with radius $t \cdot w(s_i)$. We find it convenient to regard $c_i(t)$ as a function of either time or distance since at time t every point on $c_i(t)$ is at Euclidean distance $t \cdot w(s_i)$ from s_i, i.e., at weighted distance t. We specify a point of $c_i(t)$ relative to s_i by its polar angle α and its (weighted) polar radius t and denote it by $p_i(\alpha, t)$.

For $1 \leq i < j \leq n$, consider two sites $s_i, s_j \in S$ and assume that $w(s_i) \neq w(s_j)$. Then there exists a unique closed time interval $[t_{ij}^{min}, t_{ij}^{max}]$ during which the respective offset circles of s_i, s_j intersect. We say that the two offset circles *collide* at their mutual *collision time* t_{ij}^{min}, and s_j starts to *dominate* s_i at the domination time t_{ij}^{max}. For all other times t within this interval the two offset circles $c_i(t)$ and $c_j(t)$ intersect in two disjoint points $v_{i,j}^l(t)$ and $v_{i,j}^r(t)$. These *(moving) vertices* trace out the bisector between s_i and s_j; see Figure 2. Since $v_{i,j}^l(t)$ and $v_{i,j}^r(t)$ are defined by the same pair of offset circles we refer to $v_{i,j}^l(t)$ as the *vertex married to* $v_{i,j}^r(t)$, and vice versa. Every other pair of moving vertices defined by two different pairs of intersecting offset circles is called *unmarried*. To simplify the notation, we will drop the parameter t if we do not need to refer to a specific time. Similarly, we drop the superscripts l and r if no distinction between married and unmarried vertices is necessary.

[1] Apollonius of Perga defined a circle as a set of points that have a specific distance ratio to two foci.

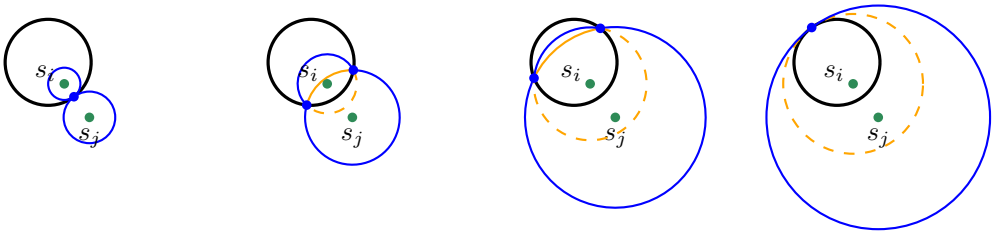


Figure 2 Two married vertices (highlighted by the blue dots) trace out the bisector b_{ij} (in black).

5 A Simple Event-Based Construction Scheme

In this section we describe a simulation of a propagation of the wavefront $\mathcal{WF}(S,t)$ to compute $\mathcal{VD}_w(S)$. Since the wavefront is given by a subset of the arcs of the arrangement of all offset circles, one could attempt to study the evolution of all arcs of that arrangement over time. However, it is sufficient to restrict our attention to a subset of arcs of that arrangement. We note that our wavefront can be seen as a kinetic data structure [7].

Clearly, the arc along $c_i(t)$ which is inside $c_j(t)$ will not belong to $\mathcal{WF}(\{s_i,s_j\},t^*)$ for any $t^* > t$. We will make use of this observation to define *inactive* and *active arcs* that are situated along the offset circles.

▶ **Definition 1** (Active point). *A point p on the offset circle $c_i(t)$ is called* inactive *at time t (relative to S) if there exists $j > i$, with $1 \le i < j \le n$, such that p lies strictly inside of $c_j(t)$. Otherwise, p is* active *(relative to S) at time t. A vertex $v_{i,j}(t)$ is an* active vertex *if it is an active point on both $c_i(t)$ and $c_j(t)$ at time t; otherwise, it is an* inactive vertex.

▶ **Lemma 2.** *If $p_i(\alpha,t)$ is inactive at time t then $p_i(\alpha,t')$ will be inactive for all times $t' \ge t$.*

An inactive point $p_i(\alpha,t)$ cannot be part of the wavefront $\mathcal{WF}(S,t)$. Lemma 2 ensures that none of its future incarnations $p_i(\alpha,t')$ can become part of the wavefront $\mathcal{WF}(S,t')$.

▶ **Definition 3** (Active arc). *For $1 \le i \le n$ and $t \ge 0$, an* active arc *of the offset circle $c_i(t)$ at time t is a maximal connected set of points on $c_i(t)$ that are active at time t. The closure of a maximal connected set of inactive points of $c_i(t)$ forms an* inactive arc *of $c_i(t)$ at time t.*

Every end-point of an active arc of $c_i(t)$ is given by the intersection of $c_i(t)$ with some other offset circle $c_j(t)$, i.e., by a moving vertex $v_{i,j}(t)$. This vertex is active, too.

▶ **Definition 4** (Arc arrangement). *The* arc arrangement (AA) *of S at time t, $\mathcal{A}(S,t)$, is the arrangement induced by all active arcs of all offset circles of S at time t.*

As time t increases, the offset circles expand. This causes the vertices of $\mathcal{A}(S,t)$ to move, but it will also result in topological changes of the arc arrangement.

▶ **Definition 5** (Collision event). *Let $p_i(\alpha,t_{ij}^{min}) = p_j(\alpha+\pi,t_{ij}^{min})$ be the point of intersection of the offset circles of s_i and s_j at the collision time t_{ij}^{min}, for some fixed angle α. A* collision event *occurs between these two offset circles at time t_{ij}^{min} if the points $p_i(\alpha,t)$ and $p_j(\alpha+\pi,t)$ have been active for all times $0 \le t \le t_{ij}^{min}$.*

At the time of a collision a new pair of married vertices $v_{i,j}^l(t)$ and $v_{i,j}^r(t)$ is created. Of course, we have $v_{i,j}^l(t_{ij}^{min}) = v_{i,j}^r(t_{ij}^{min}) = p_i(\alpha,t_{ij}^{min})$.

▶ **Definition 6** (Domination event). *Let $p_i(\alpha, t_{ij}^{max}) = p_j(\alpha, t_{ij}^{max})$ be the point of intersection of the offset circles of s_i and s_j at the domination time t_{ij}^{max}, for some fixed angle α. A* domination event *occurs between these two offset circles at time t_{ij}^{max} if the points $p_i(\alpha, t)$ and $p_j(\alpha, t)$ have been active for all times $0 \leq t \leq t_{ij}^{max}$.*

At the time of a domination event the married vertices $v_{i,j}^l(t_{ij}^{max})$ and $v_{i,j}^r(t_{ij}^{max})$ coincide and are removed.

▶ **Definition 7** (Arc event). *An* arc event *e occurs at time t_e when an active arc a_i shrinks to zero length because two unmarried vertices $v_{i,j}(t_e)$ and $v_{i,k}(t_e)$ meet in a point p_e on $c_i(t_e)$.*

Lemma 2 implies that $p_i(\alpha, t)$ has been active for all times $t \leq t_e$ if $p_i(\alpha, t_e) = p_e$. At the time of an arc event two unmarried vertices trade their places along an offset circle. Now suppose that the two unmarried vertices $v_{i,j}(t_e)$ and $v_{i,k}(t_e)$ meet in a point p_e along $c_i(t_e)$ at the time t_e of an arc event, thereby causing an active arc of $c_i(t_e)$ to shrink to zero length. Hence, the offset circles of s_i, s_j and s_k intersect at the point p_e at time t_e. If $c_j(t)$ and $c_k(t)$ did not intersect for $t < t_e$ then we also get a collision event between $c_j(t)$ and $c_k(t)$ at time t_e, see Figure 3a. (This configuration can occur for any relative order of the weights $w(s_i), w(s_j), w(s_k)$.) Otherwise, one or both of the married vertices $v_{j,k}^l(t_e)$ and $v_{j,k}^r(t_e)$ must also coincide with p_e. If both coincide with p_e then we also get a domination event between $c_j(t)$ and $c_k(t)$ at time t_e and we have $w(s_j) < w(s_k)$, see Figure 3b. The scenarios remaining for the case that only one of $v_{j,k}^l(t_e)$ and $v_{j,k}^r(t_e)$ coincides with p_e are detailed in the following lemma.

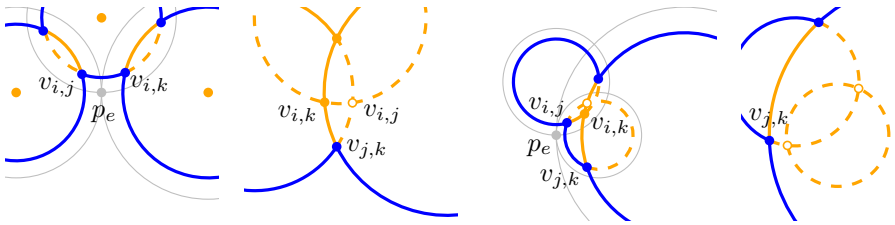
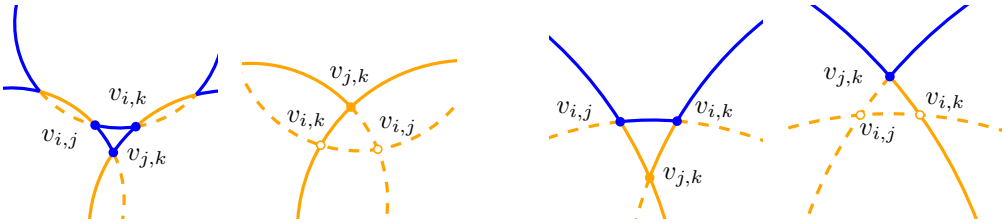


Figure 3 (a) The configuration shortly before (left) and after (right) a collision event as well as an arc event occur simultaneously at the same point p_e. In the left figure the offset arcs at the time of the event are shown in gray. Arcs and vertices that are on $\mathcal{WF}(\{s_i, s_j, s_k\}, t)$ are highlighted in blue. Other active arcs and vertices are depicted by solid orange lines and filled disks, while inactive arcs and vertices are depicted by dashed orange lines and circles. (b) The configuration shortly before and after a domination event and an arc event occur simultaneously at the same point p_e.

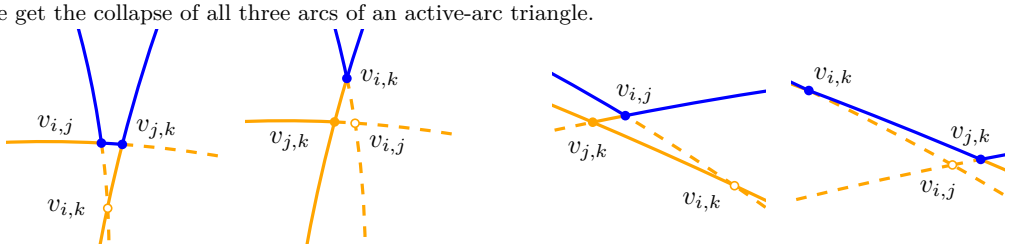
▶ **Lemma 8.** *Let $i < j < k$ and consider an arc event such that exactly the three vertices $v_{i,j}(t_e)$, $v_{i,k}(t_e)$, and $v_{j,k}(t_e)$ coincide at time t_e. Then either*

- *all three vertices were active before the event, see Figure 4a, or*
- *$v_{i,j}$ and $v_{j,k}$ were active and $v_{i,k}$ was inactive before the event, see Figure 4b, or*
- *$v_{i,k}$ and $v_{j,k}$ were active and $v_{i,j}$ was inactive before the event, see Figure 4c.*

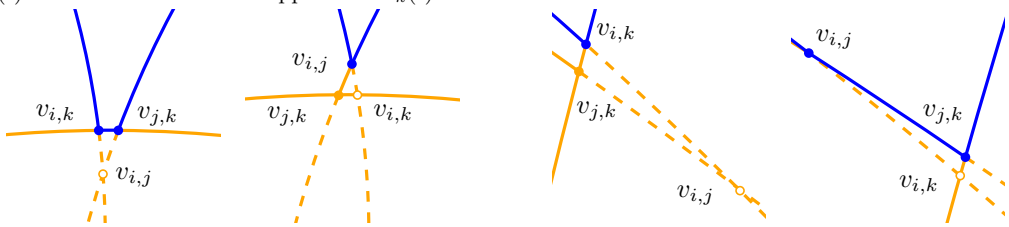
We now describe an event-handling scheme that allows us to trace out $\mathcal{VD}_w(S)$ by simulating the expansion of the arcs of $\mathcal{A}(S, t)$ as t increases, see Figure 5. We refer to this process as *arc expansion*.



(a) The two possible configurations shortly before (shown in the left figures) and after (shown in the right figures) one active arc disappears on $c_i(t)$ if no collision or domination event occurs at the same point. We get the collapse of all three arcs of an active-arc triangle.



(b) The two possible configurations shortly before (left) and after (right) one active arc disappears on $c_j(t)$ and another active arc appears on $c_k(t)$.



(c) The two possible configurations shortly before (left) and after (right) one active arc disappears on $c_k(t)$ and another active arc appears on $c_j(t)$.

Figure 4 The six different configurations that can occur for arc events for $1 \leq i < j < k \leq n$.

For each site we maintain a search data structure to keep track of all active arcs during the arc expansion. This *active offset* o_i of s_i holds the set of all arcs of $c_i(t)$ which are active at time t sorted in counter-clockwise angular order around s_i, and supports the following basic operations in time logarithmic in the number of arcs stored:

- It supports the insertion and deletion of active arcs as well as the lookup of their corresponding vertices.
- It supports point-location queries, allowing us to identify that active arc within o_i which contains a query point p on $c_i(t)$.

Every active offset contains at most $2(n-1)$ vertices and, thus, $O(n)$ active arcs. Hence, each such operation on an active offset takes $\mathcal{O}(\log n)$ time in the worst case.

Checking and handling the configurations shown in Figures 3a to 4 can be done by using only basic operations within the respective active offsets. The events themselves are stored in a priority queue $\mathcal{Q}$ ordered by the time of their occurrence. If two events take place simultaneously at the same point then collision events are prioritized higher than arc events, and arc events have to be handled before domination events. Four auxiliary operations are utilized that allow a more compact description of this process. Each one takes $\mathcal{O}(\log n)$ time.

- The *collapse-operation* takes place from $v_{i,x}$ to $v_{j,k}$ within an active offset o_x, with $x \in \{j,k\}$, in which $v_{i,x}$ and $v_{j,k}$ bound an active arc a_x that is already part of o_x; see Figure 6a. It determines the neighboring active arc a'_x of a_x that is bounded (on one side) by $v_{i,x}$, deletes a_x from o_x, and replaces $v_{i,x}$ by $v_{j,k}$ in a'_x.

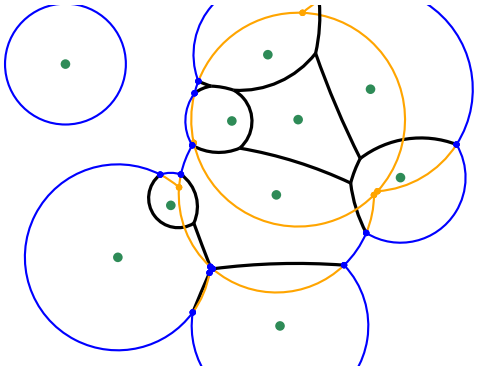


Figure 5 A snapshot of the arc expansion for the input shown in Figure 1. Active arcs that are currently not part of the wavefront are drawn in orange.

- The counterpart of the collapse-operation is the *expand-operation*; see Figure 6b. It happens from $v_{j,k}$ to $v_{i,x}$ in which $v_{j,k}$ bounds an active arc a'_x within o_x. The expansion will either move along a currently inactive or an already active portion of the offset circle of s_x. In the latter case, $v_{j,k}$ is replaced by $v_{i,x}$ in a'_x. In any case, we insert the respective active arc that is bounded by $v_{i,x}$ and $v_{j,k}$ into o_x.
- A *split-operation* involves two active offsets o_i and o_j as well as a point p_e which is situated within the active arcs $a_i := (v_{i,s}, v_{i,e})$ and $a_j := (v_{j,s'}, v_{j,e'})$ within o_i and o_j, respectively; see Figure 7a. Two married vertices $v^l_{i,j}$ and $v^r_{i,j}$ are created. Afterwards a_i and a_j are removed from o_i and o_j, respectively. Two new active arcs $(v_{i,s}, v^l_{i,j})$ and $(v^r_{i,j}, v_{i,e})$ are created and inserted into o_i. Furthermore, the three active arcs $(v_{j,e}, v^r_{i,j})$, $(v^r_{i,j}, v^l_{i,j})$, and $(v^l_{i,j}, v_{j,e})$ are inserted into o_j. If a_i and a_j were wavefront arcs then the newly created married vertices coincide with wavefront vertices and the newly inserted active arcs except $(v^r_{i,j}, v^l_{i,j})$ are marked as wavefront arcs.
- During a *merge-operation*, exactly two offset circles interact; see Figure 7b. The active arcs a_i and a_j bounded by the two corresponding married vertices $v^r_{i,j}$ and $v^l_{i,j}$ are removed from o_i and o_j, respectively. Additionally, the active arcs $(v_{j,s}, v^r_{i,j})$ and $(v^l_{i,j}, v_{j,e})$ that were adjacent to a_j within o_j are removed. Finally, a new active arc $a'_j := (v_{j,s}, v_{j,e})$ is inserted into o_j. If a_j was a wavefront arc then a'_j is also marked as a wavefront arc.

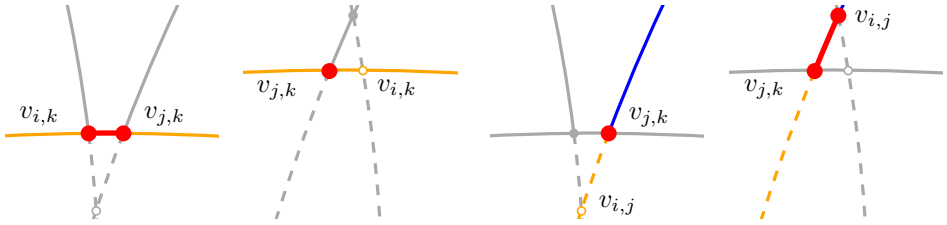


Figure 6 (a) A collapse-operation from $v_{i,k}$ to $v_{j,k}$ takes place within o_k. (b) An expand-operation happens within o_j from $v_{j,k}$ to $v_{i,j}$.

Domination events and arc events are easy to detect. The point and time of a collision is trivial to compute for any pair of offset circles, too. Unfortunately there is no obvious way to identify those pairs of circles for which this intersection will happen within portions of these offset circles which will still be active at the time of the collision. Hence, for the rest of

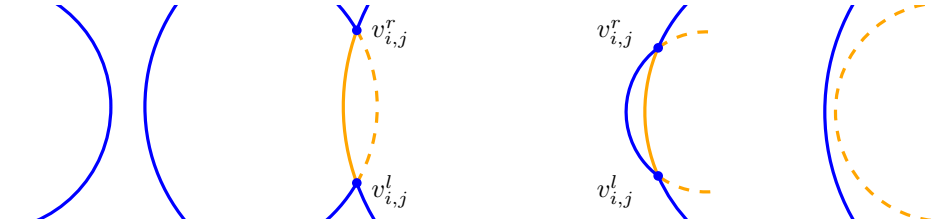


Figure 7 (a) A split-operation happens when at the time of a collision event. (b) A merge-operation happens at the time of a domination event.

this section we assume that all collisions among all pairs of offset circles are computed prior to the actual arc expansion. Lemma 9 verifies that our algorithm correctly simulates the arc expansion.

▶ **Lemma 9.** *For time $t > 0$, the arc arrangement $\mathcal{A}(S,t)$ can be obtained from $\mathcal{A}(S,0)$ by modifying it according to all collision events, domination events and arc events that occur till time t, in the order in which they appear.*

If the maximum weight of all sites is associated with only one site then there will be a time t when the offset circle of this site dominates all other offset circles, i.e., when $\mathcal{WF}(S,t)$ contains only this offset circle as one active arc. Obviously, at this time no further event can occur and the arc expansion stops. If multiple sites have the same maximum weight then $\mathcal{Q}$ can only be empty once $\mathcal{WF}(S,t)$ contains only one loop of active arcs which all lie on offset circles of these sites and if all wavefront vertices move along rays to infinity.

▶ **Lemma 10.** *An active arc or active vertex within an active offset is identified and marked as a wavefront arc (wavefront vertex, resp.) at time $t \geq 0$ if and only if it lies on $\mathcal{WF}(S,t)$.*

If we allow points in $\mathbb{R}^2$ to have the same weighted distance to more than three sites then we need to modify our strategy. In particular, we need to take care of constellations in which more than three arc events happen simultaneously at the same point. In such a case it is necessary to carefully choose the sequence in which the corresponding arc events are handled. More precisely, an arc event may only be handled (without corrupting the state of the active offsets) whenever the respective active vertices are considered neighboring within the active offsets. If the active vertices that participate in an arc event are not currently neighboring then we can always find an arc event whose active vertices are neighboring that happens simultaneously at the same location by walking along the corresponding active offsets. By dealing with the arc events in this specific order, we generate multiple coinciding Voronoi nodes of degree three. Domination events that occur simultaneously at the same point p_e are processed in increasing order of the weights. Note that this order can already be established at the time when an event is inserted into $\mathcal{Q}$, at no additional computational cost. Simultaneous multiple collision events at the same point p_e either involve arcs that are not active or coincide with arc events. These arc events automatically establish a sorted order of the active arcs around p_e, thus allowing us to avoid an explicit (and time-consuming) sorting.

▶ **Lemma 11.** *During the arc expansion $\mathcal{O}(n^2)$ collision and domination events are computed.*

We know that collision events create and domination events remove active vertices (and make them inactive for good). A collapse of an entire active-arc triangle causes two vertices to become inactive. During every other arc event at least one active vertex becomes inactive,

but at the same time one inactive vertex may become active again. In order to bound the number of arc events it is essential to determine how many vertices can be active and how often a vertex can undergo a *reactivation*, i.e., change its status from inactive to active. (Note that Lemma 2 is not applicable to a moving vertex since its polar angle does not stay constant.) We now argue that the total number of reactivations of inactive vertices is bounded by the number of different vertices that ever were active during the arc expansion.

▶ **Lemma 12.** *Every reactivation of a moving vertex during an arc event forces another moving vertex to become inactive and remain inactive for the rest of the arc expansion.*

▶ **Lemma 13.** *Let h be the number of different vertices that ever were active during the arc expansion. Then $\mathcal{O}(h)$ arc events can take place during the arc expansion.*

▶ **Theorem 14.** *The multiplicatively weighted Voronoi diagram $\mathcal{VD}_w(S)$ of a set S of n weighted point sites can be computed in $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n^2)$ space.*

Additionally, in the full version [9] we argue that the one-dimensional MWVD can be computed efficiently using a wavefront-based strategy.

▶ **Theorem 15.** *The multiplicatively weighted Voronoi diagram $\mathcal{VD}_w(S)$ of a set S of n weighted point sites in one dimension can be computed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.*

6 Reducing the Number of Collisions Computed

Experiments quickly indicate that the vast majority of pairwise collisions computed a priori never ends up on pairs of active arcs. Furthermore, the resulting Voronoi diagrams show a quadratic combinatorial complexity only for contrived input data. We make use of the following results to determine all collision events in near-linear expected time. Throughout this section, we assume that for each site $s_i \in S$ the corresponding weight $w(s_i)$ is independently sampled from some probability distribution.

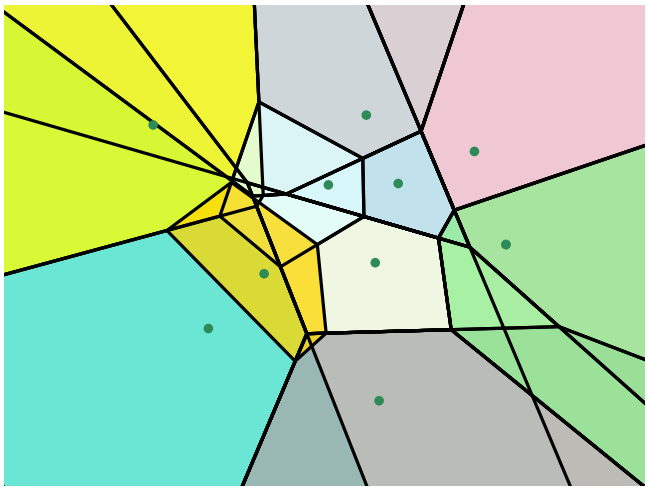


Figure 8 The overlay arrangement is generated by inserting the sites ordered by decreasing weights.

▶ **Definition 16** (Candidate Set). *Consider an arbitrary (but fixed) point $q \in \mathbb{R}^2$, and let s be its nearest neighbor in S under the weighted distance. Let $s' \in S \setminus \{s\}$ be another site. Since s is the nearest neighbor of q we know that either s has a higher weight than s' or a*

smaller Euclidean distance to q than s'. Thus, one can define a candidate set *for a weighted nearest neighbor of q which consists of all sites $s \in S$ such that all other sites in S either have a smaller weight or a larger Euclidean distance to q.*

▶ **Lemma 17** (Har-Peled and Raichel [8]). *For all points $q \in \mathbb{R}^2$, the candidate set for q among S is of size $\mathcal{O}(\log n)$ with high probability.*

▶ **Lemma 18** (Har-Peled and Raichel [8]). *Let K_i denote the Voronoi cell of s_i in the unweighted Voronoi diagram of the i-th suffix $S_i := \{s_i, \ldots, s_n\}$. Let $\mathcal{OA}$ denote the arrangement formed by the overlay of the regions $K_1, \ldots, K_n$. Then, for every face f of $\mathcal{OA}$, the candidate set is the same for all points in f.*

Figure 8 shows a sample overlay arrangement. Kaplan et al. [11] prove that this overlay arrangement has an expected complexity of $\mathcal{O}(n \log n)$. Note that their result is applicable since inserting the points in sorted order of their randomly chosen weights corresponds to a randomized insertion. These results allow us to derive better complexity bounds.

▶ **Theorem 19** (Kaplan et al. [11]). *The expected combinatorial complexity of the overlay of the minimization diagrams that arises during a randomized incremental construction of the lower envelope of n hyperplanes in $\mathbb{R}^d$, for $d \geq 2$, is $\mathcal{O}(n^{\lfloor d/2 \rfloor})$, for d even, and $O(n^{\lfloor d/2 \rfloor} \log n)$, for d odd. The bounds for d even and for $d = 3$ are tight in the worst case.*

▶ **Lemma 20.** *If a collision event occurs between the offset circles of two sites $s_i, s_j \in S$ then there exists at least one candidate set which includes both s_i and s_j.*

▶ **Theorem 21.** *All collision events can be determined in $\mathcal{O}(n \log^3 n)$ expected time by computing the overlay arrangement $\mathcal{OA}$ of a set S of n input sites.*

Thus, the number h of vertices created during the arc expansion can be expected to be bounded by $\mathcal{O}(n \log^3 n)$. Lemma 13 tells us that the number of arc events is in $\mathcal{O}(h)$. Therefore, $\mathcal{O}(n \log^3 n)$ events happen in total.

▶ **Theorem 22.** *A wavefront-based approach allows to compute the multiplicatively weighted Voronoi diagram $\mathcal{VD}_w(S)$ of a set S of n (randomly) weighted point sites in expected $\mathcal{O}(n \log^4 n)$ time and expected $\mathcal{O}(n \log^3 n)$ space.*

7 Extensions

Consider a set S' of n disjoint weighted straight-line segments in $\mathbb{R}^2$. A wavefront propagation among weighted line segments requires us to refine our notion of "collision". We call an intersection of two offset circles a *non-piercing collision event* if it marks the initial contact of the two offset circles. That is, it occurs when the first pair of moving vertices appear. We call an intersection of two offset circles a *piercing collision event* if it takes place when two already intersecting offset circles intersect in a third point for the first time; see Figure 9. In this case, a second pair of moving vertices appear.

Hence, a minor modification of our event-based construction scheme is sufficient to extend it to weighted straight-line segments; see Figure 10. We only need to check whether a piercing collision event that happens at a point p_e at time t_e currently is part of $\mathcal{WF}(S', t_e)$. In such a case the two new vertices as well as the corresponding active arc between them need to be flagged as part of $\mathcal{WF}(S', t_e)$.

An extension to additive weights can be integrated easily into our scheme by simply giving every offset circle a head-start of $w_a(s_i)$ at time $t = 0$, where $w_a(s_i) \geq 0$ denotes the real-valued additive weight that is associated with s_i.

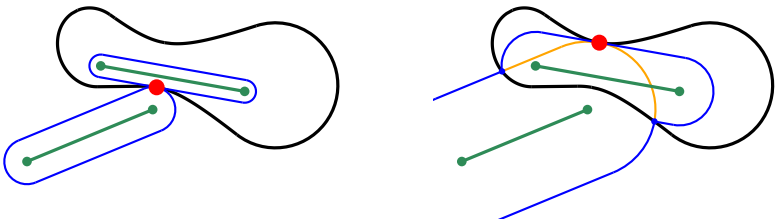


Figure 9 An example of a non-piercing (left) as well as a piercing collision event (right).

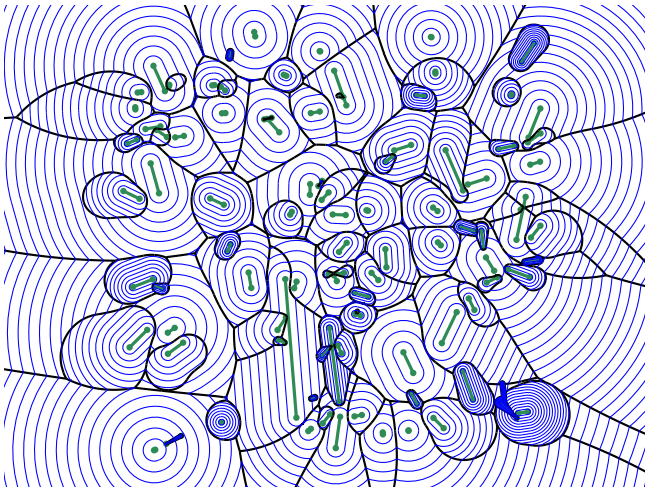


Figure 10 The MWVD of a set of weighted points and weighted straight-line segments together with a family of wavefronts for equally-spaced points in time.

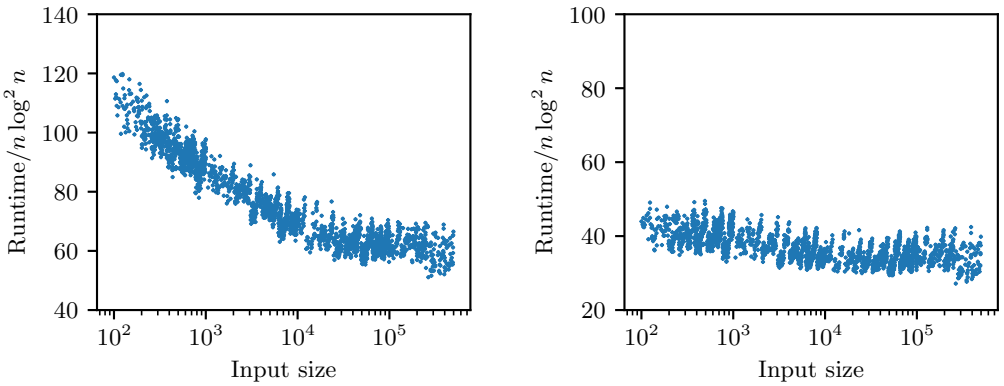
8 Experimental Evaluation

We implemented our full algorithm for multiplicatively weighted points as input sites[2], based on CGAL and exact arithmetic[3]. In particular, we use CGAL's `Arrangement_2` package for computing the overlay arrangement and its `Voronoi_diagram_2` package for computing unweighted Voronoi diagrams. The computation of the MWVD itself utilizes CGAL's `Exact_circular_kernel_2` package which is based on the `Gmpq` number type. The obvious advantage of using exact number types is that events are guaranteed to be processed in the right order even if they occur nearly simultaneously at nearly the same place. One of the main drawbacks of exact number types is their memory consumption which is significantly (and sometimes unpredictably) higher than when standard floating-point numbers are used.

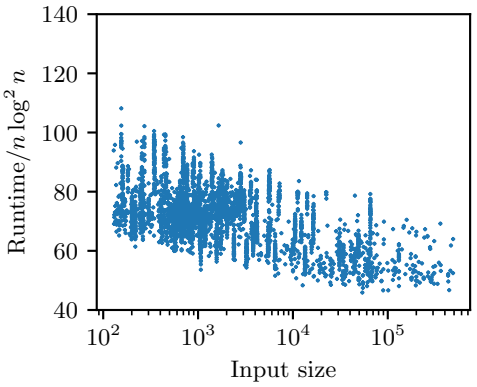
We used our implementation for an experimental evaluation and ran our code on over 8000 inputs ranging from 256 vertices to 500 000 vertices. For all inputs all weights were chosen uniformly at random from the interval $[0, 1]$. All tests were carried out with CGAL 5.0 on an Intel Core i9-7900X processor clocked at 3.3 GHz.

[2] We do also have a prototype implementation that handles both weighted points and weighted straight-line segments. It was used to generate Figure 10.

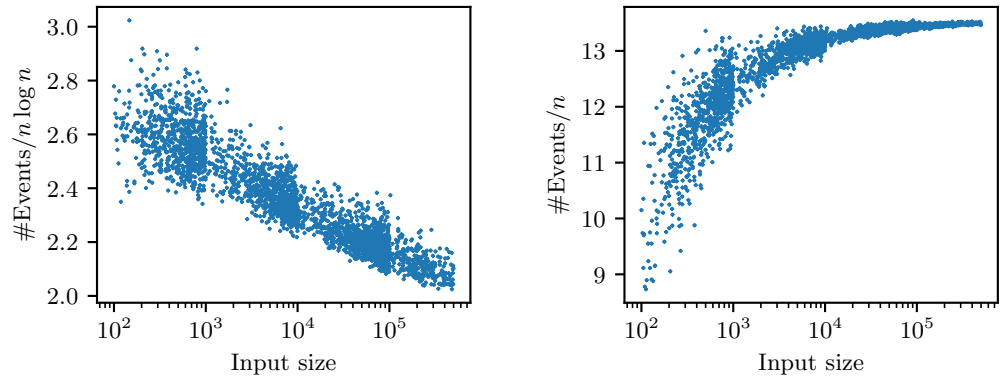
[3] We have not spent enough time on fine-tuning an implementation based on conventional floating-point arithmetic. The obvious crux is that inaccurately determined event times (and locations) may corrupt the state of the arc arrangement and, thus, cause a variety of errors during the subsequent arc expansion.



(a) Left: The overall runtime results for inputs with randomly generated weights and point coordinates. Right: The runtime consumed by the computation of the corresponding overlay arrangements. All runtimes were divided by $n \log^2 n$.



(b) The overall runtime results for inputs with randomly generated weights and vertices of real-world polygons and polygons of the Salzburg database of polygonal data [5, 6] taken as input points. The runtimes were divided by $n \log^2 n$.



(c) The left plot shows the total number of (valid and invalid) collision events (divided by $n \log n$); the right plot shows the number of arc events (divided by n) processed during the arc expansion. All point coordinates and weights were generated randomly.

Figure 11 Experimental evaluation.

In any case, the number of events is smaller than predicted by the theoretical analysis. This is also reflected by our runtime statistics: In Figures 11a and 11b the runtime that was consumed by the computation of a MWVD is plotted. We ran our tests on two different input classes: The point locations were either generated randomly, i.e., they were chosen according to either a uniform or a normal distribution, or obtained by taking the vertices of real-world polygons or polygons of the brand-new Salzburg database of polygonal data [5, 6]. Summarizing, our tests suggest an overall runtime of $\mathcal{O}(n \log^2 n)$ for both input classes. In particular, the actual geometric distribution of the sites does not have a significant impact on the runtime if the weights are chosen randomly: For real-world, irregularly distributed sites the runtimes are scattered more wildly than in the case of uniformly distributed sites, but they do not increase. The numbers of collision events and arc events that occurred during the arc expansion are plotted in Figure 11c. Our tests suggest that we can expect to see at most $3n \log n$ collision events and at most most $14n$ arc events to occur. Note that the number of arc events forms an upper bound on the number of Voronoi nodes of the final MWVD. That is, random weights seem to result in a linear combinatorial complexity of the MWVD.

It is natural to ask how much these results depend on the randomness of the weights. To probe this question we set up a second series of experiments: We sampled points uniformly within a square with side-length $\sqrt{2}$ and then tested different weights. Let $d(s)$ be the distance of the site $s \in S$ from the center of the square, and let $r(s)$ be a number uniformly distributed within the interval $[0, 1]$. Of course, $0 \leq d(s) \leq 1$. Then we assign $\alpha \cdot d(s) + \beta \cdot r(s) / (\alpha + \beta)$ as weight to s, with α and β being the same arbitrary but fixed non-negative numbers for all sites of S. Figure 12 shows the results obtained for the same sets of points and the (α, β)-pairs $(1, 0)$, $(9, 1)$, $(7, 3)$, $(1, 1)$ and $(0, 1)$. This test makes it evident that the bounds on the complexities need not hold if the weights are not chosen randomly, even for a uniform distribution of the sites. Rather, this may lead to a linear number of candidates per candidate set and a quadratic runtime complexity, as shown in Figure 12.

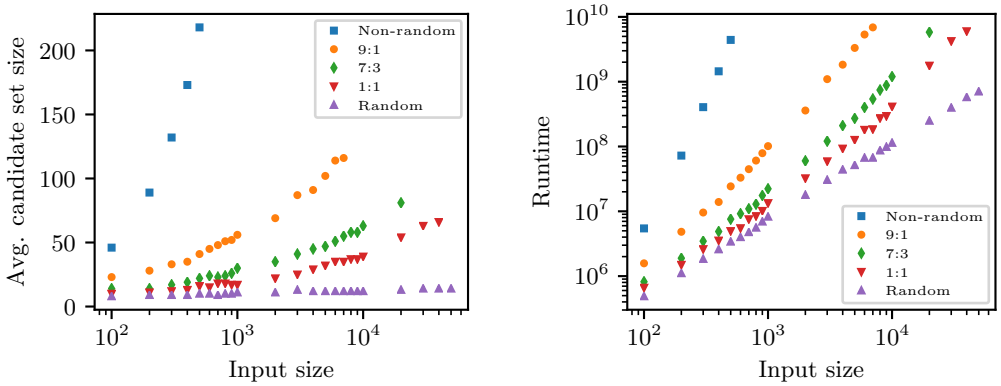


Figure 12 The plots show how the average number of candidates (left) and the total runtime (right) depend on the weights assigned to the sites. Each marker on the x-axes indicates the number n of input sites uniformly distributed within a square.

9 Conclusion

We present a wavefront-like approach for computing the MWVD of points and straight-line segments. Results by Kaplan et al. [11] and Har-Peled and Raichel [8] allow to predict an $\mathcal{O}(n \log^4 n)$ expected time complexity for point sites with random weights. We also discuss a robust, practical implementation which is based on CGAL and exact arithmetic. Extensive tests of our code indicate an average runtime of $\mathcal{O}(n \log^2 n)$ if the sites are weighted randomly. To the best of our knowledge, there does not exist any other code for computing MWVDs that is comparatively fast. A simple modification of our arc expansion scheme makes it possible to handle both additive and multiplicative weights simultaneously. Our code is publicly available on GitHub under `https://github.com/cgalab/wevo`. Figure 13 shows several examples of MWVDs computed by our implementation.

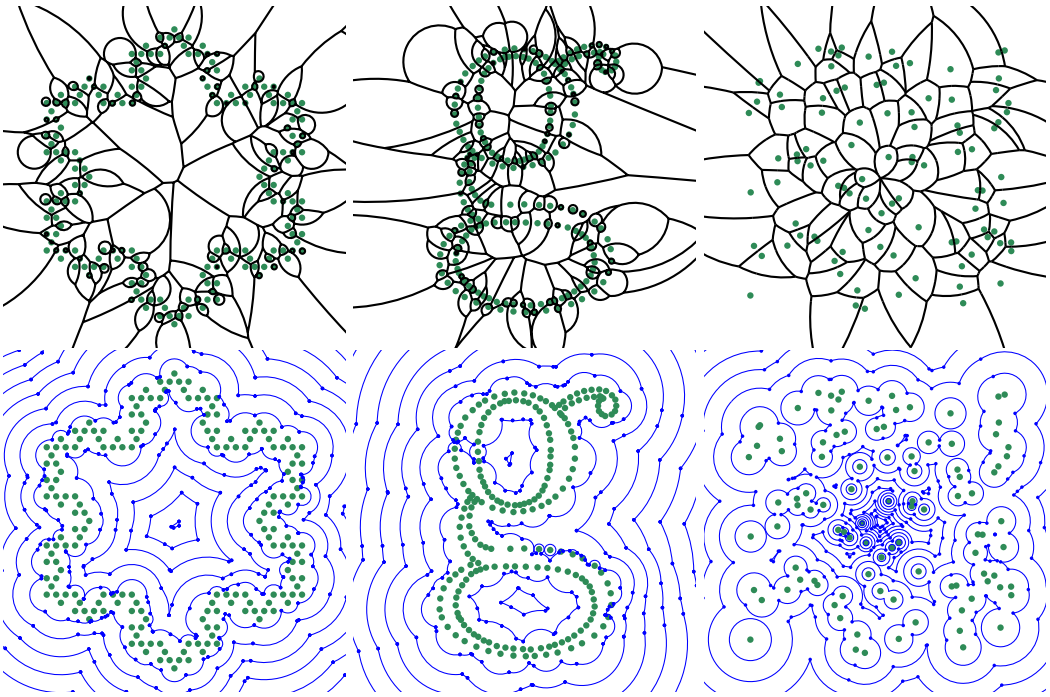


Figure 13 Several examples of MWVDs are shown in the top figures. The bottom figures illustrate a series of uniformly distributed wavefronts that have been derived from the corresponding MWVDs.

References

1 Franz Aurenhammer. The One-Dimensional Weighted Voronoi Diagram. *Information Processing Letters*, 22(3):119–123, 1986. doi:10.1016/0020-0190(86)90055-4.

2 Franz Aurenhammer and Herbert Edelsbrunner. An Optimal Algorithm for Constructing the Weighted Voronoi Diagram in the Plane. *Pattern Recognition*, 17(2):251–257, 1984. doi:10.1016/0031-3203(84)90064-5.

3 Rudi Bonfiglioli, Wouter van Toll, and Roland Geraerts. GPGPU-Accelerated Construction of High-Resolution Generalized Voronoi Diagrams and Navigation Meshes. In *Proceedings of the Seventh International Conference on Motion in Games*, pages 26–30, 2014. doi:10.1145/2668084.2668093.

4 Barry N. Boots. Weighting Thiessen Polygons. *Economic Geography*, 56(3):248–259, 1980. doi:10.2307/142716.

5 Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader. On Generating Polygons: Introducing the Salzburg Database. In *Proceedings of the 36th European Workshop on Computational Geometry*, pages 75:1–75:7, March 2020.

6 Computational Geometry and Applications Lab Salzburg. Salzburg Database of Geometric Inputs. `https://sbgdb.cs.sbg.ac.at/`, 2020.

7 Leonidas Guibas. Kinetic Data Structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, pages 23.1–23.18. Chapman and Hall/CRC, 2001. ISBN 9781584884354.

8 Sariel Har-Peled and Benjamin Raichel. On the Complexity of Randomly Weighted Multiplicative Voronoi Diagrams. *Discrete & Computational Geometry*, 53(3):547–568, 2015. `doi:10.1007/s00454-015-9675-0`.

9 Martin Held and Stefan de Lorenzo. An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams, 2020. `arXiv:2006.14298`.

10 Kenneth E. Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast Computation of Generalized Voronoi Diagrams using Graphics Hardware. In *Proceedings of the the 26th Annual International Conference on Computer Graphics and Interactive Techniques*, pages 277–286. ACM Press/Addison-Wesley Publishing Co., 1999. `doi:10.1145/311535.311567`.

11 Haim Kaplan, Edgar Ramos, and Micha Sharir. The Overlay of Minimization Diagrams in a Randomized Incremental Construction. *Discrete & Computational Geometry*, 45(3):371–382, 2011. `doi:10.1007/s00454-010-9324-6`.

12 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0 edition, 2019. URL: `https://doc.cgal.org/5.0/Manual/packages.html`.

13 Kira Vyatkina and Gill Barequet. On Multiplicatively Weighted Voronoi Diagrams for Lines in the Plane. *Transactions on Computational Science*, 13:44–71, 2011. `doi:10.1007/978-3-642-22619-9_3`.

Fully-Dynamic Coresets

Monika Henzinger
University of Vienna, Faculty of Computer Science, Austria
https://homepage.univie.ac.at/monika.henzinger/
monika.henzinger@univie.ac.at

Sagar Kale
University of Vienna, Faculty of Computer Science, Austria
https://sagark4.github.io/
sagar.kale@univie.ac.at

Abstract

With input sizes becoming massive, coresets – small yet representative summary of the input – are relevant more than ever. A weighted set C_w that is a subset of the input is an ε-coreset if the cost of any feasible solution S with respect to C_w is within $[1\pm\varepsilon]$ of the cost of S with respect to the original input. We give a very general technique to compute coresets in the fully-dynamic setting where input points can be added or deleted. Given a static (i.e., not dynamic) ε-coreset-construction algorithm that runs in time $t(n,\varepsilon,\lambda)$ and computes a coreset of size $s(n,\varepsilon,\lambda)$, where n is the number of input points and $1-\lambda$ is the success probability, we give a fully-dynamic algorithm that computes an ε-coreset with worst-case update time $O((\log n)\cdot t(s(n,\varepsilon/\log n,\lambda/n),\varepsilon/\log n,\lambda/n))$ (this bound is stated informally), where the success probability is $1-\lambda$. Our technique is a fully-dynamic analog of the merge-and-reduce technique, which is due to Har-Peled and Mazumdar [17] and is based on a technique of Bentley and Saxe [3], that applies to the insertion-only setting where points can only be added. Although, our space usage is $O(n)$, our technique works in the presence of an adaptive adversary, and we show that $\Omega(n)$ space is required when adversary is adaptive.

As a concrete implication of our technique, using the result of Braverman et al. [4], we get fully-dynamic ε-coreset-construction algorithms for k-median and k-means with worst-case update time $O(\varepsilon^{-2}k^2\log^5 n\log^3 k)$ and coreset size $O(\varepsilon^{-2}k\log n\log^2 k)$ ignoring $\log\log n$ and $\log(1/\varepsilon)$ factors and assuming that $\varepsilon=\Omega(1/\operatorname{poly}(n))$ and $\lambda=\Omega(1/\operatorname{poly}(n))$ (which are very weak assumptions made only to make these bounds easy to parse). This results in the first fully-dynamic constant-approximation algorithms for k-median and k-means with update times $O(\operatorname{poly}(k,\log n,\varepsilon^{-1}))$. Specifically, the dependence on k is only quadratic, and the bounds are worst-case. The best previous bound for both problems was *amortized* $O(n\log n)$ by Cohen-Addad et al. [10] via randomized $O(1)$-coresets in $O(n)$ space.

We also show that under the OMv conjecture [18], a fully-dynamic $(4-\delta)$-approximation algorithm for k-means must either have an amortized update time of $\Omega(k^{1-\gamma})$ or amortized query time of $\Omega(k^{2-\gamma})$, where $\gamma>0$ is a constant.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Clustering, Coresets, Dynamic Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.57

Related Version A full version of the paper is available at https://arxiv.org/abs/2004.14891.

Funding *Monika Henzinger*: The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.
Sagar Kale: Fully supported by Vienna Science and Technology Fund (WWTF) through project ICT15-003.

1 Introduction

Clustering is an ubiquitous notion that one encounters in computer-science areas such as data mining, machine learning, image analysis, bioinformatics, data compression, and computer graphics, and also in the fields of medicine, social science, marketing, etc. Today, when the input data has become massive, one would rather run an algorithm on a small but representative summary of the input, and for clustering problems, a *coreset* serves that function perfectly. The concept of a coreset was defined first in computational geometry as a small subset of a point set that approximates the shape of the point set. The word coreset has now evolved to mean an appropriately weighted subset of the input that approximates the original input with respect to solving a computational problem.

Let P be a problem for which the input is a weighted subset[1] $X_w \subseteq U$; think of U as in a metric space (U, d), so U is unweighted and d is the distance function. Let $n := |X_w|$ and $W := \sum_{x \in X_w} w(x)$. We also refer to elements of U as points. The goal in the problem P then is to output S^* that belongs to the feasible-solution space (or *query* space) Q such that the cost $c(S^*, X_w)$ is minimized. For example, in the *k-median* (respectively, *k-means*) problem, Q is the set of all (unweighted) subsets of X_w of cardinality at most k and $c(S, X_w) := \sum_{x \in X_w} w(x) \min_{s \in S} d(x, s)$ (respectively, $\sum_{x \in X_w} w(x) \min_{s \in S}(d(x, s))^2$). Then, for the problem P, a weighted set C_w such that $C_w \subseteq X_w$ is an *ε-coreset* if, for any feasible solution $S \in Q$, we have that $c(S, X_w) \in [1 \pm \varepsilon] c(S, C_w)$; we sometimes say that the *quality* of coreset C_w is ε. For many problems, fast coreset-construction algorithms exist; e.g., for k-median and k-means, $\tilde{O}(nk)$-time[2] algorithms for computing ε-coresets of size $O(\varepsilon^{-2} k \operatorname{polylog}(n))$ exist.

Throughout the paper, we assume that the cost function c for the problem P is *linear*: for any weighted subsets $Y_w^1, Y_w^2 \subseteq U$ with disjoint supports and any $S \in Q$, we have that $c(S, Y_w^1 \cup Y_w^2) = c(S, Y_w^1) + c(S, Y_w^2)$, where the union $Y_w^1 \cup Y_w^2$ is the weighted union. It is easy to see that k-median and k-means cost functions are linear.

Our goal in this paper is to give *dynamic algorithms* for computing a coreset. In the dynamic setting, the input changes over time. A dynamic algorithm is a data structure that supports three types of operations: *Insert*(p, w), which inserts a point p with weight w into X_w; *Delete*(p), which removes point p from X_w; and *Query*$()$, which outputs a coreset of X_w. Weight updates can be simulated by deleting and re-inserting a given point, or the data structure may support a separate weight-changing operation. This is known as the *fully* dynamic model as opposed to the *insertion-only* setting where a point can only be inserted. At any time instant, a coreset is maintained by the algorithm, and the complexity measure of interest is the *update* time, i.e., how fast the solution can be updated after receiving a point update, and also the *size* of the coreset, which determines the *query* time. Suppose there is a dynamic coreset-construction algorithm, say ALG_D, for a problem P. Then a solution for the problem P can be maintained dynamically by running ALG_D, and on query, a solution is computed by querying ALG_D and running a static (i.e., not dynamic) algorithm for P on the returned coreset. In this paper, we give a very general technique on how to maintain a coreset in the fully-dynamic setting: given a *static* coreset-construction algorithm for any problem P, we show how to turn it into a dynamic coreset-construction algorithm for P.

[1] When using a set operation such as union or notation such as $\subseteq$ with one or more weighted sets, we mean it for the underlying unweighted sets. Also, all weights are nonnegative.

[2] Logarithmic factors are hidden in the $\tilde{O}$ notation.

Intuitively, our technique is to the fully-dynamic setting as the merge-and-reduce technique is to the insertion-only setting. The *merge-and-reduce* technique, which is based on a technique of Bentley and Saxe [3], is due to Har-Peled and Mazumdar [17] and is a fundamental technique to obtain an insertion-only coreset-construction algorithm using a static coreset-construction algorithm, say ALG_S, as a black box. Loosely speaking, it is as follows. At any time instant, the algorithm maintains up to $\lceil \log n \rceil$ buckets. For $i \in \{1, 2, \ldots, \lceil \log n \rceil\}$, the bucket B_i has capacity 2^{i-1}, each bucket can be either *full*, (i.e. at capacity 2^{i-1}) or *empty*, and each point goes in exactly one bucket. Then at any time-instant, the current number of points uniquely determines the states of the buckets. Whenever a point is inserted, the states of the buckets change like a binary counter. That is, the new point goes into B_i, where B_i is the smallest-index empty bucket, and all the points in $\cup_{j=1}^{i-1} B_j$ are moved to B_i (*merge*). Note that this creates a full bucket B_i. Then a coreset is computed on B_i by running ALG_S on it (*reduce)*. The overall coreset is then just union of all non-empty buckets.

We show that a similar result can be achieved in the fully-dynamic setting. Our main result is the following theorem (stated slightly informally).

▶ **Theorem 1.** *Assume that there is a static coreset construction algorithm for a problem P with linear cost function that* ***a)*** *runs in time $t_P(n_s, \varepsilon_s, \lambda_s, W_s)$,* ***b)*** *always outputs a set of cardinality at most $s_P(\varepsilon_s, \lambda_s, W_s)$ and total weight at most $(1+\delta)W_s$, and* ***c)*** *has the guarantee that the output is an ε_s-coreset with probability at least $1-\lambda_s$, where n_s is the number of* integer*-weighted input points and W_s is the total weight of points.*

Then there is a fully-dynamic coreset-construction algorithm for P that, with rational*-weighted input points,* ***a)*** *always maintains an output set of cardinality at most $s_P(\varepsilon, \lambda, W)$,* ***b)*** *has the guarantee that the output is an ε-coreset with probability at least $1-\lambda$, and* ***c)*** *has worst-case update time*

$$O\left((\log n) \cdot t_P\left(s_P^*, \frac{\varepsilon}{\log n}, \frac{\lambda}{n}, W\right)\right),$$

where $W = O((1+\delta)^{\lceil \log n \rceil} \operatorname{poly}(n))$, $s_P^* = s_P\left(\frac{\varepsilon}{\log n}, \frac{\lambda}{n}, W\right)$, and n is the current number of points.

We mention below a concrete implication of the above theorem for k-median and k-means using the result of Braverman et al. [4].

▶ **Theorem 2.** *For the k-median and k-means problems, there is a fully-dynamic algorithm that maintains a set of cardinality $O(\varepsilon^{-2}k(\log n \log k \log(k\varepsilon^{-1} \log n) + \log(1/\lambda)))$, that is an ε-coreset with probability at least $1-\lambda$, and has worst-case update time*

$$O\left(\varepsilon^{-2}k^2 \log^5 n \log^3 k \log^2(\varepsilon^{-1})(\log\log n)^3\right),$$

assuming that $\varepsilon = \Omega(1/\operatorname{poly}(n))$ and $\lambda = \Omega(1/\operatorname{poly}(n))$. [3]

Ignoring $\log\log n$ and $\log(1/\varepsilon)$ above, the coreset cardinality is $O(\varepsilon^{-2}k \log n \log^2 k)$ and worst-case update time is $O(\varepsilon^{-2}k^2 \log^5 n \log^3 k)$. It can be easily proved that running an α-approximation algorithm for k-median on an ε-coreset gives a $2\alpha(1+\varepsilon)$-approximation whereas that for k-means gives a $4\alpha(1+\varepsilon)$-approximation. Any such polynomial-time static algorithm – say, e.g., $(5+\varepsilon')$-approximation algorithm for k-median by Arya et al. [2] and

[3] We make these very weak assumptions to simplify some extremely unhandy factors involving ε and λ in the expression for the update time.

16-approximation algorithm for k-means by Gupta and Tangwongsan [15] – can be run on our output coreset in $O(\text{poly}(k, \log n, \varepsilon^{-1}))$ time to obtain a constant approximation. This is the first fully-dynamic constant-approximation algorithm for k-median and k-means whose *worst-case* time per operation is polynomial in k, $\log n$, and ε^{-1}. The best previous result was a randomized algorithm with *amortized* $O(n \log n)$ update time and $O(n)$ space by Cohen-Addad et al. [10].

With a simple reduction, we also show a conditional lower bound on the time per operation for k-means. The following theorem is proved as Theorem 20 in Section 4.

▶ **Theorem 3.** *Let $\gamma > 0$ be a constant. Under the OMv conjecture [18], for any $\delta > 0$, there does not exist a fully-dynamic algorithm that maintains a $(4 - \delta)$-approximation for k-means with amortized update time $O(k^{1-\gamma})$ and query time $O(k^{2-\gamma})$ such that over a polynomial number of updates, the error probability is at most $1/3$.*

Our technique

At the core, our technique is simple. We always maintain a balanced binary tree of depth $\lceil \log n \rceil$ containing exactly n leaf nodes (recall that n is the current number of points). Each node corresponds to a subset of X_w, the current input: each leaf node corresponds to a singleton (hence n leaf nodes), and an internal node corresponds to the weighted union of the sets represented by its children. If the cardinality of the union exceeds a certain threshold, then we use the static coreset-construction algorithm to compute its coreset. The root gives a coreset of the whole input.

We next explain how we handle updates in this data structure. Point insertions are straightforward: create a new leaf node and run all the static-algorithm instances at the nodes on the leaf-to-root path. The way we handle point deletions is similar in spirit to the way delete-min works in a min-heap data structure: whenever a point at leaf-node ℓ_d is deleted, we swap contents of ℓ_d with those of the rightmost leaf-node, say ℓ_r, and delete ℓ_r, thus maintaining the balance of the tree. Then we run all the static-algorithm instances at the nodes on the two affected leaf-to-root paths.

However, there are some complications that require new techniques to make it work in *worst-case* time. To maintain guarantees for the output coreset quality and overall success probability, we need to adapt the parameters ε_s and λ_s used for the static algorithm at the internal nodes. The problem is that both depend on n, which changes over time and thus might become outdated. To show an *amortized* update-time bound, we can simply rerun the static algorithms at all internal nodes whenever n has changed by a constant factor. To achieve our worst-case bound, we use two *refresh pointers* that point at leaf nodes, and after each update operation, we rerun using the new values of ε_s and λ_s all the static-algorithm instances at the nodes on the leaf-to-root path from the leaf nodes pointed to by the refresh pointers. This keeps the outputs of the static-algorithm instances at the internal nodes always fresh. After every update, we move these pointers to the right so that they point to the next leaf nodes.

Further complications are caused by fractional weights at the leaf nodes and fractional intermediate-output weights. A problem arises when the weights in X_w are fractional, and the static algorithm expects integer-weighted input [9]. Even if the static algorithm can handle fractional weights [12, 4], there can be a problem. The output of the static algorithm at an internal node is the input for the static algorithm at its parent. Naïvely feeding these output fractional weights directly to the static algorithm at the parent may result in numbers exponential in n near the root, thus prohibitively increasing the update time. To deal with

these problems, rounding is needed for the input, i.e., at the leaf nodes, as well as for each intermediate-output at an internal node. Thus, we propose a more sophisticated rounding scheme and show that the rounding errors accumulated by our rounding are not too high.

We note that our balanced binary-tree data structure may be used to get dynamic algorithms in the following situations. Let $f : \mathbb{R}^{\dim} \to \mathbb{R}^{\dim}$ be a *multi-valued* function. Suppose for any u and v with disjoint supports and for any $f_u \in f(u)$ and $f_v \in f(v)$, we have $f_u + f_v \in f(u + v)$. Also suppose that $f(f(v)) \subseteq f(v)$ for any v. Now, given input v, we want to compute some vector in $f(v)$. If there is a static algorithm for this, then using our technique, we can maintain some vector in $f(v)$ for a dynamically changing vector v. The allowed dynamic operation on v is "add a to the ith coordinate of v," where $a \in \mathbb{R}$. The resulting dynamic algorithm is fast if the static algorithm always outputs a "small" vector; this is true for coresets because coresets are small by nature. Thinking about coresets in the above language, each point is an identity vector in $\mathbb{R}_+^{|U|}$, and then each weighted set of points naturally identifies with a vector. An ε-coreset reduces the number of points drastically. Union of coresets of two disjoint sets is a coreset of the union of those two sets (see Lemma 5). Although an ε-coreset of an ε-coreset is not an ε-coreset, it is a $(2\varepsilon + \varepsilon^2)$-coreset (see Lemma 6).

Space

In the merge-and-reduce technique, a bucket B_i will not actually contain 2^{i-1} points but just a coreset of 2^{i-1} points that would have been there otherwise at any time instant. Thus, using space just $\lceil \log n \rceil$ times the coreset size for a bucket, one can get a coreset of the whole input [17]. This makes it also applicable in the more restricted streaming model, where the input points arrive in a sequence and the goal is to compute a coreset using sublinear space. In the fully-dynamic setting, deletions also need to be handled, and hence no deterministic or randomized algorithm against an adaptive adversary that stores only a coreset is possible: the adversary generating the input could simply ask a query and then delete all points in the returned coreset. Hence, an algorithm that does not store any information about the non-coreset points would not be able to maintain a valid coreset. Even though we store all the points in our fully-dynamic technique, i.e., its space usage is $O(n)$, it works against an adaptive adversary because we never make any assumption about the next update and perform each update independently of all previous updates. By a straightforward reduction from the communication problem of INDEX, we show that $\Omega(n)$ space is required in the presence of an adaptive adversary. The proof of the following theorem appears in Section 4.

▶ **Theorem 4.** *A fully-dynamic algorithm that obtains any bounded approximation for* 1*-median or* 1*-means that works in the presence of an adaptive adversary and has success probability* $1 - 1/(8n^2)$ *must use* $\Omega(n)$ *space, where* n *is the current number of points.*

Comparison with the sparsification technique

Our technique is close to the sparsification technique of Eppstein et al. [11] that is used to speed up dynamic graph algorithms. There, one has to assume that the number of vertices in the input graph, say n_v, does not change, but the edge set changes dynamically, and the bounds are obtained in terms of n_v and m, the current number of edges. Their dynamic edge-tree structure is based on a fixed vertex-partition tree. In the vertex-partition tree, a node at level i corresponds to a vertex-set of cardinality $n_v/2^i$, and a vertex-set at a node is a union of its children's vertex sets (cf. our technique). To start using the edge tree, the vertex-partition tree has to be built first and hence the knowledge of n_v is necessary. Neither

do we need such a fixed structure nor any information about the number of points. Also, in the sparsification technique, there is no analog of weight handling/rounding. Another crucial difference is that they do not use a routine analogous to our refresh-pointers routine because their internal-node guarantees are always fresh. As we discussed before, these refresh pointers are critical for us also in making sure that the error introduced by the unavoidable rounding of output weights of the static-algorithm instances is kept in check.

1.1 Related Work

The most related work is by Cohen-Addad et al. [10] who give an $O(1)$-coreset for k-median and k-means in *amortized* update time of $O(n \log n)$.

For k-median and k-means, the first coreset-construction algorithms were by Har-Peled and Mazumdar [17] for Euclidean metrics and by Chen [9] for general metrics. Improved algorithms computing smaller coresets were later obtained by Har-Peled and Kushal [16] and by Feldman and Langberg [12]. The current known best is by Braverman et al. [4]: $O(\varepsilon^{-2}k \log k \log n)$-size coresets in $\tilde{O}(nk)$ time, who also give an excellent summary of the literature on coresets that we highly recommend. Note that by merge-and-reduce technique, each improvement also gave rise to better (insertion-only) streaming coreset constructions. For k-median and k-means, Frahling and Sohler [14] gave the first coreset-construction algorithm in the dynamic-streaming setting where points can be added or removed. It uses space and update time of $O(\text{poly}(\varepsilon^{-1}, \log m, \log \Delta))$ for constant k and dim when the points lie in the discrete Euclidean metric space $\{1, \ldots, \Delta\}^{\dim}$; for k-median, this was recently improved to $O(\varepsilon^{-2}k \,\text{poly}(\dim, \log \Delta))$ space and update time of $O(\text{poly}(\varepsilon^{-1}, k, \dim, \log \Delta))$ by Braverman et al. [5]. Coreset constructions with improvements in certain parameters in the Euclidean settings have been obtained [13, 24].

The k-median and k-means problems have received significant attention in the algorithms community [8, 20, 19, 7, 2, 23, 21, 15, 22, 1, 6]. The best approximation ratio for k-median is 2.675 by Byrka et al. [6] and that for k-means is $9 + \varepsilon$ by Ahmadian et al. [1].

2 Preliminaries

Let us fix a problem P with the input X_w, the set of feasible solutions Q, and the linear cost function $c : Q \times \mathcal{W} \to \mathbb{R}_+$, where $\mathcal{W}$ is the set of all weighted subsets[4] of X_w. All the numbers encountered are nonnegative.

The computational model

The input set X_w is a weighted set of n points having rational weights whose numerators and denominators are bounded by $O(\text{poly}(n))$. The algorithm works in the random access machine model with word size $O(\log n)$. Each memory word can be accessed in constant time. With each update, a new point is inserted, an existing point is deleted, or the weight of an existing point is modified by adding or subtracting a nonnegative number. The net weight of each point always stays nonnegative with its numerator and denominator always bounded by $O(\text{poly}(n))$.

We will prove some basic lemmas about coresets. Using these, we can take weighted union of two coresets without any loss (Lemma 5) and take a coreset of a coreset without much loss (Lemma 6).

[4] To be precise: denote unweighted version of X_w by X', then $\mathcal{W}$ is essentially $\mathbb{R}_+^{X'}$.

▶ **Lemma 5.** *If C_w^1 and C_w^2 are ε-coresets of X_w^1 and X_w^2, respectively, with respect to a linear cost function c such that $X_w^1 \cap X_w^2 = \emptyset$, then $C_w^1 \cup C_w^2$ is an ε-coreset of $X_w^1 \cup X_w^2$.*

Proof. By linearity of c: for any $S \in Q$,

$$c(S, X_w^1 \cup X_w^2) = c(S, X_w^1) + c(S, X_w^2) \in [1\pm\varepsilon]\left(c(S, C_w^1) + c(S, C_w^2)\right) = [1\pm\varepsilon]c(S, C_w^1 \cup C_w^2)\,,$$

where, recall that, $C_w^1 \cup C_w^2$ is a weighted union. ◀

▶ **Lemma 6.** *If C'_w is an ε-coreset of C_w, and C''_w is a δ-coreset of C'_w, both with respect to c, then C''_w is an $(\varepsilon + \delta + \varepsilon\delta)$-coreset of C_w with respect to c.*

Proof. For any $S \in Q$, we have $c(S, C_w) \in [1\pm\varepsilon]c(S, C'_w)$ and $c(S, C'_w) \in [1\pm\delta]c(S, C''_w)$. So,

$$c(S, C_w) \geqslant (1-\varepsilon)c(S, C'_w) \geqslant (1-\varepsilon)(1-\delta)c(S, C''_w) \geqslant (1 - \varepsilon - \delta - \varepsilon\delta)c(S, C''_w)\,,$$

and $c(S, C_w) \leqslant (1+\varepsilon)c(S, C'_w) \leqslant (1+\varepsilon)(1+\delta)c(S, C''_w) = (1 + \varepsilon + \delta + \varepsilon\delta)c(S, C''_w)$. ◀

Let C_w^1 be an ε-coreset of C_w and C_w^2 be an ε-coreset of C_w^1. Then we say that C_w^1 and C_w^2 are, respectively, 1-level and 2-level ε-coresets of C_w. Extending this notion, we define an i-level ε-coreset to be an ε-coreset of an $(i-1)$-level ε-coreset.

▶ **Lemma 7.** *If C_w^ℓ is an ℓ-level ε-coreset of C_w, then C_w^ℓ is a $\left(\sum_{i=1}^{\ell} \binom{\ell}{i}\varepsilon^i\right)$-coreset of C_w.*

Proof. The proof is by induction on ℓ. Base case is when $\ell = 1$, and by definition, a 1-level coreset is an ε-coreset. By induction hypothesis, we have that $C_w^{\ell-1}$ is a $\left(\sum_{i=1}^{\ell-1} \binom{\ell-1}{i}\varepsilon^i\right)$-coreset of C_w. Now, C_w^ℓ is an ε-coreset of $C_w^{\ell-1}$, hence C_w^ℓ is an $\left(\varepsilon + (1+\varepsilon)\sum_{i=1}^{\ell-1} \binom{\ell-1}{i}\varepsilon^i\right)$-coreset of C_w by Lemma 6. Now, use Lemma 8, which appears below, with $\alpha = \varepsilon$ to finish the proof. ◀

We prove two basic lemmas.

▶ **Lemma 8.** *For any positive integer ℓ and $\alpha \in \mathbb{R}_+$, we have $\alpha + (1+\alpha)\sum_{i=1}^{\ell-1} \binom{\ell-1}{i}\alpha^i = \sum_{i=1}^{\ell} \binom{\ell}{i}\alpha^i$.*

Proof idea. The proof is provided in Appendix A and uses elementary identities involving binomial coefficients and algebraic manipulations. ◀

▶ **Lemma 9.** *For any positive integer ℓ and $\alpha \in [0, 1]$, we have $\sum_{i=1}^{\ell} \binom{\ell}{i} \left(\frac{\alpha}{2\ell}\right)^i \leqslant \alpha$.*

Proof. $\sum_{i=1}^{\ell} \binom{\ell}{i} \left(\frac{\alpha}{2\ell}\right)^i \leqslant \sum_{i=1}^{\ell} \ell^i \frac{\alpha^i}{2^i \ell^i} = \sum_{i=1}^{\ell} \frac{\alpha^i}{2^i} \leqslant \sum_{i=1}^{\ell} \frac{\alpha}{2^i} \leqslant \alpha$. ◀

Now, as a corollary to Lemma 7, we get the following using Lemma 9.

▶ **Corollary 10.** *If C_w^ℓ is an ℓ-level $(\varepsilon/(2\ell))$-coreset of C_w, then C_w^ℓ is an ε-coreset of C_w.*

As we discussed earlier, rounding of the weights at internal nodes is needed in our dynamic algorithm to achieve the desired worst-case update time. Towards that, we need two lemmas.

In the next lemma, think of a/b as the original weight of the point, c/d as the weight that we want to approximate a/b with, and D as the cost of this point with respect to a feasible solution in Q. So the lemma says that by rounding, the cost of the point stays within $1 \pm b/d$ of the original cost.

▶ **Lemma 11.** *For positive integers a, b, and d, let $c = \lfloor ad/b \rfloor$. Then $cD/d \in [1 \pm b/d]aD/b$ for any nonnegative real D.*

Proof. By the definition of c, we have that $c/d \leqslant a/b \leqslant c/d + 1/d$, and $1/d \leqslant a/d$ because $a \geqslant 1$; hence $a/b \geqslant c/d \geqslant a/b - a/d$, which implies that $aD/b \geqslant cD/d \geqslant aD/b - aD/d = (1 - b/d)aD/b$. ◀

The proof of the following lemma is very similar. Here, think that we approximate the weight r of a point by $\lfloor r \rfloor + c/d$ and the cost of the point stays within $1 \pm 1/d$ of the original cost.

▶ **Lemma 12.** *Let $r \geqslant 1$ be a rational number, a and b be positive integers such that $a/b = r - \lfloor r \rfloor$, d be any positive integer, and $c = \lfloor ad/b \rfloor$. Then $(\lfloor r \rfloor + c/d)D \in (1 \pm 1/d)rD$ for any nonnegative real D.*

Proof. By the definition of c and using $r \geqslant 1$, we get that $a/b \geqslant c/d \geqslant a/b - r/d$; adding $\lfloor r \rfloor$ and multiplying by D finishes the proof. ◀

3 A Dynamic Coreset

We describe our dynamic algorithm for maintaining an ε-coreset for a problem P with query space Q that uses a static coreset algorithm, say, ALG_S.

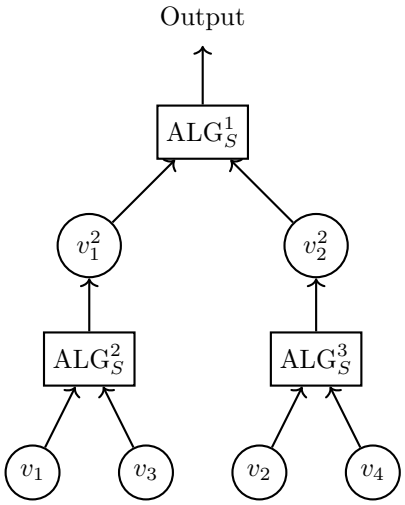


Figure 1 An ALG_S node takes input from two point-nodes. If the union of the sets has cardinality greater than s', then the ALG_S node computes a coreset of cardinality at most s' and passes it on to the point-node above it (its parent). The number of leaf nodes is always n, and the number of levels is always $O(\log n)$, where n is the current number of points.

The main idea is described in Figure 1 using a tree with a special structure. Each node is of one of the two types: a point-node representing a weighted set of points or an alg-node representing an instance of ALG_S. We sometimes use a point-node to denote the point set it represents and an alg-node to denote the ALG_S instance it represents. Each level contains either only point-nodes or only alg-nodes. All leaf nodes are point-nodes and represent a weighted singleton with an input point. Each alg-node gets as input the weighted union of its children, and its output is represented by its parent node (which is a point-node). When running ALG_S at an alg-node A, if the union of its children has cardinality larger than s', then A would compute a coreset of cardinality at most s' otherwise it would just output the weighted union. We will later fix this threshold s' for computing a coreset. An example of

how insertions and deletions are handled is shown in Figure 2 (where all weights are assumed to be one). For the ease of description, from now onwards, we will think of this tree with alg-nodes being collapsed into their parent nodes. Then each leaf node would contain a weighted singleton and each internal node would contain the output of the ALG_S instance run on the weighted union of its children's sets.

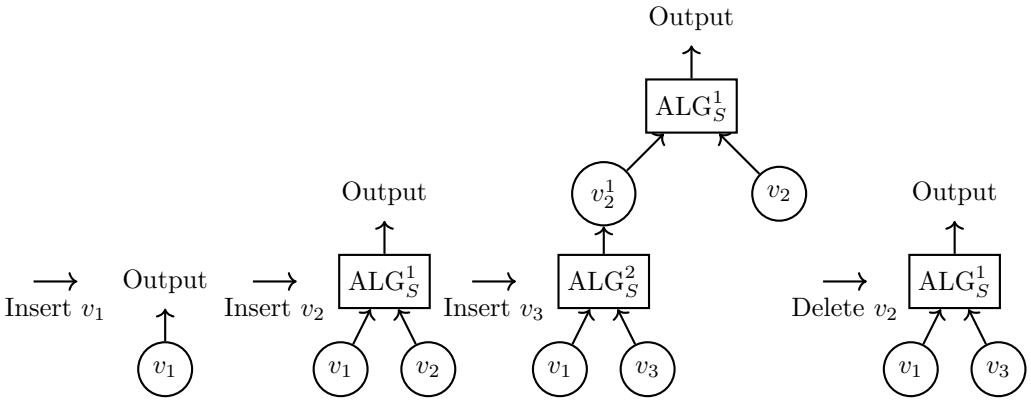


Figure 2 An example of how insertions and deletions are handled. We start with an empty tree. The first point that is inserted is represented by v_1. We use a point and the node that represents it interchangeably. Then v_2 is inserted followed by v_3. Next, if v_4 is inserted, we get exactly the tree shown in Figure 1, and if v_2 is deleted, then we get the last tree.

We guarantee that the resulting tree then will always be a *complete* binary tree, i.e., every level except possibly the lowest is completely filled, and the nodes at the lowest level are packed to the left. To describe the updates briefly, let ℓ_r denote the rightmost leaf node at the lowest level; for simplicity, assume that the lowest level is not full. Insertion is straightforward: the new point goes in a new leaf node to the right of ℓ_r. For deletion of a point at leaf node ℓ_d, if $\ell_d \neq \ell_r$, then we replace contents of ℓ_d with those of ℓ_r and delete ℓ_r. See Section 3.1 for details of these operations. For weight update, the tree does not change.

▶ Remark 13. Since a coreset will not be computed until a node has more than s' points, the tree can be modified so that each leaf node corresponds to a set of $\Theta(s')$ points. Then the number of nodes in the tree is $\Theta(n/s')$. This reduces the additional space used for maintaining this tree. This is important when the number of points is very large. See Section 3.2 for further details. This is essentially the same idea as used for asymmetric sparsification in Section 3.4 in Eppstein et al. [11].

We call the leaf nodes at the same level as that of the leftmost leaf node to be at level 0. We increment these level numbers naturally as we move upwards in the tree. Since we maintain a complete binary tree, the root, which is at the highest level, is on level $\lceil \log n \rceil$. After a point insertion, deletion, or weight update, we recompute all the nodes that are affected by running ALG_S from scratch. Once we update a leaf node, all the nodes on its leaf-to-root path are affected. Since at most two leaf nodes are updated after every point update, we run at most $2\lceil \log n \rceil$ instances of ALG_S. Finally, to reduce the cardinality of our output coreset, we run another *outer* instance of ALG_S with $\varepsilon_s = \varepsilon/3$ and $\lambda_s = \lambda/2$ with input as the output of the root. Here, ε_s and λ_s are parameters for ALG_S as described below, and our goal is to compute an ε-coreset with probability at least $1-\lambda$. The outer instance is run after every update.

The static coreset algorithm ALG_S takes as input an integer weighted set of n_s points with total weight W_s and always returns a weighted set of cardinality at most $s(\varepsilon_s, \lambda_s, W_s)$; this set is an ε_s-coreset with probability at least $1-\lambda_s$. Let the running time of ALG_S be $t(n_s, \varepsilon_s, \lambda_s, W_s)$. We assume that the functions t and s are nondecreasing in W_s and nonincreasing in ε_s and λ_s, and also that t is nondecreasing in n_s. We call such functions t and s *well-behaved.*

We note that t and s implicitly depend on the query space Q as well. In particular, for k-median and k-means, they depend on k and the dimension or the cardinality of the universe from which a solution is allowed to be picked. Also, assume that the total weight of ALG_S's output is at most $1+\delta$ times the total input weight and it outputs a coreset of points with integer weights. For the dynamic algorithm, n denotes the current number of points, and we assume that any input weight is a rational number with numerator and denominator bounded by n^c, for a fixed constant c.

▶ **Theorem 14.** *Assume that there is a static algorithm* ALG_S *that takes as input an integer-weighted set of* n_s *points with total weight* W_s *and always returns an integer-weighted set of cardinality at most* $s(\varepsilon_s, \lambda_s, W_s)$ *with total weight at most* $(1+\delta)W_s$*, and this set is an* ε_s*-coreset with probability at least* $1-\lambda_s$*. Let the running time of* ALG_S *be* $t(n_s, \varepsilon_s, \lambda_s, W_s)$*, and assume that both* s *and* t *are well-behaved. Then there is a fully-dynamic algorithm that, on* rational*-weighted input points, always maintains an* $s\left(\frac{\varepsilon}{3}, \frac{\lambda}{2}, W_p\right)$*-cardinality weighted set. This set is an* ε*-coreset with probability at least* $1-\lambda$*. Its worst-case update time is*

$$O\left(t\left(2s^*, \frac{\varepsilon}{6\lceil \log n_p\rceil}, \frac{\lambda}{2n_p}, W_p\right)\cdot\left(1+\log(1+\delta)+\frac{\log \varepsilon^{-1}}{\log n}\right)\cdot \log n\right),$$

where $W_p = (1+\delta)^{\lceil \log n_p\rceil} n_p^{c''}\lceil 1/\varepsilon\rceil$*,* c'' *is a constant,* $s^* = s\left(\frac{\varepsilon}{6\lceil \log 2n_p\rceil}, \frac{\lambda}{4n_p}, W_p\right)$*, and* $8n/3 \leqslant n_p \leqslant 8n$*.*

Proof. We first prove that the output of the algorithm is an ε-coreset if every non-outer ALG_S instance outputs an ε_s-coreset of its input for some $\varepsilon_s \leqslant \varepsilon/(6\lceil \log n\rceil)$ and the outer ALG_S instance outputs an $(\varepsilon/3)$-coreset of its input. We prove the following by induction on level number: every node at level ℓ contains a $(\sum_{i=1}^{\ell}\binom{\ell}{i}\varepsilon_s^i)$-coreset of the leaf nodes in its subtree. In the base case, a node at level 1 contains an ε_s-coreset of its input trivially. An ALG_S instance A at level i gets as input two sets, say C'_w and C''_w, each of which is a $(\sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\varepsilon_s^i)$-coreset for the leaf nodes in their respective nodes' subtrees. Hence, $C'_w \cup C''_w$ is a $(\sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\varepsilon_s^i)$-coreset for leaf nodes in the subtree rooted at A by Lemma 5. Now, A outputs an ε_s-coreset of $C'_w \cup C''_w$, hence by Lemma 6, its output is an $(\varepsilon_s + (1+\varepsilon_s)\sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\varepsilon_s^i)$-coreset of the leaf nodes in its subtree, which, by Lemma 8, means a $(\sum_{i=1}^{\ell}\binom{\ell}{i}\varepsilon_s^i)$-coreset. This completes the induction step. Hence, the root node, which is at level $\lceil \log n\rceil$, contains $(\sum_{i=1}^{\lceil \log n\rceil}\binom{\lceil \log n\rceil}{i}\varepsilon_s^i)$-coreset. Now, since $\varepsilon_s \leqslant \varepsilon/(6\lceil \log n\rceil)$, by Lemma 9, the output at the root is an $(\varepsilon/3)$-coreset. The outer ALG_S instance outputs an $(\varepsilon/3)$-coreset of this, hence, by Lemma 6, the final output is an $(2\varepsilon/3 + \varepsilon^2/9)$-coreset, which is an ε-coreset of all points.

Recall that the running time of ALG_S is $t(n_s, \varepsilon_s, \lambda_s, W_s)$ to compute an ε_s-coreset with probability at least $1-\lambda_s$, where n_s is the number of points in the input. Our output success probability will depend on λ_s, and ε depends on ε_s as proved in the previous paragraph. We will need $\varepsilon_s \leqslant \varepsilon/(6\lceil \log n\rceil)$ and $\lambda_s \leqslant \lambda/(2n)$, so these depend on n, which can change a lot over time. We now show how to maintain these guarantees for ε_s and λ_s after each update.

Towards this, we need a little tweak to our algorithm and an additional maintenance routine that we call the *refresher*. The algorithm works in phases. The refresher routine maintains two *refresh* pointers that always point to consecutive leaf nodes, say r_1 and r_2.

The refresh pointers are reset after the end of a phase as follows. If the number of leaf nodes is a power of 2, then r_1 and r_2 point to the two leftmost leaf nodes, otherwise they point to the two leftmost leaf nodes at the level above the lowest level. Assume, for completeness, that the very first phase ends after receiving two points, so the tree is just two leaf nodes and their parent as the root.

For each subsequent phase, let n_0 be the value of n at the beginning of the phase. Each phase ends after $n_0/2$ updates, and we set $n_p = 4n_0$. This guarantees that n_p is greater than n throughout the whole phase and even the next phase (details appear below). After receiving an update, we rerun all the ALG_S instances on the leaf-to-root path starting at r_1 and r_2 (at most $2\lceil \log n \rceil$ such instances). This is the refresher routine. Then we move the refresh pointers to the next two leaf nodes on the right. If we reach the right end, then we go to the next level if it exists, otherwise we stop. If we stop, then we achieved the goal of (re-)running all the ALG_S instances that are present at the end of the phase at least once in this phase (this will become clearer below). After the refresher routine, we execute the update which affects at most two leaf nodes. We rerun all the ALG_S instances that are affected by this update, again, at most $2\lceil \log n \rceil$ such instances. So in total, at most $4\lceil \log n \rceil$ of non-outer ALG_S instances are run after an update and one outer instance, which explains the $\log n$ factor in the update time. We now explain the parameters used in the ALG_S instances. For all the non-outer ALG_S instances, we use $\varepsilon_s = \varepsilon/(6\lceil \log n_p \rceil)$ and $\lambda_s = \lambda/(2n_p)$. (This explains the ε_s and λ_s parameters of the functions t and s in the theorem statement.) Note here that the running time of the outer instance is going to be less than any non-outer instance because t is non-increasing in ε_s and λ_s.

As we use $n_p = 4n_0$ and there could be at most $n_0/2$ insertions in a phase, the final value of n is at most $3n_0/2$, and, thus, n_p is always greater than n. In fact, crucially, n_p is an upper bound on n for even the next phase; in the next phase, $n \leqslant n_0 + n_0/2 + (n_0 + n_0/2)/2 = 9n_0/4 \leqslant n_p$. Also, in the current phase, $n_0/2 \leqslant n \leqslant 3n_0/2$, hence $8n/3 \leqslant n_p \leqslant 8n$, as required (cf. the theorem statement).

We now prove that any non-outer ALG_S instance uses $\varepsilon_s \leqslant \varepsilon/(6\lceil \log n \rceil)$ and $\lambda_s \leqslant \lambda/(2n)$ at any time instant. Let L be the set of leaf nodes at the beginning of the phase; therefore, $|L| = n_0$. An ALG_S instance that exists at the end of the phase is either on the leaf-to-root path for some leaf in L or it was created/updated in this phase. At the end of the phase, the refresh pointers will hit all surviving leaf nodes in L; the argument is as follows. Each phase lasts for $n_0/2$ updates, $|L| = n_0$, and we move the two refresh pointers to the right on next two leaf nodes after each update. Importantly, new leaf nodes are added only to the right of the rightmost leaf node at the lowest level, and hence, the refresher routine will have hit all surviving leaf nodes in L before hitting a newly created leaf node.

This shows that, in any case (being either hit by a point update or by the refresher routine), each ALG_S instance is run with $n_p = 4n_0$, setting up these instances for the next phase. This means that at any time instant, each ALG_S instance was created/updated in the current phase or created/updated in the previous phase, thus showing that $\varepsilon_s \leqslant \varepsilon/(6\lceil \log n \rceil)$ and $\lambda_s \leqslant \lambda/(2n)$ for all ALG_S instances at all times.

At any time instant, there are at most n non-outer instances of ALG_S, each with success probability at least $1 - \lambda/(2n)$, and the outer ALG_S instance has success probability at least $1 - \lambda/2$. Hence, the final success probability is at least $1-\lambda$ by the union bound over these $n + 1$ instances.

How to handle weights

We will need one further tweak to argue that each weight ever encountered by the algorithm can be stored using $O(1+\log(1+\delta)+\log(1/\varepsilon)/\log n)$ words, which also explains that factor in the update time. By assumption, an insertion or weight update comes with a weight that is a fraction with the numerator and the denominator bounded by n^c for some fixed constant c. After receiving such an update, we approximate the weight by a fraction that has numerator bounded by $n_p^{c'}\lceil 1/\varepsilon\rceil$, where $c'=2c+1$ is also a fixed constant, and the denominator is equal to $n_p^{c+1}\lceil 1/\varepsilon\rceil$[5]. The change in the cost due to this approximation is at most ε/n_p times the original cost; hence, by the linearity of the cost function, the output coreset quality is affected by at most an additive factor of $O(\varepsilon/n)$. More formally, the following claim holds by Lemma 11 and using $b/d\leqslant\varepsilon/n_p$ below (think of D below as cost).

▷ Claim 15. Let $d=n_p^{c+1}\lceil 1/\varepsilon\rceil$. Given a rational number a/b, where a and b are integers, $a\leqslant n_p^c$ and $b\leqslant n_p^c$, let $f=\lceil ad/b\rceil$. Then $f\leqslant n_p^{2c+1}\lceil 1/\varepsilon\rceil$ and $(f/d)D\in[1\pm\varepsilon/n_p](a/b)D$ for any nonnegative real D.

Recall that due to the refresher routine, at any time instant, the denominator of the weight at any leaf node can be one of the two: $n_p^{c+1}\lceil 1/\varepsilon\rceil$ or $n_{pp}^{c+1}\lceil 1/\varepsilon\rceil$, where n_{pp} is the value of n_p for the previous phase. When the two children of an internal node use different denominators, this complicates our rounding scheme. Thus, when taking a union of the children's sets at an internal node, for each weight, we make its numerator an integer and the denominator equal to $(n_pn_{pp})^{c+1}\lceil 1/\varepsilon\rceil$, which is a common multiple of $n_p^{c+1}\lceil 1/\varepsilon\rceil$ and $n_{pp}^{c+1}\lceil 1/\varepsilon\rceil$ – the only possible denominators of an input weight after rounding. Next, we run the ALG_S instance with integer weights as given by the numerator, then (implicitly) dividing the output weights by the denominator $(n_pn_{pp})^{c+1}\lceil 1/\varepsilon\rceil$ afterwards. Since each ALG_S instance can increase the total weight by at most a factor of $1+\delta$, the sum of the numerators of all weights at level i is always bounded by $n(1+\delta)^i(n_pn_{pp})^{c'}\lceil 1/\varepsilon\rceil$. Since $i\leqslant\lceil\log n\rceil$ and $n_{pp}=\Theta(n_p)$, there exists a constant c'', such that the sum of the numerators of all weights at any level i and all the possible numerators and denominators are bounded by $(1+\delta)^{\lceil\log n_p\rceil}n_p^{c''}\lceil 1/\varepsilon\rceil=:W_p$, and hence, can be stored in $O(1+\log(1+\delta)+\log(1/\varepsilon)/\log n)$ words as desired (see the beginning of the paragraph before Claim 15). This also justifies the W_s parameters of the functions t and s in the theorem statement.

Now we put everything together. The outer ALG_S instance outputs a weighted set of size at most $s\left(\frac{\varepsilon}{3},\frac{\lambda}{2},W_p\right)$. This set is an ε-coreset with probability at least $1-\lambda$, which we proved by a union bound over all ALG_S instances. We set $s'=s\left(\frac{\varepsilon}{6\lceil\log n_p\rceil},\frac{\lambda}{2n_p},W_p\right)$, which is the threshold for computing a coreset at each internal node, i.e., (recall that) if the number of points at an internal node is greater than s', then we run ALG_S to compute a coreset. An upper bound on the threshold for the current phase and the previous phase is $s^*=s\left(\frac{\varepsilon}{6\lceil\log 2n_p\rceil},\frac{\lambda}{4n_p},W_p\right)$ because the n_p value for the previous phase can be at most twice that of the current phase. Then the worst-case update time is dominated by the non-outer

[5] The static algorithm ALG_S expects integer-weighted input and outputs integer-weighted points, whereas our dynamic algorithm handles fractional weights. If fractional weights are naïvely stored in our dynamic algorithm, then at internal nodes, combining two fractions may result in larger magnitude numbers. E.g., naïvely handling two points with weights a/b and c/d so as to be used in ALG_S results in weights $ad/(bd)$ and $bc/(bd)$. Thus, at level i, the numerators and denominators may be as large $(\mathrm{poly}(n))^{2^i}$. Note that some rounding would be needed even if ALG_S can handle rational weights, because its output may be points with rational weights having much larger magnitude; e.g., even if the output magnitude is about only quadratic in that of the input, the blowup near the root in our dynamic algorithm would be nth power of the input. In fact, we do this rounding in the proof of Theorem 2.

ALG_S instances, each running in time $t\left(2s^*, \frac{\varepsilon}{2\lceil\log n_p\rceil}, \frac{\lambda}{2n_p}, W_p\right)$, and we run $O(\log n)$ of these after receiving an update. An additional factor of $1+\log(1+\delta)+\log(1/\varepsilon)/\log n$ appears because each weight may need memory worth $O(1+\log(1+\delta)+\log(1/\varepsilon)/\log n)$ words, and we need constant time to access each memory word. ◀

Before proving the concrete bounds for k-median and k-means that are stated in Theorem 2, we prove a weaker theorem that is a direct consequence of Theorem 14 using the static algorithm of Chen [9].

▶ **Theorem 16.** *For the k-median and k-means problems, there is a fully-dynamic algorithm that maintains a set of cardinality $O(\varepsilon^{-2}k\log^2(n/\varepsilon)(k\log n+\log(1/\lambda)))$, that is an ε-coreset with probability at least $1-\lambda$, and has worst-case update time*

$$O\left(\varepsilon^{-2}k^2\log^3 n\log^2\frac{n}{\varepsilon}\log\frac{n}{\lambda}\left(k\log n+\log\frac{n}{\lambda}\right)\log\log\frac{n}{\varepsilon}\left(1+\frac{\log\varepsilon^{-1}}{\log n}\right)\right).$$

Ignoring the $\log\log n$ factors, for $\lambda=\Omega(1/\operatorname{poly}(n))$ and $\varepsilon=\Omega(1/\operatorname{poly}(n))$, the coreset cardinality is $O(\varepsilon^{-2}k^2\log^3 n)$, and the worst-case update time is $O(\varepsilon^{-2}k^3\log^7 n)$.

Proof. Chen's algorithm takes in an integer weighted set and outputs also an integer weighted set. Its output has the same total weight as the input, so $\delta=0$ (see Theorem 14). Also, for Chen's algorithm, $s(\varepsilon_s,\lambda_s,W_s)=O(\varepsilon_s^{-2}k(k\log n+\log(1/\lambda_s))\log^2 W_s)$ and the running time $t(n_s,\varepsilon_s,\lambda_s,W_s)=O(n_sk\log(1/\lambda_s)\log\log W_s)$ (see Theorems 3.6 and 5.5 in Chen [9]), which is dominated by the computation of a bicriteria approximation. Note that both s and t are well-behaved. Using $W_p=O(\operatorname{poly}(n)/\varepsilon)$,

$$s^*=O\left(\varepsilon^{-2}k\log^2 n\log^2\frac{n}{\varepsilon}\left(k\log n+\log\frac{n}{\lambda}\right)\right),$$

and $\delta=0$ in Theorem 14 gives the desired bounds using the functions t and s above. ◀

Now we use the result of Braverman et al. [4] to get better bounds as stated in Theorem 2 in the introduction section. Unfortunately, we cannot use Theorem 14 as a complete black box for this because in this case, on integer weighted input, ALG_S does not output an integer weighted coreset. The proof of the following theorem is thus an extension of the proof of Theorem 14.

▶ **Theorem 2.** *For the k-median and k-means problems, there is a fully-dynamic algorithm that maintains a set of cardinality $O(\varepsilon^{-2}k(\log n\log k\log(k\varepsilon^{-1}\log n)+\log(1/\lambda)))$, that is an ε-coreset with probability at least $1-\lambda$, and has worst-case update time*

$$O\left(\varepsilon^{-2}k^2\log^5 n\log^3 k\log^2(\varepsilon^{-1})(\log\log n)^3\right),$$

assuming that $\varepsilon=\Omega(1/\operatorname{poly}(n))$ and $\lambda=\Omega(1/\operatorname{poly}(n))$.

Proof. Our dynamic algorithm expects to have at its disposal a static algorithm ALG_S that takes integer-weighted input and outputs an integer-weighted coreset. Since the algorithm of Braverman et al. that we use as ALG_S outputs on *integer* weighted input a coreset with *fractional* weights, we need some modifications. Hence, before ALG_S is ready to be used in the dynamic algorithm, we round its output to turn it into integers.

Weight-Rounding Modifications for $\mathbf{ALG}_S$

- Let input to ALG_S be Y_w which is a set of n_s points with integer weights $w(1), \ldots, w(n_s)$.
- We run ALG_S on the same points with scaled weights $s'w(1), \ldots, s'w(n_s)$, where s' is the desired cardinality of the output coreset (which is the same as the threshold for computing a coreset at an internal node in this case). We set s' later in a such a way that it can be computed by our dynamic algorithm. This step of multiplying input weights by s' is done to make sure that each of the fractional weights output by ALG_S is at least 1 (see Line 6 of Algorithm 2 in Braverman et al. [4]).
- Let the output C_w of ALG_S be a weighted set of s' points with fractional weights $w_o(1), \ldots, w_o(s')$. Using the rounding strategy of Lemma 12, round these fractional weights to have an integer numerator and the denominator equal to $\lceil (\log n_p)/\varepsilon \rceil$ to get weights $\tilde{w}(1), \ldots, \tilde{w}(s')$, where n_p is as defined in the proof of Theorem 14. Formally, for $i \in \{1, \ldots, s'\}$:

$$\tilde{w}(i) = \lfloor w_o(i) \rfloor + \frac{\left\lfloor (w_o(i) - \lfloor w_o(i) \rfloor) \left\lceil \frac{\log n_p}{\varepsilon} \right\rceil \right\rfloor}{\left\lceil \frac{\log n_p}{\varepsilon} \right\rceil}.$$

  Since $w_o(i) \geqslant 1$, by Lemma 12, for any real $D \geqslant 0$, we have $\tilde{w}(i)D \in [1 \pm \varepsilon/\log n_p] w_o(i) D$.
- Hence, by the linearity of the cost function, C_w with weights $\tilde{w}(1)/s', \ldots, \tilde{w}(s')/s'$ is an $(\varepsilon_s + 2\varepsilon/\log n_p)$-coreset of Y_w with weights $w(1), \ldots, w(n_s)$ if C_w with weights $w_o(1), \ldots, w_o(s')$ is an ε_s-coreset of Y_w with weights $s'w(1), \ldots, s'w(n_s)$. Note that $\tilde{w}(i)/s'$ can be represented as a fraction with an integer numerator and denominator equal to $s'\lceil (\log n_p)/\varepsilon \rceil$.
- The additive loss of $2\varepsilon/\log n_p$ in the coreset quality due to this rounding is tolerable because every non-outer ALG_S instance will be run with $\varepsilon_s = O(\varepsilon/\log n_p)$[6]. Hence, the coreset quality at internal nodes will always be $O(\varepsilon_s + \varepsilon/\log n_p) = O(\varepsilon/\log n)$, as desired.
- This rounding ensures that on integer-weighted input with total weight W, the output weights of ALG_S are fractions with integer numerator bounded by $(1+\delta)Ws'\lceil (\log n_p)/\varepsilon \rceil$ and integer denominator equal to $s'\lceil (\log n_p)/\varepsilon \rceil$. Here, $1+\delta$ is the factor by which ALG_S can increase the total weight.

To handle rational weights in the dynamic algorithm, we first proceed as described in the paragraph on how to handle weights in the proof of Theorem 14. Recall that we assume that each insertion or weight update by the adversary comes with a weight that is a fraction with the numerator and the denominator bounded by n^c for some fixed constant c, and we set $c' = 2c+1$. Also, each leaf node was created/updated in the current phase or created/updated in the previous phase and thus uses the value either n_p or n_{pp}, where n_{pp} is the value of n_p for the previous phase. We then showed the following. At any time instant, the weight of the point at a leaf node is rounded in such a way that the numerator is bounded by $n_p^{c'}\lceil 1/\varepsilon \rceil$ and the denominator is equal to $n_p^{c+1}\lceil 1/\varepsilon \rceil$, or the numerator is bounded by $n_{pp}^{c'}\lceil 1/\varepsilon \rceil$ and the denominator is equal to $n_{pp}^{c+1}\lceil 1/\varepsilon \rceil$. Due to this rounding, the output coreset quality is affected by at most an additive factor of $\max\{2\varepsilon/n_p, 2\varepsilon/n_{pp}\} = O(\varepsilon/n)$. We now prove the following more general statement towards the current proof.

---

[6] If we go for smaller additive loss, say ε/n_p, the denominators of resulting numbers due to this rounding would become exponential in n_p. And if we go for a larger additive loss, it would worsen the coreset quality at non-outer instances to $\omega(\varepsilon/\log n_p)$ resulting in the quality of the output coreset worse than ε.

▶ **Lemma 17.** *At any time instant, every weight at a node at level i has an integer numerator and a denominator that is a factor of $(n_p n_{pp})^{c+1}\lceil 1/\varepsilon\rceil(s'_p s'_{pp}\lceil(\log n_p)/\varepsilon\rceil\lceil(\log n_{pp})/\varepsilon\rceil)^i =: D(i)$, where s'_p and s'_{pp} are values of the threshold s' in the current and the previous phase, respectively.*

Proof. We prove this statement by induction over the sequence of nodes updated by the algorithm.

In the base case, the first ever node update will be due to creation of a leaf node, and the weight will have denominator $n_p^{c+1}\lceil 1/\varepsilon\rceil$. Next we discuss the induction step. Let the update be on a node at level i, so we run the *modified* ALG_S instance with all weights having a denominator that is a factor of $D(i-1)$, which is true by induction hypothesis. Then, since the modified ALG_S adds a factor of $s'_p\lceil(\log n_p)/\varepsilon\rceil$ to the denominator, all resulting output weights have a denominator that is a factor of $D(i-1)s'_p\lceil(\log n_p)/\varepsilon\rceil$, which is a factor of $D(i)$. This finishes the induction step for the case when the node update is not the last of the phase. When the node being updated is the last of the phase, we have to be careful. In this case, we need to show that for all weights in all nodes, n_{pp} or s'_{pp} do not appear in the denominator, as this will set these denominators for the next phase. Towards this, we need the following claim.

▷ Claim 18. Let u be a node at level i. Fix a time instant. Suppose, in the current phase, all nodes in the subtree rooted at u were updated and u was updated after the update of the last-updated leaf node in the subtree. Then the denominator of the weights at u is a factor of $n_p^{c+1}\lceil 1/\varepsilon\rceil(s'_p\lceil(\log n_p)/\varepsilon\rceil)^i$ at the fixed time instant.

We omit the proof of this claim as it can be proved easily by induction on the level number at any fixed time instant.

After the last node update of the phase, every node in the tree has been updated in the current phase and the premise of Claim 18 holds due to the refresher routine. Hence, by Claim 18, after the last node update of the phase, i.e., just before the new phase begins, all denominators at level i are a factor of $n_p^{c+1}\lceil 1/\varepsilon\rceil(s'_p\lceil(\log n_p)/\varepsilon\rceil)^i$. Since n_p and s'_p of this phase will become n_{pp} and s'_{pp} in the next phase, the induction hypothesis stays true for the next phase as well. This finishes the proof of Lemma 17. ◀

Since an ALG_S instance may increase the total weight by at most a factor of $1+\delta$, the sum of the numerators of weights at any level i is at most

$$n_p(1+\delta)^i(n_p n_{pp})^{c'}\left\lceil\frac{1}{\varepsilon}\right\rceil\left(s'_p s'_{pp}\left\lceil\frac{\log n_p}{\varepsilon}\right\rceil\left\lceil\frac{\log n_{pp}}{\varepsilon}\right\rceil\right)^i;$$

this can be seen by an easy induction on the level number. Using this bound, we set the threshold s' in a way similar to that in the proof of Theorem 14: we set

$$s'_p = s\left(\frac{\varepsilon}{6\lceil\log n_p\rceil}, \frac{\lambda}{2n_p}, W_p\right),$$

where

$$W_p = (1+\delta)^{\lceil\log n_p\rceil} n_p^{c_1}\left(k\left\lceil\frac{\log n_p}{\varepsilon}\right\rceil\right)^{c_2\lceil\log n_p\rceil},$$

and c_1 and c_2 are chosen to be large enough constants so that W_p upper bounds the sum of the numerators of all weights at any level. From now onwards, we assume that $\lambda = \Omega(1/\operatorname{poly}(n))$. For ALG_S, the function s is $s(\varepsilon_s, \lambda_s, W_s) = O(\varepsilon_s^{-2}k(\log k \log W_s + \log(1/\lambda_s)))$ and $\delta = O(\varepsilon)$.

Then, using $n_{pp} = \Theta(n_p)$, we get that both s'_p and s'_{pp} are $O\left(\left(k\left\lceil\frac{\log n_p}{\varepsilon}\right\rceil\right)^{c_3}\right)$, where c_3 is a fixed constant (so, independent of c_1 and c_2). Observe that W_p and thus s'_p are determined by the phase and hence can be computed by our algorithm. More concretely, we get that both s'_p and s'_{pp} are

$$O\left(\varepsilon^{-2}k\log^3 n\log k\log\frac{k\log n}{\varepsilon}\right).$$

All possible numerators and denominators encountered by the algorithm are bounded by

$$N := O\left(\operatorname{poly}(n)\left(\frac{k\log n}{\varepsilon}\right)^{O(\log n)}\right),$$

so, can be stored in $m := (\log N)/\log n = O(\log((k\log n)/\varepsilon))$ words.

The running time of ALG_S is $t(n_s, \varepsilon_s, \lambda_s, W_s) = O(n_s k\log(1/\lambda_s)\log\log W_s)$, which, similar to Chen's algorithm, is dominated by computation of a bicriteria approximation. At a non-outer ALG_S instance, $n_s = O(s'_p)$, $\varepsilon_s = O(\varepsilon/\log n_p)$, $\lambda_s = O(\lambda/n_p)$, and $W_s \leqslant W_p$. With every update, $O(\log n)$ instances of ALG_S are run, and an additional m factor appears because a weight may need up to m words. Hence, the worst-case update time assuming $\varepsilon = \Omega(1/\operatorname{poly}(n))$ and $\lambda = \Omega(1/\operatorname{poly}(n))$ is

$$O\left(t\left(s'_p, \frac{\varepsilon}{\log n}, \frac{\lambda}{n}, W_p\right)m\log n\right) = O\left(\varepsilon^{-2}k^2\log^5 n\log k\log^2\left(\frac{k\log n}{\varepsilon}\right)\log\log\left(\frac{k\log n}{\varepsilon}\right)\right)$$

and a looser, easier to parse, bound is $O\left(\varepsilon^{-2}k^2\log^5 n\log^3 k\log^2(1/\varepsilon)(\log\log n)^3\right)$. The output coreset cardinality is

$$s\left(\frac{\varepsilon}{3}, \frac{\lambda}{2}, W_p\right) = O\left(\varepsilon^{-2}k\left(\log n\log k\log\left(\frac{k\log n}{\varepsilon}\right) + \log\frac{1}{\lambda}\right)\right).$$

This finishes the proof of Theorem 2. ◀

3.1 The Binary-Tree Structure

We describe the tree structure in more detail, especially, how insertions and deletions are handled. We always maintain a complete binary tree, in which every level except possibly the lowest is completely filled, and the nodes in the lowest level are packed to the left. We also maintain the property that each internal node has exactly two children. Our data structure behaves somewhat like a heap, though a crucial difference is that we do not have keys. This structure supports insertion and deletion of a leaf node. Insertion of a new leaf-node ℓ works as follows.

- If the current number of leaf nodes is a power of 2, then let v be the leftmost leaf node,
- Else let v be the leftmost leaf node in the level above the lowest level.
- Let p be v's parent.
- Create a new node u.
- Make p to be u's parent; u replaces v, so if v was p's right (respectively, left) child, then u is now p's right (respectively, left) child.
- Make v to be u's left child and ℓ to be u's right child. This way, ℓ the rightmost leaf node at the lowest level.

Deletion of a leaf-node ℓ works as follows. Let v be the rightmost leaf node at the lowest level, p be v's parent, and v' be v's sibling. Replace ℓ's contents by v's contents and replace p's contents by the contents of v'. Delete v and v'.

3.2 Reducing the Number of Nodes

The tree can be modified to have each leaf node correspond to a set of $\Theta(s')$ points to reduce the additional space used for maintaining this tree (pointers and such). Recall that s' is the threshold for computing a coreset. To reduce the number of nodes in the tree this way, we maintain the invariant that each leaf node, except possibly one, contains a set of size s_ℓ with $s'/2 \leqslant s_\ell \leqslant s'$. To maintain this invariant, we use a pointer p_s that points to a leaf node with less than $s'/2$ elements if such a leaf node exists.

Whenever a point is inserted, we add it to the leaf node, say ℓ_e pointed to by p_s. If ℓ_e now contains at least $s'/2$ points, then we make p_s a null pointer. If p_s was a null pointer already, then we create a new leaf node, say ℓ_n, insert the new point in ℓ_n, and make p_s point to ℓ_n. The new leaf node ℓ_n is inserted in the tree as described in Section 3.1.

Whenever a point is deleted, we check if the leaf node, say ℓ_d that contains it now contains less than $s'/2$ points. If ℓ_d contains less than $s'/2$ points, and p_s points to some leaf node, say ℓ_e, then we move points in ℓ_d into ℓ_e and delete ℓ_d. (Deletion of a leaf node is handled as described in Section 3.1.) If p_s does not point to any leaf node, then we make it point to ℓ_d.

As usual, we recompute all nodes on the affected leaf-to-root path.

4 Lower Bounds

In this section, we show lower bounds. We first see a space lower bound and then a conditional lower bound on the time per operation.

4.1 Space Lower Bound

We show a simple and very general space lower bound. Consider any problem that on input X has to output a feasible solution that is a subset of X. Moreover, if X non-empty, then all feasible solutions are also non-empty. Call such a problem *compliant*. Clearly, computing any bounded approximation for k-median and k-means and the problem of constructing any coreset with bounded quality are compliant. To get a linear space lower bound for fully-dynamic algorithms that solve a compliant problem, we use the communication problem of INDEX. In INDEX_N, Alice's input is an N-bit string and Bob's input is an index $I \in \{1, 2, \ldots, N\}$. Alice sends one message to Bob, and he needs to correctly output the bit at position I. By a well-known communication complexity lower bound, Alice must send a message of size $(1 - H_2(3/4))N \geqslant 2N/11$ bits so that Bob can correctly output with a success probability of $3/4$; here H_2 is the binary entropy function.

▶ **Theorem 19.** *A fully-dynamic algorithm for a compliant problem that works in the presence of an adaptive adversary and has success probability $1 - 1/(8n^2)$ must use space $\Omega(n)$, where n is the current input size.*

Proof. We describe the reduction for any compliant problem in a metric space, such as 1-median or 1-means, but it can be naturally generalized to any compliant problem. Alice defines

$$X = \{j : j\text{th bit in her string } = 1\},$$

and distance between any two points of X to be 1. She runs the fully-dynamic algorithm on X and sends the memory snapshot to Bob. Bob queries for a solution and if X is nonempty, a nonempty solution S_1 would be returned. He deletes the points in S_1 and queries again to get S_2, and so on until $\emptyset$ is returned. There would be at most N such queries. Note that

this works because the algorithm works under an adaptive adversary. If one of the S_ℓs in this process contains I, which is Bob's input for the index problem, then Bob outputs 1, else he outputs 0. In the worst case, Bob makes N queries, where query number i would have failure probability at most $1/(8(N-i+1)^2)$. So overall failure probability by the union bound is at most

$$\sum_{i=1}^{N} \frac{1}{8(N-i+1)^2} \leqslant \frac{1}{8}\sum_{i=1}^{\infty}\frac{1}{i^2} = \frac{1}{8}\frac{\pi^2}{6} \leqslant \frac{1}{4}\,.$$

Alice communicated as many bits as the space usage of the dynamic algorithm. Then, by the INDEX_N lower bound, the space usage of the algorithm is at least $2N/11 \geqslant 2n/11$ bits. ◀

4.2 Conditional Lower Bounds on the Time Per Operation

Now, we show conditional lower bounds on the time per update and query for fully-dynamic k-means algorithms. They are based on the *OMv-conjecture* [18]: You are given an $N \times N$ Boolean matrix M that can be preprocessed in polynomial time. Then, an online sequence of N-dimensional Boolean vectors $v^1, \ldots, v^N$ is presented and the task is to compute each Mv^i (using Boolean matrix-vector multiplication) before seeing the next vector v^{i+1}. The conjecture is that finding all the N answers takes time $\Omega(N^{3-\gamma})$ for any constant $\gamma > 0$. In [18] also the following *OuMv problem* was presented: You are given an $N \times N$ Boolean matrix M that can be preprocessed in polynomial time and an online sequence of Boolean vector pairs $(u^1, v^1), \ldots, (u^N, v^N)$ with the goal to compute each $(u^i)^T M v^i$ (using Boolean matrix-vector multiplication) before seeing the next vector pair (u^{i+1}, v^{i+1}). Under the OMv conjecture, finding N answers for the OuMv problem such that the error probability is at most $1/3$ takes time $\Omega(N^{3-\gamma})$ for any constant $\gamma > 0$. We will show a reduction from the latter problem to prove the following result.

▶ **Theorem 20.** *Let $\gamma > 0$ be a constant. Under the OMv conjecture, for any $\delta > 0$, there does not exist a fully-dynamic algorithm that maintains a $(4-\delta)$-approximation for k-means with amortized update time $O(k^{1-\gamma})$ and query time $O(k^{2-\gamma})$ such that over a polynomial number of updates the error probability is at most $1/3$.*

Proof. For the ease of presentation, we assume that k is even; if k is odd, the construction can be easily adapted. We set $N = k/2$. Given an OuMv instance with $N \times N$ matrix M, we construct the following metric space with distance function d from it:

The metric space U consists of $4N$ points numbered from 1 to $4N$. For any $1 \leqslant i < j \leqslant N$ and $N+1 \leqslant i < j \leqslant 2N$, the distance $d(i,j) = 2$. Furthermore, for $1 \leqslant i \leqslant N$ and $N+1 \leqslant j \leqslant 2N$, the distance $d(i,j) = 1$ if $M_{i,j-N} = 1$, and $d(i,j) = 2$ otherwise. Additionally, all $2N$ points $2N+1, \ldots, 4N$ are at distance 100 from each other and from all the other points.

We use a k-means data structure to solve a $u^T M v$ computation as follows: Initially the set X is empty. When given a vector pair (u, v), let p be the number of ones in v and in u. Note that $p \leqslant 2N$. We insert the points i such that $u_i = 1$ and the points j such that $v_{j-N} = 1$ into X and additionally $2N+1-p$ of the points ℓ with $\ell > 2N$. Thus $|X| = 2N+1 = k+1$. Then we ask a k-means query. Afterwards, we delete the inserted points.

If $u^T M v = 1$, then there exist indices i and j such that $u_i = 1, M_{i,j} = 1$, and $v_j = 1$. Consider the optimal solution that consists of all points in X except for point i. Note that the cost of this solution for the k-means problem is 1.

If $u^T Mv = 0$, then any optimal solution must also consist of $2N + 1 - p$ of the points ℓ with $\ell > 2N$, and all but one of the other points in X. But as none of the points in X has distance smaller than 2 to any other point in X, the cost of the solution is at least 4 for k-means. Thus, any $(4 - \delta)$-approximation for k-means can distinguish between the cases $u^T Mv = 1$ and $u^T Mv = 0$. Hence, the OMv conjecture implies that it takes at least time $\Omega(N^{2-\gamma})$ time to execute the above $2N$ update operations and 1 query operation. This implies the claimed lower bound. ◀

References

1 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 61–72. IEEE Computer Society, 2017.

2 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004.

3 Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms*, pages 301–358, 1980.

4 Vladimir Braverman, Dan Feldman, and Harry Lang. New Frameworks for Offline and Streaming Coreset Constructions. *arXiv e-prints*, 2016. `arXiv:1612.00889`.

5 Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 576–585, International Convention Centre, Sydney, Australia, August 06–11 2017. PMLR.

6 Jaroslaw Byrka, Thomas W. Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2):23:1–23:31, 2017.

7 Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM J. Comput.*, 34(4):803–824, 2005.

8 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002.

9 Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009. `doi:10.1137/070699007`.

10 Vincent Cohen-Addad, Niklas Oskar D Hjuler, Nikos Parotsidis, David Saulpic, and Chris Schwiegelshohn. Fully dynamic consistent facility location. In *Advances in Neural Information Processing Systems 32*, pages 3255–3265. Curran Associates, Inc., 2019. URL: `http://papers.nips.cc/paper/8588-fully-dynamic-consistent-facility-location.pdf`.

11 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

12 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 569–578, 2011.

13 Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1434–1453. SIAM, 2013.

14 Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 209–217. ACM, 2005.

15 Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location, 2008. `arXiv:0809.2554`.

16 Sariel Har-Peled and Akash Kushal. Smaller coresets for k-median and k-means clustering. *Discret. Comput. Geom.*, 37(1):3–19, 2007.

17 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 291–300. Association for Computing Machinery, 2004.

18 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Symposium on Theory of Computing (STOC)*, pages 21–30, 2015. `doi:10.1145/2746539.2746609`.

19 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 731–740. ACM, 2002.

20 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

21 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Comput. Geom.*, 28(2-3):89–112, 2004.

22 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM J. Comput.*, 45(2):530–547, 2016.

23 Ramgopal R. Mettu and C. Greg Plaxton. Optimal time bounds for approximate clustering. *Mach. Learn.*, 56(1-3):35–60, 2004.

24 Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 802–813. IEEE Computer Society, 2018.

A Proof of Lemma 8

▶ **Lemma 8.** *For any positive integer ℓ and $\alpha \in \mathbb{R}_+$, we have*

$$\alpha + (1+\alpha)\sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\alpha^i = \sum_{i=1}^{\ell}\binom{\ell}{i}\alpha^i .$$

Proof.

$$\begin{aligned}
\alpha + (1+\alpha)\sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\alpha^i &= \alpha + \sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\alpha^i + \sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\alpha^{i+1} \\
&= \binom{\ell-1}{0}\alpha + \sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\alpha^i + \sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\alpha^{i+1} \\
&\qquad \text{using the fact } \binom{\ell-1}{0} = 1 \\
&= \binom{\ell-1}{0}\alpha + \sum_{i=1}^{\ell-1}\binom{\ell-1}{i}\alpha^i + \sum_{i=2}^{\ell}\binom{\ell-1}{i-1}\alpha^i \\
&\qquad \text{change of index in the second summation}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{\ell-1} \binom{\ell-1}{i} \alpha^i + \sum_{i=1}^{\ell} \binom{\ell-1}{i-1} \alpha^i \\
&\qquad\qquad \text{incorporating first term in second summation} \\
&= \sum_{i=1}^{\ell} \binom{\ell-1}{i} \alpha^i + \sum_{i=1}^{\ell} \binom{\ell-1}{i-1} \alpha^i \\
&\qquad\qquad \text{using the fact } \binom{\ell-1}{\ell} = 0 \\
&= \sum_{i=1}^{\ell} \left(\binom{\ell-1}{i} + \binom{\ell-1}{i-1} \right) \alpha^i \\
&= \sum_{i=1}^{\ell} \binom{\ell}{i} \alpha^i ,
\end{aligned}
$$

where we use $\binom{\ell}{i} = \binom{\ell-1}{i} + \binom{\ell-1}{i-1}$ in the last step. ◀

Dynamic Matching Algorithms in Practice

Monika Henzinger
University of Vienna, Faculty of Computer Science, Austria
monika.henzinger@univie.ac.at

Shahbaz Khan
Department of Computer Science, University of Helsinki, Finland
shahbaz.khan@helsinki.fi

Richard Paul
University of Vienna, Faculty of Computer Science, Austria
richard.paul@univie.ac.at

Christian Schulz
University of Vienna, Faculty of Computer Science, Austria
christian.schulz@univie.ac.at

Abstract

In recent years, significant advances have been made in the design and analysis of fully dynamic maximal matching algorithms. However, these theoretical results have received very little attention from the practical perspective. Few of the algorithms are implemented and tested on real datasets, and their practical potential is far from understood. In this paper, we attempt to bridge the gap between theory and practice that is currently observed for the *fully dynamic maximal matching problem*. We engineer several algorithms and empirically study those algorithms on an extensive set of dynamic instances.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Matching, Dynamic Matching, Blossom Algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.58

Supplementary Material Source code and instances are available at `https://github.com/schulzchristian/DynMatch`.

Funding The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) /ERC grant agreement No. 340506.

1 Introduction

The matching problem is one of the most prominently studied combinatorial graph problems having a variety of practical applications. A matching $\mathcal{M}$ of a graph $G = (V, E)$ is a subset of edges such that no two elements of $\mathcal{M}$ have a common end point. Many applications require matchings with certain properties, like being maximal (no edge can be added to $\mathcal{M}$ without violating the matching property) or having maximum cardinality. These problems can be solved in polynomial time. For example, Micali and Vazirani [31] compute a maximum cardinality matching in $O(m\sqrt{n})$ time. For the weighted case, the fastest algorithm is by Galil et. al [19] requiring $O(mn \log n)$ time which improves the $O(n^3)$ time algorithm [18] for sparse graphs.

However, often the underlying graphs change over time, e.g., edges are inserted or deleted in the graph as the time progresses. For example, new relations between objects of a network may be created or removed over time (for example [30]). Even though the matching problem can be solved in polynomial time, computing a new matching from scratch every time the graph changes is an expensive task on huge networks, as this ignores the previously computed

information on the given instance. Hence, in the recent years significant advances have been made in the design and analysis of fully dynamic maximal matching algorithms. These theoretical algorithmic ideas have received very little attention from the practical perspective. Only a few of the dynamic algorithms are implemented and tested on real datasets, and hence their practical potential is far from being understood.

Contribution and Outline. In this paper, we start to bridge the gap between theory and practice that is currently observed for the *fully dynamic maximal matching problem*. We engineer several dynamic maximal matching algorithms as well as an algorithm that is able to maintain the maximum matching. To this end, we look at an algorithm due to Baswana, Gupta and Sen [4], which performs edge updates in $O(\sqrt{n})$ time and maintains a 2-approximate maximum matching, the algorithm of Neiman and Solomon [33], which takes $O(\sqrt{n+m})$ time to maintain a 3/2-approximate maximum matching, as well as two *novel* dynamic algorithms: a random walk-based algorithm as well as a dynamic algorithm that searches for augmenting paths using a (depth bounded) blossom algorithm. Without depth bound, the latter algorithm is able to maintain a maximum matching. We perform extensive experiments comparing the performance of these algorithms on the real-world and artificially generated instances. Experiments indicate that maintaining optimum matchings can be done much more efficiently than the naive algorithm that recomputes maximum matchings from scratch (more than an order of magnitude faster). Second, all non-optimum dynamic algorithms that we consider in this work are able to maintain near-optimum matchings in practice while being multiple orders of magnitudes faster than the naive optimum dynamic algorithm.

2 Preliminaries

2.1 Basic Concepts

Let $G = (V = \{0, \ldots, n-1\}, E)$ be an *undirected graph* without parallel edges and self-loops. We set $n = |V|$, and $m = |E|$, $N(v) := \{u : \{v, u\} \in E\}$ denotes the *neighbors* of v. The degree of a vertex v is $d(v) := |N(v)|$. A matching $\mathcal{M} \subseteq E$ in a graph is a set of edges without common vertices. The *cardinality* or *size* of a matching is simply the cardinality of the edge subset $\mathcal{M}$. We call a matching *maximal*, if there is no edge in E that can be added to $\mathcal{M}$. A *maximum cardinality matching* $\mathcal{M}_{\text{opt}}$ is a matching that contains the largest possible number of edges of all matchings. An α-*approximate* maximum matching is a matching, that contains at least $\frac{|\mathcal{M}_{\text{opt}}|}{\alpha}$ edges. A vertex is called *free* or *unmatched* if it is not incident to an edge of the matching. Otherwise, we call it *unfree* or *matched*. For a matched vertex u with $\{u, v\} \in \mathcal{M}$, we call vertex v the *mate* of u, which we denote as $\text{mate}(u) = v$. For an unmatched vertex u, we define $\text{mate}(u) = \perp$. An *augmenting path* is defined as a cycle-free path in the graph G, that starts and ends on a *free* vertex and where edges from $\mathcal{M}$ alternate with edges from $E \setminus \mathcal{M}$. The *trivial augmenting path* is a single edge, that has both its endpoints unmatched. Throughout this paper, we call such an edge a *free* edge. If we take an augmenting path and resolve it by matching every unmatched edge and unmatching every matched edge, we increase the cardinality of the matching by one. Any matching without *augmenting paths* is a maximum matching [5] and any matching with no augmenting paths of length at most $2k-3$ is a $(k/(k-1))$-approximate maximum matching [23]. Hence, a maximal matching having no augmenting paths of length one (or free edges) is a 2-approximate maximum matching. Throughout the paper, we omit the inverse Ackermann function from complexity statements.

Our focus in this paper are *fully dynamic graphs*, where the number of vertices is fixed, but edges can be added and removed. All the algorithms evaluated can handle edge insertions as well as edge deletions. In the following, Δ denotes the maximum degree that can be found in any state of the dynamic graph.

2.2 Related Work

Computing large or maximum matchings in graphs is a well researched topic. Edmonds [17] gave an algorithm that can compute a maximum cardinality matching in a static graph in time $O(mn^2)$. This result was later improved to $O(mn^{0.5})$ by Micali and Vazirani [31]. Recently, algorithms use simple data reductions rules such as [25] to speed up computations, or shrink-trees instead of blossoms [16] to speed up computations in static graphs. In practice, these algorithms can still be time consuming for many applications involving large graphs. Hence, several near linear time approximation algorithms exist in practice such as the local max algorithm [11], the path growing algorithm [15] and the global paths algorithm [28]. As the focus of this work are dynamic graphs, we refer the reader to the quite extensive related work section of [16] for more recent static matching algorithms.

In the dynamic setting, the maximum matching problem has been prominently studied ensuring α-approximate guarantees. A major exception is the randomized algorithm by Sankowski [36] which maintains a maximum matching in $O(n^{1.495})$ update time. One can trivially maintain a maximal (2-approximate) matching in $O(n)$ update time by resolving all trivial augmenting paths of length one. Ivković and Llyod [24] designed the first fully dynamic algorithm to improve this bound to $O((n+m)^{\sqrt{2}/2})$ update time. Later, Onak and Rubinfeld [34] presented a randomized algorithm for maintaining a $O(1)$-approximate matching in a dynamic graph that takes $O(\log^2 n)$ expected amortized time for each edge update. This result led to a flurry of results in this area. Baswana, Gupta and Sen [4] improved the approximation ratio of [34] from $O(1)$ to 2 and the amortized update time to $O(\log n)$. Further, Solomon [38] improved the update time of [4] from amortized $O(\log n)$ to *constant*. However, the first deterministic data structure improving [24] was given by Bhattacharya et al. [8] maintaining $(3+\epsilon)$ approximate matching in $\tilde{O}(min(\sqrt{n}, m^{1/3}/\epsilon^2))$ amortized update time, which was further improved to $(2+\epsilon)$ requiring $O(\log n/\epsilon^2)$ update time by Bhattacharya et al. [9]. Recently, Bhattacharya et al. [7] achieved the first $O(1)$ amortized update time for a deterministic algorithm but for a weaker approximation guarantee of $O(1)$. For worst-case bounds, the best results are by Gupta and Peng [21] requiring $O(\sqrt{m}/\epsilon)$ update time for $(1+\epsilon)$ approximation, Neiman and Solomon [33] requiring $O(\sqrt{m})$ update time for 3/2 approximation, Bernstein and Stein [6] requiring $m^{1/4}/\epsilon^{2.5}$ for $(3/2+\epsilon)$ approximation. Recently, Charikar and Solomon [12], and Arar et al. [2] (using [10]), independently presented the first algorithms requiring $O(poly \log n)$ worst-case update time both maintaining $(2+\epsilon)$ approximation. Recently, Grandoni et al. [20] gave an incremental matching algorithm that achieves a $(1+\epsilon)$-approximate matching in constant deterministic amortized time. Despite this variety of different algorithms, to the best of our knowledge, there has been no effort made so far, to engineer and evaluate these algorithms on real-world instances.

3 Algorithms

We now present the fully dynamic algorithms for the maximal matching problem under consideration. We implemented and tested a variety of simple, combinatorial algorithms that seemed likely to work well in practice. We begin with random walk based dynamic algorithms, followed by dynamic algorithms based on (bounded) augmenting path search and finally

review the algorithms by Baswana, Gupta and Sen [4] and Neiman and Solomon [33]. All of the algorithms are fully dynamic. Throughout this section, we provide a brief description of these algorithms and their implementation. In each case, we explain how we handle initialization, edge insertions and edge deletions separately.

3.1 Random Walk-based Algorithms

In general finding long augmenting paths in networks is an expensive step. The main idea of the random walk based methods proposed in this section is to use random walks in order to detect augmenting paths, and hence to improve the size of the matching. We start by explaining the general idea to use random walks for finding augmenting paths and then explain how we handle edge insertions and deletions.

3.1.1 Random Walks For Augmenting Paths

The algorithm works as follows: we start at a free vertex u and randomly choose a neighbour w of u. If this neighbour is free, then we match the edge (u, w) and our random walk stops. If w is matched, we unmatch $(w, \text{mate}(w))$ and match (u, w). Note that $u \neq \text{mate}(w)$ since u is free in the beginning and therefore $\text{mate}(u) = \perp$, but $\text{mate}(\text{mate}(w)) = w$ and $w \neq \perp$. Afterwards, the previous mate of w is free. Hence, we continue our random walk at this vertex. Our random walk performs $O(\frac{1}{\epsilon})$ steps (see below). Here, ϵ basically controls the quality of the matching (see below). Since picking a random neighbor can be done in constant time, the overall time for the random walk update algorithm is $O(\frac{1}{\epsilon})$. Note that the length of the random walk is a natural parameter of the algorithm that we will investigate in the experimental evaluation.

Also note that if the algorithm does not end by matching two free vertices, the matching may not be maximal even if it was initially – this can be the case if the vertex freed last is incident to a free vertex. There are multiple possibilities to fix this. Our default is to undo all changes that have been done in this case. The overall running time of a random walk is then $O(1/\epsilon)$. Another possibility is Δ-*settling:* The algorithm tries to settle visited vertices. The algorithm scans through their neighbors to find a free vertex and stops if once it was successful or the number of steps exceeds $1/\epsilon$. If the random walk was not successful, the algorithm tries to match the last vertex touched by the random walk by scanning its neighbors instead of undoing all changes. This also ensures that the matching is maximal but requires $O(\Delta)$ additional time per visited vertex. The running time of the Δ-settling random walk is then $O(\Delta/\epsilon)$.

We now explain how we perform edge insertions and deletions.

Edge Insertion. Our algorithm handles edge insertions as follows: when inserting an edge (u, v), if both the endpoints are free, we match it. Note that the simple algorithms stops here if at least one of the endpoints is not free. The random walk based algorithms try to improve insertion by doing the following: If both endpoints are matched, thus prohibiting to match the inserted edge, we do nothing. If only one of the endpoints is matched, w.l.o.g let this be u, we unmatch u and $w := \text{mate}(u)$ and match (u, v). We then start a random walk as described above to find augmenting paths from w. If the random walk is unsuccessful to further increase the size of the matching, we undo all changes and restore the matching to the state before we unmatched u and w.

Edge Deletion. Deleting a matched edge (u, v) leaves the two endpoints u and v free. If possible, our algorithm matches them in $O(\Delta)$ time by scanning their neighbors in order to maintain a maximal matching. If u and v cannot be matched and the matching before edge deletion was maximal, then the matching remains maximal. However, a free vertex may be a starting point for an augmenting path of arbitrary length. Hence, we start a random walk as described above from u if it is free and do the same for v.

3.1.2 Analysis

The algorithm can maintain a $(1+\epsilon)$-approximation, if the random walks are of appropriate length and repeated sufficiently often. More precisely, if the algorithm uses random paths of length $2/\epsilon - 1$ and the process is repeated until successful or $\Delta^{2/\epsilon-1} \log n$ times, then with high probability the matching is a $(1+\epsilon)$ approximation of the maximum matching (at each point in time).

▶ **Lemma 1.** *The random walk based algorithm maintains a $(1+\epsilon)$-approximate maximum matching if the length of the walk is $2/\epsilon - 1$ and the walks are repeated $\Delta^{2/\epsilon-1} \log n$ times.*

Proof. If no augmenting path of length $\leq 2/\epsilon - 1$ exists, then the matching is a $(\frac{1/\epsilon+1}{1/\epsilon}) = (1+\epsilon)$-approximate maximum matching. To see this, rewrite the length of the path to $2(1/\epsilon+1)-3$ and set $k = 1/\epsilon+1$ in the approximation lemma above. If there is such a path from a free node, then the probability of finding it is $\geq (\frac{1}{\Delta})^{2/\epsilon-1}$ since one possibility is the that random walker makes the "correct" decision at every vertex of the path. The probability that λ random walks of length $2/\epsilon - 1$ do not find an augmenting path of length $2/\epsilon - 1$ is $\leq (1 - \frac{1}{\Delta^{2/\epsilon-1}})^{\lambda} \leq e^{-\frac{1}{\Delta^{2/\epsilon-1}} \cdot \lambda}$. Thus for $\lambda \geq \Delta^{2/\epsilon-1} \log n$ the probability is $\leq 1/n$. ◀

Parallelization. Note that multiple repetitions of the random walks can be easily parallelized as they are completely independent if changes are made thread-local. If one random walker finds an augmenting path, it is accepted and the other random walkers can be stopped.

3.2 Blossom-based (Optimum) Algorithms

Note that the random walk algorithm also yields a static $(1+\epsilon)$-approximate maximum matching algorithm: use a simple greedy algorithm as initialization and then run the random walks as stated above from the remaining free nodes. However, the amount of repetitions to achieve the approximation is fairly high. Simply, running a modified BFS to find augmenting paths bounded in depth by $2/\epsilon - 1$ from a free node has a theoretically faster running time $O(\Delta^{2/\epsilon-1})$ per free node. Note however that the theoretical bound for the dynamic random walk algorithm is fairly pessimistic: our algorithm stops as soon as *one* augmenting path has been found – this path can also be shorter or in practice there may be multiple possibilities for augmenting paths so that the probability of finding it increases. So the natural question arises, whether a bounded augmenting path search is superior over random walk based methods stated above. Hence, we propose the following dynamic algorithms for the dynamic matching problem.

In most implementations (such as Boost [37]) finding an augmenting path starting from a free node takes $\Omega(n+m)$ running time due to initialization of the data structures of the modified BFS. These data structures are initialized every time an augmenting path search is started. Hence, the observed performance of Edmonds blossom algorithm to find an optimum matching in libraries such as Boost is $\Theta(n(n+m))$ if no algorithm to initialize the matching is used and $O(F(n+m))$ if some greedy algorithm is used as initialization and

F is the number of remaining free nodes after greedy initialization. The later is the reason why in practice greedy initialization strategies generally help to find optimum matchings. However, finding an augmenting path can easily be implemented such that it a) stops as soon as an augmenting path is found, and b) has running time $\Theta(n' + m')$, where n' and m' refers to the number of nodes and edges touched by the augmenting path search modified BFS [39, 29]. The first augmenting path search needs time $O(n + m)$ to initialize the typical data structures. All searches then do book keeping of the changes they made in the data structures and undo them afterwards. Note that this clearly changes the behaviour of the algorithm in practice: if there are many short augmenting paths the algorithm will run much faster than $\Theta(n(n + m))$. The implementation does not change the worst-case complexity, but improves the best case to $O(m)$ [29]. In fact, in our experience the static version of our implementation scales close to linear in m in practice (as there are many short augmenting paths in real world instances). In the following, we always use this variant of augmenting path search and each of the dynamic operations does book keeping to be able to quickly search for augmenting paths.

Edge Insertion. Let (u, v) be the inserted edge. If u and v are free, then we match that edge directly. Otherwise, we start an augmenting path search from u if u is free and from v if v is free. If both u and v are not free, then we perform a breadth first search from u to find a free node reachable via an alternating path. From this node we start an augmenting path search. Note that an augmenting path must use (u, v) as both connected components did not contain an augmenting path with the component before as the algorithm maintains a maximum matching. Also note that the last case will be an expensive step in practice as the algorithm tries to maintain a maximum matching, newly inserted edges will often not result in a new augmenting path and hence the augmenting path search takes $\Theta(n + m)$ time. Without the third case of the algorithm, we call it *unsafe*. That is in case both u and v are not free, the unsafe configuration of the algorithm does nothing.

Not using the unsafe option, the algorithm maintains a maximum matching. This is due to the fact that if the graph did not contain an augmenting path before insertion, the only way we can create one is due to the insertion of the new edge. The first and second case are obvious. In the third case, after finding a single free node, the augmenting path search must use the newly inserted edge (u, v) (which is not matched, but both endpoints are non-free). Hence, it is sufficient to find a single free node. After running the augmenting path search, the matching size has either increased by one, or there was no augmenting path. Hence, the matching must be maximum. Lastly, note that the third case is only necessary if both endpoints of the inserted edge are in different connected components.

Note that when considering insertions only, the algorithm is more expensive than just running the static algorithm. This is due to the fact that the static algorithm runs an augmenting path search from each free *node* once, while our dynamic algorithm does try to find augmenting paths every time we insert an edge (since the graph may have changed at other places not close to the inserted edge). The overall worst case complexity in this case is $O(m(n + m))$ compared to $O(n(n + m))$ for the static algorithm. In our experiments, this effect is especially noticeable if we start a search from a node where a previous augmenting path search has been unsuccessful.

Hence, besides using the unsafe option which drops the property that the matching is maximum, we propose the following optimization called *lazy augmenting path search*. Here, we start an augmenting path search from u and v only if at least $m'/2$ edges have been inserted or deleted since the last augmenting path search from u or v or no augmenting path

search has been started. Note that this effectively amortizes the cost for the augmenting path search, yielding amortized constant time per edge. Our experiments indicate that this speeds up the overall time of the algorithm drastically, while being only slightly worse than the optimum algorithm. Our third optimization limits the search depth of the augmenting path search to $2/\epsilon - 1$. This ensures that there is no augmenting path of length $2/\epsilon - 1$ and hence is a deterministic $(1+\epsilon)$-approximate matching algorithm (if the deletion part algorithm ensures this as well, and the algorithm is run with the safe option). Note that the worst case complexity of the optimum version of the insertion operation is $O(n+m)$, but in practice augmenting paths (if present) are much shorter. The bounded version of our algorithm has, however, worse case complexity of $O(\Delta^{2/\epsilon-1})$.

Edge Deletion. Let (u, v) be the deleted edge. After the deletion we start an augmenting path search from any free endpoint u or v. Depending on the configuration of the algorithm this either does a full run for an augmenting path or stops when the augmenting path search reached depth $2/\epsilon - 1$. In the first case, this guarantees that the matching is maximum if it was maximum before and in the latter case, our algorithm maintains an $(1+\epsilon)$ approximate maximum matching. If case we use lazy augmenting path search, we start an (depth bounded) augmenting path search from u and v only if at least $m'/2$ edges have been inserted or deleted since the last augmenting path search from u or v. Otherwise, we limit augmenting path search from u and v to augmenting paths of length $\min(3, 2/\epsilon - 1)$.

3.3 Baswana, Gupta and Sen Algorithm

Baswana, Gupta and Sen (BGS) presented an randomized algorithm in [4], that maintains a maximal matching in a dynamic graph in *amortized* $O(\sqrt{n})$ update time with high probability. They also present a multi-level variant that runs in $O(\log n)$ amortized time. To be self contained, we briefly review the main concepts of the algorithm and follow their description closely.

Levels and Ownership of Edges. The algorithm uses the concept of *ownership* for edges. More precisely, based upon the number of edges that a vertex *owns*, the algorithm partitions the set of vertices into two levels 0 and 1. An edge is always owned by at least one of its endpoints. If both endpoints are at level 0, then both vertices own the edge. If only one endpoint is at level 1, then this endpoint owns the edge. If both endpoints are at level 1, then exactly one endpoint, namely the first mentioned vertex owns the edge. If a new edge (u, v) with $level(u) = level(v) = 1$ is inserted, it will therefore be owned by the vertex u.

Algorithm 1 RANDOM-SETTLE(u): find a random edge (u, v) from the set of owned edges of u, matches it and returns the previous mate of v.

Let (u, v) be a uniformly randomly selected edge from $\mathcal{O}_u$
forall $(v, w) \in \mathcal{O}_u$ **do**
  remove (v, w) from $\mathcal{O}_w$
if v *is matched* **then**
  $x \leftarrow \text{mate}(v)$; $M \leftarrow M \setminus \{(v, x)\}$
else
  $x \leftarrow$ NULL
$M \leftarrow M \cup \{(u, v)\}$, LEVEL$(u) \leftarrow 1$, LEVEL$(v) \leftarrow 1$
return z

The set $\mathcal{O}_u$ denotes the set of edges owned by a vertex u. The level of an edge is defined by $level(e = \{u, v\}) = \max(level(u), level(v))$. BGS maintains the following invariants: (1) Every vertex on level 1 is matched. (2) Every free vertex on level 0 has all neighbours matched. (3) Every vertex on level 0 owns less then $\sqrt{n}$ edges (at any moment of time). (4) Both endpoints of each matched edge are on same level.

Edge Insertion. Let (u, v) be the edge being inserted. If either u or v are at level 1, then there is no violation of any invariant. The algorithm adds (u, v) to $\mathcal{O}_u$ if level$(u) = 1$ and to $\mathcal{O}_v$ otherwise. If both endpoints of (u, v) are at level 0, then the algorithm proceeds as follows: If both endpoints are free, the edge is added to the matching. Adding the edge (u, v) to the sets $\mathcal{O}_u$ and $\mathcal{O}_v$ increases the number of edges owned by u and v. If at least one set $\mathcal{O}_u$ or $\mathcal{O}_v$ exceeds the threshold of $\sqrt{n}$ in size, the vertex with the higher number of owned edges will be *repaired*. Let u be that vertex. Repairing a vertex u is done by calling the procedure Random-Settle on u. As a result, u moves to level 1 and gets matched to some vertex y selected randomly uniformly from the set of owned edges $\mathcal{O}_u$. The vertex y is also moved to level 1 to satisfy invariant 4. If w and x were the earlier mates of u and y at level 0, respectively, then matching u and y has rendered w and x free. The algorithm tries to settle each of those by scanning their set of owned edges for free vertices.

Edge Deletion. Let (u, v) be an edge that is deleted. If the edge has not been matched, then after removing the edge from the graph all invariants still hold. If it has been matched, then u and v are now free. Therefore, the first invariant may be violated. If (u, v) is at level 0, then the algorithm tries to settle both endpoints by scanning their sets of owned edges. If (u, v) is at level 1, then u the algorithm does the following: First, u disowns all its edges whose other endpoint is at level 1. If $\mathcal{O}_u$ is still greater than or equal to $\sqrt{n}$, then u stays at level 1 and executes Random-Settle(u). If u owns less than $\sqrt{n}$ edges, it moves to level 0 and tries to settle it by scanning its set of owned edges. The transition of u from level 1 to 0 leads to an increase in the number of edges owned by each of its neighbors at level 0. This may violate the size constraint of owned edges for those neighbors. Hence, the algorithm calls Random-Settle for each neighbor that violates the constraint, which moves it to level 1.

3.4 Neiman and Solomon Algorithm

In contrast to the BGS algorithm [4], which is randomized, Neiman and Solomon (NS) [33] present a deterministic algorithm for maintaining a maximal matching in a dynamic graph. Their approach guarantees, that the maintained matching is a 3/2-approximate maximum matching and that update time is $O(\sqrt{m})$ in *worst case*, where m denotes the number of edges present in the graph in the moment of the update. NS maintains the following invariants: There are no augmenting paths of length ≤ 3, ensuring 3/2-approx matching. All free vertices have degree at most $\sqrt{2n + 2m}$.

▶ **Lemma 2** ([33]). *Any free vertex of degree larger than $\sqrt{m}$ can always be matched, so as to generate a free vertex with degree less than $\sqrt{m}$. This can be achieved in $O(\sqrt{m})$ time.*

Edge Insertion. Let (u, v) be the edge being inserted. If both the endpoints are free, the edge is simply added to the matching. Also, if both endpoints are matched it does not entail any further processing. However, if exactly one endpoint of the edge, say u, is matched, they try to remove a possible augmenting path of length 3 as follows. The neighbours of the mate of u say $u' = \text{mate}(u)$, are scanned for a free vertex, say x. If such a free vertex exists, an augmenting path of length 3 has been found, which is augmented increasing the matching size.

Edge Deletion. Let (u, v) be the edge being deleted. If the edge was unmatched, its deletion cannot create any new augmenting paths. However, if it was a matched edge, both the endpoints become free after the edge deletion. First, the algorithm checks for both freed vertices whether they have free neighbours and if so matches the freed vertices with those free neighbours. Now, in order to eliminate augmenting paths of length 3 starting from a free vertex, say u, all neighbours w of u are scanned checking if mate(w) has a free neighbour. By providing appropriate data structures, this can be done in $O(\sqrt{n})$ time. If an augmenting path has been found, it is augmented increasing the size of the matching by one. If no augmenting path has been found, vertex u remains free, but only if its degree is at most $\sqrt{2m}$. If the degree of u exceeds $\sqrt{2m}$, using Lemma 2 a surrogate can be found in $O(\sqrt{m})$.

The overall update time of the algorithm is bound by the bounded degree of all free vertices, making any linear search through the neighbourhood $N(u)$ of a vertex u cost at most $O(\sqrt{n+m})$. Bounding the degree can further be achieved in $O(\sqrt{m})$ time using Lemma 2.

4 Experimental Evaluation

Implementation and System

We implemented the algorithms described in the previous section. The codes are written in C++ and have been compiled using g++-7.3.0 with flags `-O3`. All codes are sequential. We plan to further improve the codes and then to release them to make it available to a larger audience. Our experiments are conducted on one core of a machine with AMD Opteron Processors 6174 with 2.2GHz and 256GB of RAM. *Dynamic Graph Data Structure:* our algorithms use the following dynamic graph data structure. For each node v, we maintain a vector L_v of adjacent nodes, and a hash table $\mathcal{H}_v$ that maps a vertex u that is incident to v to its position in L_v. This data structure allows for expected constant time insertion and deletion as well as a constant time operation to select a random neighbor of v. The deletion operation on (v, u) is implemented as follows: get the position of u in L_v via a lookup in $\mathcal{H}_v(u)$. Swap the element in L_v with the last element w in the vector and update the position of w in $\mathcal{H}_v$. Finally, pop the last element (now u) from L_v and delete its entry from $\mathcal{H}_v$.

Instances and Methodology

By default we perform ten repetitions per instance. We measure the total time taken to compute all edge insertions and deletions and generally use the *geometric mean* when averaging over different instances in order to give every instance a comparable influence on the final result. In order to compare different algorithms, we use *performance profiles* [14]. These plots relate the matching size / running time of all algorithms to the corresponding matching size / running time produced / consumed by each algorithm. More precisely, the y-axis shows $\#\{\text{objective} \geq \tau * \text{best}\}/\#\text{instances}$, where objective corresponds to the result of an algorithm on an instance and best refers to the best result of any algorithm shown within the plot. When we look at running time, the y-axis shows $\#\{\text{t} \leq \text{fastest}/\tau\}/\#\text{instances}$ (as a function of the parameter τ), where t corresponds to the time of an algorithm on an instance and fastest refers to the time of the fastest algorithm on that instance. The parameter $\tau \leq 1$ in this equation is plotted on the x-axis. For each algorithm, this yields a non-decreasing, piecewise constant function. Thus, if we are interested in the number of instances where an algorithm is the best/fastest, we only need to look at $\tau = 1$.

Instances

We evaluate our algorithms on a number of large graphs. These graphs are collected from [3, 13, 27, 26, 35]. Table 3 summarizes the main properties of the benchmark set. Our benchmark set includes a number of graphs from numeric simulations as well as complex networks. These include static graphs as well as real dynamic graphs. As our algorithms do only handle undirected graphs, we consider all input graphs to be undirected by ignoring edge directions and we remove self-loops and parallel edges. We perform *two different types* of experiments. First, we use the algorithms using insertions only, i.e. we start with an empty graph and insert all edges of the static graph in a random order. We do this with all graphs from Table 3. Second, we use real dynamic instances from Table 4. Most of these instances, however, only feature insertions (with the exception being `dewiki` and `wiki-simple-en` which have both real insert and real delete operations). Hence, we perform additional experiments with fully dynamic graphs from these inputs, by undoing x percent of the update operations performed last.

4.1 Random Walk and Blossom-based Algorithms

In this section, we use our algorithms with random insertions only. More precisely, we use the static graphs from Table 3. For each experiment, we start with an empty graph and insert edges of the static input in random order until all edges are insert all edges are inserted compare the result of our dynamic algorithms the maximum matching on the final graph Edmond [17].

Random Walk-Based Algorithms. We start with random walk-based algorithms. Preliminary experiments have shown that decreasing ϵ is more effective in getting better solutions than performing more repeated random walks at the start node. Hence, we exclude algorithms that perform multiple repetitions of random walks per insert operation here from the evaluation and focus on the different values of ϵ. We vary $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$. Recall that the path length of a single random walk is then bounded by $2/\epsilon - 1$. If *all* paths of that length were explored, the algorithms would be guaranteed to give a $(1+\epsilon)$-approximation. Figure 1 summarizes the result. It is not surprising that the algorithm needs more running time for smaller ϵ, but also yields better results with increasing path lengths. On average, the algorithm is 2.4%, 3.2%, 4.2%, 5.5%, 11.5% percent away from the optimum for $\epsilon = 0.1, 0.25, 0.5, 1, 2$ respectively. Thus, even though the algorithms are not guaranteed to explore all paths of length $2/\epsilon - 1$, they achieve in practice an approximation that is much better than the theoretical bound for algorithms that explore all such paths. The strongest configuration ($\epsilon = 0.1$) is at most 1% away from the optimum matching size in 50% of the cases. Note that the random walk algorithm does not achieve the guarantee of 1% approximation as claimed by Lemma 1 since we did not perform the vast amount of repetitions necessary to get the result in expectation – instead we performed a single repetition of the random walker for each insertion. As excepted the running time does increase with decreasing ϵ. However, due to random walks that can finish early because they managed to match an edge, the effect is less visible than theory expects. The running time increase over the random walk using $\epsilon = 2$ (which is essentially a random walker not allowed to move, and hence boils down to the very simple greedy algorithm), is 12%, 17%, 21%, 27% for $\epsilon = 1, 0.5, 0.25, 0.1$, respectively.

Enabling Δ-settling generally improves the result. On average, the random walk with Δ-settling is now 1.1%, 1.4%, 1.8%, 2.2%, 3.7% away from the optimum matching for $\epsilon = 0.1, 0.25, 0.5, 1, 2$, respectively. On average in our experiments using Δ-settling has a negligible impact on running time. Hence, we recommend to use Δ-settling when using random walk-based algorithms and do so in the following unless otherwise mentioned.

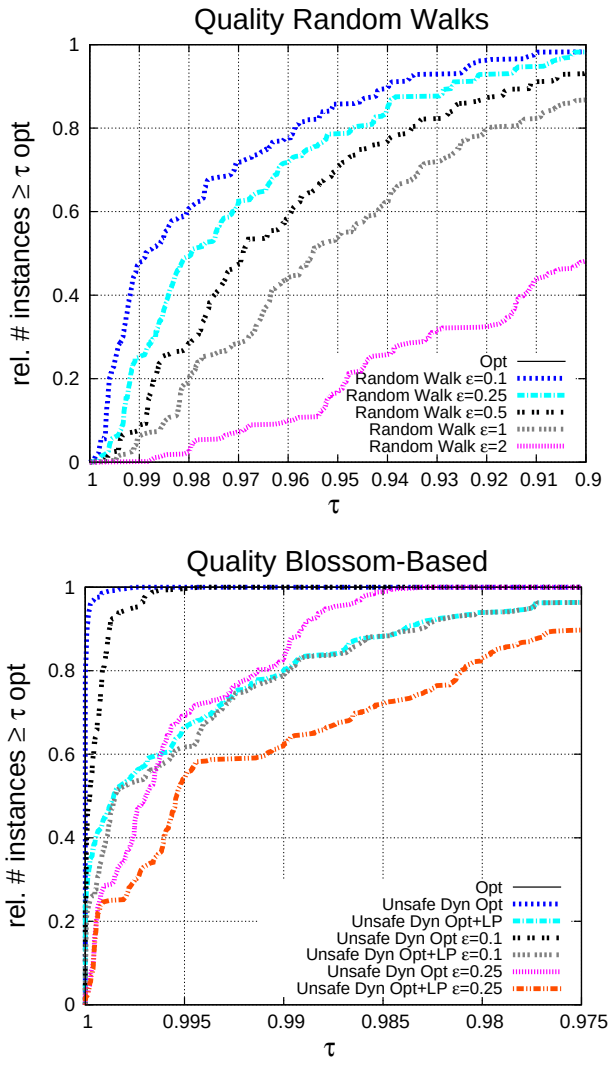


Figure 1 Performance profile for matching size $|\mathcal{M}|$ for Random Walk and for Unsafe Dyn Opt.

(Optimum) Blossom-Based Algorithms. We now consider dynamic blossom-based algorithms from Section 3.2. We start this section with the version of the algorithm that maintains the optimum matching, and compare it to the naive dynamic optimum matching algorithm that recomputes a maximum matching from scratch each time an edge is inserted. Since the running time of the naive optimum algorithm is fairly excessive, we run it only on the graphs of our benchmark set having less than 25k nodes. First of all, our dynamic algorithm that maintains the optimum matching is more than an order of magnitude faster than the naive optimum algorithm (roughly a factor 12). We expect that the difference will be even more pronounced if even larger graphs are used. Running our dynamic optimum algorithm with the unsafe option indeed significantly speeds up the algorithm – the lazy augmenting path search configuration is more than two orders of magnitude over the safe version of our algorithm (roughly a factor 115). The improvements in running time stem

from the fact that our algorithms try to maintain a very large matching. Hence, the case that is executed by the safe option often does not find an augmenting path which implies that the augmenting path search has to look at the overall network and hence reaches its worst-case complexity. Of course, the unsafe option does not have a guarantee on optimality anymore. In our experiments, the unsafe option computes matchings that are 0.02% worse than the optimum on average. We conclude that the algorithm maintains near-optimum matchings while being three orders of magnitude faster than the naive optimum dynamic algorithm. Henceforth, we only consider the unsafe version of our algorithm.

We now switch our set of graphs back to all of our benchmark graphs from Table 3. Using lazy augmenting path search in the unsafe algorithm additionally speedups up computations. Unsafe+LP is on average 20.5 faster than the unsafe algorithm without lazy augmenting path search – again at the cost of solution quality. The algorithm is already only 30% slower than running the static algorithm a *single* time on the final graph that contains all edges. On the other hand, the unsafe dynamic algorithm using lazy augmenting path search computes 0.6% worse matchings that the unsafe algorithm without lazy augmenting path search.

Lastly, we focus on the third variation of the algorithm, which is to bound the depth of the augmenting path search to that is done during update operations. The depth is bounded to $2/\epsilon - 1$ so that given ϵ and running the safe option of the algorithm would maintain a $1 + \epsilon$ approximate matching. We, however, only consider the unsafe version of the algorithm. We use same values of $\epsilon = 0.1, 0.25, 0.5, 1$ as in the random walk-based algorithms section, but do not consider $\epsilon = 2$, since this is again essentially the very simple greedy algorithm. Moreover, we run the algorithm with and without the lazy augmenting path search.

First of all, running without lazy augmenting path search, the algorithm indeed maintains the approximation guarantee. On average, the algorithm is 0.1%, 0.4%, 1.5% and 3.6% worse than the optimum algorithm for $\epsilon = 0.1, 0.25, 0.5, 1$, respectively. Using the lazy augmenting path search, speeds up to algorithm by a factor of 5.52, 2.65, 2.03, 1.76 for $\epsilon = 0.1, 0.25, 0.5, 1$, respectively. With lazy augmenting path search, the algorithm is 0.6%, 1.00%, 2.2%, 4.3% worse than the optimum for $\epsilon = 0.1, 0.25, 0.5, 1$, respectively (and hence still achieves the approximation guarantee). The algorithm using $\epsilon = 0.1$ is only 0.2% worse than the algorithm not bounding the depth. However, the algorithm is also not much faster. On average, bounding the search depth with $\epsilon = 0.1$ improves running time by 6%. Figure 1 shows a summarizing performance profile.

4.2 Comparison of Algorithms

Dynamic Sequences from Static Graphs. We now compare all of the different non-optimal algorithms against each other for the insertion-only case. For random-walks, we always enable Δ-settling, for blossom-based algorithm always use the unsafe option and with and without lazy augmenting path search. Table 1 shows average results for matching size and running time after all edges and operations have been performed. Figure 2 shows performance profiles for running time and for matching size.

First of all, both the blossom-based (with lazy augmenting path search) and random walk-based algorithms dominate the algorithms by Neiman Solomon and Baswana Gupta Sen (BGS). The algorithms find consistently larger matchings and do so in less time. However, note that the real-world instances we look at rarely have nodes with more than $\sqrt{n}$ neighbors, so that the BGS algorithm is roughly similar to the simple greedy algorithm. We also try to use $c \cdot \sqrt{n}$ as a threshold for different values of c, but this always resulted in worse matching sizes.

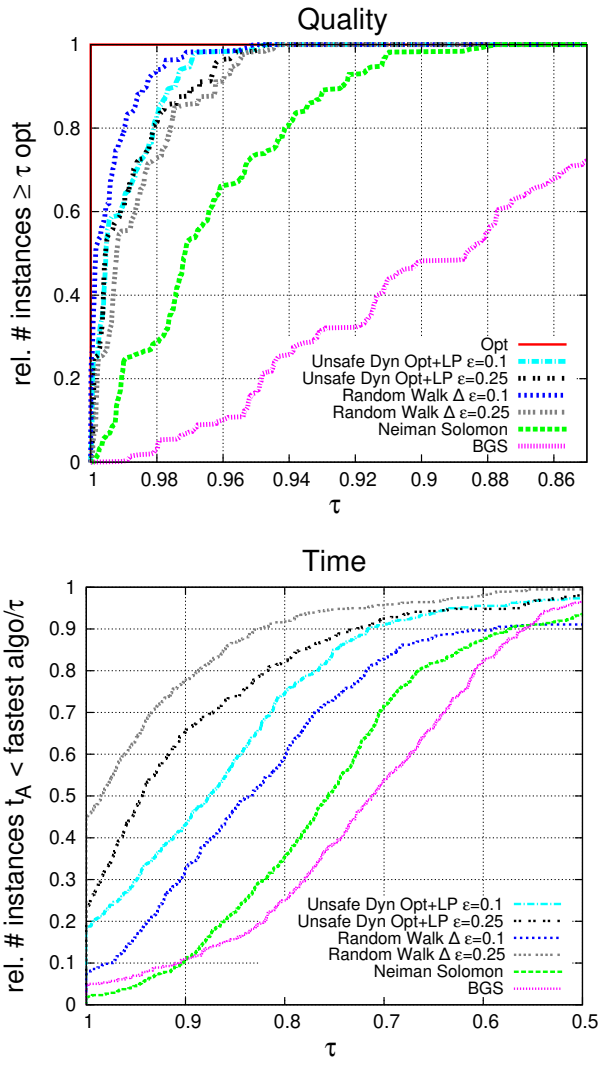


Figure 2 Performance profile for matching size $|\mathcal{M}|$ and time for all algorithms. In all cases, if an algorithm has a curve closer to the upper left corner, then the algorithm is better.

In general, performance differences in running time are not very big (except if we don't use lazy augmenting path search in the blossom-based algorithms). Secondly, for the same values of ϵ the blossom-based algorithms compute slightly better results than their random walk-based counter parts. This is not surprising as the blossom-based algorithms explore larger subgraphs for each edge that has been inserted. We conclude here that both types of algorithms are feasible in practice and have an advantage in solution quality over Neiman Solomon and Baswana Gupta Sen on graphs with random insertions. Moreover, both of these algorithm yield a clear trade-off between running time and solution quality via the ϵ parameter. On the other hand, all of the algorithms considered here are roughly five orders of magnitude faster than the naive dynamic optimum algorithm (only considering instances having less than 25k nodes) and except Baswana Gupta Sen, all of these algorithms are within a 4% range of the optimum matching size.

Real-World Dynamic Instances. We now switch to the real-world dynamic instances. As already mentioned, most of these instances are insertion-only. Hence, we perform additional experiments with fully dynamic graphs from these inputs, by undoing x percent of the update operations performed last (call them $\mathcal{O}_x$). More precisely, we perform the operations in $\mathcal{O}_x$ in reverse order. More precisely, if an edge operation was an insertion in $\mathcal{O}_x$, we perform a delete operation and if it was a delete operation we insert it. As before, we compute the update on the graph after each removal/insertion. The connection to practice in this case, is that with undoing operations, we want to restore a previous state. Table 2 summarizes the results of the experiment and Figure 3 compares the algorithms on the two real-world dynamic graphs dewiki and wiki_simple_en. Overall, the situation is similar to experiments with random insertions that we have seen before. The random walk with Δ-settling and $\epsilon = 0.5$ dominates Baswana, Gupta, Sen and Neiman Solomon in terms of running time *and* matching size for every number of undo operations. The blossom-based algorithm with lazy path search, however, yields smaller matchings if no operations are undone. We believe that this is due to the edges not being inserted randomly and hence the lazy augmenting path search heuristic is less effective, and misses augmenting paths that have been created over time. If operations are undone, the blossom-based algorithms outperform Neiman Solomon in terms of matching size, but are also considerably slower as the deletion operations search for augmenting paths of lengths three (except for $\epsilon = 1$). The blossom-based algorithm without lazy augmenting path search get very close to the optimum solutions. The best algorithm here is blossom-based algorithm without lazy augmenting path search for $\epsilon = 1/3$. On average, it computes solutions that are $< 0.6\%$ away from the optimum (for every amount of undo operations done).

In general, all algorithms improve quality relative to the optimum matching size, if we undo operations. This is due to the fact that the matching may have changed over time and hence new (short) augmenting paths may be found. In case of random-walks this is also simply due to the fact that additional work is performed and the likelihood to find an augmenting path is increased by running additional random walks. Summing up, all of the algorithms, except Baswana, Gupta, Sen, compute/maintain very large matchings. Blossom-based and random walk-based algorithm are highly flexible and are able to trade solution quality for time. Overall, random walk-based algorithms seem to be the method of choice in practice.

Table 1 Random insertions from static graphs: mean of the matching size relative to optimum after all operations have been done as well as mean increase in running time over Random Walk,Δ,$\epsilon = 0.25$.

algorithm		mean $\lvert\mathcal{M}\rvert$ / $\lvert\mathcal{M}_{\text{opt}}\rvert$	rel. time
BGS		0.885	32%
Neiman Solomon		0.964	28%
Unsafe Dyn Opt+LP	$\epsilon = 0.1$	0.994	27%
Unsafe Dyn Opt+LP	$\epsilon = 0.25$	0.990	11%
Unsafe Dyn Opt	$\epsilon = 0.1$	0.999	613%
Unsafe Dyn Opt	$\epsilon = 0.25$	0.996	192%
Unsafe Dyn Opt	$\epsilon = 0.5$	0.985	101%
Unsafe Dyn Opt	$\epsilon = 1$	0.964	67%
Random Walk,Δ	$\epsilon = 0.1$	0.989	5%
Random Walk,Δ	$\epsilon = 0.25$	0.986	1

Table 2 Real-world dynamic instances: mean of the matching size relative to optimum after all operations have been done as well as the mean increase in running time over Random Walk,Δ,$\epsilon = 0.5$.

# undo op		0	5%	10%	25%
algorithm		mean $\lvert\mathcal{M}\rvert$ / $\lvert\mathcal{M}_{\text{opt}}\rvert$			
BGS		0.845	0.847	0.848	0.851
Neiman Solomon		0.968	0.971	0.973	0.976
Unsafe Dyn Opt+LP	$\epsilon = 0.1$	0.947	0.985	0.990	0.996
Unsafe Dyn Opt+LP	$\epsilon = 0.25$	0.942	0.982	0.988	0.993
Unsafe Dyn Opt	$\epsilon = 0.\overline{33}$	0.994	0.996	0.997	0.998
Unsafe Dyn Opt	$\epsilon = 0.5$	0.988	0.991	0.992	0.994
Unsafe Dyn Opt	$\epsilon = 1$	0.968	0.971	0.973	0.976
Random Walk,Δ	$\epsilon = 0.1$	0.982	0.984	0.985	0.986
Random Walk,Δ	$\epsilon = 0.25$	0.981	0.983	0.984	0.985
Random Walk,Δ	$\epsilon = 0.5$	0.978	0.980	0.981	0.982
algorithm		rel. time			
BGS		4%	14%	13%	18%
Neiman Solomon		64%	82%	92%	112%
Unsafe Dyn Opt+LP	$\epsilon = 0.1$	82%	306%	383%	633%
Unsafe Dyn Opt+LP	$\epsilon = 0.25$	23%	149%	200%	346%
Unsafe Dyn Opt	$\epsilon = 0.\overline{33}$	1 551%	1 551%	1 598%	1814%
Unsafe Dyn Opt	$\epsilon = 0.5$	679%	682%	713%	800%
Unsafe Dyn Opt	$\epsilon = 1$	210%	212%	223%	250%
Random Walk,Δ	$\epsilon = 0.1$	25%	26%	24%	24%
Random Walk,Δ	$\epsilon = 0.25$	10%	11%	9%	11%
Random Walk,Δ	$\epsilon = 0.5$	1	1	1	1

5 Conclusion

We looked at several dynamic matching algorithms including Baswana, Gupta and Sen [4], Neiman and Solomon [33], as well as random walk-based algorithms and blossom-based algorithms. We performed extensive experiments comparing the performance of these algorithms on the real-world and artificially generated instances. In terms of results, first we have shown that maintaining optimum matchings can be done much more efficiently than the naive algorithm that recomputes maximum matchings from scratch. Second, we have seen that all non-optimum dynamic algorithms that we considered in this work are able to maintain near-optimum matchings in practice while being multiple orders of magnitudes faster than the naive optimum dynamic algorithm. In practice, random walk-based algorithms with Δ-settling will be the method of choice.

In future work, it may be interesting to transfer results to the weighted case, and to combine our algorithms with simple data reductions rules such as [25]. It could be interesting to use these dynamic matching algorithms to derive dynamic multilevel algorithms for example for graph partitioning [32, 1]. Another direction will be to explore the parallelization potential of random walk-based algorithms. Lastly, it may be interesting to incorporate dynamic transitive closure algorithms (e.g. [22]) into the delete operation of the dynamic optimum matching algorithm to further reduce the number of augmenting path searches.

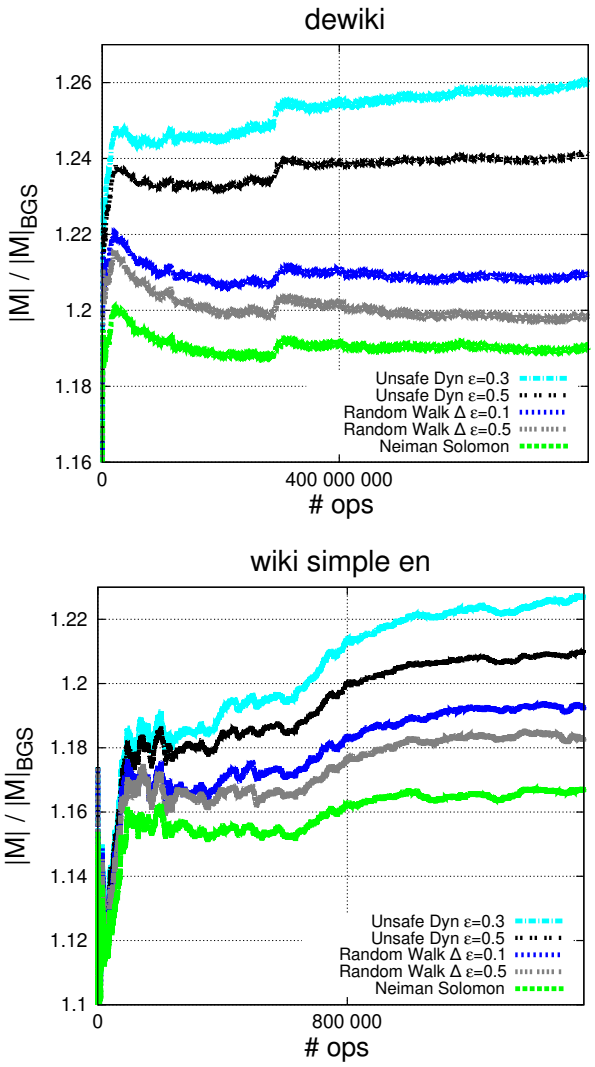


Figure 3 Matching size over time compared to Baswana, Gupta, Sen on the two real dynamic instances dewiki and wiki_simple_en.

References

1 Yaroslav Akhremtsev, Peter Sanders, and Christian Schulz. High-quality shared-memory graph partitioning. In Marco Aldinucci, Luca Padovani, and Massimo Torquati, editors, *Euro-Par 2018: Parallel Processing - 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings*, volume 11014 of *Lecture Notes in Computer Science*, pages 659–671. Springer, 2018. doi:10.1007/978-3-319-96983-1_47.

2 Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 7:1–7:16, 2018.

3 D. Bader, A. Kappes, H. Meyerhenke, P. Sanders, C. Schulz, and D. Wagner. Benchmarking for Graph Clustering and Partitioning. In *Encyclopedia of Social Network Analysis and Mining*. Springer, 2014.

4 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. *SIAM J. Comput.*, 44(1):88–113, 2015.

5 Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957. doi:10.1073/pnas.43.9.842.

6 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Symposium on Discrete Algorithms SODA*, pages 692–711. SIAM, 2016. doi:10.1137/1.9781611974331.ch50.

7 Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In *19th International Conf. on Integer Programming and Combinatorial Optimization IPCO*, pages 86–98, 2017.

8 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM J. Comput.*, 47(3):859–887, 2018.

9 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual Symposium on Theory of Computing*, pages 398–411. ACM, 2016.

10 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms SODA*, pages 470–489. SIAM, 2017.

11 Marcel Birn, Vitaly Osipov, Peter Sanders, Christian Schulz, and Nodari Sitchinava. Efficient parallel and external matching. In *Euro-Par 2013*, volume 8097 of *LNCS*, pages 659–670. Springer, 2013. doi:10.1007/978-3-642-40047-6_66.

12 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In *45th International Colloquium on Automata, Languages, and Programming ICALP*, pages 33:1–33:14, 2018.

13 T. Davis. The University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices, 2008. URL: http://www.cise.ufl.edu/research/sparse/matrices/.

14 Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, 2002. doi:10.1007/s101070100263.

15 D. Drake and S. Hougardy. A Simple Approximation Algorithm for the Weighted Matching Problem. *Information Processing Letters*, 85:211–213, 2003.

16 Andre Droschinsky, Petra Mutzel, and Erik Thordsen. Shrinking trees not blossoms: A recursive maximum matching approach. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020*, pages 146–160. SIAM, 2020. doi:10.1137/1.9781611976007.12.

17 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

18 Harold Neil Gabow. *Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs*. PhD thesis, Stanford University, Stanford, CA, USA, 1974.

19 Zvi Galil, Silvio Micali, and Harold N. Gabow. An O(|E||V| log |V|) algorithm for finding a maximal weighted matching in general graphs. *SIAM Journal Computing*, 15(1):120–130, 1986.

20 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1 + \epsilon)$-approximate incremental matching in constant deterministic amortized time. In *Proceedings of the 20th Symposium on Discrete Algorithms*, pages 1886–1898. SIAM, 2019. doi:10.1137/1.9781611975482.114.

21 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *54th Symposium on Foundations of Computer Science, FOCS*, pages 548–557. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.65.

22 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Faster fully dynamic transitive closure in practice. In Simone Faro and Domenico Cantone, editors, *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy*, volume 160 of *LIPIcs*, pages 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SEA.2020.14.

23 J. E. Hopcroft and R. M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite. In *12th Annual Symposium on Switching and Automata Theory (SWAT)*, pages 122–125, 1971. doi:10.1109/SWAT.1971.1.

24 Zoran Ivkovic and Errol L. Lloyd. Fully dynamic maintenance of vertex cover. In *19th International Workshop Graph-Theoretic Concepts in Computer Science*, volume 790 of *LNCS*, pages 99–111, 1993.

25 Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. In *26th European Symposium on Algorithms ESA*, volume 112 of *LIPIcs*, pages 53:1–53:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.53.

26 Jérôme Kunegis. KONECT: the koblenz network collection. In Leslie Carr, Alberto H. F. Laender, Bernadette Farias Lóscio, Irwin King, Marcus Fontoura, Denny Vrandecic, Lora Aroyo, José Palazzo M. de Oliveira, Fernanda Lima, and Erik Wilde, editors, *22nd World Wide Web Conference, WWW '13*, pages 1343–1350. International World Wide Web Conferences Steering Committee / ACM, 2013. doi:10.1145/2487788.2488173.

27 J. Lescovec. Stanford Network Analysis Package (SNAP). http://snap.stanford.edu/index.html.

28 J. Maue and P. Sanders. Engineering Algorithms for Approximate Weighted Matching. In *Proceedings of the 6th Workshop on Experimental Algorithms (WEA'07)*, volume 4525 of *LNCS*, pages 242–255. Springer, 2007. doi:10.1007/978-3-540-72845-0_19.

29 Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999. URL: http://www.mpi-sb.mpg.de/%7Emehlhorn/LEDAbook.html.

30 Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized on-line matching. In *46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 264–273. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.12.

31 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *21st Symposium on Foundations of Computer Science*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

32 Orlando Moreira, Merten Popp, and Christian Schulz. Evolutionary multi-level acyclic graph partitioning. In Hernán E. Aguirre and Keiki Takadama, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pages 332–339. ACM, 2018. doi:10.1145/3205455.3205464.

33 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Trans. Algorithms*, 12(1):7:1–7:15, 2016.

34 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC*, pages 457–464, 2010. doi:10.1145/1806689.1806753.

35 Julia Preusse, Jérôme Kunegis, Matthias Thimm, Thomas Gottron, and Steffen Staab. Structural dynamics of knowledge networks. In *Proc. Int. Conf. on Weblogs and Social Media*, 2013.

36 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126, 2007. doi:10.1145/1283383.1283397.

37 Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library - User Guide and Reference Manual*. C++ in-depth series. Pearson / Prentice Hall, 2002.

38 Shay Solomon. Fully dynamic maximal matching in constant update time. In *57th Symposium on Foundations of Computer Science FOCS*, pages 325–334, 2016.

39 Robert Endre Tarjan. *Data structures and network algorithms*, volume 44 of *CBMS-NSF regional conference series in applied mathematics*. SIAM, 1983. doi:10.1137/1.9781611970265.

A Instances

Table 3 Basic properties of the benchmark set of static graphs obtained from [3, 13, 27].

graph	n	m	graph	n	m
144	144 649	1 074 393	eu-2005	862 664	16 138 468
3elt	4 720	13 722	fe_4elt2	11 143	32 818
4elt	15 606	45 878	fe_body	45 087	163 734
598a	110 971	741 934	fe_ocean	143 437	409 593
add20	2 395	7 462	fe_pwt	36 519	144 794
add32	4 960	9 462	fe_rotor	99 617	662 431
amazon-2008	735 323	3 523 472	fe_sphere	16 386	49 152
as-22july06	22 963	48 436	fe_tooth	78 136	452 591
as-skitter	554 930	5 797 663	finan512	74 752	261 120
auto	448 695	3 314 611	in-2004	1 382 908	13 591 473
bcsstk29	13 992	302 748	loc-brightkite_edges	56 739	212 945
bcsstk30	28 924	1 007 284	loc-gowalla_edges	196 591	950 327
bcsstk31	35 588	572 914	m14b	214 765	1 679 018
bcsstk32	44 609	985 046	memplus	17 758	54 196
bcsstk33	8 738	291 583	p2p-Gnutella04	6 405	29 215
brack2	62 631	366 559	PGPgiantcompo	10 680	24 316
citationCiteseer	268 495	1 156 647	rgg_n_2_15_s0	32 768	160 240
cnr-2000	325 557	2 738 969	soc-Slashdot0902	28 550	379 445
coAuthorsCiteseer	227 320	814 134	t60k	60 005	89 440
coAuthorsDBLP	299 067	977 676	uk	4 824	6 837
coPapersCiteseer	434 102	16 036 720	vibrobox	12 328	165 250
coPapersDBLP	540 486	15 245 729	wave	156 317	1 059 331
crack	10 240	30 380	web-Google	356 648	2 093 324
cs4	22 499	43 858	whitaker3	9 800	28 989
cti	16 840	48 232	wiki-Talk	232 314	1 458 806
data	2 851	15 093	wing	62 032	121 544
email-EuAll	16 805	60 260	wing_nodal	10 937	75 488
enron	69 244	254 449	wordassociation-2011	10 617	63 788

Table 4 Basic properties of the benchmark set of dynamic graphs with number of update operations $\mathcal{O}$. Most of the graphs only feature insertions. The only two exceptions are marked with a *. All of these graphs have been obtained from the KONECT graph database [35].

graph	n	$\mathcal{O}$
amazon-ratings	2 146 058	5 838 041
citeulike_ui	731 770	2 411 819
dewiki*	2 166 670	86 337 879
dnc-temporalGraph	2 030	39 264
facebook-wosn-wall	46 953	876 993
flickr-growth	2 302 926	33 140 017
haggle	275	28 244
lastfm_band	174 078	19 150 868
lkml-reply	63 400	1 096 440
movielens10m	69 879	10 000 054
munmun_digg	30 399	87 627
proper_loans	89 270	3 394 979
sociopatterns-infections	411	17 298
stackexchange-stackoverflow	545 197	1 301 942
topology	34 762	171 403
wikipedia-growth	1 870 710	39 953 145
wiki_simple_en*	100 313	1 627 472
youtube-u-growth	3 223 590	9 375 374

Finding All Global Minimum Cuts in Practice

Monika Henzinger
University of Vienna, Faculty of Computer Science, Austria
monika.henzinger@univie.ac.at

Alexander Noe
University of Vienna, Faculty of Computer Science, Austria
alexander.noe@univie.ac.at

Christian Schulz
University of Vienna, Faculty of Computer Science, Austria
christian.schulz@univie.ac.at

Darren Strash
Hamilton College, Department of Computer Science, Clinton, NY, USA
dstrash@hamilton.edu

Abstract

We present a practically efficient algorithm that finds all global minimum cuts in huge undirected graphs. Our algorithm uses a multitude of kernelization rules to reduce the graph to a small equivalent instance and then finds all minimum cuts using an optimized version of the algorithm of Nagamochi, Nakao and Ibaraki. In shared memory we are able to find all minimum cuts of graphs with up to billions of edges and millions of minimum cuts in a few minutes. We also give a new linear time algorithm to find the most balanced minimum cuts given as input the representation of all minimum cuts.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Graph algorithms; Mathematics of computing → Network flows

Keywords and phrases Minimum Cut, Graph Algorithm, Algorithm Engineering, Cut Enumeration, Balanced Cut, Global Minimum Cut, Large-scale Graph Analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.59

Supplementary Material `https://github.com/VieCut/VieCut`

Funding The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) /ERC grant agreement No. 340506.
Partially supported by DFG grant SCHU 2567/1-2.

1 Introduction

We consider the problem of finding *all minimum cuts* of an undirected network where edges are weighted by positive integers. A *minimum cut* in a graph is a partition of the vertices into two sets so that the total weight of edges crossing the boundary between the blocks is minimized. The problem of finding all minimum cuts has applications in many fields. In particular, minimum cuts in similarity graphs can be used to find clusters [57, 25]. As the minimum cut is often highly skewed, a variety of techniques to find more balanced bipartitions with small cuts were developed [16, 24, 53]. However, these balanced variations of the problem are generally NP-complete. In contrast to that, we find all minimum cuts in practice in a similar timescale than finding an arbitrary minimum cut and can thus output the minimum cut that is least skewed. In community detection, the absence of a small cut inside a cluster can indicate a likely community in a social network [10]. Other applications for finding all minimum cuts can be found in network reliability [34, 51], where a minimum cut in a network

has the highest risk to disconnect the network; in VLSI design [40] and graph drawing [32], in which minimum cuts are used to separate the network; and finding all minimum cuts is an important subproblem for edge-connectivity augmentation algorithms [19, 47].

The problem of finding all minimum cuts is closely related to the *(global) minimum cut problem*, which aims to find *some* minimum cut in the graph. The classical algorithm of Gomory and Hu [23] solves the minimum cut problem by solving n minimum-s-t cut problems. Using the push-relabel algorithm [22] this results in a total running time of $\mathcal{O}\left(n^2 m \log \frac{n^2}{m}\right)$. Nagamochi et al. [43, 46] give an algorithm for the minimum cut problem, which is based on edge contractions instead of maximum flows. Their algorithm has a worst case running time of $\mathcal{O}\big(nm + n^2 \log n\big)$ but performs far better in practice on many graph classes [12, 31, 28]. Henzinger et al. [26] give a fast shared-memory parallel algorithm for the minimum cut problem based on their algorithm which finds a minimum cut very fast in practice.

Even though a graph can have up to $\binom{n}{2}$ minimum cuts [33], there is a compact representation of all minimum cuts of a graph called *cactus graph* with $\mathcal{O}(n)$ vertices and edges. A cactus graph is a graph in which each edge belongs to at most one simple cycle. Karzanov and Timofeev [36] give the first polynomial time algorithm to construct the cactus representation for all minimum cuts. Picard and Queyranne [50] show that all minimum cuts separating two specified vertices can be found from a maximum flow between them. Thus, similar to the classical algorithm of Gomory and Hu [23] for the minimum cut problem, we can find all minimum cuts in $n-1$ maximum flow computations. The algorithm of Karzanov and Timofeev [36] combines all those minimum cuts into a cactus graph representing all minimum cuts. Nagamochi and Kameda [44] give a representation of all minimum cuts separating two vertices s and t in a so-called (s,t)-cactus representation. Based on this (s,t)-cactus representation, Nagamochi et al. [45] give an algorithm that finds all minimum cuts and gives the minimum cut cactus in $\mathcal{O}\big(nm + n^2 \log n + n^* m \log n\big)$, where n^* is the number of vertices in the cactus.

Karger and Stein [35] give a randomized algorithm to find all minimum cuts in $\mathcal{O}\big(n^2 \log^3 n\big)$ time by contracting random edges. Based on the algorithm of Karzanov and Timofeev [36] and its parallel variant given by Naor and Vazirani [48] they show how to give the cactus representation of the graph in the same asymptotic time. Ghaffari et al. [20] recently gave an algorithm that finds all *non-trivial minimum cuts* of a simple unweighted graph in $\mathcal{O}\big(m \log^2 n\big)$ time. Using the techniques of Karger and Stein the algorithm can trivially give the cactus representation of all minimum cuts in $\mathcal{O}\big(n^2 \log n\big)$. While there are multiple implementations of the algorithm of Karger and Stein [12, 21, 28] for the minimum cut problem, to the best of our knowledge there are no published implementations of either of the algorithms to find the cactus graph representing all minimum cuts (with or without data reduction techniques).

In the last two decades significant advances in FPT algorithms have been made: an NP-hard graph problem is fixed-parameter tractable (FPT) if large inputs can be solved efficiently and provably optimally, as long as some problem parameter is small. This has resulted in an algorithmic toolbox that are by now well-established. Few of the new techniques are implemented and tested on real datasets, and their practical potential is far from understood. However, recently the engineering part in area has gained some momentum. There are several experimental studies in the area that take up ideas from FPT or kernelization theory, e.g. for independent sets (or equivalently vertex cover) [2, 11, 14, 41, 29, 30], for cut tree construction[3], for treewidth computations [7, 54, 39], for the feedback vertex set problem [37, 18], for the dominating set problem [1], for the minimum cut [26, 28], for the multiterminal cut problem [27], for the maximum cut problem [17] and for the cluster editing problem [8]. Recently, this type of data reduction techniques is also applied for problem in P such as matching [38].

Our Results

We give an algorithm that finds all minimum cuts on very large graphs. Our algorithm is based on the recursive algorithm of Nagamochi et al. [45]. We combine the algorithm with a multitude of techniques to find edges that can be contracted without affecting any minimum cut in the graph. Some of these techniques are adapted from techniques for the global minimum cut problem [49, 46]. Using these and newly developed reductions we are able to decrease the running time by up to multiple orders of magnitude compared to the algorithm of Nagamochi et al. [45] and are thus able to find all minimum cuts on graphs with up to billions of edges in a few minutes. Based on the cactus representation of all minimum cuts, we are able to find the most balanced minimum cut in time linear to the size of the cactus. As our techniques are able to find the most balanced minimum cut (and all others too) of graphs with billions of edges in mere minutes, this allows the use of minimum cuts as a subroutine in sophisticated data mining and graph analysis tools.

2 Basic Concepts

Let $G = (V, E, c)$ be a weighted undirected simple graph with vertex set V, edge set $E \subset V \times V$ and non-negative edge weights $c : E \to \mathbb{N}$. We extend c to a set of edges $E' \subseteq E$ by summing the weights of the edges; that is, let $c(E') := \sum_{e=(u,v)\in E'} c(u, v)$ and let $c(u)$ denote the sum of weights of all edges incident to vertex v. Let $n = |V|$ be the number of vertices and $m = |E|$ be the number of edges in G. The *neighborhood* $N(v)$ of a vertex v is the set of vertices adjacent to v. The *weighted degree* of a vertex is the sum of the weights of its incident edges. For brevity, we simply call this the *degree* of the vertex. For a set of vertices $A \subseteq V$, we denote by $E[A] := \{(u, v) \in E \mid u \in A, v \in V \setminus A\}$; that is, the set of edges in E that start in A and end in its complement. A cut $(A, V \setminus A)$ is a partitioning of the vertex set V into two non-empty *partitions* A and $V \setminus A$, each being called a *side* of the cut. The *capacity* or *weight* of a cut $(A, V \setminus A)$ is $c(A) = \sum_{(u,v)\in E[A]} c(u, v)$. A *minimum cut* is a cut $(A, V \setminus A)$ that has smallest capacity $c(A)$ among all cuts in G. We use $\lambda(G)$ (or simply λ, when its meaning is clear) to denote the value of the minimum cut over all $A \subset V$. For two vertices s and t, we denote $\lambda(G, s, t)$ as the capacity of the smallest cut of G, where s and t are on different sides of the cut. $\lambda(G, s, t)$ is also known as the *minimum s-t-cut* of the graph. If all edges have weight 1, $\lambda(G, s, t)$ is also called the *connectivity* of vertices s and t. The connectivity $\lambda(G, e)$ of an edge $e = (s, t)$ is defined as $\lambda(G, s, t)$, the connectivity of its incident vertices. At any point in the execution of a minimum cut algorithm, $\hat{\lambda}(G)$ (or simply $\hat{\lambda}$) denotes the smallest upper bound of the minimum cut that the algorithm discovered until that point. For a vertex $u \in V$ with minimum vertex degree, the size of the *trivial cut* $(\{u\}, V \setminus \{u\})$ is equal to the vertex degree of u. Many algorithms tackling the minimum cut problem use *graph contraction*. Given an edge $e = (u, v) \in E$, we define $G/(u, v)$ (or G/e) to be the graph after *contracting edge* (u, v). In the contracted graph, we delete vertex v and all edges incident to this vertex. For each edge $(v, w) \in E$, we add an edge (u, w) with $c(u, w) = c(v, w)$ to G or, if the edge already exists, we give it the edge weight $c(u, w) + c(v, w)$.

A graph with n vertices can have up to $\Omega(n^2)$ minimum cuts [33]. To see that this bound is tight, consider an unweighted cycle with n vertices. Each set of 2 edges in this cycle is a minimum cut of G. This yields a total of $\binom{n}{2}$ minimum cuts. However, all minimum cuts can be represented by a cactus graph C_G with up to $2n$ vertices and $\mathcal{O}(n)$ edges [45]. A cactus graph is a connected graph, in which any two simple cycles have at most one vertex in common. In a cactus graph, each edge belongs to at most one simple cycle.

To represent all minimum cuts of a graph G in an edge-weighted cactus graph $C_G = (V(C_G), E(C_G))$, each vertex of C_G represents a possibly empty set of vertices of G and each vertex in G belongs to the set of one vertex in C_G. Let Π be a function that assigns to each vertex of C_G it set of vertices of G. Then every cut $(S, V(C_G)\backslash S)$ corresponds to a minimum cut $(A, V\backslash A)$ in G where $A = \cup_{x \in S} \Pi(x)$. In C_G, all edges that do not belong to a cycle have weight λ and all cycle edges have weight $\frac{\lambda}{2}$. A minimum cut in C_G consists of either one tree edge or two edges of the same cycle. We denote by n^* the number of vertices in C_G and m^* the number of edges in C_G. The weight $c(v)$ of a vertex $v \in C_G$ is equal to the number of vertices in G that are assigned to v.

3 Algorithm Description

Our algorithm combines a variety of techniques and algorithms in order to find all minimum cuts in a graph. The algorithm is based on the contractions of edges which cannot be part of a minimum cut. Thus, we first show that an edge e that is not part of any minimum cut in graph G can be contracted and all minimum cuts of G remain in the resulting graph G/e.

▶ **Lemma 1** ([35]). *If an edge $e = (u, v)$ is not part of any minimum cut in graph G, all minimum cuts of G remain in the resulting graph G/e.*

Proof. Let (A, B) be an arbitrary minimum cut of G. For an edge $e = (u, v)$, which is not part of any minimum cut, we know that $e \notin E[A]$, so either u and v are both in vertex set A or both in vertex set B. This is still the case in G/e. Thus, the edge e can be contracted even if we aim to find every minimum cut of G. ◀

Lemma 1 is very useful to reduce the size of the graph by a multitude of techniques to identify such edges. We first give a short overview of our algorithm and then explain the techniques in more detail. First, we use the shared-memory parallel heuristic minimum cut algorithm VieCut [28] in order to find an upper bound $\hat{\lambda}$ for the minimum cut which is very likely to be the correct value. VieCut is a multilevel algorithm that uses the label propagation algorithm to contract heavily connected clusters. Having a tight bound for the minimum cut allows the contraction of many edges, as multiple reduction techniques depend on the value of the minimum cut. We adapt contraction techniques originally developed by Nagamochi et al. [43, 46] and Padberg et al. [49] to the problem of finding all minimum cuts. Section 3.1 explains these contraction routines. On the resulting graph we find all minimum cuts using an optimized variant of the algorithm of Nagamochi, Nakao and Ibaraki [45] and return the cactus graph which represents them all. A short description of the algorithm and an explanation of our engineering effort are given in Section 3.2. Afterwards, in Section 3.3 we show how we combine the parts into a fast algorithm to find all minimum cuts of large networks.

3.1 Edge Contraction

As shown in Lemma 1, edges that are not part of any minimum cut can be safely contracted. We build a set of techniques that aim to find contractible edges and run these in alternating order until neither of them finds any more contractible edges. We now give a short introduction to these.

For efficiency, we perform contractions in bulk. If our algorithm finds an edge that can be contracted, we merge the incident vertices in a thread-safe union-find data structure [5]. After each run of a contraction technique that finds contractible edges, we create the contracted

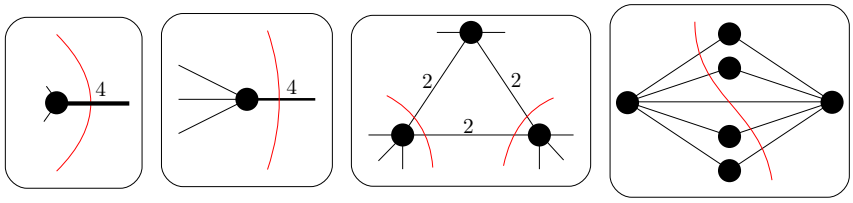


Figure 1 Contraction: (1) `HeavyEdge`, (2) `ImbalancedVertex`, (3) `ImbalancedTriangle`, (4) `HeavyNeighborhood`.

graph using a shared-memory parallel hash table [42]. In this contracted graph, each set of vertices of the original graph is merged into a single node. The contraction of this vertex set is equivalent to contracting a spanning tree of the set. After contraction we check whether a vertex in the contracted graph has degree $< \hat{\lambda}$. If it does, we found a cut of smaller value and update $\hat{\lambda}$ to this value.

3.1.1 Connectivity-based Contraction

The connectivity of an edge $e = (s, t)$ is the weight of the minimum cut that separates s and t, i.e. the *minimum s-t-cut*. For an edge that has connectivity $> \hat{\lambda}$, we thus know that there is no cut separating s and t (i.e. no cut that contains e) that has value $\leq \hat{\lambda}$. Thus, we know that there cannot be a minimum cut that contains e, as $\hat{\lambda}$ is by definition at least as large as λ. However, solving the minimum s-t-cut problem takes significant time, so computing the connectivity of each edge does not scale to large networks. Hence, as part of their algorithm for the global minimum cut problem, Nagamochi et al. [43, 46] give an algorithm that computes a lower bound $q(e)$ for the connectivity of every edge e of G in a total running time of $\mathcal{O}(m + n \log n)$. Each of the edges whose connectivity lower bound is already larger than $\hat{\lambda}$ can be contracted as it cannot be part of any minimum cut. Their algorithm builds *edge-disjoint maximum spanning forests* and contracts all edges that are not in the first $\lambda - 1$ spanning forests, as those connect vertices that have connectivity at least λ [26]. This is possible as the incident vertices of any such edge e are connected in each of the first $\lambda - 1$ spanning forests and by e and thus have a connectivity of at least λ. In other words, there can not be any cut smaller than λ which contains e.

Henzinger et al. [26] give a fast shared-memory parallel variant of their algorithm. As both of these algorithms only aim to find a single minimum cut, they also contract edges that have connectivity equal to $\hat{\lambda}$, as they only want to see whether there is a cut better than the best cut known previously. As we want to find all minimum cuts, we can only contract edges whose connectivity is strictly larger than $\hat{\lambda}$. Nagamochi et al. could prove that at least one edge has value $\hat{\lambda}$ in their routine and can thus be contracted. We do not have such a guarantee when trying to find edges that have connectivity $> \hat{\lambda}$. Consider for example an unweighted tree, whose minimum cut has a value of 1 and each edge has connectivity 1 as well.

3.1.2 Local Contraction Criteria

Padberg and Rinaldi [49] give a set of *local reduction* routines which determine whether an edge can be contracted without affecting the minimum cut. Their reductions routines were shown to be very useful in order to find a minimum cut fast in practice [12, 31, 28]. We adapt the routines originally developed for the minimum cut problem so that they hold for

the problem of for finding all minimum cuts. Thus, we have to make sure that we do not contract cuts of value $\hat{\lambda}$, as they might be minimal and additionally make sure that we do not contract edges incident to vertices that could have a *trivial minimum cut*, i.e. a minimum cut, where one side contains only a single vertex. Figure 1 depicts the contraction routines and Lemma 2 gives a more formal definition of them.

▶ **Lemma 2.** *For an edge $e = (u, v) \in E$, e is not part of any minimum cut, if e fulfills at least one of the following criteria. Thus, all minimum cuts of G are still present in G/e and e can be contracted.*

1. *`HeavyEdge`:* $c(e) > \hat{\lambda}$
2. *`ImbalancedVertex`:*
   - $c(v) < 2c(e)$ *and* $c(v) > \hat{\lambda}$, *or*
   - $c(u) < 2c(e)$ *and* $c(u) > \hat{\lambda}$
3. *`ImbalancedTriangle`:*
   $\exists w \in V$ *with*
   - $c(v) < 2\{c(v, w) + c(e)\}$ *and* $c(v) > \hat{\lambda}$, *and*
   - $c(u) < 2\{c(u, w) + c(e)\}$ *and* $c(u) > \hat{\lambda}$
4. *`HeavyNeighborhood`:*
   $c(e) + \sum_{w \in V} min\{c(v, w), c(u, w)\} > \hat{\lambda}$

Proof.

1. If $c(e) > \hat{\lambda}$, every cut that contains e has capacity $> \hat{\lambda}$. Thus it can not be a minimal cut.
2. Without loss of generality let v be the vertex in question. The condition $c(v) < 2c(e)$ means that e is heavier than all other edges incident to v combined. Thus, for any non-trivial cut that contains e, we can find a lighter cut by replacing e with all other incident edges to v, i.e. moving v to the other side of the cut. As this is not true for the trivial minimum cut $(v, V \setminus v)$, we cannot contract an edge incident to a vertex that has weight $\leq \hat{\lambda}$.
3. This condition is similar to (2). Let there be a triangle u, v, w in the graph in which it holds for both u and v that the two incident triangle edges are heavier than the sum of all other incident edges. Then, every cut that separates u and v can be improved by moving u and v into the same side. As the cut could have vertex w on either side, both vertices need to fulfill this condition. To make sure that we do not contract any trivial minimum cut, we check that both v and u have weight $> \hat{\lambda}$ and thus can not represent a trivial minimum cut.
4. In this condition we check the whole shared neighborhood of vertices u and v. Every cut that separates u and v must contain e and for each shared neighbor w at least one of the edges connecting them to w. Thus, we sum over the lighter edge connecting them to the shared neighbors and have a lower bound of the minimum cut that separates u and v. If this is heavier than $\hat{\lambda}$, we know that no minimum cut separates u and v. ◀

The conditions `HeavyEdge` and `ImbalancedVertex` can both be checked for the whole graph in a single run in linear time. While we can check condition `ImbalancedTriangle` when summing up the lighter incident edges for condition `HeavyNeighborhood`, exhaustively checking all triangles incurs a strictly worse than linear runtime, as a graph can have up to $\Theta(m^{3/2})$ triangles [52]. Thus, we only perform linear-time runs as developed by Chekuri et al. [12] by marking the neighborhood of u and v while we check the conditions and do not perform the test on marked vertices.

3.1.3 Vertices with one Neighbor

Over the run of the algorithm, we occasionally encounter vertices that have only a single neighbor. Let v be this vertex with one neighbor and $e = (v, w)$ be the only incident edge. As we update $\hat{\lambda}$ to the minimum degree whenever we perform a bulk edge contraction, $c(e) \geq \hat{\lambda}$: for an edge whose weight is $> \hat{\lambda}$, condition `HeavyEdge` will contract it. For an edge whose weight is $\hat{\lambda}$, the edge represents a trivial minimum cut iff $\hat{\lambda} = \lambda$. This is the only minimum cut that contains e, as every non-trivial cut containing e has higher weight. Thus, we can contract e for now and remember that it was contracted. If $\hat{\lambda}$ is decreased, we can forget about these vertices as the cuts are not minimal. When we are finished, we can re-insert all contracted vertices that have a trivial minimum cut. We perform this reinsertion in a bottom-up fashion (i.e. in reverse order to how they were contracted), as the neighbor w could be contracted in a later contraction.

3.2 Finding all Minimum Cuts

We apply the reductions in the previous section exhaustively until they are not able to find a significant number of edges to contract. On the remaining graph we aim to find the cactus representation of all minimum cuts. Our algorithm for this purpose is based on the algorithm of Nagamochi, Nakao and Ibaraki [45]. While there is a multitude of algorithms for the problem of finding all minimum cuts, to the best of our knowledge there are no implementations accessible to the public and there is no practical experimentation on finding all minimum cuts. We base our algorithm on the algorithm of Nagamochi, Nakao and Ibaraki [45], as their algorithm allows us to run the reduction routines previously detailed in between recursion steps.

We give a quick sketch of their algorithm, for details we refer the reader to [45]. To find all minimum cuts in graph G, the algorithm chooses an edge $e = (s, t)$ in G and uses a maximum flow f to find the minimum s-t-cut $\lambda(s, t)$. If $\lambda(s, t) > \lambda$ there is no minimum cut that separates s and t and thus e can be contracted. If $\lambda(s, t) = \lambda$, the edge is part of at least one minimum cut. They show that the strongly connected components $(V_1, \ldots, V_k)$ of the residual graph G_f represent all minimum cuts that contain e (and potentially some more). For each connected component V_i, they build a graph C_i, in which all other connected components are contracted into a single vertex. We recurse on these component subgraphs and afterwards combine the minimum cut cactus graphs of the recursive calls to a cactus representation for G.

The combination of the cactus graphs begins by building a cactus graph C representing the set of strongly connected components, in which each V_i is represented by a single vertex v_i. Each cactus C_i is then merged with C by replacing v_i with C_i. For details we refer the reader to [45].

As the contraction routines in Section 3.1 usually mark a large amount of edges that can be contracted in bulk, we represent the graph in the compressed sparse row format [56]. This allows for fast and memory-efficient accesses to vertices and edges, however, we need to completely rebuild the graph in each bulk contraction and also keep vertex information about the whole graph hierarchy to be able to see which vertices in the original graph are encompassed in a vertex in a coarser vertex and to be able to re-introduce the cactus edges that were removed. While this is efficient for the bulk contractions performed in the previous section, in this section we often perform single-edge contractions or contract a small block of vertices. For fast running times these operations should not incur a complete rebuild of the graph data structure. We therefore use a mutable adjacency list data structure where

each vertex is represented by a dynamic array of edges to neighboring vertices. Each edge stores its weight, target and the ID of its reverse edge (as we look at undirected graphs). This allows us to contract edges and small blocks in time corresponding to the sum of vertex degrees. For each vertex in the original graph, we store information which vertex currently encompasses it and every vertex keeps a list of currently encompassed vertices of the original graph. This information is updated during each edge contraction. Inside this algorithm we re-run the contraction routines of Section 3.1. As they incur some computational cost and the graph does not change too much over different recursion steps, we only run the contraction routines every 10 recursion steps.

3.2.1 Edge Selection

The recursive algorithm of Nagamochi, Nakao and Ibaraki [45] selects an arbitrary edge for the maximum flow problem in each recursion step. If this edge has connectivity equal to the minimum cut, we create a recursive subproblem for each connected component of the residual graph. In order to reduce the graph size - and thus the amount of work necessary - quickly, we aim to select edges in which the largest connected component of the residual graph is as small as possible. The edge selection strategy `Heavy` searches for the highest degree vertex v and chooses the edge from v to its highest degree neighbor. The strategy `WeightedHeavy` does the same, but uses the vertices whose weighted degree is highest. The idea is that an edge between high-degree vertices is most likely 'central' to the graph and thus manages to separate sizable chunks from the graph. The edge selection strategy `Central` aims to find a central edge more directly: we aim to find two vertices u and v with a high distance and take the central edge in their shortest paths. We find those vertices by performing a breadth-first search from a random vertex w, afterwards performing a breadth-first search from the vertex encountered last. We then take the central edge in the shortest path (as defined from the second breadth-first search) from the two vertices encountered last in the two breadth-first searches. The edge selection strategy `Random` picks a random edge.

3.2.2 Degree-two Reductions

Over the course of this recursive contraction-based algorithm, we routinely encounter vertices with just two neighbors. Let v be the vertex in question, which is connected to u_0 by edge e_0 and to u_1 by edge e_1. We look at four cases, each looking at whether the weight of e_0 being equal to the weight of e_1 and $c(v)$ being equal to λ, both conditions that can be checked in constant time. In three out of four cases, we are able to contract an incident edge.

$c(e_0) \neq c(e_1)$ and $c(v) > \lambda$: Without loss of generality let e_0 be the heavier edge. As $c(v) > \lambda$, the trivial cut $(\{v\}, V \setminus \{v\})$ is not a minimum cut. As by definition no cut in G is smaller than λ, $\lambda(u_0, u_1) \geq \lambda$. Thus, excluding the path through v, they have a connectivity of $\geq \lambda - c(e_1)$ and any cut containing e_0 has weight $\geq \lambda - c(e_1) + c(e_0) > \lambda$ and can thus not be minimal. We therefore know that e_0 is not part of any minimum cuts and can be contracted according to Lemma 1.

$c(e_0) \neq c(e_1)$ and $c(v) = \lambda$: Without loss of generality let e_0 be the heavier edge. Analogously to the previous case we can show that no nontrivial cut contains e_0. In this case, where $c(v) = \lambda$, the trivial cut $(\{v\}, V \setminus \{v\})$ is minimal and therefore should be represented in the cactus graph. For all other minimum cuts that contain e_1, we know that v and u_0 will be in the same block (as $c(e_0) > c(e_1)$). Thus, v will be represented in the cactus as a leaf incident to u_0. We contract e_0 calling the resulting vertex u^* and store which vertices of the original graph are represented by v. Then we recurse. On return from the recursion we check which cactus vertex now encompasses u^* and add an edge from this vertex to a newly added vertex representing all vertices encompassed by v.

Algorithm 1 Algorithm to find all minimum cuts.

```
1: procedure FindAllMincuts(G = (V, E))
2:     λ̂ ← VieCut(G) [28]
3:     while not converged do
4:         (G, D_1, λ̂) ← contract degree-one vertices(G, λ̂)
5:         (G, λ̂) ← connectivity-based contraction(G, λ̂)
6:         (G, λ̂) ← local contraction(G, λ̂)
7:     λ ← FindMinimumCutValue(G)
8:     C ← RecursiveAllMincuts(G, λ) ([45])
9:     C ← reinsert vertices(C, D_1)
10:    return (C, λ)
```

$c(e_0) = c(e_1)$ and $c(v) > \lambda$: in this case we are not able to contract any edges without further connectivity information.

$c(e_0) = c(e_1)$ and $c(v) = \lambda$: as $c(v) = \lambda$, the trivial cut $(\{v\}, V \setminus \{v\})$ is minimal. If there are other minimum cuts that contain either e_0 or e_1 (e.g. that separate u_0 and u_1), we know that by replacing e_0 with e_1 (or vice-versa) the cut remains minimal. Such a minimum cut exists iff $\lambda(u_0, u_1) = \lambda$. We contract e_0 and remember this decision. As e_1 is still in the graph (merged with (u_0, u_1)), we are able to find each cut that separates u_0 and u_1. If none exists, $\lambda(u_0, u_1) > \lambda$ and u_0 and u_1 will be contracted into a single vertex later in the algorithm. When leaving the recursion, we can thus re-introduce vertex v as a leaf connected to the vertex encompassing u_0 and u_1. If u_0 and u_1 are in different vertices after leaving the recursion, there is at least one nontrivial cut that contains e_1. We thus re-introduce v as a cycle vertex connected to u_0 and u_1, each with weight $\frac{\lambda}{2}$, and subtract $\frac{\lambda}{2}$ from $c(u_0, u_1)$.

In three out of the four cases presented here, we are able to contract an edge incident to a degree-two vertex. We can check these conditions in total time $\mathcal{O}(n)$ for the whole graph. Over the course of the algorithm, we perform edge contractions and thus routinely encounter vertices whose neighborhood has been contracted and thus have a degree of two. Thus, these reductions are able to reduce the size of the graph significantly even if the initial graph is rather dense and does not have a lot of low degree vertices.

3.3 Putting it all together

Algorithm 1 gives an overview over our algorithm to find all minimum cuts. Over the course of the algorithm we keep an upper bound $\hat{\lambda}$ for the minimum cut, initially set to the result of the inexact variant of the `VieCut` minimum cut algorithm [28]. While the `VieCut` algorithm also offers an exact version [26], we use the inexact version, as it is considerably faster and gives a low upper bound for the minimum cut, usually equal to the minimum cut. As described in Section 3.1, we use this bound to contract degree-one vertices, high-connectivity edges and edges whose local neighborhood guarantees that they are not part of any minimum cut. We repeat this process until it is converged, as an edge contraction can cause other edges in the neighborhood to also become safely contractible. As this process often incurs a long tail of single edge contractions, we stop if the number of vertices was decreased by less than 1% over a run of all contraction routines.

We then use the minimum cut algorithm of Nagamochi, Ono and Ibaraki [43, 46] on the remaining graph, as the following steps need the correct minimum cut. To find all minimum cuts in the contracted graph, we call our optimized version of the algorithm of Nagamochi et al. [45], as sketched in Section 3.2, and afterwards re-insert all minimum cut edges that were

previously deleted. Before each recursive call of the algorithm of Nagamochi et al. [45], we contract edges incident to degree-one and eligible degree-two vertices. Every 10 recursion levels we additionally check for connectivity-based edge contractions and local contractions.

3.4 Shared-Memory Parallelism

Algorithm 1 employs shared-memory parallelism in every step. When we run the algorithm in parallel, we use the parallel variant of VieCut [28]. Local contraction and marking of degree one vertices are parallelized using OpenMP [13]. For the first round of connectivity-based contraction, we use the parallel connectivity certificate used in the shared-memory parallel minimum cut algorithm by Henzinger et al. [26]. This connectivity certificate is essentially a parallel version of the connectivity certificate of Nagamochi et al. [43, 46], in which the processors divide the work of computing the connectivity bounds for all edges of the graph. In subsequent iterations every processor runs an independent run of the connectivity certificate of Nagamochi et al. on the whole graph starting from different random vertices in the graph. As the connectivity bounds given by the algorithm heavily depend on the starting vertex, this allows us to find significantly more contractible edges per round than running the connectivity certificate only once.

We use the shared-memory parallel minimum cut algorithm of Henzinger et al. [26] to find the exact minimum cut of the graph. The algorithm of Nagamochi et al. [45] is not shared-memory parallel, however we usually manage to contract the graph to a size proportional to the minimum cut cactus before calling them. Unfortunately it is not beneficial to perform the recursive calls embarrassingly parallel, as in almost all cases one of the connected components of the residual graph contains the vast majority of vertices and thus also has the overwhelming majority of work.

4 Applications

We can use the minimum cut cactus C_G to find a minimum cut fulfilling certain balance criteria, such as a most balanced minimum cut, e.g. a minimum cut $(A, V \setminus A)$ that maximizes $\min(|A|, |V \setminus A|)$. Note that this is not equal to the most balanced s-t-cut problem, which is NP hard [9]. Following that we show how to modify the algorithm to find the optimal minimum cut for other optimization functions.

One can find a most balanced minimum cut trivially in time $\mathcal{O}\big((n^*)^3\big)$, as one can enumerate all $\mathcal{O}\big((n^*)^2\big)$ minimum cuts [33] and add up the number of vertices of the original graph G on either side. We now show how to find a most balanced minimum cut of a graph G in $\mathcal{O}(n^* + m^*)$ time, given the minimum cut cactus graph C_G.

For every cut $(A, V\setminus A)$, we define the balance $b(A)$ (or $b(V\setminus A)$) of the cut as the number of vertices of the original graph encompassed in the lighter side of the cut. Recall that for any node $v \in V_G$, $c(v)$ is the number of vertices of G represented by v. For a leaf $v \in V_G$, we set its weight $w(v) = c(v)$ and set the balance $b(v)$ to be the minimum of $w(v)$ and $n - w(v)$. We root C_G in an arbitrary vertex and depending on that root define $w(v)$ as the sum of vertex weights in the subcactus rooted in v; and $b(v)$ accordingly. For a cycle $C = \{c_1, \dots, c_i\}$, we define $b(c_j, \dots, c_{k \mod i})$ with $0 \geq j \geq k$ analogously as the balance of the minimum cut splitting the cycle so that the sub-cacti rooted in $c_j, \dots, c_{k \mod i}$ are on one side of the cut and the rest are on the other side (see blue line in Figure 2 for an example).

Let T_G be the tree representation of C_G where each cycle in C_G is contracted into a single vertex. We perform a depth-first search on T_G rooted on an arbitrary vertex and check the balance of every cut in T_G when backtracking.

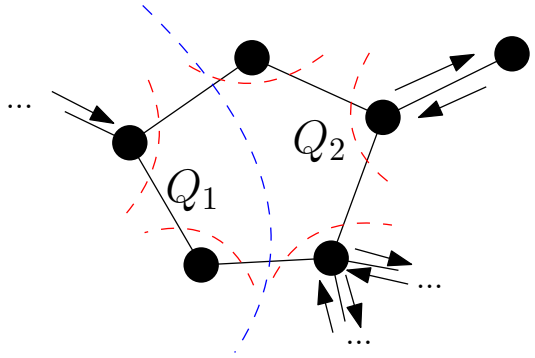


Figure 2 Cycle check in balanced cut algorithm.

As C_G is not necessarily a tree, we might encounter cycles and we explain next how to extend the depth first search to handle such cycles. Let $\mathcal{C} = \{c_0, \ldots, c_{i-1}\}$ be a cycle and c_0 be the vertex encountered first by the DFS. Due to the cactus graph structure of C_G, the depth-first search backtracks from a vertex v_{cy} in T_G that represents $\mathcal{C}$ only after all subtrees rooted in $\mathcal{C}$ are explored. Thus, we know the weight of all subtrees rooted in vertices $c_1, \ldots, c_{i-1}$ when backtracking. The weight of c_0 is equal to n minus the sum of these sub-cactus weights.

Examining all cuts in the cycle would take i^3 time, but as we only want to find the most balanced cut, we can check only a subset of them, as shown in Algorithm 2. Q_1 and Q_2 are *queues*, thus elements are ordered and the following operations are supported: *queue* adds an element to the back of the queue, called the *tail* of the queue, *dequeue* removes the element at the front of the queue, called the *head* of the queue. We implicitly use the fact that queues can only be appended to, thus an element q was added to the queue after all elements that are closer to the head of the queue and before all elements that are closer to its tail.

Algorithm 2 Algorithm to find most balanced cut in cycle $\{c_0, \ldots, c_{i-1}\}$.

```
1:  procedure BalanceInCycle(G = (V, E), C = {c_1, ..., c_i})
2:      b_OPT ← 0
3:      Q_1 = Queue({})
4:      Q_2 = Queue({c_0, c_1, ..., c_{i-1}})
5:      while c_0 not Q_1.head() for second time do
6:          b_OPT ← checkBalance(Q_1, Q_2)
7:          if w(Q_1) > w(Q_2) then
8:              Q_2.queue(Q_1.dequeue())
9:          else
10:             Q_1.queue(Q_2.dequeue())
11:     return b_OPT
```

The weight of a queue $w(Q)$ is denoted as the weight of its contents. For queue $Q = \{c_{j \bmod i}, \ldots, c_{k \bmod i}\}$ with $0 \leq j \leq k$, we use the notation $w_{j \bmod i, k \bmod i}$ to denote the weight of Q and $\overline{w_{j \bmod i, k \bmod i}}$ as the weight of the queue that contains all cycle vertices not in Q.

In every step of the algorithm, the cut represented by the current state of the queues consists of the two edges connecting the queue heads to the tails of the respective other queue. Initially Q_1 is empty and Q_2 contains all elements, in order from c_0 to c_{i-1}. In every

step of the algorithm, we dequeue one element and queue it in the other queue. Thus, at every step each cycle vertex is in exactly one queue. When we check the balance of a cut, we compute the weight of each queue at the current point in time; and update b_{OPT}, the best balance found so far, if (Q_1, Q_2) is more balanced. As we only move one cycle vertex in each step, we can check the balance of an adjacent cut in constant time by adding and subtracting the weight of the moved vertex to the weights of each set.

▶ **Lemma 3.** *Algorithm 2 terminates after $O(i)$ steps.*

Proof. In each step of Algorithm 2, one queue head is moved to the other queue. The algorithm terminates when c_0 is the head of Q_1 for the second time. In the first step, c_0 is moved to Q_1, as the empty queue Q_1 is the lighter one. The algorithm terminates after c_0 then performs a full round through both queues and is the head of Q_1 again. At termination, c_0 was thus moved a total of three times, twice from Q_2 to Q_1 and once the other way. As no element can 'overtake' c_0 in the queues, every vertex will be moved at most three times. Thus, we enter the loop at most $3i$ times, each time only using a constant amount of time. ◀

In Algorithm 2, we only check the balance of a subset of cuts represented by edges in the cycle C. Lemma 5 shows that none of the disregarded cuts can have balance better than b_{OPT} and we thus find the most balanced minimum cut. We call a cut disregarded if its balance was never checked (Line 6), and considered otherwise. In order to prove correctness of Algorithm 2, we first show the following Lemma:

▶ **Lemma 4.** *Each vertex in the cycle is dequeued from Q_1 at least once in the algorithm.*

Proof. The algorithm terminates when c_0 is the head of Q_1 for the second time. For this, it needs to be moved from Q_2 to Q_1 twice. As we queue elements to the back of a queue, all vertices are dequeued from Q_2 before c_0 is dequeued from it for the second time. In order for c_0 to become the head of Q_1 again, all elements that were added beforehand need to be dequeued from Q_1. ◀

▶ **Lemma 5.** *Algorithm 2 finds the most balanced minimum cut represented by cycle C.*

Proof. We now prove for each $c_l \in \mathcal{C}$ that all disregarded cuts containing the cycle edge separating c_l from $c_{(l-1) \bmod i}$ are not more balanced than the most balanced cut found so far. As no disregarded cut can be more balanced than the most balanced cut considered in the algorithm, the output of the algorithm is the most balanced minimum cut; or one of them if multiple cuts of equal balance exist.

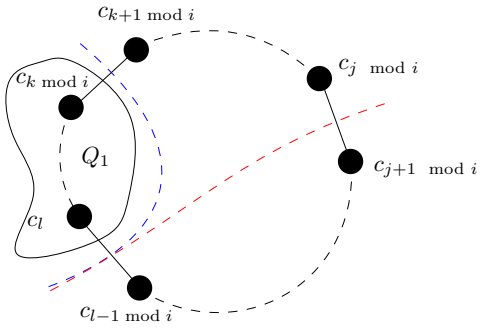


Figure 3 State of Q_1 at time t_l (cut in blue). Cut in red denotes cut considered at time t^*.

Let t_l be the time that c_l becomes the head of Q_1 for the first time. Figure 3 shows the state of Q_1 at that point in time. Let $c_{k \bmod i}$ be the tail of Q_1 at time t_l for some integer k. Right before t_l, $c_{l-1 \bmod i}$ was head of the heavier queue Q_1 and thus dequeued, i.e. $Q_1 = \{c_{l-1 \bmod i}, \ldots, c_{k \bmod i}\}$ has weight $w_{l-1 \bmod i, k \bmod i} \geq \overline{w_{l-1 \bmod i, k \bmod i}}$ and c_l is now head of Q_1.

From this point t_l the algorithm considers cuts that separate c_l from $c_{l-1 \bmod i}$. While Q_1 is not heavier than Q_2, we add more elements to the tail of Q_1 (and check the respective cuts) until Q_1 is the heavier queue. Let t^* be the time when this happens and $c_{j \bmod i}$ with $j \geq k$ be the tail of Q_1 at this point. Note that at time t^*, c_l is about to be dequeued from Q_1. The red cut in Figure 3 shows the cut at time t^*, where $w_{c_l, c_{j \bmod i}} > \overline{w_{c_l, c_{j \bmod i}}}$.

We now prove that all cuts in which c_l is the head of Q_1 and its tail is not between $c_{k \bmod i}$ and $c_{j \bmod i}$ cannot be more balanced than the most balanced cut considered so far.

For all cuts where c_l is head of Q_1 and Q_1 also contains $c_{j+1 \bmod i}$, Q_1 is heavier than $w_{l,j \bmod i}$, as it contains all elements in $c_l, \ldots, c_{j \bmod i}$ plus at least one more. As $w_{l,j \bmod i} > \overline{w_{l,j \bmod i}}$, i.e. Q_1 is already heavier when $c_{j \bmod i}$ is its tail, all of these cuts are less balanced than $(\{c_l, \ldots, c_{j \bmod i}\}, \mathcal{C} \backslash \{c_l, \ldots, c_{j \bmod i}\})$.

For the cuts in which $c_{k \bmod i}$ is in Q_2, i.e. Q_1 is lighter than at time t_l, we need to distinguish two cases, depending on whether $w_{l,k \bmod i}$ is larger than $\overline{w_{l,k \bmod i}}$ or not.

If $w_{l,k \bmod i} \leq \overline{w_{l,k \bmod i}}$, all cuts in which c_l is the head of Q_1 and $c_{k \bmod i}$ is in Q_2 are less balanced than $(\{c_l, \ldots, c_{k \bmod i}\}, \mathcal{C} \backslash \{c_l, \ldots, c_{k \bmod i}\})$, as Q_1 is lighter than it is at t_l, where it was already not the heavier queue.

If $w_{l,k \bmod i} > \overline{w_{l,k \bmod i}}$, there might be cuts in which c_l is the head of Q_1 that are more balanced than $(\{c_l, \ldots, c_{k \bmod i}\}, \mathcal{C} \backslash \{c_l, \ldots, c_{k \bmod i}\})$ in which Q_1 is lighter than at time t_l. Thus, consider time t' when $c_{k \bmod i}$ was added to Q_1. Such a time must exist, since Q_1 is initially empty. As $c_{k \bmod i}$ is already the tail of Q_1 at time t_l, $t' < t_l$. At that time Q_1 contained $c_{l-1 \bmod i}, \ldots, c_{k-1 \bmod i}$ and potentially more vertices.

Still, $w_{l-1 \bmod i, k-1 \bmod i} \leq \overline{w_{l-1 \bmod i, k-1 \bmod i}}$, as otherwise $c_{k \bmod i}$ would not have been added to Q_1. Obviously $w_{l-1 \bmod i, k-1 \bmod i} > w_{l, k-1 \bmod i}$, as Q_1 is even lighter when $c_{l-1 \bmod i}$ is dequeued. As $w_{l-1 \bmod i, k-1 \bmod i}$ is already not heavier than its complement, $(\{c_l, \ldots, c_{k-1 \bmod i}\}, \mathcal{C} \backslash \{c_l, \ldots, c_{k-1 \bmod i}\})$ is more imbalanced than the cut examined just before time t'. Thus, all cuts where c_l is the head of Q_1 and $c_{k-1 \bmod i}$ is in Q_2 are even more imbalanced, as Q_1 is even lighter.

Coming back to the outline shown in Figure 3, we showed that for all cuts in which c_l is head of Q_1 and Q_1 is lighter than at time t_l (left of blue cut) and all cuts where Q_1 is heavier than at time t^* (below red cut) can be safely disregarded, as a more balanced cut than any of them was considered at some point between t' and t^*. The algorithm considers next all cuts with c_l as head of Q_1 and the tail of Q_1 between $c_{k \bmod i}$ and $c_{j \bmod i}$. Thus, the algorithm will return a cut that is at least as balanced as the most balanced cut that separates c_l and $c_{l-1 \bmod i}$. This is true for every cycle vertex $v_l \in \mathcal{C}$, which concludes the proof. ◀

This allows us to perform the depth-first search and find the most balanced minimum cut in C_G in time $\mathcal{O}(n^* + m^*)$. This algorithm can be adapted to find the minimum cut of any other optimization function of a cut that only depends on the (weight of the) edges on the cut and the (weight of the) vertices on either side of the cut. In order to retain the linear running time of the algorithm, the function needs to be evaluable in constant time on a neighboring cut. For example, we can find the minimum cut of lowest conductance. The conductance of a cut $(S, V \setminus S)$ is defined as $\frac{\lambda(S,(V \setminus S))}{min(a(S),a(V \setminus S))}$, where $a(S)$ is the sum of degrees for all vertices in set S. Note that this is not the minimum conductance cut problem,

which is NP-hard [4], as we only look at the minimum cuts. To find the minimum cut of lowest conductance, we set the weight of a vertex $v_{C_G} \in C_G$ to the sum of vertex degrees encompassed in v_{C_G}. Otherwise the algorithm remains the same.

5 Experiments and Results

We now perform an experimental evaluation of the proposed algorithms. This is done in the following order: first analyze the impact of algorithmic components on our minimum cut algorithm in a non-parallel setting, i.e. we compare different variants for edge selection and see the impact of the various optimizations detailed in this work. Afterwards, we report parallel speedup on a variety of large graphs.

Experimental Setup and Methodology

We implemented the algorithms using C++-17 and compiled all code using g++ version 8.3.0 with full optimization (`-O3`). Our experiments are conducted on a machine with two Intel Xeon Gold 6130 processors with 2.1GHz with 16 CPU cores each and 256 GB RAM in total. We perform five repetitions per instance and report average running time. In this section we first describe our experimental methodology. Afterwards, we evaluate different algorithmic choices in our algorithm and then we compare our algorithm to the state of the art. When we report a mean result we give the geometric mean as problems differ significantly in cut size and time. Our code is freely available under the permissive MIT license [1].

Instances

We use a variety of graphs from the 10th DIMACS Implementation challenge [6] and the SuiteSparse Matrix Collection [15]. These are social graphs, web graphs, co-purchase matrices, cooperation networks and some generated instances. Table 2 shows a set of smaller instances and Table 3 shows a set of larger and harder to solve instances. All instances are undirected. If the original graph is directed, we generate an undirected graph by removing edge directions and then removing duplicate edges. If a network has multiple connected components, we run on the largest.

As most large real-world networks have cuts of size 1, finding all minimum cuts becomes essentially the same as finding all bridges, which can be solved in linear time using depth-first search [55]. However, usually there is usually one huge block that is connected by minimum cuts to a set of small and medium size blocks. Thus, we use our algorithm to generate a more balanced set of instances. We find all minimum cuts and contract each edge that does not connect two vertices of the largest block. Thus, the remaining graph only contains the huge block and is guaranteed to have a minimum cut value $> \lambda$. We use this method to generate multiple graphs with different minimum cuts for each instance.

5.1 Edge Selection

Figure 4 shows the results for the graphs in Table 2. We compute the cactus graph representing all minimum cuts using the edge selection variants `Random`, `Central`, `Heavy` and `HeavyWeighted`, as detailed in Section 3.2. As we want a majority of the running time in the recursive algorithm of Nagamochi et al. [45], where we actually select edges, we run a variant of our algorithm that only contracts edges using connectivity-based contraction and then runs the algorithm of Nagamochi et al. [45].

[1] `https://github.com/VieCut/VieCut`

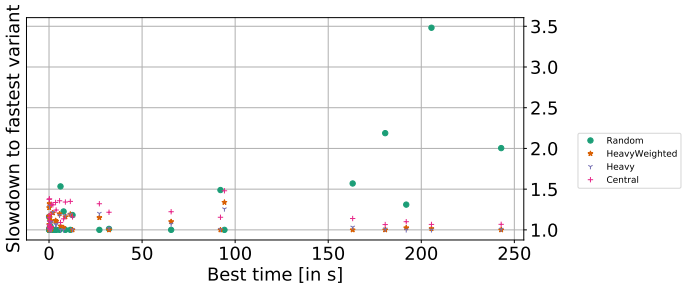


Figure 4 Effect of edge selection strategies.

We can see that in the graphs which cannot be contracted quickly, `Random` is significantly slower than all other variants. On `cnr-2000`, `Random` takes over 700 seconds in average, whereas all other variants finish in approximately 200 seconds. This happens independently of the random seed used, there is no large deviation in the running time on any of the graphs. On almost all graphs, the variants `Heavy` and `HeavyWeighted` are within 3% of each other, which is not surprising, as the variants are almost identical. While it optimizes for 'edge centrality' very directly, `Central` has two iterations of breadth-first search in each edge selection and thus a sizable overhead. For this reason it is usually $5-15\%$ slower than `Heavy` and is not the fastest algorithm on any graph. On graphs with large n^*, all three variants manage to shrink the graph significantly faster than `Random`.

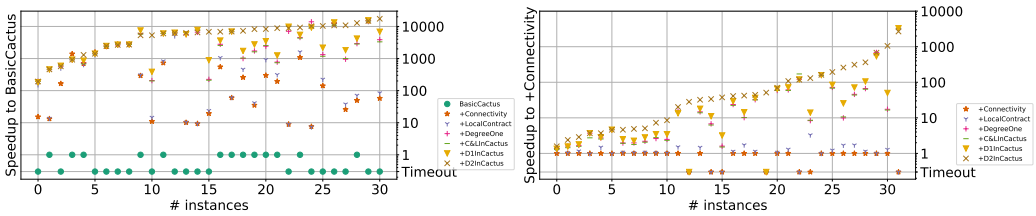
On graphs with a low value of n^*, we can see that `Random` is slightly faster than the other variants. There is no significant difference in the shrinking of the graph, as almost all selected edges have connectivity larger than λ and thus only trigger a single edge contraction anyway. Thus, not spending the extra work of finding a "good" edge results in a slightly lower running time. In the following we will use variant `Heavy`, which is the only variant that is never more than 30% slower than the fastest variant on any graph.

5.2 Optimization

We now examine the effect of the different optimizations. For this purpose, we benchmarks different variants on a variety of graphs. We hereby compare the following variants that build on one another: as a baseline, `BasicCactus` runs the algorithm of Nagamochi, Nakao and Ibaraki [45] on the input graph. `+Connectivity` additionally runs VieCut [28] to find an upper bound for the minimum cut and uses this to contract high-connectivity edges as described in Section 3.1.1. In addition to this, `+LocalContract` also contracts edges whose neighborhood guarantees that they are not part of any minimum cut, as described in Section 3.1.2 and Lemma 2. `+DegreeOne` runs also the last remaining contraction routine from Algorithm 1, contraction and re-insertion of degree-one vertices as described in Section 3.1.3.

`+C&LInCactus` additionally runs high-connectivity and local contraction in every tenth recursion step. `+D1InCactus` additionally contracts and re-inserts degree-one vertices in every recursion step. `FullAlgorithm` also runs the degree-two contraction as described in Section 3.2.2. We compare these variants on the graphs in Tables 2 and 3. We use a timeout of 30 minutes on these graphs. If the baseline algorithm does not finish in the allotted time, we report speedup to the timeout, so a lower bound for the actual speedup.

Figure 5a shows the speedup of all variants to the baseline variant `BasicCactus` on all graphs in Table 2. We can see that already just adding `+Connectivity` gives a speedup of more than an order of magnitude for each of the graphs in the dataset. Most of the



(a) Speedup to `BasicC` on small graphs (Table 2) **(b)** Speedup to `+Conn` on large graphs (Table 3)

Figure 5 Speedup to respective basic version.

other optimizations manage to improve the running time of at least some instances by a large margin. Especially `+DegreeOne`, which is the first contraction for edges that are in a minimum cut, has speedups of multiple orders of magnitude in some instances. This is the case as minimum cut edges that are incident to a degree-one vertex previously incur a flow problem on the whole graph each. However, it is very easy to see that the edge will be part of exactly one minimum cut, thus we can contract and re-insert it in constant time. Especially in graphs whose minimum cut is 1, all edges can be quickly contracted, as they will either be incident to a degree-one vertex or be quickly certified to have a connectivity value of > 1.

While rerunning `Connectivity` and `LocalContract` inside of the recursive algorithm of Nagamochi et al. [45] does usually not yield a large speedup, many graphs develop degree-one vertices by having their whole neighborhood contracted. Thus, `+D1InCactus` has a significant speedup for most graphs in which n^* is sufficiently large. `FullAlgorithm` has an even larger speedup on these graphs, even when the minimum cut is significantly higher than 2, as there are often cascading effects where the contraction of an edge incident to a degree-two vertex often lowers the degree of neighboring vertices to two.

Figure 5b shows the speedup of all variants. As variant `BasicCactus` is not able to solve any of these instances in 30 minutes, we use `+Connectivity` as a baseline. The results are similar to Figure 5a, but we can see even clearer how useful the contraction of degree-two vertices is in finding all minimum cuts: `FullAlgorithm` often has a speedup of more than an order of magnitude to all other variants and is the only variant that never times out.

5.3 Shared-memory Parallelism

Table 1 shows the average running times of our algorithm both sequential and with 16 threads on huge social and web graphs. Each of these graphs has more than a billion of edges and more than a million vertices in the cactus graph depicting all minimum cuts. On these graphs we have a parallel speedup factor of 5.7x to 9.1x using 16 threads. On all of these graphs, a large part of the running time is spent in the first iteration of the kernelization routines, which already manages to contract most dense blocks in the graph. Thus, all subsequent operations can be performed on significantly smaller problems and are therefore much faster.

Table 1 Huge social and web graphs. n^* denotes number of vertices in cactus graph, max n and max m denote size of smaller block in most balanced cut.

Name	n	m	n^*	λ	max. n	max. m	seq. t	par. t
friendster	65.6M	1.81B	13.99M	1	897	1 793	1266.35s	138.34s
twitter7	41.7M	1.20B	1.93M	1	47	1 893	524.86s	72.51s
uk-2007-05	104.3M	3.29B	9.66M	1	49 984	13.8M	229.18s	40.16s

6 Conclusion

We engineered an algorithm to find all minimum cuts in large undirected graphs. Our algorithm combines multiple kernelization routines with an engineered version of the algorithm of Nagamochi, Nakao and Ibaraki [45] to find all minimum cuts of the reduced graph. Our experiments show that our algorithm can find all minimum cuts of huge social networks with up to billions of edges and millions of minimum cuts in a few minutes on shared memory. We found that especially the contraction of high-connectivity edges and efficient handling of low-degree vertices can give huge speedups. Additionally we give a linear time algorithm to find the most balanced minimum cut given the cactus graph representation of all minimum cuts. Future work includes finding near-minimum cuts.

References

1 Faisal N. Abu-Khzam, Shaowei Cai, Judith Egan, Peter Shaw, and Kai Wang. Turbocharging dominating set with an FPT subroutine: Further improvements and experimental analysis. In *Proc. 14th Annual Conference on Theory and Applications of Models of Computation (TAMC 2017)*, volume 10185 of *LNCS*, pages 59–70. Springer, 2017. doi:10.1007/978-3-319-55911-7_5.

2 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609, Part 1:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.

3 Takuya Akiba, Yoichi Iwata, Yosuke Sameshima, Naoto Mizuno, and Yosuke Yano. Cut tree construction from massive graphs. In *Proc. 16th International Conference on Data Mining (ICDM 2016)*, pages 775–780, 2016. doi:10.1109/ICDM.2016.0089.

4 Reid Andersen and Kevin J Lang. An algorithm for improving graph partitions. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 651–660. SIAM, 2008. URL: https://dl.acm.org/doi/10.5555/1347082.1347154.

5 Richard J Anderson and Heather Woll. Wait-free parallel algorithms for the union-find problem. In *Proc. 23rd ACM Symposium on Theory of Computing*, STOC '91, pages 370–380. ACM, 1991. doi:10.1145/103418.103458.

6 David A Bader, Henning Meyerhenke, Peter Sanders, Christian Schulz, Andrea Kappes, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. *Encyclopedia of Social Network Analysis and Mining*, pages 73–82, 2014. doi:10.1007/978-1-4614-7163-9_23-1.

7 Max Bannach and Sebastian Berndt. Practical access to dynamic programming on tree decompositions. In *Proc. 26th European Symposium on Algorithms (ESA'18)*, volume 112 of *LIPIcs*, pages 6:1–6:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.6.

8 Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011. doi:10.1007/s00453-009-9339-7.

9 Paul Bonsma. Most balanced minimum cuts. *Discrete Appl. Math.*, 158(4):261–276, February 2010. doi:10.1016/j.dam.2009.09.010.

10 Deng Cai, Zheng Shao, Xiaofei He, Xifeng Yan, and Jiawei Han. Mining hidden community in heterogeneous social networks. In *Proc. 3rd International Workshop on Link Discovery (LinkKDD '05)*, pages 58–65. ACM, 2005. doi:10.1145/1134271.1134280.

11 Lijun Chang, Wei Li, and Wenjie Zhang. Computing a near-maximum independent set in linear time by reducing-peeling. In *Proc. 2017 ACM International Conference on Management of Data (SIGMOD '17)*, pages 1181–1196. ACM, 2017. doi:10.1145/3035918.3035939.

12 Chandra S. Chekuri, Andrew V. Goldberg, David R. Karger, Matthew S. Levine, and Cliff Stein. Experimental study of minimum cut algorithms. In *Proc. 8th Symposium on Discrete Algorithms (SODA '97)*, pages 324–333. SIAM, 1997. URL: https://dl.acm.org/doi/10.5555/314161.314315.

13 Leonardo Dagum and Ramesh Menon. OpenMP: An industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998. doi:10.1109/99.660313.

14 Jakob Dahlum, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Accelerating local search for the maximum independent set problem. In *Proc. 15th International Symposium on Experimental Algorithms (SEA 2016)*, volume 9685 of *LNCS*, pages 118–133. Springer, 2016. doi:10.1007/978-3-319-38851-9_9.

15 Timothy A Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Trans. Mathematical Software (TOMS)*, 38(1):1, 2011. doi:10.1145/2049662.2049663.

16 Chris HQ Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proc. 2001 IEEE International Conference on Data Mining (ICDM 2001)*, pages 107–114. IEEE, 2001. doi:10.1109/ICDM.2001.989507.

17 Damir Ferizovic, Demian Hespe, Sebastian Lamm, Matthias Mnich, Christian Schulz, and Darren Strash. Engineering kernelization for maximum cut. In *Proc. 22nd Symposium on Algorithm Engineering and Experiments (ALENEX 2020)*, pages 27–41. SIAM, 2020. doi:10.1137/1.9781611976007.3.

18 Rudolf Fleischer, Xi Wu, and Liwei Yuan. Experimental study of FPT algorithms for the directed feedback vertex set problem. In *Proc. 17th European Symposium on Algorithms (ESA 2009)*, volume 5757 of *LNCS*, pages 611–622. Springer, 2009. doi:10.1007/978-3-642-04128-0_55.

19 Harold N Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *Proc. 32nd Symposium of Foundations of Computer Science (FOCS '91)*, pages 812–821. IEEE, 1991. doi:10.1109/SFCS.1991.185453.

20 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proc. 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 1260–1279. SIAM, 2020. doi:10.1137/1.9781611975994.77.

21 Lukas Gianinazzi, Pavel Kalvoda, Alessandro De Palma, Maciej Besta, and Torsten Hoefler. Communication-avoiding parallel minimum cuts and connected components. In *Proc. 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 219–232. ACM, 2018. doi:10.1145/3200691.3178504.

22 Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988. doi:10.1145/48014.61051.

23 Ralph E. Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. doi:10.1137/0109047.

24 Lars Hagen and Andrew B Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, 1992. doi:10.1109/43.159993.

25 Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4-6):175–181, 2000. doi:10.1016/S0020-0190(00)00142-3.

26 Monika Henzinger, Alexander Noe, and Christian Schulz. Shared-memory exact minimum cuts. In *Proc. 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2019)*, pages 13–22. IEEE, 2019. doi:10.1109/IPDPS.2019.00013.

27 Monika Henzinger, Alexander Noe, and Christian Schulz. Shared-memory branch-and-reduce for multiterminal cuts. In *Proc. 22nd Symposium on Algorithm Engineering and Experiments (ALENEX 2020)*, pages 42–55. SIAM, 2020. doi:10.1137/1.9781611976007.4.

28 Monika Henzinger, Alexander Noe, Christian Schulz, and Darren Strash. Practical minimum cut algorithms. *ACM Journal of Experimental Algorithmics*, 23, 2018. doi:10.1145/3274662.

29 Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. We got you covered: The winning solver from the PACE 2019 challenge, vertex cover track. In *Proc. SIAM Workshop on Combinatorial Scientific Computing 2020 (CSC20)*, pages 1–11. SIAM, 2020. doi:10.1137/1.9781611976229.1.

30 Demian Hespe, Christian Schulz, and Darren Strash. Scalable kernelization for maximum independent sets. In *Proc. 20th Workshop on Algorithm Engineering and Experiments (ALENEX 2018)*, pages 223–237, 2018. doi:10.1137/1.9781611975055.19.

31 Michael Jünger, Giovanni Rinaldi, and Stefan Thienel. Practical performance of efficient minimum cut algorithms. *Algorithmica*, 26(1):172–195, 2000. doi:10.1007/s004539910009.

32 Goossen Kant. *Algorithms for drawing planar graphs*. PhD thesis, Utrecht University, 1993. URL: https://www.persistent-identifier.nl/urn:nbn:nl:ui:10-1874-842.

33 David R Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, 2000. doi:10.1145/331605.331608.

34 David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Review*, 43(3):499–522, 2001. doi:10.1137/S0036144501387141.

35 David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, 1996. doi:10.1145/234533.234534.

36 Alexander V Karzanov and Eugeniy A Timofeev. Efficient algorithm for finding all minimal edge cuts of a nonoriented graph. *Cybernetics and Systems Analysis*, 22(2):156–162, 1986. doi:10.1007/BF01074775.

37 Krzysztof Kiljan and Marcin Pilipczuk. Experimental evaluation of parameterized algorithms for feedback vertex set. In *Proc. 17th International Symposium on Experimental Algorithms (SEA 2018)*, volume 103 of *LIPIcs*, pages 12:1–12:12, 2018. doi:10.4230/LIPIcs.SEA.2018.12.

38 Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. In *Proc. 26th European Symposium on Algorithms (ESA 2018)*, volume 112 of *LIPIcs*, pages 53:1–53:13, 2018. doi:10.4230/LIPIcs.ESA.2018.53.

39 Arie MCA Koster, Hans L Bodlaender, and Stan PM Van Hoesel. Treewidth: computational experiments. *Electronic Notes in Discrete Mathematics*, 8:54–57, 2001. doi:10.1016/S1571-0653(05)80078-2.

40 Balakrishnan Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. on Computers*, 33(5):438–446, 1984. doi:10.1109/TC.1984.1676460.

41 Sebastian Lamm, Christian Schulz, Darren Strash, Robert Williger, and Huashuo Zhang. Exactly solving the maximum weight independent set problem on large real-world graphs. In *Proc. 21st Workshop on Algorithm Engineering and Experiments (ALENEX 2019)*, pages 144–158. SIAM, 2019. doi:10.1137/1.9781611975499.12.

42 Tobias Maier, Peter Sanders, and Roman Dementiev. Concurrent hash tables: Fast and general(?)! *ACM Trans. Parallel Comput.*, 5(4), 2019. doi:10.1145/3309206.

43 Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992. doi:10.1137/0405004.

44 Hiroshi Nagamochi and Tiko Kameda. Canonical cactus representation for minimum cuts. *Japan Journal of Industrial and Applied Mathematics*, 11(3):343–361, 1994. doi:10.1007/BF03167227.

45 Hiroshi Nagamochi, Yoshitaka Nakao, and Toshihide Ibaraki. A fast algorithm for cactus representations of minimum cuts. *Japan Journal of Industrial and Applied Mathematics*, 17(2):245–264, 2000. doi:10.1007/BF03167346.

46 Hiroshi Nagamochi, Tadashi Ono, and Toshihide Ibaraki. Implementing an efficient minimum capacity cut algorithm. *Math. Prog.*, 67(1):325–341, 1994. doi:10.1007/BF01582226.

47 Dalit Naor, Dan Gusfield, and Charles Martel. A fast algorithm for optimally increasing the edge connectivity. *SIAM Journal on Computing*, 26(4):1139–1165, 1997. doi:10.1137/S0097539792234226.

48 Dalit Naor and Vijay V Vazirani. Representing and enumerating edge connectivity cuts in RNC. In *Proc. 2nd Workshop on Algorithms and Data Structures (WADS 1991)*, volume 519 of *LNCS*, pages 273–285. Springer, 1991. doi:10.1007/BFb0028269.

49 Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991. doi:10.1137/1033004.

50 Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. In *Combinatorial Optimization II*, pages 8–16. Springer, 1980. doi:10.1007/BFb0120902.

51 Aparna Ramanathan and Charles J Colbourn. Counting almost minimum cutsets with reliability applications. *Mathematical Programming*, 39(3):253–261, 1987. doi:10.1007/BF02592076.

52 Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Proc. 4th International Workshop on Experimental and Efficient Algorithms (WEA 2005)*, volume 3503 of *LNCS*, pages 606–609. Springer, 2005. doi:10.1007/11427186_5.

53 Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. doi:10.1109/34.868688.

54 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. In *Proc. 25th European Symposium on Algorithms (ESA 2017)*, volume 87 of *LIPIcs*, pages 68:1–68:13, 2017. doi:10.4230/LIPIcs.ESA.2017.68.

55 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

56 Reginald P Tewarson. *Sparse matrices*. Academic Press, 1973.

57 Zhenyu Wu and Richard Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 1101–1113, 1993. doi:10.1109/34.244673.

A Graph Instances

Table 2 Set of small graphs.

Name	n	m	λ	n^*
amazon	64 813	153 973	1	10 068
auto	448 695	3.31M	4	43
	448 529	3.31M	5	102
	448 037	3.31M	6	557
	444 947	3.29M	7	1 128
	437 975	3.24M	8	2 792
	418 547	3.10M	9	5 814
caidaRouterLevel	190 914	607 610	1	49 940
cfd2	123 440	1.48M	7	15
citationCiteseer	268 495	1.16M	1	43 031
	223 587	1.11M	2	33 423
	162 464	862 237	3	23 373
	109 522	435 571	4	16 670
	73 595	225 089	5	11 878
	50 145	125 580	6	8 770
cnr-2000	325 557	2.74M	1	87 720
	192 573	2.25M	2	33 745
	130 710	1.94M	3	11 604
	110 109	1.83M	4	9 256
	94 664	1.77M	5	4 262
	87 113	1.70M	6	5 796
	78 142	1.62M	7	3 213
	73 070	1.57M	8	2 449
coAuthorsDBLP	299 067	977 676	1	45 242
cs4	22 499	43 858	2	2
delaunay_n17	131 072	393 176	3	1 484
fe_ocean	143 437	409 593	1	40
kron-logn16	55 319	2.46M	1	6 325
luxembourg	114 599	239 332	1	23 077
vibrobox	12 328	165 250	8	625
wikipedia	35 579	495 357	1	2 172

Table 3 Set of large graphs.

Name	n	m	λ	n^*
amazon-2008	735 323	3.52M	1	82 520
	649 187	3.42M	2	50 611
	551 882	3.18M	3	35 752
	373 622	2.12M	5	19 813
	145 625	582 314	10	64 657
coPapersCiteseer	434 102	16.0M	1	6 372
	424 213	16.0M	2	7 529
	409 647	15.9M	3	7 495
	379 723	15.5M	5	6 515
	310 496	13.9M	10	4 579
eu-2005	862 664	16.1M	1	52 232
	806 896	16.1M	2	42 151
	738 453	15.7M	3	21 265
	671 434	13.9M	5	18 722
	552 566	11.0M	10	23 798
hollywood-2009	1.07M	56.3M	1	11 923
	1.06M	56.2M	2	17 386
	1.03M	55.9M	3	21 890
	942 687	49.2M	5	22 199
	700 630	16.8M	10	19 265
in-2004	1.35M	13.1M	1	278 092
	909 203	11.7M	2	89 895
	720 446	9.2M	3	45 289
	564 109	7.7M	5	33 428
	289 715	5.1M	10	12 947
uk-2002	18.4M	261.6M	1	2.5M
	15.4M	254.0M	2	1.4M
	13.1M	236.3M	3	938 319
	10.6M	207.6M	5	431 140
	7.6M	162.1M	10	298 716
	657 247	26.2M	50	24 139
	124 816	8.2M	100	3 863

Approximate Turing Kernelization for Problems Parameterized by Treewidth

Eva-Maria C. Hols
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
eva-maria.hols@fkie.fraunhofer.de

Stefan Kratsch
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
kratsch@informatik.hu-berlin.de

Astrid Pieterse
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
astrid.pieterse@informatik.hu-berlin.de

Abstract

We extend the notion of lossy kernelization, introduced by Lokshtanov et al. [STOC 2017], to approximate Turing kernelization. An α-approximate Turing kernel for a parameterized optimization problem is a polynomial-time algorithm that, when given access to an oracle that outputs c-approximate solutions in $\mathcal{O}(1)$ time, obtains an $\alpha \cdot c$-approximate solution to the considered problem, using calls to the oracle of size at most $f(k)$ for some function f that only depends on the parameter.

Using this definition, we show that Independent Set parameterized by treewidth ℓ has a $(1+\varepsilon)$-approximate Turing kernel with $\mathcal{O}(\frac{\ell^2}{\varepsilon})$ vertices, answering an open question posed by Lokshtanov et al. [STOC 2017]. Furthermore, we give $(1+\varepsilon)$-approximate Turing kernels for the following graph problems parameterized by treewidth: Vertex Cover, Edge Clique Cover, Edge-Disjoint Triangle Packing and Connected Vertex Cover.

We generalize the result for Independent Set and Vertex Cover, by showing that all graph problems that we will call *friendly* admit $(1+\varepsilon)$-approximate Turing kernels of polynomial size when parameterized by treewidth. We use this to obtain approximate Turing kernels for Vertex-Disjoint H-packing for connected graphs H, Clique Cover, Feedback Vertex Set and Edge Dominating Set.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Approximation, Turing kernelization, Graph problems, Treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.60

Related Version A full version of the paper [24] is available at `https://arxiv.org/abs/2004.12683`.

Funding *Eva-Maria C. Hols* and *Astrid Pieterse*: Supported by DFG Emmy Noether-grant (KR 4286/1).

1 Introduction

Many important computational problems are NP-hard and, thus, they do not have efficient algorithms unless P = NP. At the same time, it is well known that *efficient preprocessing* can greatly speed up (exponential-time) algorithms for solving NP-hard problems. The notion of a *kernelization* from parameterized complexity has allowed a rigorous and systematic study of this important paradigm. The central idea is to relate the effectiveness of preprocessing to the structure of the input instances, as quantified by suitable parameters.

A *parameterized problem* consists of any (classical) problem together with a choice of one or more parameters; we use (x, k) to denote an instance with input data x and parameter k. A *kernelization* is an efficient algorithm that on input of (x, k) returns an equivalent instance

(x', k') of size upper bounded by $f(k)$, where f is a computable function. For a polynomial kernelization we require that the size bound $f(k)$ is polynomially bounded in k. The study of which parameterized problems admit (polynomial) kernelizations has turned into a very active research area within parameterized complexity (see, e.g., [1, 5, 7, 8, 16, 23, 27, 28, 29, 31, 35] and the recent book [17]). An important catalyst for this development lies in the ability to prove lower bounds for kernelizations, e.g., to conditionally rule out polynomial kernels for a problem, which was initiated through work of Bodlaender et al. [4] and Fortnow and Santhanam [18].

Unfortunately, the lower bound tools have also revealed that many fundamental parameterized problems do not admit polynomial kernelizations (unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ and the polynomial hierarchy collapses). These include a variety of problems like Connected Vertex Cover [12], Disjoint Cycle Packing [6], Multicut [11], and k-Path [4] parameterized by solution size, but also essentially any NP-hard problem parameterized by width parameters such as treewidth. This has motivated the study of relaxed forms of kernelization, notably Turing kernelization [3] and lossy (or approximate) kernelization [30].

Given an input (x, k), a *Turing kernelization* may create $|x|^{\mathcal{O}(1)}$ many instances of size at most $f(k)$ each, and the answer for (x, k) may depend on solutions for all those instances. This is best formalized as an efficient algorithm that solves (x, k) while being allowed to ask questions of size at most $f(k)$ to an oracle. A priori, this is much more powerful than regular kernelization, which creates only a single output instance. Nevertheless, there are only few polynomial Turing kernelizations known for problems without (regular) polynomial kernelization (e.g., [3, 26, 25, 34]). Moreover, a hardness-based approach of Hermelin et al. [22] gives evidence that many problems are unlikely to admit polynomial Turing kernels.

More recently, Lokshtanov et al. [30] proposed a framework dedicated to the study of *lossy kernelization*. This relaxes the task of the kernelization by no longer requiring that an optimal solution to the output (x', k') yields an optimal solution for (x, k). Instead, for an α-approximate kernelization any c-approximate solution to (x', k') can be lifted to an $\alpha \cdot c$-approximate solution for (x, k). Amongst others, they show that Connected Vertex Cover and Disjoint Cycle Packing admit approximate kernelizations. In contrast, they were able to show, e.g., that k-Path has no α-approximate kernelization for any $\alpha \geq 1$ (unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$). Subsequent works have shown approximate kernelizations for other problems [13, 14, 32], in particular, further problems with connectivity constraints, which are often an obstruction for the existence of polynomial kernelizations.

Lokshtanov et al. [30] ask whether Independent Set parameterized by treewidth admits a polynomial-size approximate Turing kernelization with constant approximation ratio. In the present work, we answer this question affirmatively and in fact provide an efficient polynomial size approximate Turing kernelization scheme (EPSATKS). Moreover, extending the ideas for Independent Set, we provide similar results for a variety of other problems.

Our results. We prove that there is an EPSATKS for a wide variety of graph problems when parameterized by treewidth. The simplest problems we consider are the Vertex Cover and Independent Set problem. Observe that both problems parameterized by treewidth can be shown to be $\mathsf{MK}[2]$-hard, by a simple reduction from CNF-Sat with unbounded clause size.[1] As such, for both problems we indeed do not expect polynomial Turing kernels [22]. We show that Vertex Cover has a $(1+\varepsilon)$-approximate Turing kernel with $\mathcal{O}(\frac{\ell}{\varepsilon})$ vertices, and Independent Set has a kernel with $\mathcal{O}(\frac{\ell^2}{\varepsilon})$ vertices.

[1] A variant of the well-known NP-hardness proof of Independent Set (or Vertex Cover) suffices, where we add two vertices v_x and $v_{\bar{x}}$ for each variable x and connect them. Add a clique for each clause, that has a vertex u_ℓ for each literal ℓ in the clause. Connect u_ℓ to v_x if $\ell = \neg x$, connect u_ℓ to $v_{\bar{x}}$ if $\ell = x$. Observe that the treewidth is bounded by twice the number of variables.

Both approximate Turing kernels follow a similar strategy, based on using separators (originating from the tree decomposition) that separate a piece from the rest of the graph, such that the solution size in this piece is appropriately bounded. For this reason, we formulate a set of conditions on a graph problem and we call graph problems that satisfy these conditions *friendly*. We then show that all friendly graph optimization problems have polynomial-size $(1+\varepsilon)$-approximate Turing kernels for all $\varepsilon > 0$, when parameterized by treewidth. Precise bounds on the size of the obtained approximate Turing kernels depend on properties of the considered problem, such as the smallest-known (approximate) kernel when parameterized by solution size plus treewidth. In particular, applying the general result for Vertex Cover indeed shows that it has an EPSATKS of size $\mathcal{O}(\frac{\ell}{\varepsilon})$. Using this general technique, we obtain approximate Turing kernels for Clique Cover, Vertex-Disjoint H-Packing for connected graphs H, Feedback Vertex Set, and Edge Dominating Set.

Finally, we prove that Edge Clique Cover and Edge-Disjoint Triangle Packing have an EPSATKS and show that Connected Vertex Cover has a polynomial-size $(1+\varepsilon)$-approximate Turing kernel. These problems do not satisfy our definition of a friendly problem and require a more problem-specific approach. In particular, for Connected Vertex Cover we will need to consider subconnected tree decompositions [19] and carefully bound the size difference between locally optimal connected vertex covers, and intersections of (global) connected vertex covers with parts of the graph.

Overview. We start in Section 3 by illustrating the general technique using the Vertex Cover problem as an example. We continue by giving the approximate Turing kernels for Edge Clique Cover, Connected Vertex Cover, and Edge-Disjoint Triangle Packing. In Section 4 we state and prove our general theorem and then show that it allows us to give approximate Turing kernels for a number of different graph problems. For statements marked with a ★, the (full) proof can be found in the full version of the paper [24].

2 Preliminaries

We use $\mathbb{N}$ to denote the non-negative integers. Let $[n]$ be defined as the set containing the integers 1 to n. We assume that all graphs are simple and undirected, unless mentioned otherwise. A graph G has vertex set $V(G)$ and edge set $E(G)$. For $v \in V(G)$ we let $d_G(v)$ denote the degree of v. For $X \subseteq V(G)$, we use $G[X]$ to denote the graph induced by vertex set X, we use $G - X$ to denote $G[V(G) \setminus X]$. For $F \subseteq E(G)$, we use $G \setminus F$ to denote the graph resulting from deleting all edges in F from G.

We say that a set $X \subseteq V(G)$ *separates* vertex sets $A \subseteq V(G)$ and $B \subseteq V(G)$ if every path from some vertex in A to some vertex in B contains a vertex in X.

Treewidth. Recall the definition of treewidth.

▶ **Definition 1** ([10]). *A* tree decomposition *of a graph G is a tuple $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree in which each node $t \in V(T)$ has an assigned set of vertices $X_t \subseteq V(G)$, also referred to as the* bag *of node t, such that the following three conditions hold:*

- $\bigcup_{t \in V(T)} X_t = V(G)$, *and*
- *for every edge $\{u, v\} \in E(G)$ there exists $t \in V(T)$ such that $u, v \in X_t$, and*
- *for all $v \in V(G)$ the set $T_v := \{t \in V(T) \mid v \in X_t\}$ induces a connected subtree of T.*

The width *of a tree decomposition of G is the size of its largest bag minus one. The* treewidth *of G is the minimum width of any tree decomposition of G.*

In the remainder of the paper, we will always assume that a tree decomposition is given on input, as treewidth is NP-hard to compute. If it is not, we may use the result below to obtain an approximation of the treewidth and a corresponding tree decomposition in polynomial time. Doing so will weaken any given size bounds in the paper, as it is not a constant-factor approximation. The theorem below is part of [15, Theorem 6.4].

▶ **Theorem 2** ([15, Theorem 6.4]). *There exists a polynomial time algorithm that finds a tree decomposition of width at most $\mathcal{O}(\sqrt{\log tw(G)} \cdot tw(G))$ for a general graph G.*

Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a tree decomposition. Let $t \in V(T)$, we use V_t to denote the set of vertices from G that are contained in some bag of a node in the subtree of T that is rooted at t. It is well-known that for all $t \in V(T)$, the set X_t separates V_t from the rest of the graph. A rooted tree decomposition with root r is said to be *nice* if it satisfies the following properties (cf. [10]).

- **(i)** $X_r = \emptyset$ and $X_t = \emptyset$ for every leaf t of T.
- **(ii)** Every other node is of one of the following three types:
  - The node $t \in V(T)$ has exactly two children t_1 and t_2, and $X_t = X_{t_1} = X_{t_2}$. We call such a node a *join* node, or
  - the node $t \in V(T)$ has exactly one child t_1, and there exist $v \in V(G)$ such that $X_t = X_{t_1} \cup \{v\}$ (in this case t is an *introduce* node) or such that $X_{t_1} = X_t \cup \{v\}$ (in which case t is a *forget* node).

One can show that a tree decomposition of a graph G of width ℓ can be transformed in polynomial time into a nice tree decomposition of the same width and with $\mathcal{O}(\ell|V(G)|)$ nodes, see for example [10].

To deal with the Connected Vertex Cover problem we need the tree decomposition to preserve certain connectivity properties. Let a *subconnected tree decomposition* [19] be a tree decomposition where $G[V_t]$ is connected for all $t \in V(T)$. We observe the following.

▶ **Theorem 3** (cf. [19, Theorem 1]). *There is an $\mathcal{O}(n\ell^3)$-algorithm that, given a nice tree decomposition on n nodes of width ℓ of a connected graph G, returns an $\mathcal{O}(n \cdot \ell)$-node subconnected tree decomposition of G, of width at most ℓ such that each node in T has at most $2\ell + 2$ children.*

Proof. Without the additional bound on the degrees of nodes in T, the result is immediate from [19, Theorem 1]. We obtain a subconnected tree decomposition by only executing Phase 1 of Algorithm `make-it-connected` in [19]. Is is shown in [19, Claim 1] that this procedure results in a tree decomposition of width ℓ that is subconnected. It remains to analyze the maximum node degree. The only relevant step of the algorithm is the application of the `split` operation on nodes t from the original tree. Observe that every node in the original tree is visited at most once, and newly introduced nodes are never `split`. If t has parent s, the `split` operation only modifies the degree of s, and any newly introduced nodes. The newly introduced nodes will have degree at most $d_T(t)$. In particular, if s had degree a before the `split` operation on t, it will have degree $a - 1 + p$ after the `split` operation, where p is the number of connected components of $G[V_t]$. We will show that the number of connected components of $G[V_t]$ is bounded by $|X_t|$ if G is a connected graph. We do this by showing that each connected component contains at least one vertex from X_t. Suppose not. Let C be such a component. But since $C \cap X_t = \emptyset$, and X_t is a separator in G, it follows that there are no connections from C to $G[V(G) \setminus V_t]$. If $V_t = V(G)$, then $G[V_t]$ is connected and we are done, otherwise, vertices in $V(G) \setminus V_t$ are not connected to C in G, contradicting that G is connected. Thus, $p \leq |X_t| \leq \ell + 1$. Since in a nice tree decomposition every node has only two children, in the worst case `split` is applied to both these children. Thus, every node in T has degree at most $2\ell + 2$. ◀

Approximation, Kernelization, and Turing Kernelization. We recall the framework for approximate kernelization by Lokshtanov et al. [30] following Fomin et al. [17]. We then introduce suitable definitions for approximate Turing kernelization.

▶ **Definition 4** ([17]). *A* parameterized optimization problem $\mathcal{Q}$ *is a computable function*

$$\mathcal{Q}\colon \Sigma^* \times \mathbb{N} \times \Sigma^* \to \mathbb{R} \cup \{\pm\infty\}.$$

The instances *of a parameterized optimization problem are pairs* (I, k) *where* k *is the parameter. A* solution *to* (I, k) *is simply a string* $s \in \Sigma^*$*, such that* $|s| \leq |I| + k$*. The* value *of a solution* s *is given by* $\mathcal{Q}(I, k, s)$*. Using this, we may define the optimal value for the problem as*

$$\mathrm{OPT}_{\mathcal{Q}}(I, k) = \min\{\mathcal{Q}(I, k, s) \mid s \in \Sigma^*, |s| \leq |I| + k\},$$

for minimization problems and as

$$\mathrm{OPT}_{\mathcal{Q}}(I, k) = \max\{\mathcal{Q}(I, k, s) \mid s \in \Sigma^*, |s| \leq |I| + k\},$$

for maximization problems.

An *optimization problem* $\mathcal{P}\colon \Sigma^* \times \Sigma^* \to \mathbb{R} \cup \{\pm\infty\}$ is defined similarly, but without the parameter. In both cases we will say that s is a *solution* for instance I, if its value is not ∞ (or $-\infty$, in case of maximization problems).

We say that an algorithm for a (regular) minimization problem $\mathcal{P}$ is a *c-approximation algorithm* if for all inputs x it returns a solution s such that the value of s is at most $c \cdot \mathrm{OPT}_{\mathcal{P}}(x)$. Similarly, for a maximization problem we require that s has value at least $\frac{1}{c}\mathrm{OPT}_{\mathcal{P}}(x)$.

When a problem is parameterized by the value of the optimal solution, the definitions of parameterized optimization problems and lossy kernels will cause problems. As such, we use the following interpretation [30, p.229]. Given an optimization problem $\mathcal{P}$ that we want to parameterize by a sum of (potentially multiple) parameters, one of which is the solution value, we define the following corresponding parameterized optimization problem:

$$\mathcal{P}^{\perp}(I, k, s) := \min\{\mathcal{P}(I, s), k + 1\}.$$

In cases where we consider $\mathcal{P}$ parameterized by the treewidth of the input graph, we simply use

$$\mathcal{P}^{\perp}(I, k, s) := \mathcal{P}(I, s).$$

▶ **Definition 5** (α-Approximate kernelization [17]). *Let* $\alpha \geq 1$ *be a real number, let* g *be a computable function and let* $\mathcal{Q}$ *be a parameterized optimization problem. An* α-approximate kernelization $\mathcal{A}$ *of size* g *for* $\mathcal{Q}$ *is a pair of polynomial-time algorithms. The first one is called the* reduction algorithm *and computes a map* $\mathcal{R}_{\mathcal{A}}\colon \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$*. Given as input an instance* (I, k) *of* $\mathcal{Q}$*, the reduction algorithm computes another instance* $(I', k') = \mathcal{R}_{\mathcal{A}}(I, k)$ *such that* $|I'|, k' \leq g(k)$*.*

The second is called the solution-lifting algorithm*. This algorithm takes as input an instance* $(I, k) \in \Sigma^* \times \mathbb{N}$ *of* $\mathcal{Q}$*, together with* $(I', k') := \mathcal{R}_{\mathcal{A}}(I, k)$ *and a solution* s' *to* (I', k')*. In time polynomial in* $|I| + |I'| + k + k' + |s|$*, it outputs a solution* s *to* (I, k) *such that if* $\mathcal{Q}$ *is a minimization problem, then*

$$\frac{\mathcal{Q}(I, k, s)}{\mathrm{OPT}_{\mathcal{Q}}(I, k)} \leq \alpha \cdot \frac{\mathcal{Q}(I', k', s')}{\mathrm{OPT}_{\mathcal{Q}}(I', k')}.$$

For maximization problems we require

$$\frac{\mathcal{Q}(I, k, s)}{\mathrm{OPT}_{\mathcal{Q}}(I, k)} \geq \frac{1}{\alpha} \cdot \frac{\mathcal{Q}(I', k', s')}{\mathrm{OPT}_{\mathcal{Q}}(I', k')}.$$

We say that a problem admits a *Polynomial Size Approximate Kernelization Scheme (PSAKS)* [30] if it admits an α-approximate polynomial kernel for all $\alpha > 1$.

We recall the definition of a Turing kernel, so that we can show how to naturally generalize the notion of approximate kernelization to Turing kernels.

▶ **Definition 6** (Turing kernelization [17]). *Let $\mathcal{Q}$ be a parameterized problem and let $f : \mathbb{N} \to \mathbb{N}$ be a computable function. A* Turing kernelization *for $\mathcal{Q}$ of size f is an algorithm that decides whether a given instance $(x, k) \in \Sigma^* \times \mathbb{N}$ belongs to $\mathcal{Q}$ in time polynomial in $|x|+k$, when given access to an oracle that decides membership of $\mathcal{Q}$ for any instance (x', k') with $|x'|, k' \leq f(k)$ in a single step.*

In the following definition, we combine the notions of lossy kernelization and Turing kernelization into one, as follows.

▶ **Definition 7** (Approximate Turing kernelization). *Let $\alpha \geq 1$ be a real number, let f be a computable function and let $\mathcal{Q}$ be a parameterized optimization problem. An* α-approximate Turing kernel of size f *for $\mathcal{Q}$ is an algorithm that, when given access to an oracle that computes a c-approximate solution for instances of $\mathcal{Q}$ in a single step, satisfies the following.*

- *It runs in time polynomial in $|I| + k$, and*
- *given instance (I, k), outputs a solution s such that $\mathcal{Q}(I, k, s) \leq \alpha \cdot c \cdot \mathrm{OPT}_{\mathcal{Q}}(I, k)$ if $\mathcal{Q}$ is a minimization problem and $\mathcal{Q}(I, k, s) \cdot \alpha \cdot c \geq \mathrm{OPT}_{\mathcal{Q}}(I, k)$ is $\mathcal{Q}$ is a minimization problem, and*
- *it only uses oracle-queries of size bounded by $f(k)$.*

Note that, in the definition above, the algorithm does not depend on c, just like in lossy kernelization. We say that a parameterized optimization problem $\mathcal{Q}$ has an *EPSATKS* when it has a polynomial-size $(1 + \varepsilon)$-approximate Turing kernel for every $\varepsilon > 0$, of size $f(\varepsilon) \cdot \mathrm{poly}(k)$ where f is a function that depends only on ε.

3 Approximate Turing kernels for specific problems

In this section we will give approximate Turing kernels for a number of graph problems parameterized by treewidth. We start by discussing the Vertex Cover problem, since the approximate Turing kernels for all other problems will follow the same overall structure.

3.1 Vertex Cover

In this section we discuss an approximate Turing kernel for Vertex Cover parameterized by treewidth ℓ. The overall idea will be to use the treewidth decomposition of the graph, and find a subtree rooted at a node t such that $G[V_t \setminus X_t]$ has a large (but not too large) vertex cover. A vertex cover of the entire graph will then be obtained by taking a vertex cover of $G[V_t \setminus X_t]$, adding all vertices in X_t, and recursing on the graph that remains after removing V_t. This produces a correct vertex cover because X_t is a separator in the graph. Furthermore, taking all of X_t into the vertex cover is not problematic as X_t is ensured to be comparatively small. To obtain a vertex cover of $G[V_t \setminus X_t]$, we will use the following lemma.

▶ **Lemma 8** (★). *Let G be a graph with $OPT_{VC}(G) \leq k$. Then there is a polynomial-time algorithm returning vertex cover of G of size at most $c \cdot \mathrm{OPT}_{VC}(G)$, when given access to c-approximate oracle that solves vertex cover on graphs with at most $\mathcal{O}(k)$ vertices.*

Proof sketch. First apply the LP-based kernel [9] for Vertex Cover parameterized by solution size to (G, k). This gives an instance G' with at most $2k$ vertices. We can then apply the oracle to obtain a c-approximate vertex cover of G'. We show in the complete proof that it is straightforward to lift this solution to a c-approximate vertex cover of the original graph G, concluding the proof. ◀

Using this, we can now give the $(1+\varepsilon)$-approximate Turing kernel for Vertex Cover. While the theorem statement requires $\varepsilon \leq 1$, this does not really impose a restriction: if $\varepsilon > 1$ one may simply reset it to be 1. It simply shows that the bounds do not continue improving indefinitely as ε grows larger than 1. Note however that Vertex Cover is 2-approximable in polynomial time, such that choosing ε larger than one is likely not useful.

▶ **Theorem 9.** *For every $0 < \varepsilon \leq 1$, Vertex Cover parameterized by treewidth ℓ has a $(1+\varepsilon)$-approximate Turing kernel with $\mathcal{O}(\frac{\ell}{\varepsilon})$ vertices.*

Proof. Consider Algorithm 1, we use the well-known 2-approximation algorithm for Vertex Cover. First of all, we show how to do Step 8 of the algorithm efficiently.

Algorithm 1 An approximate Turing kernel for Vertex Cover.

```
1: procedure ApproxVC(G, T, ε)
2:     Turn T into a nice tree decomposition of G
3:     Obtain a 2-approximate solution S̃ for VC in G
4:     if |S̃| ≤ 8(ℓ+1)/ε then
5:         Determine a c-approximate solution S to VC in G using Lemma 8
6:         return S
7:     else
8:         Find t ∈ V(T) s.t. (ℓ+1)/ε ≤ OPT_VC(G[V_t \ X_t]) ≤ 8(ℓ+1)/ε
9:         Determine a c-approximate solution S_t to VC in G[V_t \ X_t] using Lemma 8
10:        G' ← G − V_t
11:        Let T' be T after removing the subtree rooted at t and all vertices in X_t
12:        S' ← ApproxVC(G', T', ε)
13:        return S_t ∪ X_t ∪ S'
14:    end if
15: end procedure
```

▷ Claim. There is a polynomial-time algorithm that, given graph G such that $\mathrm{OPT}_{\mathrm{VC}}(G) \geq \frac{(\ell+1)}{\varepsilon}$, with a nice tree decomposition $\mathcal{T}$ of width at most ℓ, outputs a node $t \in V(T)$ such that $\frac{(\ell+1)}{\varepsilon} \leq \mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t]) \leq \frac{8(\ell+1)}{\varepsilon}$.

Proof. Let $\mathcal{T}$ be a nice tree decomposition with root r. We start from $t := r$, maintaining that $\mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t]) \geq \frac{\ell+1}{\varepsilon}$. Note that this is initially true since $G_r = G$.

Check whether the 2-approximation returns a vertex cover of size at most $\frac{8(\ell+1)}{\varepsilon}$ for $G[V_t \setminus X_t]$. If yes, we are done. If not, then $\mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t]) > \frac{4(\ell+1)}{\varepsilon}$. We show that t has a child on which we will recurse. We do a case distinction on the type of node of t.

- t is a leaf node. In this case, $|V_t \setminus X_t| = 0$, contradicting that $\mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t]) > \frac{4(\ell+1)}{\varepsilon} \geq 0$.
- t is a forget or introduce node. This implies t has one child t_1 and the size of $V_t \setminus X_t$ and $V_{t_1} \setminus X_{t_1}$ differs by at most one. Therefore, $\mathrm{OPT}_{\mathrm{VC}}(G[V_{t_1} \setminus X_{t_1}]) \geq \mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t]) - 1 \geq \mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t])/2$.

- t is a join node with children t_1 and t_2. Observe that $G[V_t \setminus X_t]$ is the disjoint union of the graphs $G[V_{t_1} \setminus X_{t_1}]$ and $G[V_{t_2} \setminus X_{t_2}]$ (note $X_t = X_{t_1} = X_{t_2}$). As such, for one of the two children, without loss of generality let this be t_1, running the 2-approximation algorithm for vertex cover returns a value of at least $\mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t])/2$, meaning that $\mathrm{OPT}_{\mathrm{VC}}(G[V_{t_1} \setminus X_{t_1}]) \geq \mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t])/4$.

Thus, there is a child t_1 such that $\mathrm{OPT}_{\mathrm{VC}}(G[V_{t_1} \setminus X_{t_1}]) \geq \mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t])/4 \geq \frac{\ell+1}{\varepsilon}$. Continue with $t := t_1$. ◁

We will now show the correctness of the algorithm by induction on $|V(G)|$. Let G be a graph with nice tree decomposition $\mathcal{T}$. If the algorithm returns a Vertex Cover in Step 5, the result is immediate. If not, then it follows that the algorithm returns in Step 13, and that $\mathrm{OPT}_{\mathrm{VC}}(G) > \frac{4(\ell+1)}{\varepsilon}$. The algorithm then returns a vertex cover S_t for $G[V_t \setminus X_t]$ together with X_t and a vertex cover $S' = \textsc{ApproxVC}(G', \mathcal{T}', \varepsilon)$ in the remainder of the graph. It is easy to see that the returned set is indeed a vertex cover of the graph. Furthermore, one may verify that the oracle is only used for graphs with at most $\mathcal{O}(\frac{\ell}{\varepsilon})$ vertices. It remains to verify the approximation ratio. Recall that $G' := G - V_t$. Then

$$\begin{aligned}|S_t| + |S'| + |X_t| &\leq c \cdot \mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t]) + c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{VC}}(G') + \ell + 1 \\ &\leq c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{VC}}(G[V_t \setminus X_t]) + c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{VC}}(G') \\ &\leq c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{VC}}(G).\end{aligned}$$ ◀

3.2 Edge Clique Cover

In this section, we obtain an approximate Turing kernel for Edge Clique Cover, which is defined as follows.

Edge Clique Cover (ECC) **Parameter:** ℓ
Input: A graph G with tree decomposition $\mathcal{T}$ of width ℓ.
Output: The minimum value for $k \in \mathbb{N}$ such that there exists a family S of subsets of $V(G)$ such that $\lvert S\rvert \leq k$, $G[C]$ is a clique for all $C \in S$, and for all $\{u, v\} \in E(G)$ there exists $C \in S$ such that $u, v \in S$?

To obtain an approximate Turing kernel, we will separate suitably-sized subtrees from the graph using the tree decomposition, as we did in the approximate Turing kernel for Vertex Cover. To show that this results in the desired approximation bound, we will need the following lemma. It basically shows that if we find a node t of the tree decomposition such that X_t is "small" compared to $\mathrm{OPT}(V_t)$, we will be able to combine an edge clique cover in $G[V_t]$ with one in $G - (V_t \setminus X_t)$ to obtain a clique cover of the entire graph that is not too far from optimal.

▶ **Lemma 10.** *Let G be a graph, let $X_1, X_2 \subseteq V(G)$ such that $X_1 \cup X_2 = V(G)$ and $X = X_1 \cap X_2$ separates X_1 from X_2 in G. Then*

$$OPT_{ECC}(G) \geq OPT_{ECC}(G[X_1]) + OPT_{ECC}(G[X_2]) - \binom{|X|}{2}.$$

Proof. Let S be an edge clique cover of G. We show how to obtain clique covers S_1 and S_2 for $G[X_1]$ and $G[X_2]$ such that $|S_1| + |S_2| \leq |S| + \binom{|X|}{2}$. First define

$$S_1' := \{C \mid C \cap (X_1 \setminus X) \neq \emptyset, C \in S\} \cup \{C \mid C \subseteq X, C \in S\},$$

similarly, define

$$S_2' := \{C \mid C \cap (X_2 \setminus X) \neq \emptyset, C \in S\}.$$

For $j \in [2]$, define $S_j := S_j' \cup S_j''$, where $S_j'' := \{\{u,v\} \in E(G[X]) \mid \{u,v\} \text{ not covered by } S_j'\}$.

We start by showing that S_j is an edge clique cover of $G[X_j]$ for $i \in [2]$. First of all, we will verify that $C \subseteq X_j$ and that C forms a clique in $G[X_j]$ for all $C \in S_j$. For $C \in S_j''$ this is trivial, for $C \in S_j'$, observe that C is a clique in G and any clique in G containing a vertex from $X_j \setminus X$ cannot contain a vertex from $V(G) \setminus X_j$, since X is a separator. Thus $C \subseteq X_j$. The fact that C is a clique in $G[X_j]$ is immediate from C being a clique in G.

It remains to show that S_j covers all edges in $G[X_j]$. Let $\{u,v\} \in E(G[X_j])$. If $u,v \in X$, then the edge is covered by definition. Without loss of generality, suppose $u \in X_j \setminus X$. Let $C \in S$ be a clique that covered edge $\{u,v\}$. Then clearly $u \in C \cap (X_j \setminus X)$ and thus $C \cap (X_j \setminus X) \neq \emptyset$, implying $C \in S_j$. Thus, the edge $\{u,v\}$ is indeed covered by S_j.

It remains to show that $|S_1| + |S_2| \leq |S| + \binom{|X|}{2}$. Start by observing that $|S_1'| + |S_2'| \leq |S|$, since a clique cannot contain both a vertex from $X_1 \setminus X$ and $X_2 \setminus X$. Since every edge $\{u,v\} \in E(G[X])$ is covered by S, it is easy to observe from the definition that $\{u,v\}$ is covered by S_1' or S_2'. As such, $S_1'' \cap S_2'' = \emptyset$. Since $G[X]$ has at most $\binom{|X|}{2}$ edges, it follows that $|S_1''| + |S_2''| \leq \binom{|X|}{2}$ and indeed $|S_1| + |S_2| \leq |S_1'| + |S_1''| + |S_2'| + |S_2''| \leq |S| + \binom{|X|}{2}$. ◀

Before giving the approximate Turing kernel, we show that there exists a node t in the tree decomposition such that the size of the subtree rooted at t falls within certain size bounds. We use this to split off subtrees, similar to the strategy we used for Vertex Cover earlier.

▶ **Lemma 11 (★).** *There is a polynomial-time algorithm that, given a graph G with $|V(G)| \geq 2\frac{1+\varepsilon}{\varepsilon}(\ell+1)^4$, a nice tree decomposition $\mathcal{T}$ of width ℓ, and $\varepsilon > 0$, outputs a node $t \in V(T)$ such that $2\frac{1+\varepsilon}{\varepsilon}(\ell+1)^4 \leq |V_t \setminus X_t| \leq 4\frac{1+\varepsilon}{\varepsilon}(\ell+1)^4$.*

Using the lemma above, we can now give the approximate Turing kernel for Edge Clique Cover.

▶ **Theorem 12.** *For every $0 < \varepsilon \leq 1$, Edge Clique Cover parameterized by treewidth ℓ has a $(1+\varepsilon)$-approximate Turing kernel with $\mathcal{O}(\frac{\ell^4}{\varepsilon})$ vertices.*

Proof. Consider Algorithm 2, we show that it is a $(1+\varepsilon)$-approximate Turing kernel for ECC. Observe that Step 2 can be done efficiently while maintaining a valid tree decomposition, as one may simply restrict the bags of the decomposition to the relevant connected component of G. It is easy to verify that the procedure runs in polynomial time, using that $|V_t \setminus X_t|$ is always non-empty and thus the recursive call is on a strictly smaller graph. Finally, we can verify the size-bound, as the oracle is only applied to G if $|V(G)| \leq \mathcal{O}(\frac{\ell^4}{\varepsilon})$ or to $G[V_t]$ when $|V_t \setminus X_t| \leq \mathcal{O}(\frac{\ell^4}{\varepsilon})$, implying that $|V_t| \leq |V_t \setminus X_t| + \ell + 1 = \mathcal{O}(\frac{\ell^4}{\varepsilon})$.

We continue by showing that Algorithm 2 returns an edge clique cover of G. If the algorithm returns in Step 6, this is immediate. Otherwise, observe that since X_t separates V_t and $V(G')$ in G, it follows that any edge in G is in $E(G[V_t])$ or in $E(G')$. Thus, such an edge is covered by S_t or S', implying that $S = S_t \cup S'$ is an edge clique cover of G. We now bound $|S_t| + |S'|$, to show that the algorithm indeed approximates the optimum ECC.

$$\begin{aligned} |S_t| + |S'| &\leq c \cdot \mathrm{OPT}_{\mathrm{ECC}}(G[V_t]) + |S'| \\ &= c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{ECC}}(G[V_t]) - c \cdot \varepsilon \cdot \mathrm{OPT}_{\mathrm{ECC}}(G[V_t]) + |S'| \end{aligned}$$

Algorithm 2 An approximate Turing Kernel for Edge Clique Cover.

procedure ApproxECC($G, \mathcal{T}, \varepsilon$)
  If G is not connected, split G into its connected components and treat them separately.
  Turn $\mathcal{T}$ into a nice tree decomposition.
  if $|V(G)| \leq \frac{2(1+\varepsilon)}{\varepsilon}(\ell+1)^4$ **then**
    Apply the c-approximate oracle to obtain an ECC S of G
    return S
  else
    Find $t \in V(T)$ s.t. $2\frac{(1+\varepsilon)}{\varepsilon}(\ell+1)^4 \leq |V_t \setminus X_t| \leq \frac{4(1+\varepsilon)}{\varepsilon}(\ell+1)^4$ (by Lemma 11)
    Determine a c-approximate solution S_t to ECC in $G[V_t]$ using the oracle
    $G' \leftarrow G - (V_t \setminus X_t)$
    Let $\mathcal{T}'$ be $\mathcal{T}$ after removing the subtree rooted at t except for t
    $S' \leftarrow$ ApproxECC($G', \mathcal{T}', \varepsilon$)
    return $S_t \cup S'$
  end if
end procedure

Observe that every clique covers at most $\binom{\ell+1}{2}$ edges, since it has at most $\ell+1$ vertices, since the treewidth of G is bounded by ℓ. Thus $\mathrm{OPT}_{\mathrm{ECC}}(G[V_t]) \geq |E(G[V_t])|/\binom{\ell+1}{2}$.

$$\leq c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{ECC}}(G[V_t]) - c \cdot \varepsilon \cdot |E(G[V_t])|/\binom{\ell+1}{2} + |S'|$$

Observe that $V_t \setminus X_t$ cannot contain vertices that are isolated in $G[V_t]$, since G is connected and X_t separates V_t from the remainder of G. Thus, $|E(G[V_t])| \geq |V_t \setminus X_t|/2$.

$$\leq c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{ECC}}(G[V_t]) - c \cdot \varepsilon \cdot \frac{|V_t \setminus X_t|}{2(\ell+1)^2} + |S'|$$
$$\leq c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{ECC}}(G[V_t]) - c \cdot (1+\varepsilon) \cdot (\ell+1)^2 + |S'|$$

using $\ell + 1 \geq |X_t|$

$$\leq c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{ECC}}(G[V_t]) - c \cdot (1+\varepsilon) \cdot \binom{|X_t|}{2} + |S'|$$
$$\leq c \cdot (1+\varepsilon) \cdot (\mathrm{OPT}_{\mathrm{ECC}}(G[V_t]) + \mathrm{OPT}_{\mathrm{ECC}}(G') - \binom{|X_t|}{2})$$

By Lemma 10

$$\leq c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{ECC}}(G). \quad \blacktriangleleft$$

3.3 Edge-Disjoint Triangle Packing

In this section we give an approximate Turing kernel for the Edge-Disjoint Triangle Packing problem, defined as follows.

Edge-Disjoint Triangle Packing (ETP) **Parameter:** ℓ

Input: A graph G with tree decomposition $\mathcal{T}$ of width ℓ.

Output: The maximum value for $k \in \mathbb{N}$ such that there exists a family S of size-3 subsets of $V(G)$ such that $|S| \geq k$, $G[X]$ is a triangle for all $X \in S$, and X and Y are edge-disjoint for all $X, Y \in S$?

Observe that the problem has a 3-approximation by taking any maximal edge-disjoint triangle packing S, which can be greedily constructed. This packing then uses $3|S|$ edges. If there is a solution S' with $|S'| > 3|S|$, then there is a triangle in S' that contains no edge covered by S, contradicting that S is maximal. We now give the approximate Turing kernel.

▶ **Theorem 13.** *For every $0 \leq \varepsilon \leq 1$, Edge-Disjoint Triangle Packing parameterized by treewidth ℓ, has a $(1+\varepsilon)$-approximate Turing kernel with $\mathcal{O}(\frac{\ell^2}{\varepsilon})$ vertices.*

Proof. We will use the following claim.

▷ Claim 14 (★). Let G be a graph with $\mathrm{OPT}_{\mathrm{ETP}}(G) \leq k$. There is a polynomial-time algorithm that when given access to a c-approximate oracle, outputs a c-approximate solution for G using calls to the oracle with at most $\mathcal{O}(k)$ vertices.

We now describe the algorithm. Start by computing a 3-approximate solution $\tilde{S}$ to Edge-Disjoint Triangle Packing in G. If $|\tilde{S}| \leq 18\frac{(\ell+1)^2}{\varepsilon}$, we obtain an approximate solution to triangle packing using Claim 14.

Otherwise, for $t \in V(T)$ define G_t as $G[V_t] \setminus E(G[X_t])$, i.e., the graph $G[V_t]$ from which the edges between vertices in X_t have been removed. We show how to find $t \in T$ such that

$$\frac{(\ell+1)^2}{\varepsilon} \leq \mathrm{OPT}_{\mathrm{ETP}}(G_t) \leq 18\frac{(\ell+1)^2}{\varepsilon},$$

together with an approximate solution S_t in G_t. Start with $t := r$, observe that initially $\mathrm{OPT}_{\mathrm{ETP}}(G_t) > \frac{18(\ell+1)^2}{\varepsilon}$ since $G_r = G$ and $\mathrm{OPT}_{\mathrm{ETP}}(G_t) \geq |\tilde{S}|$. So suppose we are at some node t with $\mathrm{OPT}_{\mathrm{ETP}}(G_t) \geq \frac{(\ell+1)^2}{\varepsilon}$. Compute a 3-approximate solution in G_t. If this solution has value at most $\frac{6(\ell+1)^2}{\varepsilon}$, we obtain an approximate solution S_t to triangle packing in G_t using Claim 14. Otherwise, we will recurse on a child t_1 of t for which $\mathrm{OPT}_{\mathrm{ETP}}(G_{t_1}) \geq \frac{(\ell+1)^2}{\varepsilon}$, we show how to find such a child by doing a case distinction on the type of node of t.

- t is a leaf node. This is a contradiction with the assumption that $\mathrm{OPT}_{\mathrm{ETP}}(G_t) > 6\frac{(\ell+1)^2}{\varepsilon}$, since G_t is empty.
- t has exactly one child t_1 and $X_t = X_{t_1} \cup \{v\}$ for some $v \in V(G)$. This means in particular that $G_{t_1} = G_t - \{v\}$. Furthermore, we can show that v is isolated in G_t. After all, there are no edges between vertices in X_t and v by definition of G_t. Furthermore, there are no edges between v and vertices not in X_t, by correctness of the tree decomposition. Therefore, trivially, $\mathrm{OPT}_{\mathrm{ETP}}(G_t) = \mathrm{OPT}_{\mathrm{ETP}}(G_{t_1})$ and we continue with $t \leftarrow t_1$.
- t has exactly one child t_1 and $X_t = X_{t_1} \setminus \{v\}$ for some $v \in V(G)$. In this case G_{t_1} can be obtained by G_t by removing all edges between vertices in v and vertices in X_t. This removes at most $(\ell+1)$ edges from the graph, and thus $\mathrm{OPT}_{\mathrm{ETP}}(G_{t_1}) \geq \mathrm{OPT}_{\mathrm{ETP}}(G_t) - \ell \geq \frac{(\ell+1)^2}{\varepsilon}$, and we continue with $t \leftarrow t_1$.
- t is a join node with children t_1 and t_2. Observe that X_t separates G_t and that $\mathrm{OPT}_{\mathrm{ETP}}(G_t) = \mathrm{OPT}_{\mathrm{ETP}}(G_{t_1}) + \mathrm{OPT}_{\mathrm{ETP}}(G_{t_2})$. As such, there is a child of G_t, w.l.o.g. let this be t_1, such that $\mathrm{OPT}_{\mathrm{ETP}}(G_{t_1}) \geq \mathrm{OPT}_{\mathrm{ETP}}(G_t)/2 \geq \frac{3(\ell+1)^2}{\varepsilon}$. Using the 3-approximation on both children, find a child where the returned solution size is at least $\frac{3(\ell+1)^2}{3\varepsilon} = \frac{(\ell+1)^2}{\varepsilon}$. Continue with this child.

Using t and the obtained solution S_t in G_t, we now do the following. Let $G' := G - (V_t \setminus X_t)$. Obtain a solution S' in G' using the algorithm above on the smaller graph G'. Output $S := S_t \cup S'$. Since G' and G_t are edge-disjoint subgraphs of G, it is easy to observe that S is an edge-disjoint triangle packing in G.

It remains to show that S has the desired size. Observe that the size of an edge-disjoint triangle packing in G can be bounded by considering the triangles whose edges are in G_t, those whose edges are in G', and those with at least one edge with both endpoints in X_t. Using that there are at most $\binom{X_t}{2}$ edges between vertices in X_t, we get

$$\begin{aligned}\mathrm{OPT}_{\mathrm{ETP}}(G) &\leq \mathrm{OPT}_{\mathrm{ETP}}(G_t) + \mathrm{OPT}_{\mathrm{ETP}}(G') + \binom{X_t}{2} \\ &\leq (1+\varepsilon)\mathrm{OPT}_{\mathrm{ETP}}(G_t) + \mathrm{OPT}_{\mathrm{ETP}}(G') \\ &\leq c \cdot (1+\varepsilon)|S_t| + c \cdot (1+\varepsilon)|S'| \\ &\leq c \cdot (1+\varepsilon)|S|.\end{aligned}$$

◀

The strategy used to obtain a kernel for Edge-Disjoint Triangle Packing can be generalized to packing larger cliques, as long as these problems have polynomial kernels when parameterized by solution size. Generalizing to the more general question of packing edge-disjoint copies of some other graph H may be more difficult. In this case, there can be copies of H that have vertices in both sides of the graph after removing the edges within a separator, and one needs to be careful to not discard too many of these.

3.4 Connected Vertex Cover

The Connected Vertex Cover (CVC) problem asks, given a graph G and tree decomposition $\mathcal{T}$, for the minimum size of a vertex cover S in G such that $G[S]$ is connected. It is known that CVC has a $(1+\varepsilon)$-approximate kernel of polynomial size [30].

▶ **Theorem 15** ([30]). *Connected Vertex Cover parameterized by solution size k admits a strict time efficient PSAKS with $\mathcal{O}(k^{\lceil \frac{\alpha}{\alpha-1} \rceil} + k^2)$ vertices.*

To obtain an approximate Turing kernel, we will use a similar strategy to the Turing kernel for Vertex Cover described in Theorem 9. However, the connectivity constraint makes this kernel somewhat more complicated. We deal with this by changing the procedure in two places. First of all, we will use a subconnected tree decomposition, to ensure that $G[V_t]$ is connected for any node t. We will then again find a subtree with a suitably-sized solution. In this case however, we will contract the separator between the subtree and the rest of the graph to a single vertex. The next lemma shows that this does not reduce the connected vertex cover size in the subtree by more than twice the size of the separator.

▶ **Lemma 16.** *Let G be a connected graph and let $X \subseteq V(G)$. Given a connected vertex cover S of G_X where G_X is obtained from X by identifying all vertices from X into a single vertex z, there is a polynomial-time algorithm that finds a connected vertex cover S' of size at most $|S| + 2|X|$ of G.*

Proof. Let S be a connected vertex cover of G_X. Let $S'' := S \cup X \setminus \{z\}$. Observe that S'' is a vertex cover of G, such that every connected component of $G[S'']$ contains at least one vertex from X; thus, there are at most $|X|$ connected components. If $G[S'']$ is connected, we are done. Otherwise, we show that there is a single vertex $v \in V(G)$ such that $G[S'' \cup \{v\}]$ has strictly fewer connected components than $G[S'']$. It is then straightforward to obtain S' by repeatedly adding such a vertex, until $G[S'']$ is connected. For any vertex $u \in S''$ define C_u as the connected component of vertex u in $G[S'']$.

Let x and x' be in two distinct components in $G[S'']$, consider the shortest path P from x to x' in G. Refer to Figure 1 for a sketch of the situation. By this definition, $C_x \neq C_{x'}$. Let y be the first vertex in P such that $y \in S''$ but $y \notin C_x$, let y' be the vertex on P before

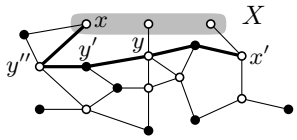


Figure 1 A graph with a vertex cover S'' (indicated in white) that is connected when all vertices in X are identified into a single vertex. Shown are x, x',y,y',y'', and P (indicated in bold) as used in the proof of Lemma 16.

y, observe that $y' \notin S''$ since otherwise $y' \in S''$ and $y' \notin C_x$ which is a contradiction with the fact that y is the first such vertex in P. Let y'' be the vertex on the path before y', such that $P = (x, \ldots, y'', y', y, \ldots, x')$, where possibly $x = y''$ or $y = x'$. Observe that $y'' \in S''$ as otherwise edge $\{y'', y'\}$ is not covered, and therefore $y'' \in C_x$ since y is the first vertex on P that is in S'' but not in C_x. Therefore, adding vertex y' to S'' will merge connected components C_x and C_y, such that the number of connected components in $G[S'' \cup \{y'\}]$ is strictly smaller than the number of connected components in $G[S'']$. In total, we add less than $|X|$ vertices to S'' obtain a connected vertex cover S' and thus $|S'| \leq |S| + |X|$. ◀

We now prove the main result of this section.

▶ **Theorem 17.** *For every $0 < \varepsilon \leq 1$, Connected Vertex Cover parameterized by treewidth ℓ has a $(1+\varepsilon)$-approximate Turing Kernel with $\mathcal{O}((\frac{\ell^2}{\varepsilon})^{\lceil \frac{3+\varepsilon}{\varepsilon} \rceil})$ vertices.*

Proof. We will use the PSAKS for Connected Vertex Cover from Theorem 15. Recall that such a PSAKS consists of a reduction algorithm $R_{\mathcal{A}}$ together with a solution lifting algorithm $S_{\mathcal{A}}$. We will use the following claim.

▷ Claim 18. Given $0 < \delta \leq 1$ and a connected graph G with tree decomposition of width ℓ, there is a polynomial-time algorithm to determine a d-approximate solution for CVC or correctly decide that $\mathrm{OPT}_{\mathrm{CVC}}(G) > \frac{100\ell^2}{\delta}$, when given access to a c-approximate CVC-oracle that allows calls using graphs with at most $\mathcal{O}((\frac{\ell^2}{\delta})^{\lceil \frac{1+\delta}{\delta} \rceil})$ vertices, where $d = \min(c \cdot (1+\delta), 2)$.

Proof. Using the fact that CVC is 2-approximable in polynomial time [33], obtain a 2-approximate solution $\tilde{S}$ in G. If $|\tilde{S}| > 200\ell^2/\delta$, return NO and halt. Otherwise, continue by running $R_{\mathcal{A}}$ on $(G, |\tilde{S}|)$ to obtain (G', k'). Observe that G' has at most $\mathcal{O}((\frac{\ell^2}{\delta})^{\lceil \frac{1+\delta}{\delta} \rceil})$ many vertices. Apply the c-approximate oracle on G' to obtain CVC S' in G'. Obtain an approximate solution S in G by using the solution lifting algorithm on G' and S'. Output the smallest solution of S and $\tilde{S}$, let this be $\hat{S}$. We show that this has the desired approximation factor, which requires an argument since the PSAKS works for $\mathrm{CVC}^{\perp}$ instead of CVC (recall $\mathrm{CVC}^{\perp}(G, k, S) = \min\{k+1, \mathrm{CVC}(G, S)\}$). Observe that $|\hat{S}| \leq |\tilde{S}|$, by definition. Therefore, $|\hat{S}| \leq \mathrm{CVC}^{\perp}(G, |\tilde{S}|, S)$. Thus

$$\frac{|\hat{S}|}{\mathrm{OPT}_{\mathrm{CVC}}(G)} \leq \frac{\mathrm{CVC}^{\perp}(G, |\tilde{S}|, S)}{\mathrm{OPT}_{\mathrm{CVC}}(G)} \leq \frac{\mathrm{CVC}^{\perp}(G, |\tilde{S}|, S)}{\mathrm{OPT}_{\mathrm{CVC}^{\perp}}(G, |\tilde{S}|)}.$$

By correctness of the solution lifting algorithm, we get

$$\frac{\mathrm{CVC}^{\perp}(G, |\tilde{S}|, S)}{\mathrm{OPT}_{\mathrm{CVC}^{\perp}}(G, |\tilde{S}|)} \leq (1+\delta)\frac{\mathrm{CVC}^{\perp}(G', k', S')}{\mathrm{OPT}_{\mathrm{CVC}^{\perp}}(G', k')} \leq (1+\delta)\frac{|S'|}{\mathrm{OPT}_{\mathrm{CVC}}(G')} \leq c \cdot (1+\delta),$$

by correctness of the oracle. ◁

Algorithm. The algorithm now proceeds as follows. Our goal is to find a subtree of T for which on the one hand, the local optimum CVC is small enough to find an approximate solution using Claim 18, but also large enough to be able to (among other things) add the entire set X_t to the solution, without introducing a too large error. Let $\delta := \varepsilon/3$.

For any vertex $t \in V(T)$, let G_t be the graph given by $G[V_t]$ after identifying all vertices from X_t into a single vertex z_t. Apply Claim 18 to G, if it returns an approximate connected vertex cover of G, we are done. Otherwise, $\mathrm{OPT}_{\mathrm{CVC}}(G) > \frac{100\ell^2}{\delta}$. We now aim to find a vertex t such that Claim 18 returns an approximate solution in G_t of size at least $\frac{10\ell}{\delta}$.

▷ Claim 19. There is a polynomial-time algorithm that, given G with tree decomposition $\mathcal{T}$ of width ℓ such that $\mathrm{OPT}_{\mathrm{CVC}}(G) > \frac{100\ell^2}{\delta}$, finds $t \in V(T)$ for which Claim 18 returns an approximate solution S_t with $|S_t| \geq \frac{10\ell}{\delta}$, using calls to a c-approximate oracle of size at most $\mathcal{O}((\frac{\ell^2}{\delta})^{\lceil \frac{1+\delta}{\delta} \rceil})$.

Proof. Start with $t := r$, note that since $\mathrm{OPT}_{\mathrm{CVC}}(G) > \frac{100\ell^2}{\delta}$ and $G_r = G$, we have that $\mathrm{OPT}_{\mathrm{CVC}}(G_r) > \frac{100\ell^2}{\delta}$, where r is the root of T. We search through the graph maintaining $\mathrm{OPT}_{\mathrm{CVC}}(G_t) > \frac{100\ell^2}{\delta}$. Let $t_1, \ldots, t_m$ be the children of t, recall that we may assume $m \leq 2\ell + 2$ by Theorem 3. For each t_i, apply Claim 18. Consider the following possibilities.

- There exists $i \in [m]$ such that the claim determines $\mathrm{OPT}_{\mathrm{CVC}}(G_{t_i}) > \frac{100\ell^2}{\delta}$, in this case, recurse with this t_i.
- There exists $i \in [m]$ such that the claim returns a $\min\{2, (1+\delta)\cdot c\}$-approximate solution S_{t_i} of size at least $\frac{10\ell}{\delta}$ for CVC. In this case, return $t := t_i$.
- Otherwise. Thus, for every $i \in [m]$, the algorithm returns a connected vertex cover S_i of size at most $\frac{10\ell}{\delta}$ for CVC in G_{t_i}. Obtain a connected vertex cover S'_i of $G[V_{t_i}]$ of size at most $|S_i| + 2(\ell+1)$ using Lemma 16. We will argue that in this case $CVC(G_t) < \frac{55\ell^2}{\delta}$, which is a contradiction. We obtain a connected vertex cover of G_t as follows. Let $\hat{S}_t := \bigcup_{i\in[m]}(S'_i) \cup \{z_t\}$. Observe that $\hat{S}_t$ has size at most $(2\ell+2)\cdot\frac{13\ell}{\delta} + 1 \leq \frac{55\ell^2}{\delta}$. It is easy to observe that $\hat{S}_t$ is indeed a connected vertex cover of G_t.

Observe that from the steps above, we always get a connected vertex cover S_t of G_t, that is a $(1+\delta)\cdot c$-approximation of $\mathrm{OPT}_{\mathrm{CVC}}(G_t)$ and has size at least $\frac{10\ell}{\delta}$. ◁

Using Claim 19, we obtain a node t and a connected vertex cover S_t of G_t, that is a $\min\{(1+\delta)\cdot c, 2\}$-approximation of $\mathrm{OPT}_{\mathrm{CVC}}(G_t)$ and has size at least $\frac{10\ell}{\delta}$. Use Lemma 16 to obtain a connected vertex cover S'_t of $G[V_t]$ of size at most $|S_t| + 2(\ell+1)$, containing X_t.

We now obtain graph G' by removing all vertices in $V_t \setminus X_t$ from G and then contracting all vertices in X_t to a single vertex z_t. Let $\mathcal{T}'$ to be a tree decomposition of G', one may obtain $\mathcal{T}'$ by replacing occurrences of vertices in V_t by z_t in $\mathcal{T}$. Since G' is strictly smaller than G, we may use the algorithm described above to obtain a $c\cdot(1+\varepsilon)$-approximate solution S' for $\mathrm{OPT}_{\mathrm{CVC}}(G')$, using $\mathcal{T}'$. Output $S := S' \cup S'_t \setminus \{z_t\}$.

Correctness. We start by showing that S is a connected vertex cover. Verify that it is indeed a vertex cover of G: any edge within G' is covered as $S' \subseteq S$, any edge in G_t is covered since $S'_t \subseteq S$ and any other edge has at least one endpoint in $X_t \subseteq S$ and is thereby covered. It remains to verify that $G[S]$ is connected. Clearly, $G[V_t \cap S]$ is connected since it corresponds to $G[S'_t]$. Let $\tilde{G} := G - (V_t \setminus X_t)$. We show that every connected component of $\tilde{G}[S]$ contains at least one vertex from X_t, such that the entire graph is connected as $X_t \subseteq S$ and the vertices in X_t are in the same connected component as observed earlier. Suppose not, let C be such a component not containing any vertex in X_t. Consider $G'[S']$. Observe that C is also a connected component of $G'[S']$. Furthermore, vertex z_t is not adjacent to

any vertex in C, as otherwise there is an edge from some vertex in C to some vertex in X_t in $\tilde{G}$, since $X_t \subseteq S$ this contradicts that C contains no vertex from X_t. Since G' is connected however, z_t has an incident edge $\{z_t, u\}$ for some $u \in V(G')$ and thus $u \in S'$ or $z_t \in S'$. In both cases there is a vertex in S' that is not in connected component C, a contradiction with the assumption that S' is a connected vertex cover of G'.

We now show that we indeed achieve the desired approximation factor.

▷ Claim 20. $|S| \leq c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{CVC}}(G)$

Proof. Let S^* be a minimum connected vertex cover of G. Assume for now $|S^* \cap V(G_t)| \geq 4/\delta$.

$$\begin{aligned}|S| &\leq |S'_t| + |S'| \\ &\leq |S_t| + 2(\ell+1) + c \cdot (1+\varepsilon)\mathrm{OPT}_{\mathrm{CVC}}(G')\end{aligned}$$

Using $|S_t| \geq \frac{10\ell}{\delta}$

$$\begin{aligned}&\leq |S_t| + \frac{\delta}{2}|S_t| + c \cdot (1+\varepsilon)\mathrm{OPT}_{\mathrm{CVC}}(G') \\ &\leq c \cdot (1+\delta)(1+\delta/2)\mathrm{OPT}_{\mathrm{CVC}}(G_t) + c \cdot (1+\varepsilon)|(S^* \cap V(G')) \cup \{z_t\}| \\ &\leq c \cdot (1+\delta)(1+\delta/2)|(S^* \cap V(G_t)) \cup \{z_t\}| + c \cdot (1+\varepsilon)|(S^* \cap V(G')) \cup \{z_t\}| \\ &\leq c \cdot (1+\delta)(1+\delta/2)(|S^* \cap V(G_t)| + 1) + c \cdot (1+\varepsilon)|(S^* \cap V(G')) \cup \{z_t\}|\end{aligned}$$

By assuming $|S^* \cap V(G_t)| \geq 4/\delta$, and then using $\delta = \varepsilon/3$

$$\begin{aligned}&\leq c \cdot (1+\delta)(1+\delta/2)(1+\delta/4)(|S^* \cap V(G_t)|) + c \cdot (1+\varepsilon)|(S^* \cap V(G')) \cup \{z_t\}| \\ &\leq c \cdot (1+\varepsilon)(|S^* \cap V(G_t)|) + c \cdot (1+\varepsilon)|(S^* \cap V(G')) \cup \{z_t\}|\end{aligned}$$

Observe that since G_t and G' are non-empty, S^* must contain a vertex from X_t

$$\leq c \cdot (1+\varepsilon)|S^*| = c \cdot (1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{CVC}}(G).$$

It remains to observe that $|S^* \cap V(G_t)| \geq 4/\delta$ is a reasonable assumption. Suppose not, then $\mathrm{OPT}_{\mathrm{CVC}}(G_t) \leq |S^* \cap V(G_t)| + 1 \leq 4/\delta + 1$. However, $S_t \geq \frac{10\ell}{\delta} \geq 2 \cdot \mathrm{OPT}_{\mathrm{CVC}}(G_t)$, meaning that S_t is not a 2-approximation in G_t, which is a contradiction. ◁

Having shown the correctness of the procedure, it remains to argue the size of this Turing kernel. Observe that the oracle is only used when applying Claim 18. As such, we may bound the size of the kernel by $\mathcal{O}((\frac{\ell^2}{\delta})^{\lceil \frac{1+\delta}{\delta} \rceil}) = \mathcal{O}((\frac{\ell^2}{\varepsilon})^{\lceil \frac{3+\varepsilon}{\varepsilon} \rceil})$, recall that $\delta = \frac{\varepsilon}{3}$. ◀

4 Meta result

In this section we will describe a wide range of graph problems for which approximate Turing kernels can be obtained. The problems we will consider satisfy certain additional constraints, such that the general strategy already described for the Vertex Cover problem can be applied. Informally speaking, we need the following requirements. First of all, the problems should behave nicely with respect to taking the disjoint union of graphs. Secondly, we want to look at what happens for induced subgraphs. We will only consider problems whose value cannot increase when taking an induced subgraph. Furthermore, we restrict how much the optimal value can decrease when taking an induced subgraph. Finally, we require existence of a PSAKS and an approximation algorithm for the problem. We use the following definitions.

▶ **Definition 21.** *Let $\varphi\colon \mathbb{R}\times\mathbb{N}\to\mathbb{R}$ be a function. A* φ-approximation algorithm *for a problem $\mathcal{P}$ is a polynomial-time algorithm that, given an instance G with tree decomposition $\mathcal{T}$ of width ℓ, outputs a solution S such that (for minimization problems) $\mathcal{P}(G,S)\leq\varphi(\mathrm{OPT}_{\mathcal{P}}(G),\ell)$, and (for maximization problems) $\varphi(\mathcal{P}(G,S),\ell)\geq\mathrm{OPT}_{\mathcal{P}}(G)$.*

▶ **Definition 22.** *Let $\mathcal{P}$ be an optimization problem whose input is a graph. We will say that it is* friendly *if it satisfies the following conditions.*

1. *For all graphs G, G_1, and G_2 such that G is the disjoint union of graphs G_1 and G_2, $\mathrm{OPT}_{\mathcal{P}}(G)=\mathrm{OPT}_{\mathcal{P}}(G_1)+\mathrm{OPT}_{\mathcal{P}}(G_2)$. In particular, if S_1 is a solution for G_1 and S_2 is a solution for G_2, then $S_1\cup S_2$ is a solution for G and*

$$\mathcal{P}(G,S_1\cup S_2)=\mathcal{P}(G_1,S_1)+\mathcal{P}(G_2,S_2).$$

   In the other direction, given solution S in G it can efficiently be split into solutions S_1 in G_1 and S_2 in G_2 satisfying the above. For consistency, we require that the size of the optimal solution in the empty graph is zero.
2. *There exists a non-decreasing polynomial function f such that for all graphs G, for all $X\subseteq V(G)$:*

$$\mathrm{OPT}_{\mathcal{P}}(G)\leq\mathrm{OPT}_{\mathcal{P}}(G-X)+f(|X|),\text{ and }\mathrm{OPT}_{\mathcal{P}}(G-X)\leq\mathrm{OPT}_{\mathcal{P}}(G).$$

   In particular, for minimization problems there is a polynomial-time algorithm $\mathcal{A}$ that, given a solution S' in $G-X$, outputs a solution S for G such that $\mathcal{P}(G,S)\leq\mathcal{P}(G-X,S')+f(|X|)$. For maximization problems we require that any solution S for $G-X$ is also a solution for G and $\mathcal{P}(G,S)=\mathcal{P}(G-X,S)$.
3. *$\mathcal{P}^{\perp}$ parameterized by $k+\ell$, where k is the solution value and ℓ is the treewidth, has a $(1+\delta)$-approximate kernel for all $\delta>0$, that has $h(\delta,k+\ell)$ vertices for some function h that is polynomial in its second parameter.*
4. *$\mathcal{P}$ has a φ-approximation algorithm for some polynomial function φ such that $\alpha\cdot\varphi(k,\ell)<\varphi(\alpha\cdot k,\ell)$ for all $\alpha>1$, and φ is non-decreasing in its first parameter.*

Observe that many well-known vertex subset problems fit in this framework. As an example, let us verify them for the Vertex Cover problem. The first point is immediate. For the second point, let $\mathcal{A}(G,X,S)$ output $S':=S\cup X$. Verify that indeed this satisfies the conditions with $f(|X|)=|X|$. The third point follows with some extra work from the fact that Vertex Cover has a kernel with $2k$ vertices, this kernel can then be shown to be 1-approximate. For the last point, it is well-known that Vertex Cover has a 2-approximation algorithm.

▶ **Lemma 23 (★).** *Let $\mathcal{P}$ be a friendly graph optimization problem. There is a polynomial-time algorithm $\mathcal{B}$ with access to a c-approximate oracle. It takes as input a graph G with nice tree decomposition $\mathcal{T}$ of width ℓ and a number $0<\delta\leq 1$, and outputs either*

- *a node t such that $\mathrm{OPT}_{\mathcal{P}}(G[V_t\setminus X_t])\geq\frac{f(\ell+1)}{\delta}$ together with a $(c\cdot(1+\delta))$-approximate solution S_t to $\mathcal{P}$ in $G[V_t\setminus X_t]$, or*
- *a $c\cdot(1+\delta)$-approximate solution for G,*

using calls to the oracle on graphs with at most $h(\delta,\varphi(k,\ell)+\ell)$ vertices, where $k=\frac{2f(\ell+1)}{\delta}+f(1)$.

We will prove the result separately for maximization and minimization problems (see [24, Lemma 25] for the minimization case).

Proof of Lemma 23: Maximization problems. Let r be the root of $\mathcal{T}$, and observe that $G = G[V_r \setminus X_r]$ since $X_r = \emptyset$. Let $k := \frac{2f(\ell+1)}{\delta} + f(1)$. Compute a φ-approximate solution $\tilde{S}$ in G. We do a case distinction on the value of this solution.

If $\mathcal{P}(G, \tilde{S}) \leq k$, then apply the PSAKS with approximation ratio $1+\delta$ to $(G, \varphi(k,\ell)+\ell)$ and obtain instance (G', k') with at most $h(\delta, \varphi(k,\ell)+\ell)$ vertices. Obtain solution S' by applying the c-approximate oracle on G'. Apply the solution lifting algorithm to S' to obtain a solution S for G. We start by showing that S is the desired approximate solution. Clearly, $\mathcal{P}(G', S') \geq \frac{1}{c} \cdot \mathrm{OPT}_{\mathcal{P}}(G')$ by correctness of the oracle. If $\mathcal{P}(G', S') > k'$, then $\mathcal{P}^{\perp}(G', k', S') = k' + 1$ and thus $\mathcal{P}^{\perp}(G', k', S') \geq \mathrm{OPT}_{\mathcal{P}^{\perp}}(G', k')$. Otherwise, we have $\mathcal{P}^{\perp}(G', k', S') = \mathcal{P}(G', S') \geq \frac{1}{c} \cdot \mathrm{OPT}_{\mathcal{P}}(G') \geq \frac{1}{c} \cdot \mathrm{OPT}_{\mathcal{P}^{\perp}}(G', k')$. From the properties of the solution lifting algorithm, it now follows that $\mathcal{P}^{\perp}(G, \varphi(k,\ell)+\ell, S) \geq \frac{1}{c(1+\delta)}\mathrm{OPT}_{\mathcal{P}^{\perp}}(G, \varphi(k,\ell)+\ell)$. Observe that since $\mathcal{P}(G, \tilde{S}) \leq k$ and φ non-decreasing in its first parameter, we get that $\mathrm{OPT}_{\mathcal{P}}(G) \leq \varphi(\mathcal{P}(G, \tilde{S}), \ell) \leq \varphi(k, \ell)$ and thereby $\mathrm{OPT}_{\mathcal{P}}(G) = \mathrm{OPT}_{\mathcal{P}^{\perp}}(G, \varphi(k,\ell)+\ell)$. It follows that $\mathcal{P}(G, S) \geq \mathcal{P}^{\perp}(G, \varphi(k,\ell)+\ell, S) \geq \frac{1}{c(1+\delta)}\mathrm{OPT}_{\mathcal{P}^{\perp}}(G, \varphi(k,\ell)+\ell) = \frac{1}{c(\delta+1)}\mathrm{OPT}_{\mathcal{P}}(G)$.

Suppose $\mathcal{P}(G, \tilde{S}) > k$. For every node $t \in T$, compute a φ-approximate solution $\tilde{S}_t$ for graph $G[V_t \setminus X_t]$. We start by showing how to find a node $t \in V(T)$ such that both $\mathcal{P}(G[V_t \setminus X_t], \tilde{S}_t) \leq k$, and $\mathrm{OPT}_{\mathcal{P}}(G[V_t \setminus X_t]) \geq \frac{f(\ell+1)}{\delta}$. Start by observing that for the leaf vertices, it holds that $\mathcal{P}(G[V_t \setminus X_t], \tilde{S}_t) = 0 \leq k$. On the other hand, for the root, we found that $\mathcal{P}(G[V_r \setminus X_r], \tilde{S}_r) = \mathcal{P}(G, \tilde{S}) > k$. As such, we can find a node p such that $\mathcal{P}(G[V_p \setminus X_p], \tilde{S}_p) > k$, while for all of its children t it holds that $\mathcal{P}(G[V_t \setminus X_t], \tilde{S}_t) \leq k$. We show that one of the children of p has the desired properties. The result that $\mathcal{P}(G[V_t \setminus X_t], \tilde{S}_t) \leq k$ for all children of p is immediate. On the other hand, observe that $\mathrm{OPT}_{\mathcal{P}}(G[V_p \setminus X_p]) \geq \mathcal{P}(G[V_p \setminus X_p], \tilde{S}_p) \geq k \geq \frac{2f(\ell+1)}{\delta}$, by assumption. We do a case distinction on the type of node that p is in the nice tree decomposition.

- p is an introduce or forget node. In this case, p has exactly one child t and $V_t \setminus X_t = V_p \setminus X_p$, or $V_t \setminus X_t = (V_p \setminus X_p) \setminus \{v\}$ for some $v \in V(G)$. Since $\mathcal{P}$ is friendly, we get that $\mathrm{OPT}_{\mathcal{P}}(G[V_t \setminus X_t]) \geq \mathrm{OPT}_{\mathcal{P}}(G[V_p \setminus X_p]) - f(1) \geq \frac{f(\ell+1)}{\delta}$.
- p is a join node. In this case, p has exactly two children t_1 and t_2 and $G[V_p \setminus X_p]$ is the disjoint union of $G[V_{t_1} \setminus X_{t_1}]$ and $G[V_{t_2} \setminus X_{t_2}]$. Obtain S_1 and S_2 such that $\tilde{S}_p = S_1 \cup S_2$ and S_1 is a solution in $G[V_{t_1} \setminus X_{t_1}]$, S_2 in $G[V_{t_2} \setminus X_{t_2}]$, and $\mathcal{P}(G[V_p \setminus X_p], \tilde{S}_p) = \mathcal{P}(G[V_{t_1} \setminus X_{t_1}], S_1) + \mathcal{P}(G[V_{t_2} \setminus X_{t_2}], S_2)$. This can be done since $\mathcal{P}$ is friendly.
  Therefore, there is $i \in [2]$ such that $\mathrm{OPT}_{\mathcal{P}}(G[V_{t_i} \setminus X_{t_i}]) \geq \mathcal{P}(G[V_{t_i} \setminus X_{t_i}], S_i) \geq \mathcal{P}(G[V_p \setminus X_p], \tilde{S}_p)/2 \geq \frac{f(\ell+1)}{\delta}$.

So, we have obtained a node t such that $\mathcal{P}(G[V_t \setminus X_t], \tilde{S}_t) \leq k$, and $\mathrm{OPT}_{\mathcal{P}}(G[V_t \setminus X_t]) \geq \frac{f(\ell+1)}{\delta}$. We now show how to obtain S_t. Apply the PSAKS with ratio $1+\delta$ to $(G[V_t \setminus X_t], \varphi(k,\ell)+\ell)$ and obtain instance (G', k'). Apply the c-approximate oracle on G' to obtain a solution S''. Apply the solution lifting algorithm to S'' to obtain solution S_t in $G[V_t \setminus X_t]$. With similar arguments as before, S_t is a $c(1+\delta)$-approximate solution in $G[V_t \setminus X_t]$. Output t and S_t. ◀

The next theorem gives a polynomial-size $(1+\varepsilon)$-approximate Turing kernel with parameter treewidth for any friendly optimization problem $\mathcal{P}$. The Turing kernel follows the same ideas as the Turing kernels presented earlier in this paper, using Lemma 23 to find a node in the tree decomposition where we can split the graph.

▶ **Theorem 24 (★).** *Let $\mathcal{P}$ be a friendly optimization problem on graphs. Then $\mathcal{P}$ parameterized by treewidth has a $(1+\varepsilon)$-approximate Turing kernel with $h(\frac{\varepsilon}{3}, \varphi(\frac{6f(\ell+1)}{\varepsilon} + f(1), \ell) + \ell)$ vertices, for all $0 < \varepsilon \leq 1$.*

While the description of the Turing kernel is mostly the same for maximization and minimization problems, the correctness proof will differ quite significantly. Therefore, these cases will be proven separately, the proof for minimization problems can be found in the full version of the paper.

Proof of Theorem 24: Maximization problems. Let $\mathcal{P}$ be a friendly maximization problem. We show that Algorithm 3 is the desired approximate Turing kernel, where we let $\mathcal{A}(G, X_t, S' \cup S_t)$ return $S' \cup S_t$.

Algorithm 3 An approximate Turing kernel for friendly optimization problems $\mathcal{P}$.

procedure ApproxP$(G, \mathcal{T}, \varepsilon)$
  Turn $\mathcal{T}$ into a nice tree decomposition
  Apply Lemma 23 for $\delta := \varepsilon/3$
  if this outputs an approximate solution S for G **then**
    return S
  else // *We obtained $t \in V(T)$, $c(1+\delta)$-approximate solution S_t for $\mathcal{P}$ in $G[V_t \setminus X_t]$ such that* $\mathrm{OPT}_{\mathcal{P}}(G[V_t \setminus X_t]) \geq \frac{f(\ell+1)}{\delta}$
    Let $G' := G - V_t$.
    Obtain $\mathcal{T}'$ from $\mathcal{T}$ by removing the subtree rooted at t and all vertices in X_t
    Let $S' :=$ ApproxP$(G', \mathcal{T}', \varepsilon)$
    return $S := \mathcal{A}(G, X_t, S' \cup S_t)$
  end if
end procedure

It is easy to see that since $\mathcal{P}$ is friendly, the algorithm indeed returns a correct solution for $\mathcal{P}$ in G, it remains to prove the size bound.

$$\begin{aligned}
\mathrm{OPT}_{\mathcal{P}}(G) &\leq \mathrm{OPT}_{\mathcal{P}}(G - X_t) + f(\ell+1) \\
&= \mathrm{OPT}_{\mathcal{P}}(G - V_t) + \mathrm{OPT}_{\mathcal{P}}(G[V_t \setminus X_t]) + f(\ell+1) \\
&\leq \mathrm{OPT}_{\mathcal{P}}(G - V_t) + (1+\delta) \cdot \mathrm{OPT}_{\mathcal{P}}(G[V_t \setminus X_t]) \\
&\leq c \cdot (1+\varepsilon) \cdot \mathcal{P}(G - V_t, S') + c \cdot (1+\delta)^2 \cdot \mathcal{P}(G[V_t \setminus X_t], S_t) \\
&\leq c \cdot (1+\varepsilon) \cdot (\mathcal{P}(G - V_t, S') + \mathcal{P}(G[V_t \setminus X_t], S_t)) \\
&= c \cdot (1+\varepsilon) \cdot (\mathcal{P}(G - X_t, S' \cup S_t)) = c \cdot (1+\varepsilon) \cdot (\mathcal{P}(G, S' \cup S_t)).
\end{aligned}$$

◀

4.1 Consequences

We show that a number of considered problems are friendly in the next lemma.

▶ **Lemma 25.** *The following problems are friendly (with respect to the following bounds).*

- Independent Set *with* $f(x) = x$, $h(\delta, m) = (m+1)^2$, $\varphi(s, \ell) = (\ell+1) \cdot s$.
- Vertex-Disjoint H-packing *for connected graphs* H, *with* $|V(H)|$ *constant, with* $f(x) = x$, $h(\delta, k) = \mathcal{O}(k^{|V(H)|-1})$, $\varphi(s, \ell) = |V(H)| \cdot s$.
- Vertex Cover *with* $f(x) = x$, $h(\delta, k) = 2k$, $\varphi(s, \ell) = 2s$.
- Clique Cover *with* $f(x) = x$, $h(\delta, m) = m(m+1)$, $\varphi(s, \ell) = (\ell+1) \cdot s$
- Feedback Vertex Set *with* $f(x) = x$, $h(\delta, k) = 4k^2$, $\varphi(s, \ell) = 2s$.
- Edge Dominating Set *with* $f(x) = x$, $h(\delta, k) = 4k^2 + 4k$, $\varphi(s, \ell) = 2s$.

Proof.

Independent Set. Clearly, if G is the disjoint union of two graphs G_1 and G_2, then the union of an independent set in G_1 and an independent set in G_2 forms an independent set in G. Conversely, restricting an independent set in G to $V(G_1)$ (respectively $V(G_2)$) results in an independent set in G_1 (respectively, G_2). Furthermore, if X is a subset of G it is easy to verify that $\mathrm{OPT}_{\mathrm{IS}}(G) \leq \mathrm{OPT}_{\mathrm{IS}}(G-X) + |X|$ and that $\mathrm{OPT}_{\mathrm{IS}}(G-X) \leq \mathrm{OPT}_{\mathrm{IS}}(G)$ as any independent set in $G-X$ is an independent set in G. The PSAKS parameterized by $m := k+\ell$ is as follows. It is known that any graph of treewidth ℓ has an independent set of size at least $|V(G)|/(\ell+1)$. This can be seen from the fact that such graphs are ℓ-degenerate, meaning that there is an order of the vertices $v_1, \ldots, v_n$ such that v_i has degree at most ℓ in $G[v_1, \ldots, v_i]$. As such, an independent set of size $|V(G)|/(\ell+1)$ can be greedily constructed.

Thus, if $|V(G)| > (m+1)^2$, we simply let G' be the graph consisting of an independent set of size $m+1$. The solution lifting algorithm can then simply find a size-$(m+1)$ independent set and output it. This is always an optimal solution for $\mathcal{P}^{\perp}$, since it does not distinguish between solutions of size larger than m. Otherwise, we obtain that $|V(G)| \leq (m+1)^2$ and the PSAKS will not modify G. In both cases, we output a graph on at most $(m+1)^2$ vertices.

It remains to show that there is an approximation algorithm, the idea is equivalent to the PSAKS. Return an independent set in G of size at least $|V(G)|/(\ell+1)$. Then indeed $\varphi(|V(G)|/(\ell+1), \ell) = |V(G)| \geq \mathrm{OPT}_{\mathrm{IS}}(G)$.

Vertex-Disjoint H-Packing. Requirements 1 and 2 are easily verified for $f(|X|) = |X|$, as any vertex in X could be contained in at most one graph in any copy of H.

A simple approximation algorithm for Vertex-Disjoint H-Packing is to simply return any maximal H-packing S. We show that $|S| \geq \frac{1}{|V(H)|}\mathrm{OPT}_{\mathcal{P}}(G)$, such that this is an φ-approximation algorithm with $\varphi(s,\ell) = |V(H)| \cdot s$. Suppose there is an optimal solution S^* with $|S^*| > |V(H)| \cdot |S|$. Since the copies of H in S are vertex-disjoint, S uses exactly $|V(H)| \cdot |S|$ vertices. Since S^* contains more than $|V(H)| \cdot |S|$ elements, it follows that there is $s \in S^*$ that uses no vertices used by S, contradicting that S is maximal.

The existence of a PSAKS is shown in [24, Lemma 31, Appendix A].

Vertex Cover. Requirements 1 and 2 are easily verified for vertex cover, let algorithm $\mathcal{A}$ simply output the union of the given solution with set X. As (implicitly) observed in the proof of Lemma 8, Vertex Cover has a 1-approximate kernel of size $2k$. Furthermore, it is well-known to be 2-approximable.

Clique Cover. Requirement 1 is easy to verify. We show Requirement 2. Let $X \subseteq V(G)$. Let S be a clique cover of G, it is easy to see that $\{s \setminus X \mid s \in S\}$ is a clique cover of $G-X$, of size at most $|S|$. Therefore, $\mathrm{OPT}_{\mathcal{P}}(G) \geq \mathrm{OPT}_{\mathcal{P}}(G-X)$. Furthermore, let algorithm $\mathcal{A}$ when given G, clique cover S of $G-X$ and X output the clique cover $S \cup \{\{x\} \mid x \in X\}$. Then this is a clique cover of G and it has size at most $|S| + |X| \leq |S| + f(|X|)$.

To show Requirement 3, we obtain a 1-approximate kernel for Clique Cover in a somewhat similar way as for Independent Set. Observe that any n-vertex graph with treewidth ℓ has a minimum clique cover of size at least $\frac{n}{\ell+1}$. So, given G and parameter $m := k+\ell$, if $n > m(m+1) \geq k \cdot (\ell+1)$, we know for sure that G does not have a minimum clique cover of size k. The reduction algorithm reduces G to an independent set of size $m+1$. The solution lifting algorithm (irrespective of the solution given for G') outputs $V(G)$. Otherwise, if $n \leq m(m+1)$ we simply let G be the output of the reduction algorithm. Since the graph does not change, the solution lifting algorithm simply outputs the solution it is given. In both cases, the reduced instance has size at most $m(m+1)$.

It remains to verify that there is a φ-approximation algorithm for Clique Cover. Given a graph G of treewidth ℓ, we simply output $\{\{v\} \mid v \in V(G)\}$. Clearly, this is a valid clique cover of G of size $|V(G)|$. Observe that since G has treewidth ℓ, G contains no cliques of size larger than $\ell+1$, thus any clique in the optimal clique cover of G covers at most $\ell+1$ vertices. As such, the optimal solution contains at least $\frac{|V(G)|}{\ell+1}$ cliques, and thus $|S| \leq (\ell+1)\mathrm{OPT}_{\mathcal{P}}(G)$.

Feedback Vertex Set. Requirements 1 and 2 are straightforward to verify. The problem has a 1-approximate kernel with $4k^2$ vertices and therefore a PSAKS by [24, Lemma 29, Appendix A], showing Requirement 3. It is also known that the Feedback Vertex Set problem has a 2-approximation algorithm [2], showing Requirement 4.

Edge Dominating Set. Requirement 1 is again straightforward. For the second requirement, let G be a graph and let $X \subseteq V(G)$. We start by showing that $\mathrm{OPT}_{\mathcal{P}}(G) \geq \mathrm{OPT}_{\mathcal{P}}(G-X)$. Let S be an edge-dominating set in G. We obtain an edge-dominating set S' for $G-X$ as follows. Initialize S' as the set of edges with both endpoints in $V(G) \setminus X$, so $S' := \{e \in S \mid e \cap X = \emptyset\}$. For every edge $\{x,v\} \in S$ with $x \in X, v \notin X$, choose one arbitrary edge $\{u,v\} \in E(G-X)$ and add $\{u,v\}$ to S'. If no such edge exists, do nothing. Clearly, $|S'| \leq |S|$. Furthermore, we show that S' is indeed an edge dominating set. Suppose for contradiction that $e = \{u,v\}$ is not dominated in $G-X$ by S'. Let $\{w,v\} \in S$ be the edge dominating $\{u,v\}$ in G. Then, since $\{w,v\} \notin S'$, we have $w \in X$. But then some edge with endpoint v was added to S', a contradiction.

We continue by showing that $\mathrm{OPT}_{\mathcal{P}}(G) \leq \mathrm{OPT}_{\mathcal{P}}(G-X) + |X|$ and that algorithm $\mathcal{A}$ exists. Let S be a solution for $G-X$, then algorithm $\mathcal{A}$ will output S together with one edge $\{x,v\} \in E(G)$ for all $x \in X$. In the case that $x \in X$ is isolated in G, no edge is added for this vertex. By this definition, the output has size at most $|S|+|X|$. Furthermore, any edge with vertices in $V(G-X)$ is dominated by S. Any edge with at least one endpoint in X is dominated by the additional edges.

Edge Dominating Set has a kernel that outputs a graph G' of size at most $4k^2+4k$ such that G' is an induced subgraph of G and any size-k edge dominating set in G' is also an edge dominating set in G [21]. We can see that this is a 1-approximate kernel. Let the solution lifting algorithm simply output the solution for G' as a solution for G. Since any solution of size at most k in G' is a solution in G, and obviously any solution in G is a solution for G', it is clear that $\mathrm{OPT}_{\mathcal{P}^\perp}(G',k) = \mathrm{OPT}_{\mathcal{P}^\perp}(G,k)$. As such, the approximation ratio is preserved by the solution lifting algorithm.

It is known that even the weighted version of Edge Dominating Set can be 2-approximated [20], such that the problem has a φ-approximation for $\varphi(s,\ell) = 2s$. ◀

As an immediate consequence of Lemma 25 and Theorem 24, we obtain approximate Turing kernels for a large number of graph problems. These results are summarized in the corollary below, the size bounds are obtained by substituting the relevant bounds given by Lemma 25 into Theorem 24.

▶ **Corollary 26.** *The following problems have a polynomial $(1+\varepsilon)$-approximate Turing kernel for all $0 < \varepsilon \leq 1$, of the given size (in number of vertices), when parameterized by treewidth ℓ.*

- *Independent Set, of size $\mathcal{O}(\frac{\ell^4}{\varepsilon^2})$.*
- *Vertex-Disjoint H-packing for connected graphs H, of size $\mathcal{O}((\frac{\ell}{\varepsilon})^{|V(H)|-1})$.*
- *Vertex Cover of size $\mathcal{O}(\frac{\ell}{\varepsilon})$.*
- *Clique Cover of size $\mathcal{O}(\frac{\ell^4}{\varepsilon^2})$.*
- *Feedback Vertex Set of size $\mathcal{O}((\frac{\ell}{\varepsilon})^2)$.*
- *Edge Dominating Set of size $\mathcal{O}((\frac{\ell}{\varepsilon})^2)$.*

We observe that the bounds for Independent Set and Clique Cover can be improved to $\mathcal{O}(\frac{\ell^2}{\varepsilon})$ by a more careful analysis. Instead of using that the problem is friendly and applying Lemma 23, one may simply find t such that the number of vertices in $G[V_t \setminus X_t]$ is between $\frac{(\ell+1)^2}{\delta}$ and $\frac{10(\ell+1)^2}{\delta}$, and use that an optimal solution has size at least $|V(G)|/(\ell+1)$ for graphs of treewidth ℓ. There is no need to apply a kernelization in this case.

5 Conclusion

In this paper we have provided approximate Turing kernels for various graph problems when parameterized by treewidth. Furthermore, we give a general result that can be used to obtain approximate Turing kernels for all friendly graph problems parameterized by treewidth.

While the notion of being friendly captures many known graph problems, some interesting problems do not fit this definition. In particular, it is not clear whether the Dominating Set problem has a polynomial-size constant-factor approximate Turing kernel when parameterized by treewidth. We leave this as an open problem.

References

1 Akanksha Agrawal, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Interval vertex deletion admits a polynomial kernel. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1711–1730. SIAM, 2019. doi:10.1137/1.9781611975482.103.

2 Ann Becker and Dan Geiger. Optimization of pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1):167–188, 1996. doi:10.1016/0004-3702(95)00004-6.

3 Daniel Binkele-Raible, Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. *ACM Trans. Algorithms*, 8(4):38:1–38:19, 2012. doi:10.1145/2344422.2344428.

4 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

5 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. doi:10.1145/2973749.

6 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.

7 Yixin Cao, Ashutosh Rai, R. B. Sandeep, and Junjie Ye. A polynomial kernel for diamond-free editing. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 10:1–10:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.10.

8 Steven Chaplick, Fedor V. Fomin, Petr A. Golovach, Dusan Knop, and Peter Zeman. Kernelization of graph hamiltonicity: Proper h-graphs. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2019. doi:10.1007/978-3-030-24766-9_22.

9 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. In Peter Widmayer, Gabriele Neyer, and Stephan Eidenbenz, editors, *Graph-Theoretic Concepts in Computer Science*, pages 313–324, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

10 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Marek Cygan, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Clique cover and graph separation: New incompressibility results. *ACM Trans. Comput. Theory*, 6(2):6:1–6:19, 2014. doi:10.1145/2594439.

12 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014. doi:10.1145/2650261.

13 Eduard Eiben, Danny Hermelin, and M. S. Ramanujan. Lossy kernels for hitting subgraphs. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.MFCS.2017.67.

14 Eduard Eiben, Mithilesh Kumar, Amer E. Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy kernels for connected dominating set on sparse graphs. *SIAM J. Discrete Math.*, 33(3):1743–1771, 2019. doi:10.1137/18M1172508.

15 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008. doi:10.1137/05064299X.

16 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.

17 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

18 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

19 Pierre Fraigniaud and Nicolas Nisse. Connected treewidth and connected graph searching. In *LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006, Proceedings*, pages 479–490, 2006. doi:10.1007/11682462_45.

20 Toshihiro Fujito and Hiroshi Nagamochi. A 2-approximation algorithm for the minimum weight edge dominating set problem. *Discrete Applied Mathematics*, 118(3):199–207, 2002. doi:10.1016/S0166-218X(00)00383-8.

21 Torben Hagerup. Kernels for edge dominating set: Simpler or smaller. In *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, pages 491–502, 2012. doi:10.1007/978-3-642-32589-2_44.

22 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (turing) kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/s00453-014-9910-8.

23 Eva-Maria C. Hols and Stefan Kratsch. On kernelization for edge dominating set under structural parameters. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 36:1–36:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.36.

24 Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Approximate turing kernelization for problems parameterized by treewidth. *CoRR*, abs/2004.12683, 2020. arXiv:2004.12683v1.

25 Bart M. P. Jansen. Turing kernelization for finding long paths and cycles in restricted graph classes. *J. Comput. Syst. Sci.*, 85:18–37, 2017. doi:10.1016/j.jcss.2016.10.008.

26 Bart M. P. Jansen and Dániel Marx. Characterizing the easy-to-find subgraphs from the viewpoint of polynomial-time algorithms, kernels, and turing kernels. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 616–629. SIAM, 2015. doi:10.1137/1.9781611973730.42.

27 Bart M. P. Jansen and Astrid Pieterse. Polynomial kernels for hitting forbidden minors under structural parameterizations. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 48:1–48:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.48.

28 Bart M. P. Jansen, Marcin Pilipczuk, and Erik Jan van Leeuwen. A deterministic polynomial kernel for odd cycle transversal and vertex multiway cut in planar graphs. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 39:1–39:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.39.

29 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.

30 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237, 2017. doi:10.1145/3055399.3055456.

31 Rolf Niedermeier and Christophe Paul, editors. *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: http://www.dagstuhl.de/dagpub/978-3-95977-100-9.

32 M. S. Ramanujan. An approximate kernel for connected feedback vertex set. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 77:1–77:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.77.

33 Carla D. Savage. Depth-first search and the vertex cover problem. *Inf. Process. Lett.*, 14(5):233–237, 1982. doi:10.1016/0020-0190(82)90022-9.

34 Stéphan Thomassé, Nicolas Trotignon, and Kristina Vuskovic. A polynomial turing-kernel for weighted independent set in bull-free graphs. *Algorithmica*, 77(3):619–641, 2017. doi:10.1007/s00453-015-0083-x.

35 Jouke Witteveen, Ralph Bottesch, and Leen Torenvliet. A hierarchy of polynomial kernels. In Barbara Catania, Rastislav Královic, Jerzy R. Nawrocki, and Giovanni Pighizzini, editors, *SOFSEM 2019: Theory and Practice of Computer Science - 45th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 27-30, 2019, Proceedings*, volume 11376 of *Lecture Notes in Computer Science*, pages 504–518. Springer, 2019. doi:10.1007/978-3-030-10801-4_39.

The Fine-Grained Complexity of Median and Center String Problems Under Edit Distance

Gary Hoppenworth
Department of Computer Science, University of Central Florida, Orlando, FL, USA
gary.hoppenworth@gmail.com

Jason W. Bentley
Department of Mathematics, University of Central Florida, Orlando, FL, USA
jason.bentley@ucf.edu

Daniel Gibney
Department of Computer Science, University of Central Florida, Orlando, FL,USA
https://www.cs.ucf.edu/~dgibney/
daniel.j.gibney@gmail.com

Sharma V. Thankachan
Department of Computer Science, University of Central Florida, Orlando, FL, USA
http://www.cs.ucf.edu/~sharma/
sharma.thankachan@ucf.edu

Abstract

We present the first fine-grained complexity results on two classic problems on strings. The first one is the k-Median-Edit-Distance problem, where the input is a collection of k strings, each of length at most n, and the task is to find a new string that minimizes the sum of the edit distances from itself to all other strings in the input. Arising frequently in computational biology, this problem provides an important generalization of edit distance to multiple strings and is similar to the multiple sequence alignment problem in bioinformatics. We demonstrate that for any $\varepsilon > 0$ and $k \geq 2$, an $O(n^{k-\varepsilon})$ time solution for the k-Median-Edit-Distance problem over an alphabet of size $O(k)$ refutes the Strong Exponential Time Hypothesis (SETH). This provides the first matching conditional lower bound for the $O(n^k)$ time algorithm established in 1975 by Sankoff.

The second problem we study is the k-Center-Edit-Distance problem. Here also, the input is a collection of k strings, each of length at most n. The task is to find a new string that minimizes the maximum edit distance from itself to any other string in the input. We prove that the same conditional lower bound as before holds. Our results also imply new conditional lower bounds for the k-Tree-Alignment and the k-Bottleneck-Tree-Alignment problems studied in phylogenetics.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Edit Distance, Median String, Center String, SETH

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.61

Funding Supported in part by the U.S. National Science Foundation (NSF) under CCF-1703489.

1 Introduction

Recent years have seen a remarkable increase in our understanding of the hardness of problems in the complexity class P. By establishing conditional lower bounds based on popular conjectures, researchers have been able to identify which problems are unlikely to yield algorithms significantly faster than what is known, at least not without solving other long-standing open questions. We contribute to this growing body of research here by establishing tight conditional hardness results for the k-Median-Edit-Distance problem. This generalizes the seminal work by Backurs and Indyk in STOC 2015, which showed that conditioned on the Strong Exponential Time Hypothesis (SETH), there does not exist a strongly subquadratic algorithm for computing the edit distance between two strings [10].

▶ **Problem 1** (k-Median-Edit-Distance). *Given a set $\mathcal{S}$ of k strings, each of length at most n, find a string s^* (called a median string) that minimizes the sum of edit distances from the strings in $\mathcal{S}$ to s^*. This sum is called the median edit distance.*

When $k = 2$ this problem is equivalent to the well-known edit distance problem, whose famous dynamic programming solution was first given in 1965 by Vintsyuk [44]. An algorithm for solving this problem on k strings in time $O(n^k)$ was then given by Sankoff in 1975 [41] in the more general context of tree alignment (mutation trees). Since Sankoff's solution, no algorithms with significantly better time complexity have been developed. This is despite the problem being of practical importance as well as the subject of extensive study [29, 30, 33, 38]. Compelling reasons for this were finally given 25 years later by Higuera and Casacuberta in 2000 who showed the NP-completeness of the problem over unbounded alphabets [20]. This result was later strengthened to finite alphabets in [42] and then even to binary alphabets in [39]. In [39] it was also shown that the problem is W[1]-hard in k. This last result implies it is highly unlikely to find an algorithm with time complexity of the form $f(k) \cdot N^{O(1)}$, where N is the sum of the lengths of the k strings. None of these hardness results, however, rule out the possibility of algorithms where the time complexity is of the form $O(n^{k-\varepsilon})$. Nearly five decades after its creation, this paper gives a convincing argument as to why a significant improvement over Sankoff's algorithm is unlikely. Specifically, we show that an $O(n^{k-\varepsilon})$ time algorithm for any $\varepsilon > 0$ would refute SETH. We also prove that the same lower bound holds for a related problem known as the k-Center-Edit-Distance.

▶ **Problem 2** (k-Center-Edit-Distance). *Given a set $\mathcal{S}$ of k strings, each of length at most n, find a string s^* (called a center string) that minimizes the maximum of edit distances from the strings in $\mathcal{S}$ to s^*. The maximum edit distance from s^* to any string in $\mathcal{S}$ is called the center edit distance.*

Like k-Median-Edit-Distance, the k-Center-Edit-Distance problem is known to be NP-complete and W[1]-hard in k [39]. Additionally, k-Center-Edit-Distance has been shown to have an $O(n^{2k})$ time solution [39]. However, ours are the first fine-grained complexity results for both these problems. Finally, we note that our results imply similar conditional lower bounds for two classic tree alignment problems from phylogenetics called k-Tree-Alignment and k-Bottleneck-Tree-Alignment [18, 28, 43, 45]. The k-Tree-Alignment (resp. k-Bottleneck-Tree-Alignment) problem is defined as follows: given a tree $\mathcal{T}$ with k leaves where each leaf is labeled with a string of length n, find an assignment of strings to all internal vertices of $\mathcal{T}$ such that the sum (resp. max) of edit distances between adjacent strings/vertices over all edges is minimal. Note that the median (resp. center) edit distance problem on k strings is a special case of the k-Tree-Alignment (resp. k-Bottleneck-Tree-Alignment) problem, specifically when the tree has only one internal vertex.

1.1 Related Work

Recent progress in the field of fine-grained complexity has given us conditional hardness results for many popular problems. The list of problems includes those related to graphs, computational geometry, and strings [1, 3, 4, 6, 7, 8, 10, 15, 17, 19, 21, 24, 23, 31, 32]. Reductions based on SETH, such as the one considered here, tend to have a very similar structure. The Orthogonal Vectors problem [46] is typically used as an intermediate step in the reduction. The proof we provide here works off a generalized variant of the Orthogonal Vectors problem as used in [2]. Our work contributes to a growing list of conditional lower bounds for string problems which we describe in more detail below.

Along with the SETH-based lower bound for edit distance by Backurs and Indyk in [10], there has been a number of newly appearing conditional lower bounds for string related problems [9, 12, 14, 16]. Bringmann and Künnemann created a framework by which any string problem which allowed for a particular gadget construction has similar SETH-based lower bounds proven for it [13]. This framework includes the problems of the longest common subsequence, dynamic time warping, and edit distance under a binary alphabet (less than the four symbols used in the original reduction by Backurs and Indyk). Further work to extend these types of lower bounds to more than two strings was undertaken in [2], where it was shown that an algorithm which could find the longest common subsequence on k strings in time $O(n^{k-\varepsilon})$ for any $\varepsilon > 0$ would refute SETH. The study of conditional hardness of problems on k strings also includes [22], where the longest common increasing subsequence on k strings, k-LCIS, was studied. Likewise in [7] the local alignment problem on k strings under sum of pairs was considered. In both of the last two works mentioned, it was shown that an $O(n^{k-\varepsilon})$ algorithm would refute SETH.

Another notable achievement in this direction is in [5], where it was shown that it is possible to weaken the assumptions used to achieve many of these results. They showed that under much weaker conjectures than SETH regarding circuit complexity, many of the same hardness results still hold. In fact, for any problem where the gadgetry of Bringmann and Künnemann can be applied, having a strongly sub-quadratic time algorithm would have drastic implications for our ability to solve satisfiability problems on Boolean circuits much more complex than those required for 3-SAT. Furthermore, their work also demonstrated that if one could shave off arbitrarily large logarithmic factors, it would have drastic implications in the field of circuit complexity. In this same work, they showed that their reduction from branching programs to string problems can be adapted to k-LCS, implying circuit-based hardness results apply for LCS on k strings.

There exists a close relationship between LCS and edit distance on two strings. Namely, on two strings of lengths n and m, the edit distance with only the insertion and deletion operations is equal to $n + m - 2\ell$, where ℓ is the length of the strings' longest common subsequence. For more than two strings, such a clear relationship (in terms of just lengths and number of edits) seems unlikely. In fact, there exist collections of k strings where the lengths of the longest common subsequences are equal, but the median edit distances are not, e.g., with $k = 3$ and $n \geq 1$, the sets $\{a^n, a^n, b^n\}$ and $\{a^n, b^n, c^n\}$ both have a longest common subsequence of length zero, while the first has median edit distance n and the second has median edit distance $2n$. Because of this, it seems difficult to parlay the hardness results proven for k-LCS into hardness results for k-Median-Edit-Distance, even under only insertions and deletions. Hence, the hardness of k-Median-Edit-Distance was left open. On the other hand, a 2-approximation for k-Median-Edit-Distance can be easily obtained in $O(k^2n^2)$ time: simply choose the string within the collection that minimizes the sum of edit distances from itself to the other strings.

The problem of finding the center string of a set of k strings, the string which minimizes the maximum distance from itself to any string in the set, has more often been studied under the Hamming distance metric than the edit distance metric. In this context the problem is typically called the closest string problem [25, 27, 35, 36]. The problem under the Hamming distance metric is NP-complete [34], whereas the median version under Hamming distance can be easily solved in polynomial time. In the cases where this problem has been studied under the edit distance metric, it has made use of a parameter d, the maximum distance any solution is allowed to have from an input string. The problem is fixed parameter tractable in d, which is the basis of many solutions [11, 26, 37].

2 Hardness for k-Median-Edit-Distance

Our reduction will be from the k-Most-Orthogonal-Vectors problem, which was first introduced in [2]. It was shown that if it could be solved in $\mathcal{O}(n^{k-\varepsilon})$ time for some constant $\varepsilon > 0$, it would imply new upper bounds for MAX-CNF-SAT that would violate SETH.

▶ **Problem 3** (k-Most-Orthogonal-Vectors). *Given $k \geq 2$ sets $S_1, S_2, \ldots, S_k$ each containing n binary vectors $v \in \{0,1\}^d$, and an integer $r < d$, are there k vectors $v_1, v_2, \ldots, v_k$ with $v_i \in S_i$ such that their inner product, defined as $\sum_{h=1}^{d} \prod_{t\in[1,k]} v_t[h]$, is at most r? A collection of vectors that satisfies this property will be called r-far, and otherwise called r-close.*

Modifying the Vectors. In our reduction we apply a modification to the vectors in our input sets $S_1, S_2, \ldots, S_k$. We prepend $(r+1)$ 0's to each vector $v \in S_1$ and $(r+1)$ 1's to each vector $v \in S_i$ where $i > 1$. Every vector is now of dimension $d + r + 1 \leq 2d$ and the k-Most-Orthogonal-Vectors problem is identical on the original and modified sets.

2.1 Technical Overview

Given sets $S_1, S_2, \ldots, S_k$ of binary vectors, we will design strings $T_1, T_2, \ldots, T_k$ such that if there exists a collection of r-far vectors in the input, then their median edit distance will be at most a constant E^-. Otherwise, if there does not exist any collection of r-far vectors in the input, their median edit distance will be equal to E^+, where $E^- < E^+$. Our strings will be constructed in three levels of increasing scope: coordinate level, vector level, and set level. We use $\text{EDIT}(x_1, x_2, \ldots, x_k)$ to denote the *median edit distance* of k strings $x_1, x_2, \ldots, x_k$.

- **Coordinate Level:** Given k bits $b_1, b_2, \ldots, b_k$, we construct *coordinate gadget* strings $\text{CG}_i(b_i)$ that can distinguish between the case when $b_1 b_2 \cdots b_k = 0$ and $b_1 b_2 \cdots b_k = 1$. Specifically, we will show that there exist constants C^- and C^+ with $C^- < C^+$ such that if $b_1 b_2 \cdots b_k = 0$, then $\text{EDIT}(\text{CG}_1(b_1), \text{CG}_2(b_2), \ldots, \text{CG}_k(b_k)) = C^-$, and else if $b_1 b_2 \cdots b_k = 1$, then $\text{EDIT}(\text{CG}_1(b_1), \text{CG}_2(b_2), \ldots, \text{CG}_k(b_k)) = C^+$.
- **Vector Level:** Given vectors $v_1, v_2, \ldots, v_k \in \{0,1\}^{d+r+1}$, we construct *vector gadget* strings $\text{VG}_i(v_i)$ for $i \in [2,k]$ and a slightly more complicated *decision gadget* string $\text{DG}_1(v_1)$ out of our coordinate gadgets. Together these gadgets can determine if the k vectors are r-far or not. Specifically, we will show that if $v_1, v_2, \ldots, v_k$ are r-far, then $\text{EDIT}(\text{DG}_1(v_1), \text{VG}_2(v_2), \ldots, \text{VG}_k(v_k)) \leq D^-$ and else if $v_1, v_2, \ldots, v_k$ are r-close, then $\text{EDIT}(\text{DG}_1(v_1), \text{VG}_2(v_2), \ldots, \text{VG}_k(v_k)) = D^+$, where D^+ and $D^- < D^+$ are constants. Our construction here is a generalization of the work in [10] to k strings.
- **Set Level:** In the set level step of the reduction, we will build our final strings $T_1, T_2, \ldots, T_k$ by concatenating our vector level gadgets and adding special $\$_i$ symbols. Our final strings will be designed so that if there is an r-far collection of vectors $v_1, v_2, \ldots, v_k$ with $v_i \in S_i$, then the corresponding gadgets $\text{DG}_1(v_1), \text{VG}_2(v_2), \text{VG}_3(v_3), \ldots, \text{VG}_k(v_k)$ will align in an optimal edit sequence of our strings. These vector gadgets will have a lower median edit distance, resulting in $\text{EDIT}(T_1, T_2, \ldots, T_k) \leq E^-$. Otherwise, $\text{EDIT}(T_1, T_2, \ldots, T_k) = E^+$, where $E^- < E^+$.

We now present a definition and an associated fact.

▶ **Definition 4** (Alignment). *Given a particular edit sequence (a sequence of insertions, substitutions, and deletions) on strings $x_1, x_2, \ldots, x_k$, we say symbol α in x_i is aligned with symbol β in another string x_j if neither α nor β is deleted but are instead preserved or substituted to correspond to the same symbol. We say a substring s of x_i is aligned with substring t of x_j, if there exists a pair of aligned characters in s and t.*

The following observation will be used implicitly throughout.

▶ **Fact 5** (No criss-crossed alignments). *Consider an edit sequence on a set of strings containing strings x and y. Let $i_1 < j_1$ and $i_2 < j_2$ be indices on these strings. If $x[i_1]$ is aligned with $y[j_2]$, then $x[i_2]$ cannot be aligned with $y[j_1]$.*

2.2 Coordinate level reduction

For $i \in [1, k]$, we define coordinate gadget strings CG_i over the alphabet $\Sigma = \{2_1, 2_2, \ldots, 2_k, 3, 4\}$. Let $\ell_1 = 10k^2$. For bits $b_1, b_2, \ldots, b_k \in \{0, 1\}$, we define

$$\mathrm{CG}_i(b_i) \coloneqq f_i(b_i) \circ 4^{\ell_1} \circ g_i(b_i) \circ 4^{\ell_1} \circ h_i(b_i) \quad \text{for } i \in [1, k], \text{ where}$$

$$f_i(b_i) = \begin{cases} 2_{i+1}^{k-1} & \text{if } b_i = 1, i < k \\ 2_1^{k-1} & \text{if } b_i = 1, i = k \\ 2_i^{k-1} & \text{if } b_i = 0 \end{cases} \quad g_i(b_i) = \begin{cases} 3^{k-1} & \text{if } b_i = 1 \\ 2_i^{k-1} & \text{if } b_i = 0 \end{cases} \quad h_i(b_i) = \begin{cases} 2_i^k & \text{if } b_i = 1 \\ \bigcirc_{j=1}^{k} 2_j & \text{if } b_i = 0 \end{cases}$$

We present the following examples on $k = 3$ to aid in the understanding of our $\mathrm{CG}_i(b_i)$.

b_1, b_2, b_3	$f_1(b_1), f_2(b_2), f_3(b_3)$	$g_1(b_1), g_2(b_2), g_3(b_3)$	$h_1(b_1), h_2(b_2), h_3(b_3)$	$\mathrm{EDIT}(\mathrm{CG}_1(b_1), \cdot, \cdot)$
$1, 1, 1$	$2_2 2_2, 2_3 2_3, 2_1 2_1$	$33, 33, 33$	$2_1 2_1 2_1, 2_2 2_2 2_2, 2_3 2_3 2_3$	$4 + 0 + 6 = 10$
$0, 1, 1$	$2_1 2_1, 2_3 2_3, 2_1 2_1$	$2_1 2_1, 33, 33$	$2_1 2_2 2_3, 2_2 2_2 2_2, 2_3 2_3 2_3$	$2 + 2 + 4 = 8$
$0, 0, 0$	$2_1 2_1, 2_2 2_2, 2_3 2_3$	$2_1 2_1, 2_2 2_2, 2_3 2_3$	$2_1 2_2 2_3, 2_1 2_2 2_3, 2_1 2_2 2_3$	$4 + 4 + 0 = 8$

▶ **Lemma 6.** *Let $C^- = 2(k-1)^2$ and let $C^+ = C^- + (k-1) = (2k-1)(k-1)$. Then*

$$\mathrm{EDIT}(\mathrm{CG}_1(b_1), \mathrm{CG}_2(b_2), \ldots, \mathrm{CG}_k(b_k)) = \begin{cases} C^+ & \text{if } b_1 b_2 \cdots b_k = 1 \\ C^- & \text{otherwise} \end{cases}$$

Proof. For the remainder of this proof, let $\pi = b_1 + b_2 + \cdots + b_k \in [0, k]$.

▷ Claim 7. The median edit distance of our f_i gadgets is

$$\mathrm{EDIT}(f_1(b_1), \ldots, f_k(b_k)) = \begin{cases} (k-1)^2 & \text{if } \pi = 0 \text{ or } k \\ (k-1)(k-2) & \text{otherwise} \end{cases}$$

▷ Claim 8. The median edit distance of our g_i gadgets is

$$\mathrm{EDIT}(g_1(b_1), \ldots, g_k(b_k)) = \begin{cases} (k-1)^2 & \text{if } \pi = 0 \\ (k-1)(k-\pi) & \text{otherwise} \end{cases}$$

▷ Claim 9. The median edit distance of our h_i gadgets is $\mathrm{EDIT}(h_1(b_1), \ldots, h_k(b_k)) = (k-1)\pi$.

We have chosen ℓ_1 to be sufficiently large that all f_i, g_i, and h_i gadgets align only with gadgets of their own type. Therefore,

$$\mathrm{EDIT}(\mathrm{CG}_1(b_1), \ldots, \mathrm{CG}_k(b_k)) = \begin{cases} (k-1)^2 + (k-1)^2 + 0 & \pi = 0 \\ (k-1)(k-2) + (k-1)(k-\pi) + (k-1)\pi & 0 < \pi < k \\ (k-1)^2 + 0 + (k-1)k & \pi = k \end{cases}$$

A simple calculation will show that $\mathrm{EDIT}(\mathrm{CG}_1(b_1), \ldots, \mathrm{CG}_k(b_k))$ is C^- when $\pi < k$ (and hence $b_1 b_2 \cdots b_k = 0$) and is C^+ when $\pi = k$ (and hence $b_1 b_2 \cdots b_k = 1$). ◀

2.3 Vector level reduction

At this step of the reduction we are given binary vectors $v_1, v_2, \ldots, v_k \in \{0,1\}^{d+r+1}$ and we want to determine whether or not they are r-far. We accomplish this by constructing vector level gadgets that will have a "lower" median edit distance if the vectors are r-far. Let integer parameters $\ell_2 = 10d\ell_1$ and $\ell_3 = (10\ell_2)^2$. For vectors $v_1, v_2, \ldots, v_k$, we define

$$\mathrm{VG}_i(v_i) \coloneqq 6^{\ell_3} \circ M_i(v_i) \circ 6^{\ell_3} \qquad \text{where} \qquad M_i(v_i) \qquad \coloneqq \bigcirc_{j\in[1,d+r+1]}(5^{\ell_2} \circ \mathrm{CG}_i(v_i[j]) \circ 5^{\ell_2})$$

Observe that the vector gadget of a vector v_i is just the concatenation of the coordinate gadgets corresponding to each coordinate in v_i, along with some additional padding symbols. It follows that the median edit distance of $\mathrm{VG}_1(v_1), \mathrm{VG}_2(v_2), \ldots, \mathrm{VG}_k(v_k)$ will be proportional to the inner product of $v_1, v_2, \ldots, v_k$. This is promising because we can now argue about whether or not $v_1, v_2, \ldots, v_k$ are r-far based on the median edit distance of the $\mathrm{VG}_i(v_i)$'s (a "lower" distance implies the vectors are r-far and a "higher" distance implies the vectors are r-close). Unfortunately, vectors with a very large inner product will result in a large median edit distance, which could interfere with our ability to detect r-far vectors in the next step of our reduction. What is desired here is to have vector level gadgets with a fixed "higher" median edit distance when the vectors are r-close. We achieve this by replacing $\mathrm{VG}_1(v_1)$ with a decision gadget $\mathrm{DG}_1(v_1)$ that will ensure that no matter how large the inner product of a collection of r-close vectors, the median edit distance of their corresponding gadgets will be a constant D^+. For vector v_1, we define

$$\mathrm{DG}_1(v_1) \coloneqq 7^{\ell_3} \circ M_1(v_1) \circ 6^{\ell_3} \circ M_1(\theta) \circ 7^{\ell_3}, \quad \theta \in \{0,1\}^{d+r+1} \text{ and } \theta[i] = \begin{cases} 1 & i \le r+1 \\ 0 & \text{else} \end{cases}$$

The key properties of our vector level gadgets are captured in Lemma 10 and Lemma 11. In both proofs we let $m = |M_i| = (d+r+1)(2\ell_2 + 2\ell_1 + 3k - 2)$, and we define $D^- = 2\ell_3 + m + (d+1)C^- + rC^+$ and $D^+ = D^- + (k-1)$.

▶ **Lemma 10.** *For any given r-far vectors $v_1, v_2, \ldots, v_k \in \{0,1\}^{d+r+1}$,* $\mathrm{EDIT}(\mathrm{DG}_1(v_1), \mathrm{VG}_2(v_2), \mathrm{VG}_3(v_3), \ldots, \mathrm{VG}_k(v_k)) \le D^-$.

Proof. To upper bound the median edit distance of our k strings by D^-, we must give a complete edit sequence of our strings that requires D^- or fewer edits. Let $v_1, v_2, \ldots, v_k$ be r-far vectors. We decide to align $\mathrm{VG}_2(v_2), \mathrm{VG}_3(v_3), \ldots, \mathrm{VG}_k(v_k)$ with the $7^{\ell_3} \circ M_1(v_1) \circ 6^{\ell_3}$ substring of $\mathrm{DG}_1(v_1)$ as in Figure 1.

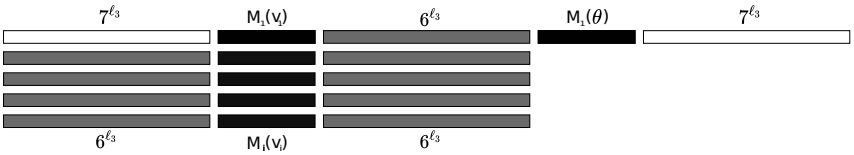


Figure 1 An optimal alignment of $\mathrm{DG}_1(v_1), \mathrm{VG}_2(v_2), \ldots, \mathrm{VG}_k(v_k)$ when $v_1, v_2, \ldots, v_k$ are r-far.

First we delete $M_1(\theta) \circ 7^{\ell_3}$ from $\mathrm{DG}_1(v_1)$ in $m + \ell_3$ edits. Then we substitute all the 7 symbols in the 7^{ℓ_3} prefix of $\mathrm{DG}_1(v_1)$ to 6 symbols in ℓ_3 edits. Finally, we must edit substrings $M_1(v_1), M_2(v_2), \ldots, M_k(v_k)$ to be the same. Each $M_i(v_i)$ contains $d+r+1$ coordinate gadgets, and for $j \in [1, d+r+1]$, we choose to align the jth leftmost coordinate gadgets of all $M_i(v_i)$ for $i \in [1,k]$. Note that the inner product of $v_1, v_2, \ldots, v_k$ is less than or equal to

r because the vectors are r-far. It follows that we will have no more than r alignments of coordinate gadgets with cost C^+ and at least $d+1$ alignments with cost C^- (recall Lemma 6). Then $\text{EDIT}(M_1(v_1), M_2(v_2), \ldots, M_k(v_k)) \leq (d+1)C^- + rC^+$. The total number of edits performed in this edit sequence is at most $2\ell_3 + m + (d+1)C^- + rC^+ = D^-$. ◀

We note that if $v_1, v_2, \ldots, v_k$ are r-close and as a result have an inner product greater than r, the optimal edit sequence of $\text{DG}_1(v_1), \text{VG}_2(v_2), \ldots, \text{VG}_k(v_k)$ will align strings $\text{VG}_2(v_2), \text{VG}_3(v_3), \ldots, \text{VG}_k(v_k)$ with the $6^{\ell_3} \circ M_1(\theta) \circ 7^{\ell_3}$ substring of $\text{DG}_1(v_1)$ as in Fig. 2.

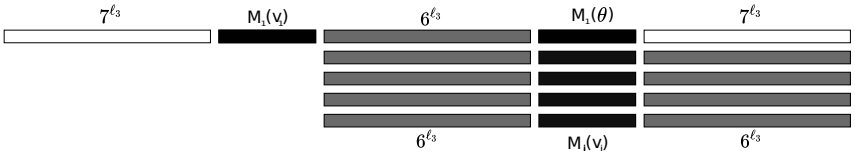


Figure 2 An optimal alignment of $\text{DG}_1(v_1), \text{VG}_2(v_2), \ldots, \text{VG}_k(v_k)$ when $v_1, v_2, \ldots, v_k$ are r-close.

▶ **Lemma 11.** *For any given r-close vectors $v_1, v_2, \ldots, v_k \in \{0,1\}^{d+r+1}$,* $\text{EDIT}(\text{DG}_1(v_1), \text{VG}_2(v_2), \text{VG}_3(v_3), \ldots, \text{VG}_k(v_k)) = D^+$.

Proof. The proof of Lemma 11 is a straightforward generalization of the vector gadget proof in [10] to k strings. In the course of this proof we will make use of the fact that for any subset $x_{i_1}, x_{i_2}, \ldots, x_{i_j}$ of strings $x_1, x_2, \ldots, x_k$, $\text{EDIT}(x_{i_1}, x_{i_2}, \ldots, x_{i_j}) \leq \text{EDIT}(x_1, x_2, \ldots, x_k)$.

▷ Claim 12. $\text{EDIT}(\text{DG}_1(v_1), \text{VG}_2(v_2), \text{VG}_3(v_3) \ldots, \text{VG}_k(v_k)) \leq D^+$

Subproof. Note that the inner product of $\theta, v_2, v_3, \ldots, v_k$ is equal to $r+1$ by the definition of θ and our modifications to the input vectors. Then we can align $\text{VG}_2(v_2), \text{VG}_3(v_3), \ldots, \text{VG}_k(v_k)$ with the $6^{\ell_3} \circ M_1(\theta) \circ 7^{\ell_3}$ substring of $\text{DG}_1(v_1)$ in a manner analogous to our edit sequence in Lemma 10. ◁

Now we "just" need to prove that $\text{EDIT}(\text{DG}_1(v_1), \text{VG}_2(v_2), \ldots, \text{VG}_k(v_k)) \geq D^+$. We proceed by cases on the alignments of the $M_i(v_i)$ substrings.

▷ Claim 13. $\text{EDIT}(\text{DG}_1(v_1), \text{VG}_2(v_2), \text{VG}_3(v_3) \ldots, \text{VG}_k(v_k)) \geq D^+$

Subproof. We have the following cases to consider.

- **Case 1:** The $M_i(v_i)$ substring of some $\text{VG}_i(v_i)$ gadget with $i > 1$ has alignments with both substrings $7^{\ell_3} \circ M_1(v_1)$ and $M_1(\theta) \circ 7^{\ell_3}$ of $\text{DG}_1(v_1)$. In this case, the cost induced by the symbols in the 7^{ℓ_3} prefix and suffix of $\text{DG}_1(v_1)$ and the 6^{ℓ_3} substring of $\text{DG}_1(v_1)$ is ℓ_3 each, so $\text{EDIT}(\text{VG}_i(v_i), \text{DG}_1(v_1)) \geq 3\ell_3 > D^+$. Our lower bound is satisfied. Note that since the inequality is strict, this case will not occur in an optimal edit sequence.
- **Case 2:** The $M_i(v_i)$ substring of some $\text{VG}_i(v_i)$ gadget with $i > 1$ does not have any alignments with the $7^{\ell_3} \circ M_1(v_1)$ substring of $\text{DG}_1(v_1)$.
- **Case 2.1:** The $M_j(v_j)$ substring of some $\text{VG}_j(v_j)$ gadget with $j > 1$ does not have any alignments with substring $M_1(\theta) \circ 7^{\ell_3}$ of $\text{DG}_1(v_1)$. We will consider $\text{EDIT}(\text{VG}_i(v_i), \text{VG}_j(v_j), \text{DG}_1(v_1))$, which is the same as $\text{EDIT}(\text{VG}_i(v_i), \text{DG}_1(v_1))$ when $i = j$. The $M_i(v_i)$ substring of $\text{VG}_i(v_i)$ has no alignments with the $7^{\ell_3} \circ M_1(v_1)$ substring of $\text{DG}_1(v_1)$. Therefore at least $D_1 = \ell_3 + m$ edits need to be performed between the 6^{ℓ_3} prefix of $\text{VG}_i(v_i)$ and the $7^{\ell_3} \circ M_1(v_1)$ prefix of $\text{VG}_1(v_1)$. Likewise, the $M_j(v_j)$ substring of $\text{VG}_j(v_j)$ has no alignments with the $M_1(\theta) \circ 7^{\ell_3}$ substring of $\text{DG}_1(v_1)$, and so at least D_1 edits need to be performed between the 6^{ℓ_3} suffix of $\text{VG}_j(v_j)$ and the $M_1(\theta) \circ 7^{\ell_3}$ suffix of $\text{DG}_1(v_1)$. The above edit costs are disjoint, and it follows that $\text{EDIT}(\text{VG}_i(v_i), \text{VG}_j(v_j), \text{DG}_1(v_1)) \geq 2D_1 > D^+$. Our lower bound is satisfied.

- **Case 2.2:** We consider the complement of Case 2.1: the $M_i(v_i)$ substrings of all $\mathrm{VG}_i(v_i)$ gadgets with $i > 1$ have alignments with the substring $M_1(\theta) \circ 7^{\ell_3}$ of $\mathrm{DG}_1(v_1)$. By our analysis in Case 1, we may now assume that the $M_i(v_i)$ substrings of all $\mathrm{VG}_i(v_i)$ gadgets with $i > 1$ do not have alignments with the $7^{\ell_3} \circ M_1(v_1)$ substring of $\mathrm{DG}_1(v_1)$. Then by our argument in Case 2.1, at least D_1 edits must be performed on the 6^{ℓ_3} prefix of $\mathrm{VG}_i(v_i)$ and the $7^{\ell_3} \circ M_1(v_1)$ prefix of $\mathrm{VG}_1(v_1)$. Additionally, note that all $\mathrm{VG}_i(v_i)$ share the suffix 6^{ℓ_3}, whereas $\mathrm{DG}_1(v_1)$ has suffix 7^{ℓ_3}. It follows that at least $D_2 = \ell_3$ edits are needed to edit $\mathrm{DG}_1(v_1), \mathrm{VG}_2(v_2), \ldots, \mathrm{VG}_k(v_k)$ to have the same suffix. Furthermore, these edits are disjoint from the D_1 edits performed on the prefixes of $\mathrm{DG}_1(v_1)$ and the $\mathrm{VG}_i(v_i)$. We have shown that at least $D_1 + D_2 = 2\ell_3 + m$ edits are required to align $\mathrm{DG}_1(v_1), \mathrm{VG}_2(v_2), \ldots, \mathrm{VG}_k(v_k)$. Now all we must do is lower bound the edits internal to our $M_i(v_i)$ substrings. Recall that our $M_i(v_i)$ substrings are composed of $d + r + 1$ coordinate gadgets $\mathrm{CG}_i(v_i[j])$.
  - **Case 2.2.1:** There is some $\mathrm{VG}_i(v_i)$ gadget with $i > 1$ such that there are some $j, \ell \in [1, d + r + 1]$ with $j \neq \ell$ such that the jth leftmost coordinate gadget of $M_i(v_i)$ is aligned with the ℓth leftmost coordinate gadget of the $M_1(\theta)$ in $\mathrm{VG}_1(v_1)$. Then we incur an edit cost of at least $2\ell_2$ from the 5 symbols between the coordinate gadgets. It follows that $\mathrm{EDIT}(\mathrm{DG}_1(v_1), \mathrm{VG}_2(v_2), \ldots, \mathrm{VG}_k(v_k)) \geq D_1 + D_2 + 2\ell_2 > D^+$. Our lower bound is satisfied.
  - **Case 2.2.2:** We now consider the complement of Case 2.2.1. For all $i \in [1, d + r + 1]$, the ith leftmost coordinate gadget of $M_j(v_j)$ for all $j > 1$ is either aligned with the ith leftmost coordinate gadget of $M_1(\theta)$ or it's not aligned with any coordinate gadget of $M_1(\theta)$.
    - For all $i \in [1, d + r + 1]$ we analyze the edit costs of the ith leftmost coordinate gadgets in $M_1(\theta), M_2(v_2), \ldots, M_k(v_k)$. If the ith leftmost coordinate gadgets of all $M_j(v_j)$ for $j > 1$ are aligned with the ith leftmost coordinate gadget of $M_1(\theta)$. Then by the transitivity of the alignment relation, we have that the ith coordinate gadgets of $M_1(\theta), M_2(v_2), \ldots, M_k(v_k)$ are aligned. By our analysis of the coordinate gadgets in Lemma 6, this alignment of coordinate gadgets will incur cost at least C^- if $\theta[i]v_2[i]v_3[i] \ldots v_k[i] = 0$, and else incur cost at least C^+ if $\theta[i]v_2[i]v_3[i] \ldots v_k[i] = 1$.
    - Else for some $M_j(v_j)$ with $j > 1$, the ith leftmost coordinate gadget $\mathrm{CG}_j(v_j[i])$ is not aligned with any coordinate gadget of $M_1(\theta)$, then it incurs cost $|\mathrm{CG}_j(v_j[i])| \geq C^+$.

    Combining our case analysis for all $d + r + 1$ coordinate gadgets, we see that they collectively incur a cost of at least $D_3 = (r + 1)C^+ + dC^-$, since the inner product of vectors $\theta, v_2, v_3, \ldots, v_k$ is $r + 1$ (follows from our modification of the input vectors and our definition of θ). Then $D_1 + D_2 + D_3 = D^+$, and since the edits from D_1, D_2, and D_3 are all necessarily disjoint, we have that $\mathrm{EDIT}(\mathrm{DG}_1(v_1), \mathrm{VG}_2(v_2), \ldots, \mathrm{VG}_k(v_k)) \geq D^+$.
- **Case 3:** The $M_i(v_i)$ substring of some $\mathrm{VG}_i(v_i)$ with $i > 1$ does not have alignments with the $M_1(\theta) \circ 7^{\ell_3}$ substring of $\mathrm{DG}_1(v_1)$. This case is symmetric to Case 2, with the only difference being that we have substring $M_1(v_1)$ as opposed to $M_1(\theta)$. Since we assumed that $v_1, v_2, \ldots, v_k$ are r-close and hence have an inner product greater than or equal to $r + 1$, it must be the case that $\mathrm{EDIT}(\mathrm{DG}_1(v_1), \mathrm{VG}_2(v_2), \ldots, \mathrm{VG}_k(v_k)) \geq D^+$.

We have shown in every case that $\mathrm{EDIT}(\mathrm{DG}_1(v_1), \mathrm{VG}_2(v_2), \ldots, \mathrm{VG}_k(v_k)) \geq D^+$, so we conclude that $\mathrm{EDIT}(\mathrm{DG}_1(v_1), \mathrm{VG}_2(v_2), \ldots, \mathrm{VG}_k(v_k)) = D^+$. ◁

This completes the proof of Lemma 11. ◀

2.4 Set level reduction

In this step of the reduction we will construct our final strings $T_1, T_2, \ldots, T_k$ that can detect r-far vectors in our input sets $S_1, S_2, \ldots, S_k$. We will accomplish this by embedding in string T_i the vector level gadgets of the vectors belonging to set S_i for $i \in [1, k]$. Then if an r-far collection of vectors exists, we can align their corresponding vector gadgets and give our strings $T_1, T_2, \ldots, T_k$ a "lower" median edit distance.

We will construct our final strings in several steps. We start by padding our vector level gadgets to discourage them from aligning with more than one vector level gadget in any given string. We define integer parameter $\ell_4 = 10000k^4 d\ell_3$, and we add a new padding symbol 8 to our alphabet. For all $v \in \{0,1\}^{d+r+1}$, let

$$\mathrm{DG}'_1(v) \coloneqq 8^{\ell_4} \circ \mathrm{DG}_1(v) \circ 8^{\ell_4} \quad \text{and} \quad \mathrm{VG}'_i(v) \coloneqq 8^{\ell_4} \circ \mathrm{VG}_i(v) \circ 8^{\ell_4} \quad \text{for } i \in [1, k]$$

We now concatenate our vector level gadgets DG'_1 and VG'_i. Define

$$P_1 \coloneqq \bigcirc_{v \in S_1} \mathrm{DG}'_1(v) \quad \text{and} \quad P_i \coloneqq \bigcirc_{v \in S_i} \mathrm{VG}'_i(v) \quad \text{for } i \in [2, k]$$

Strings $P_1, P_2, \ldots, P_k$ now contain all the vectors from our input sets. However, they are not sufficient to complete the reduction. To solve k-Most-Orthogonal-Vectors we must be able to check all n^k collections of vectors in $S_1 \times S_2 \times \cdots \times S_k$ for r-far-ness. Likewise, we must be able to align all n^k corresponding vector level gadgets in our final strings. In $P_1, P_2, \ldots, P_k$ this is not always possible without incurring a large additional edit cost. For example, there is no optimal edit sequence of $P_1, P_2, \ldots, P_k$ that aligns the leftmost vector level gadget of a string P_i with the rightmost vector level gadget of another string P_j – the number of insertions or deletions necessary would be too high.

Our strings $P_1, P_2, \ldots, P_k$ are rigid, but we can give them the freedom to slide around by making the lengths of all strings distinct. Specifically, we will add a varying number of vector level gadgets to each string so that P_{i+1} will have more vector level gadgets than P_i for all $i \in [1, k-1]$. We define the *dummy vector* ϕ to be a vector of all ones of length $d+r+1$. Let

$$L_1 \coloneqq \mathrm{VG}'_1(\phi)^{(50k+1)n} \circ \mathrm{DG}'_1(\phi)^{50kn} \quad \text{and} \quad R_1 \coloneqq \mathrm{DG}'_1(\phi)^{50kn} \circ \mathrm{VG}'_1(\phi)^{(50k+1)n}$$
$$L_i \coloneqq \mathrm{VG}'_i(\phi)^{(100k+i)n} \quad \text{and} \quad R_i \coloneqq \mathrm{VG}'_i(\phi)^{(100k+i)n} \quad \text{for } i \in [2, k]$$

Strings L_i and R_i will pad the left side and the right side of our P_i.

$$P'_i \coloneqq L_i \circ P_i \circ R_i \text{ for } i \in [1, k]$$

Observe that string P'_{i+1} has $2n$ more (dummy) vector level gadgets than P'_i for $i \in [1, k-1]$. This gives $P'_1, P'_2, \ldots, P'_k$ a pyramid-like shape as in Figure 3. We will see that this allows the sort of sliding between strings necessary to complete our reduction.

Figure 3 Final strings $T_1, T_2, \ldots, T_k$ when $k = 5$ shown from top to bottom. The vector gadgets corresponding to vectors from our input sets are shown in black, whereas the vector gadgets corresponding to dummy vectors ϕ are shown in gray. The special $\$_i$ symbols are shown in white.

However, because our strings $P'_1, P'_2, \ldots, P'_k$ are of different lengths, any complete edit sequence will require inserting or deleting vector level gadgets. This is problematic because it is difficult to reason about the edit costs of our vector level gadgets if they are inserted or

deleted in the optimal edit sequence. To solve this problem we add special $\$_i$ symbols to our strings. We will see that the $\$_i$ symbols "absorb" all the edits needed to make the lengths of the final strings equal and that no vector level gadgets will be inserted or deleted in the optimal edit sequence. We add $\$_1, \$_2, \ldots, \$_{k-1}$ to our alphabet, and we let $\ell_5 = 1000kn\ell_4$. Define

$$T_i \coloneqq \$_i^{\ell_5} \circ P'_i \circ \$_i^{\ell_5} \quad \text{for } i \in [1, k-1] \qquad \text{and} \qquad T_k \coloneqq P'_k$$

This completes the construction of our final strings $T_1, T_2, \ldots, T_k$. The length of each string as well as the time for their construction is $\mathcal{O}(nd^{O(1)})$. Their properties are summarized in Lemma 14 and Lemma 15 (proofs are deferred to Section 2.5 and Section 2.6, respectively).

▶ **Lemma 14.** *For any given sets $S_1, \ldots, S_k$ such that there is some collection $v_1, v_2, \ldots, v_k$ of r-far vectors with $v_i \in S_i$ for $i \in [1, k]$, $\text{EDIT}(T_1, T_2, \ldots, T_k) \leq E^-$, where $E^- = D^- + (100kn + n - 1)D^+ + 101k(k-1)(2k-1)(d+r+1)n + 2(k-1)\ell_5$.*

▶ **Lemma 15.** *For any given sets $S_1, S_2, \ldots, S_k$ such that there is no collection $v_1, v_2, \ldots, v_k$ of r-far vectors with $v_i \in S_i$ for $i \in [1, k]$, $\text{EDIT}(T_1, T_2, \ldots, T_k) = E^+$, where $E^+ = E^- + (k-1)$.*

▶ **Theorem 16.** *If there is an $\varepsilon > 0$, an integer $k \geq 2$, and an algorithm that can solve k-Median-Edit-Distance on strings, each of length at most n, over an alphabet of size $\mathcal{O}(k)$ in $O(n^{k-\varepsilon})$ time, then* SETH *is false.*

Proof. Follows from Lemma 14 and Lemma 15. ◀

2.5 Proof of Lemma 14

Statement: *For any given sets $S_1, S_2, \ldots, S_k$ such that there is some collection $v_1, v_2, \ldots, v_k$ of r-far vectors with $v_i \in S_i$ for $i \in [1, k]$, $\text{EDIT}(T_1, T_2, \ldots, T_k) \leq E^-$, where $E^- = D^- + (100kn + n - 1)D^+ + 101k(k-1)(2k-1)(d+r+1)n + 2(k-1)\ell_5$.*

To upper bound the median edit distance of $T_1, T_2, \ldots, T_k$ by E^-, we must give an edit sequence of at most E^- edits. Initially, we will only edit the substrings $P'_1, P'_2, \ldots, P'_k$ and thus exclude the $\$_i$ symbols from consideration. We start by aligning the vector level gadgets.

Vector Level Gadget Alignment. We have assumed vectors $v_1, v_2, \ldots, v_k$ are r-far, and we choose to align their corresponding vector level gadgets $\text{DG}_1(v_1), \text{VG}_2(v_2), \ldots, \text{VG}_k(v_k)$. We then align the rest of our vector level gadgets using the following rules:

1. Each vector level gadget in T_i aligns to exactly one vector level gadget in T_j for $j > i$.
2. If two vector level gadgets are adjacent in T_i, then they will be aligned to adjacent vector level gadgets in T_j for $j > i$.

Feasibility. We must demonstrate that this alignment is always achievable no matter how the vector level gadgets of $v_1, v_2, \ldots, v_k$ are embedded in strings $T_1, T_2, \ldots, T_k$. Recall that the vector level gadgets corresponding to vectors from our input sets are located in substrings P_i of T_i for all $i \in [1, k]$. Our construction gives paddings L_{i+1} and R_{i+1} exactly n more dummy vector level gadgets than L_i and R_i respectively for $i \in [1, k-1]$. It follows that even if the leftmost (resp. rightmost) vector level gadget in P_i is aligned with the rightmost (resp. leftmost) vector level gadget in P_{i+1}, the rules above remain satisfied.

Edit Cost for Vector Level Gadgets. There are $100kn + n$ decision gadgets DG_1 in T_1, so our edit sequence will yield $100kn + n$ alignments of $\mathrm{DG}_1, \mathrm{VG}_2, \ldots, \mathrm{VG}_k$, of which at least one such alignment will have cost D^- and the rest at most D^+. This gives an edit cost of at most $E_1^- = D^- + (100kn + n - 1)D^+$. At this point, all vector level gadgets in $P_1, P_2, \ldots, P_k$ have been edited (refer to Figure 4).

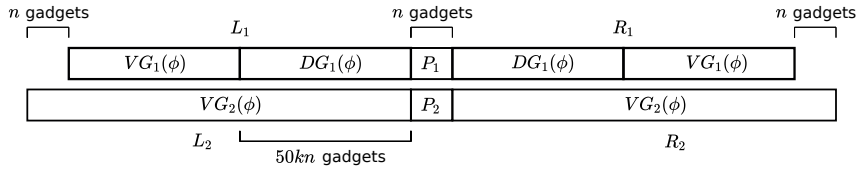


Figure 4 Strings P_1' and P_2'. All vector gadgets in P_2 align with decision gadgets DG_1 in P_1'.

Then there are exactly $2(50k + 1)n$ alignments of $\mathrm{VG}_1(\phi), \mathrm{VG}_2(\phi), \ldots, \mathrm{VG}_k(\phi)$ gadgets, and for all $i \in [2, k]$ there are exactly $2n$ alignments containing precisely the gadgets $\mathrm{VG}_i(\phi), \mathrm{VG}_{i+1}(\phi), \ldots, \mathrm{VG}_k(\phi)$. We will count the minimal number of edits needed to make these dummy vector gadgets identical. Let $F_i = (d + r + 1)(2k - 1)(k - i)$.

▷ Claim 17. For all $i \in [1, k]$, $\mathrm{EDIT}(\mathrm{VG}_i(\phi), \mathrm{VG}_{i+1}(\phi), \ldots, \mathrm{VG}_k(\phi)) = F_i$.

Proof. Each dummy vector gadget $\mathrm{VG}_j(\phi)$ is composed of $d + r + 1$ coordinate gadgets. Each alignment of the coordinate gadgets $\mathrm{CG}_i(1), \mathrm{CG}_{i+1}(1), \ldots, \mathrm{CG}_k(1)$ will incur $(2k - 1)(k - i)$ total edits, with $(k - 1)(k - i)$ edits from f gadgets and $k(k - i)$ edits from h gadgets. ◁

Denote the sum of the internal edit costs of all alignments of $\mathrm{VG}_i, \mathrm{VG}_{i+1}, \ldots, \mathrm{VG}_k$ gadgets for $i \in [1, k]$ by

$$E_2^- = 2(50k + 1)n \cdot F_1 + \sum_{i \in [2,k]} 2n \cdot F_i = 101k(k - 1)(2k - 1)(d + r + 1)n$$

This completes our edits on all vector level gadgets.

Total Edit Cost. All substrings $P_1', P_2', \ldots, P_k'$ have been edited to $P_1^*, P_2^*, \ldots, P_k^*$, respectively, so that P_i^* is a substring of P_j^* for all $i < j$. We will now edit the $\$_i$ symbols in order to complete the edit sequence of $T_1, T_2, \ldots, T_k$. In particular, we will edit all k strings to be equal to P_k^* by substituting and deleting $\$_i$ symbols. For the ith string, we will perform substitutions on $|P_k^*| - |P_i^*|$ of the $\$_i$ symbols in T_i and delete the remaining $\$_i$ symbols. Since we substitute or delete every $\$_i$ symbol, this will incur an edit cost of $E_3^- = 2(k - 1)\ell_5$. The total number of edits performed in our edit sequence is no more than $E_1^- + E_2^- + E_3^- = E^-$. This completes the proof.

2.6 Proof of Lemma 15

Statement: *For any given sets $S_1, S_2, \ldots, S_k$ such that there is no collection $v_1, v_2, \ldots, v_k$ of r-far vectors with $v_i \in S_i$ for $i \in [1, k]$,* $\mathrm{EDIT}(T_1, T_2, \ldots, T_k) = E^+ = E^- + (k - 1)$.

▷ Claim 18. $\mathrm{EDIT}(T_1, T_2, \ldots, T_k) \leq E^+$

Proof. We can achieve this upper bound by giving an edit sequence identical to the edit sequence in Lemma 14. Note that the only difference now is that there is no longer an r-far collection of vectors, so the edit cost of D^- in Lemma 14 is now D^+. This yields a complete edit sequence with $E^- + (D^+ - D^-) = E^+$ edits, so our inequality holds. ◁

We must now prove that $\text{EDIT}(T_1, T_2, \ldots, T_k) \geq E^+$. Our lower bound on the number of edits comes from two disjoint sources: the edits incurred by the $\$_i$ symbols and the edits incurred by alignments between vector level gadgets.

▷ Claim 19. The $\$_i$ symbols for $i \in [1, k-1]$ incur a cost of at least $E_1^+ = 2(k-1)\ell_5$ edits in a complete edit sequence of $T_1, T_2, \ldots, T_k$.

Proof. First note that symbols $\$_i$ for $i \in [1, k-1]$ have $E_1^+ = 2(k-1)\ell_5$ occurrences in $T_1, T_2, \ldots, T_k$. We will show that each of these $\$_i$ symbols incurs one edit and that this edit is disjoint from the edits of any other $\$_j$ symbol. If a $\$_i$ symbol is deleted or substituted, then it certainly incurs one edit. Furthermore, these deletions and substitutions are necessarily disjoint. Otherwise, suppose that a $\$_i$ symbol is not substituted or deleted, but remains unmodified in the edit sequence. Then because there are no $\$_i$ symbols in string T_k, we must incur at least one edit in T_k. This edit must be disjoint from any other edits incurred by other $\$_i$ symbols. ◁

Now we will reason about the lower bound on the edits incurred by vector level gadgets by considering every possible configuration of alignments between vector level gadgets. In order to do this, we define a graph G whose vertices correspond to vector level gadgets. More specifically, for the jth leftmost vector level gadget in T_i, we add a vertex x_i^j to G for $i \in [1, k]$. Thus vertices $x_i^1, x_i^2, \ldots, x_i^{(200k+2i+1)n}$ correspond to the $2(100k+i)n + n$ vector level gadgets in T_i from left to right. Now for a particular edit sequence, we define G to have an unordered edge $(x_{i_1}^{j_1}, x_{i_2}^{j_2})$ if the j_1th vector level gadget of T_{i_1} is aligned with the j_2th vector level gadget of T_{i_2} in the edit sequence. Also, we say that $x_{i_1}^{j_1}$ and $x_{i_2}^{j_2}$ are from the same row if $i_1 = i_2$.

Every edit sequence now corresponds to a graph G. This graph can be decomposed into a set of connected components $\mathcal{C}$. For a component $c \in \mathcal{C}$, we define $\#(c, i)$ as the number of vertices belonging to string T_i in c. We say that $\text{width}(c)$ of a component c is $\max_{i \in [1,k]} \#(c, i)$. We let $|c|$ denote the number of vertices in a component c. We now partition $\mathcal{C}$ into the following sets:

- $\mathcal{C}_1$ is the set of all components c with $\text{width}(c) > 1$
- $\mathcal{C}_2$ is the set of all components c with $\text{width}(c) = 1$ and $\#(c, k) = 0$
- $\mathcal{C}_3$ is the set of all components c with $\text{width}(c) = 1$ and $\#(c, k) = 1$

We now lower bound the edit costs of components in $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. Let $Q = 10kd\ell_3$.

▶ **Lemma 20.** *Every component c in $\mathcal{C}_1$ incurs at least $Q \cdot \text{width}(c)$ edits.*

Proof. Because our component c is connected, the case illustrated in Figure 5 must occur at least $\text{width}(c) - 1$ times. Then at least $2\ell_4(\text{width}(c) - 1)$ edits must be performed on the padding 8 symbols between the vector level gadgets of c. Observe that because $\ell_4 > Q$, this cost is greater than $Q \cdot \text{width}(c)$. These edits are disjoint from the edits of the $\$_i$ symbols. ◀

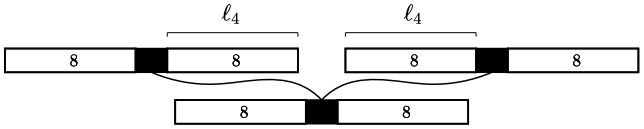


Figure 5 Case: one vector gadget in a string T_i is aligned with two vector gadgets in a string T_j. This alignment requires $2\ell_4$ edits of 8 symbols.

▶ **Lemma 21.** *Every component c in $\mathcal{C}_2$ incurs at least Q edits.*

Proof. By definition, the vector level gadgets in component c have no alignments with any vector level gadget VG_k in T_k. It follows that we incur a cost of at least $|\mathrm{VG}_k| > Q$. Furthermore, this edit cost is disjoint from the E_1^+ edit cost of our $\$_i$ symbols because there are no $\$_i$ symbols in T_k. ◀

We have given lower bounds for the edit costs of every component in $\mathcal{C}_1$ and $\mathcal{C}_2$, and these edit costs are disjoint by nature. Now we bound the costs of every component in $\mathcal{C}_3$. It will be useful to partition the components in $\mathcal{C}_3$ into the following sets:

- $\mathcal{C}_{3.1}$ is the set of all components c containing a vertex corresponding to a DG_1 gadget
- $\mathcal{C}_{3.2}$ is the remaining components in $\mathcal{C}_3$.

▶ **Lemma 22.** *All components c in $\mathcal{C}_{3.1}$ incur an edit cost of D^+.*

Proof. Our proof makes use of the following claim.

▷ Claim 23. No optimal edit sequence aligns a decision gadget DG_1 with any $\$_i$ symbol.

Subproof. Suppose some decision gadget DG_1 is aligned with a $\$_i$ symbol in string T_i for some $i \in [2, k-1]$. We will show that this incurs an edit cost greater than our upper bound E^+ established in Claim 18, implying this cannot occur in an optimal edit sequence. We may assume w.l.o.g. that DG_1 is aligned with a $\$_i$ symbol on the left side of T_i. It follows that the substring $\mathrm{VG}'_1(\phi)^{(50k+1)n}$ of T_1 must occur to the left of the alignment, and the substring P'_i of T_i must occur to the right of the alignment (see Figure 4). Then this alignment of T_1 and T_i has a combined length greater than or equal to $|\mathrm{VG}'_1(\phi)^{(50k+1)n}| + |P'_i|$. We observe that $|\mathrm{VG}'_1(\phi)^{(50k+1)n}| > 100kn\ell_4$ and $|P'_i| > 400kn\ell_4$, so our alignment of T_1 and T_i has a combined length greater than $500kn\ell_4$. On the other hand, $|T_k| = (202k+1)n|\mathrm{VG}'_k| < 203kn(3\ell_3 + 2\ell_4)$. Our alignment of T_1 and T_i must be edited to have the same length as T_k in every complete edit sequence, so it follows that $\mathrm{EDIT}(T_1, T_i, T_k) > 500kn\ell_4 - 203kn(3\ell_3 + 2\ell_4) = kn(94\ell_4 - 609\ell_3) > 1000k^4dn\ell_3$. Then our edit sequence requires $1000k^4dn\ell_3 + E_1^+ > E^+$ edits, so this alignment cannot occur in an optimal edit sequence. ◁

Let c be a component in $\mathcal{C}_{3.1}$. Suppose $\#(c, i) = 0$ for some $i \in [2, k-1]$. Then by definition, our gadgets in c have no alignments with any vector level gadget in T_i. It follows that we must perform at least $|\mathrm{VG}_i| > D^+$ edits among the vector gadgets in c. This is because the vector gadgets in c are either aligned with no symbols in T_i and therefore require at least $|\mathrm{VG}_i|$ insertions or deletions in c to make all strings the same length, or the vector gadgets in c are aligned exclusively with 8 symbols in T_i and therefore require at least $|\mathrm{VG}_i|$ substitutions to make them the same. Note that the vector gadgets in c cannot be aligned with any $\$_i$ symbols in T_i by Claim 23. This is key for proving that these edits are disjoint from the E_1^- cost of editing the $\$_i$ symbols.

Now consider the case that $\#(c, i) \neq 0$ for all $i \in [2, k-1]$. Then we have that $\#(c, i) = 1$ for all $i \in [1, k]$, and by our analysis in Lemma 14, the edit cost of aligning the k vector level gadgets is at least D^+. ◀

▶ **Lemma 24.** *Let c be a component in $\mathcal{C}_{3.2}$ and let $\lambda = |c|$, then the edit cost incurred by the vector gadgets in c is $(d + r + 1)(2k - 1)(\lambda - 1)$.*

Proof. Here we make use of the following claim, which has proof similar to Claim 23.

▷ Claim 25. Let $v_i \in S_i$ for some $i \in [2, k]$, then no optimal edit sequence aligns the vector gadget $\mathrm{VG}_i(v_i)$ in T_i with a $\$_1$ symbol in T_1, nor a dummy vector gadget $\mathrm{VG}_1(\phi)$ in T_1.

Subproof. Suppose some vector gadget $\mathrm{VG}_i(v_i)$ in string T_i with $i \in [2,k]$ and $v_i \in S_i$ is aligned with a dummy vector gadget $\mathrm{VG}_1(\theta)$ in string T_1. We will show that this incurs an edit cost greater than our upper bound E^+, implying this cannot occur in an optimal edit sequence. We may assume w.l.o.g. that $\mathrm{VG}_i(v_i)$ is aligned with a $\mathrm{VG}_1(\theta)$ gadget on the left side of T_1. It follows that the substring L_i of T_i must occur to the left of the alignment and the substring $\mathrm{DG}'_1(\phi)^{50kn} \circ P_1 \circ R_1$ of T_1 must occur to the right of the alignment. Then we can consider this alignment of T_i and T_1 to have a combined length greater than or equal to $|L_i| + |\mathrm{DG}'_1(\phi)^{50kn} \circ P_1 \circ R_1|$.

We observe that $|L_i| > 200kn\ell_4$ and $|\mathrm{DG}'_1(\phi)^{50kn} \circ P_1 \circ R_1| > 400kn\ell_4$, so our alignment of T_i and T_1 has a combined length greater than $600kn\ell_4$. On the other hand, $|T_k| = (202k+1)n|\mathrm{VG}'_k| < 203kn(3\ell_3 + 2\ell_4)$.

Our alignment of T_i and T_1 must be edited to have the same length as T_k in every complete edit sequence, so it follows that $\mathrm{EDIT}(T_1, T_i, T_k) > 600kn\ell_4 - 203kn(3\ell_3 + 2\ell_4) = kn(194\ell_4 - 609\ell_3) > 1000k^4dn\ell_3$. Then our edit sequence requires $1000k^4dn\ell_3 + E_1^+ > E^+$ edits, so this alignment cannot occur in an optimal edit sequence. It follows that $\mathrm{VG}_i(v_i)$ in T_i cannot align with a $\mathrm{VG}_1(\theta)$ gadget (and by extension a $\$_1$ symbol) in T_1. ◁

Let c be in $\mathcal{C}_{3.2}$. Suppose there is some $v_i \in S_i$ for $i \in [2,k]$ such that vector gadget $\mathrm{VG}_i(v_i)$ corresponds to a vertex in component c. Then the gadgets in our component cannot align with any decision gadgets DG_1, vector gadgets $\mathrm{VG}_1(\phi)$, or $\$_1$ symbols in T_1. It follows that we must perform at least $|\mathrm{VG}_i| > (d+r+1)(2k-1)(\lambda-1)$ insertions in T_i. Else, all vertices in component c correspond only to vector gadgets $\mathrm{VG}_i(\phi)$ for $i \in [1,k]$. By a similar argument as in Claim 17, the edit cost of component c is $(d+r+1)(2k-1)(\lambda-1)$. ◀

We have lower bounded the edit cost of all components in $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$. Now we must combine our component level arguments to obtain an overall lower bound on the edit cost. Let $W = \sum_{c \in \mathcal{C}_1 \cup \mathcal{C}_2} \mathrm{width}(c)$. Then we know that the components in $\mathcal{C}_1 \cup \mathcal{C}_2$ incur a cost of at least $E_2^+ = WQ$ edits by Lemma 20 and Lemma 21.

We now lower bound the total number of edits from components in $\mathcal{C}_3$. Note that components in $\mathcal{C}_{3.1}$ incur a much higher cost than components in $\mathcal{C}_{3.2}$. Then to lower bound the edits in $\mathcal{C}_3$, we must assume the least possible number of components in $\mathcal{C}_{3.1}$. There are $(100k+1)n$ decision gadgets DG_1 in our final strings and at most W decision gadgets in components in $\mathcal{C}_1 \cup \mathcal{C}_2$, so there must be at least $Z_1 = (100k+1)n - W$ components in $\mathcal{C}_{3.1}$. Note that if $W \geq (100k+1)n$, then $E_1^+ + E_2^+ \geq E^+$, so we may assume Z_1 is positive. Then components from $\mathcal{C}_{3.1}$ incur a cost of at least $E_3^+ = Z_1 D^+$ by Lemma 22.

There are at most $V_0 = kW$ vertices in components in $\mathcal{C}_1 \cup \mathcal{C}_2$, and there are at most $V_1 = kZ_1$ vertices in $\mathcal{C}_{3.1}$. Furthermore, there are $k(201k+2)n$ vertices in our graph G. It follows that there must be at least $V_2 = k(201k+2)n - V_1 - V_0 = k(101k+1)n$ vertices in all components in $\mathcal{C}_{3.2}$.

Because our edit cost lower bound for every component in $\mathcal{C}_{3.2}$ is linear in the component size, we have the following.

▷ Claim 26. Suppose there are Z components in $\mathcal{C}_{3.2}$ and a total of V vertices in all components in $\mathcal{C}_{3.2}$. Then the components in $\mathcal{C}_{3.2}$ incur $(d+r+1)(2k-1)(V-Z)$ edits.

Proof. By Lemma 24, each component of size λ in $\mathcal{C}_{3.2}$ incurs cost $(d+r+1)(2k-1)(\lambda-1)$. Let z_i denote the size of the ith component in $\mathcal{C}_{3.2}$ for $i \in [1,Z]$. Then we may sum the edit costs of all components in $\mathcal{C}_{3.2}$:

$$\sum_{i \in [1,Z]} (d+r+1)(2k-1)(z_i - 1) = (d+r+1)(2k-1)(V-Z)$$

where $z_i > 0$ for $i \in [1,Z]$ and $z_1 + z_2 + \cdots + z_Z = V$. ◁

Claim 26 proves that the edit cost of all the components in $\mathcal{C}_{3.2}$ decreases with the number of components Z. Then to achieve our lower bound we must upper bound the number of components in $\mathcal{C}_{3.2}$. There are exactly $(202k+1)n$ vector level gadgets in T_k, so there can be at most $Z_2 = (202k+1)n - Z_1$ components in $\mathcal{C}_{3.2}$. It follows that the total edit cost contributed by the components of $\mathcal{C}_{3.2}$ is at least $E_4^+ = (d+r+1)(2k-1)(V_2 - Z_2)$.

Then since the edit costs contributed by E_1^+, E_2^+, E_3^+, and E_4^+ are disjoint, we achieve a lower bound $\text{EDIT}(T_1, T_2, \ldots, T_k) \geq E_1^+ + E_2^+ + E_3^+ + E_4^+$. Straightforward calculation will show that $E_1^+ + E_2^+ + E_3^+ + E_4^+ \geq E^+$ for all $W > 0$. It follows that $\text{EDIT}(T_1, \ldots, T_k) = E^+$.

3 Reduction from k-Median-Edit-Distance to k-Center-Edit-Distance

We now provide a simple, yet previously unknown reduction from k-Median-Edit-Distance to k-Center-Edit-Distance. Given a set of strings $X = \{x_1, x_2, \ldots, x_k\}$, each of length at most n over an alphabet Σ, we define another set of strings $Y = \{y_1, y_2, \ldots, y_k\}$ over an alphabet $\Sigma' = \Sigma \cup \{\$\}$ (where $\$ \notin \Sigma$) as follows (fix $\ell = k^2 n$):

$$y_1 = x_1 \circ \$^\ell \circ x_2 \circ \$^\ell \circ \cdots \circ \$^\ell \circ x_{k-1} \circ \$^\ell \circ x_k$$
$$y_2 = x_2 \circ \$^\ell \circ x_3 \circ \$^\ell \circ \cdots \circ \$^\ell \circ x_k \circ \$^\ell \circ x_1$$
$$\vdots$$
$$y_k = x_k \circ \$^\ell \circ x_1 \circ \$^\ell \circ \cdots \circ \$^\ell \circ x_{k-2} \circ \$^\ell \circ x_{k-1}$$

Let $\text{CENTER-EDIT}(y_1, y_2, \ldots, y_k)$ denote the center edit distance of strings $y_1, y_2, \ldots, y_k$. We will prove the following, which will complete the reduction.

▶ **Lemma 27.** $\text{EDIT}(x_1, x_2, \ldots, x_k) = \text{CENTER-EDIT}(y_1, y_2, \ldots, y_k)$

Proof. Suppose that $\text{EDIT}(x_1, x_2, \ldots, x_k) = E$, and there is an optimal edit sequence on $x_1, x_2, \ldots, x_k$ that performs E_i edits on x_i for $i \in [1, k]$. It follows that $E_1 + E_2 + \cdots + E_k = E$.

▷ Claim 28. $\text{EDIT}(y_1, y_2, \ldots, y_k) = kE$

Subproof. It can be seen that $\text{EDIT}(y_1, y_2, \ldots, y_k) \leq kE$ since we may align all \$ symbols in the y_i in zero edits, and then we have k alignments of $x_1, x_2, \ldots, x_k$ substrings, each incurring E edits, for a total of kE edits.

Now note that no optimal edit sequence of $y_1, y_2, \ldots, y_k$ will delete an entire series of \$ symbols because it would incur cost ℓ greater than kE, our upper bound. It follows that for all $i \neq j$ the hth leftmost series of \$ symbols in y_i is aligned with the hth leftmost series of \$ symbols in y_j for $h \in \{1, \ldots, k-1\}$. Then the \$ alignments "lock" the x_i substrings into place so that we have k alignments of $x_1, x_2, \ldots, x_k$ substrings, and because no x_i contains the \$ symbol, it follows that each alignment of the x_i incurs cost greater than or equal to E. Then $\text{EDIT}(y_1, y_2, \ldots, y_k) \geq kE$. ◁

We now have that $\text{EDIT}(y_1, y_2, \ldots, y_k) = kE$. Furthermore, there is an optimal edit sequence that performs exactly E edits on every string in $y_1, y_2, \ldots, y_k$. This can be seen because in every alignment of substrings $x_1, x_2, \ldots, x_k$ in our edit sequence of $y_1, y_2, \ldots, y_k$, we may choose to perform E_i edits on each x_i. Then there exists an optimal edit sequence where for every string y_i with $i \in [1, k]$, we perform $E_i + E_{i+1} + \cdots + E_k + E_1 + E_2 + \cdots + E_{i-1} = E$ edits on y_i.

It follows that $\text{CENTER-EDIT}(y_1, y_2, \ldots, y_k) \leq E$. Furthermore, suppose that $\text{CENTER-EDIT}(y_1, y_2, \ldots, y_k) < E$. Then $\text{EDIT}(y_1, y_2, \ldots, y_k) < kE$, a contradiction. We conclude that $\text{CENTER-EDIT}(y_1, y_2, \ldots, y_k) = E$ and our reduction is complete. Note that for all $i \in [1, k]$, $|y_i| = (k-1)k^2 n + kn = \mathcal{O}(n)$. ◀

Lemma 27 directly implies the following result.

▶ **Theorem 29.** *If there is an $\varepsilon > 0$, a constant $k \geq 2$, and an algorithm that can solve k-Center-Edit-Distance on strings, each of length at most n, over an alphabet of size $\mathcal{O}(k)$ in $\mathcal{O}(n^{k-\varepsilon})$ time, then* SETH *is false.*

4 Discussion

Based on SETH, we have shown conditional hardness results for median string, center string, tree alignment, and bottleneck tree alignment problems, all under edit distance. These results suggest that the algorithms for the median string and tree-alignment problems are optimal (up to logarithmic factors). However, for the center string and bottleneck tree alignment problem, they leave an intriguing gap between the best known upper bounds. For center string (or the star instance of the bottleneck tree alignment) the best known dynamic programming algorithm works in time $O(n^{2k})$ [40], and as far as the authors know, no such algorithm exists for bottleneck tree alignment on more general trees. We conclude by asking: is an $O(n^k)$ algorithm waiting to be found for these problems, or does there exist a more efficient reduction which can prove that an $O(n^{2k-\varepsilon})$ algorithm is highly improbable?

References

1. Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. *ACM Trans. Algorithms*, 14(3):27:1–27:23, 2018. doi:10.1145/3093239.
2. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.
3. Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Seth-based lower bounds for subset sum and bicriteria path. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 41–57, 2019. doi:10.1137/1.9781611975482.3.
4. Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697. SIAM, 2015. doi:10.1137/1.9781611973730.112.
5. Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 375–388, 2016. doi:10.1145/2897518.2897653.
6. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014. doi:10.1109/FOCS.2014.53.
7. Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014. doi:10.1007/978-3-662-43948-7_4.

8 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. doi:10.1137/15M1050987.

9 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466, 2016. doi:10.1109/FOCS.2016.56.

10 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

11 Christina Boucher and Mohamed Omar. On the hardness of counting and sampling center strings. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 9(6):1843–1846, 2012. doi:10.1109/TCBB.2012.84.

12 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

13 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015. doi:10.1109/FOCS.2015.15.

14 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1216–1235, 2018. doi:10.1137/1.9781611975031.79.

15 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.

16 Yi-Jun Chang. Hardness of RNA folding problem with four symbols. *Theor. Comput. Sci.*, 757:11–26, 2019. doi:10.1016/j.tcs.2018.07.010.

17 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 21–40, 2019. doi:10.1137/1.9781611975482.2.

18 Yen Hung Chen and Chuan Yi Tang. On the bottleneck tree alignment problems. *Inf. Sci.*, 180(11):2134–2141, 2010. doi:10.1016/j.ins.2010.02.008.

19 Raphaël Clifford, Allan Grønlund, Kasper Green Larsen, and Tatiana Starikovskaya. Upper and lower bounds for dynamic data structures on strings. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.22.

20 Colin de la Higuera and Francisco Casacuberta. Topology of strings: Median string is np-complete. *Theor. Comput. Sci.*, 230(1-2):39–48, 2000. doi:10.1016/S0304-3975(97)00240-5.

21 Erik D. Demaine, Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Virginia Vassilevska Williams. Fine-grained I/O complexity via reductions: New lower bounds, faster algorithms, and a time hierarchy. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 34:1–34:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.34.

22 Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. *Algorithmica*, 81(10):3968–3992, 2019. doi:10.1007/s00453-018-0485-7.

23 Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of string matching for graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.55.

24 Isaac Goldstein, Moshe Lewenstein, and Ely Porat. On the hardness of set disjointness and set intersection with bounded universe. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 7:1–7:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ISAAC.2019.7.

25 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, 37(1):25–42, 2003. doi:10.1007/s00453-003-1028-3.

26 Franziska Hufsky, Léon Kuchenbecker, Katharina Jahn, Jens Stoye, and Sebastian Böcker. Swiftly computing center strings. In *Algorithms in Bioinformatics, 10th International Workshop, WABI 2010, Liverpool, UK, September 6-8, 2010. Proceedings*, pages 325–336, 2010. doi:10.1007/978-3-642-15294-8_27.

27 Franziska Hufsky, Léon Kuchenbecker, Katharina Jahn, Jens Stoye, and Sebastian Böcker. Swiftly computing center strings. *BMC Bioinformatics*, 12:106, 2011. doi:10.1186/1471-2105-12-106.

28 Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees - an alternative to tree edit. In Maxime Crochemore and Dan Gusfield, editors, *Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Asilomar, California, USA, June 5-8, 1994, Proceedings*, volume 807 of *Lecture Notes in Computer Science*, pages 75–86. Springer, 1994. doi:10.1007/3-540-58094-8_7.

29 Xiaoyi Jiang, Horst Bunke, and Janos Csirik. Median strings: A review. In *Data Mining in Time Series Databases*, pages 173–192. World Scientific, 2004.

30 Teuvo Kohonen. Median strings. *Pattern Recognition Letters*, 3(5):309–313, 1985. doi:10.1016/0167-8655(85)90061-3.

31 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287, 2016. doi:10.1137/1.9781611974331.ch89.

32 Robert Krauthgamer and Ohad Trabelsi. Conditional lower bounds for all-pairs max-flow. *ACM Trans. Algorithms*, 14(4):42:1–42:15, 2018. doi:10.1145/3212510.

33 Ferenc Kruzslicz. Improved greedy algorithm for computing approximate median strings. *Acta Cybern.*, 14(2):331–339, 1999. URL: http://www.inf.u-szeged.hu/actacybernetica/edb/vol14n2/Kruzslicz_1999_ActaCybernetica.xml.

34 J. Kevin Lanctôt, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Inf. Comput.*, 185(1):41–55, 2003. doi:10.1016/S0890-5401(03)00057-9.

35 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *J. ACM*, 49(2):157–171, 2002. doi:10.1145/506147.506150.

36 Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. *SIAM J. Comput.*, 39(4):1432–1443, 2009. doi:10.1137/080739069.

37 Hiromitsu Maji and Taisuke Izumi. Listing center strings under the edit distance metric. In *Combinatorial Optimization and Applications - 9th International Conference, COCOA 2015, Houston, TX, USA, December 18-20, 2015, Proceedings*, pages 771–782, 2015. doi:10.1007/978-3-319-26626-8_57.

38 Carlos D. Martínez-Hinarejos, Alfons Juan, Francisco Casacuberta, and Ramón Alberto Mollineda. Reducing the computational cost of computing approximated median strings. In Terry Caelli, Adnan Amin, Robert P. W. Duin, Mohamed S. Kamel, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops SSPR 2002 and SPR 2002, Windsor, Ontario, Canada, August 6-9, 2002, Proceedings*, volume 2396 of *Lecture Notes in Computer Science*, pages 47–55. Springer, 2002. doi:10.1007/3-540-70659-3_4.

39 François Nicolas and Eric Rivals. Hardness results for the center and median string problems under the weighted and unweighted edit distances. *J. Discrete Algorithms*, 3(2-4):390–415, 2005. doi:10.1016/j.jda.2004.08.015.

40 R. Ravi and John D. Kececioglu. Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree. *Discrete Applied Mathematics*, 88(1-3):355–366, 1998. doi:10.1016/S0166-218X(98)00079-1.

41 David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975.

42 Jeong Seop Sim and Kunsoo Park. The consensus string problem for a metric is np-complete. *J. Discrete Algorithms*, 1(1):111–117, 2003. doi:10.1016/S1570-8667(03)00011-X.

43 Andrés Varón and Ward C. Wheeler. The tree alignment problem. *BMC Bioinformatics*, 13:293, 2012. doi:10.1186/1471-2105-13-293.

44 T. K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968. Russian Kibernetika 4(1):81-88 (1968).

45 Lusheng Wang and Dan Gusfield. Improved approximation algorithms for tree alignment. *J. Algorithms*, 25(2):255–273, 1997. doi:10.1006/jagm.1997.0882.

46 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

Capacitated Sum-Of-Radii Clustering: An FPT Approximation

Tanmay Inamdar
Department of Computer Science, University of Iowa, Iowa City, IA, USA
tanmay-inamdar@uiowa.edu

Kasturi Varadarajan
Department of Computer Science, University of Iowa, Iowa City, IA, USA
kasturi-varadarajan@uiowa.edu

Abstract

In sum of radii clustering, the input consists of a finite set of points in a metric space. The problem asks to place a set of k balls centered at a subset of the points such that every point is covered by some ball, and the objective is to minimize the sum of radii of these balls. In the capacitated version of the problem, we want to assign each point to a ball containing it, such that no ball is assigned more than U points, where U denotes the *capacity* of the points. While constant approximations are known for the uncapacitated version of the problem, there is no work on the capacitated version. We make progress on this problem by obtaining a constant approximation using a Fixed Parameter Tractable (FPT) algorithm. In particular, the running time of the algorithm is of the form $2^{O(k^2)} \cdot n^{O(1)}$. As a warm-up for this result, we also give a constant approximation for the uncapacitated sum of radii clustering problem with matroid constraints, thus obtaining the first FPT approximation for this problem.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Sum-of-radii Clustering, Capacitated Clustering

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.62

Funding This research is partially supported by the National Science Foundation under Grant CCF-1615845.

1 Introduction

Clustering problems have received a great deal of attention in theoretical as well as practical research. Different ways of modeling a clustering problem have been proposed. A common way to model clustering problems is to assume that the data is represented as a set of points in a finite metric space, and the distance between a pair of points is a measure of similarity between the corresponding data points. Now, we want to partition the set of input points, such that the points belonging to each group are more similar to each other than the points outside the group. In the following, we focus on a particular set of three related clustering objective functions – k-center, k-median, and sum of radii clustering. We first describe the general setup.

Let P be a set of n input points in a metric space, and let d be the corresponding distance function. Let k denote the number of desired clusters, where k is a parameter of the problem. We want to find a set $C \subseteq P$ of at most k centers, such that a certain clustering objective function $\sigma(C, P)$ is minimized. In the k-center problem, the objective function is the largest distance of a point to its nearest center; whereas in the k-median problem, it is the sum of all such distances.

Now we describe the closely related sum of radii objective. For any $c \in P$, and $\rho \geq 0$, let $B(c, \rho) = \{p \in P : d(c, p) \leq \rho\}$ denote the ball of radius ρ centered at c. In the sum of radii objective, we want to additionally compute a *radius assignment* $r : C \to \mathbb{R}^+$, such that the corresponding set of balls $\mathcal{B} = \{B(c, r(c)) : c \in C\}$ *covers* the entire set of points P. The objective is to minimize the sum of radii of the balls, i.e., $\sum_{c \in C} r(c)$.

The sum of radii problem was studied by Charikar and Panigrahy [8], who gave a constant approximation. This result is obtained by first obtaining a constant approximation via the primal-dual technique to a closely related problem, which is the Lagrangian relaxation of the original sum of radii problem. Subsequently, Behsaz and Salavatipour [4] improved the approximation ratio in a restricted setting, and Gibson et al. [14] gave a $(1+\epsilon)$-approximation in quasi-polynomial time. In light of the latter result, the problem is likely not APX-hard, under standard complexity theoretic assumptions. There has also been significant work on certain generalizations of this problem. More general objective functions, such as the sum of α'th powers of the radii for a fixed $\alpha \geq 1$, and more general constraints, such as multi-covering of points, have been addressed, but these generalizations are outside the scope of this article.

Constant approximations are also known for the metric k-center and k-median problems, and unlike the sum of radii clustering problem, these problems are known to be APX-hard – see [15, 17, 5] and the references therein for these results. We now focus on a particular generalization of clustering problems which is the focus of this work.

1.1 Capacitated Clustering

A commonly considered generalization of clustering problems is the capacitated clustering, which models the situation where a center is able to provide a certain service to a specific number of points, which are sometimes referred to as *clients* in this context. In the uniform capacitated clustering problem, we are given an integer $1 \leq U \leq n$, which represents the *capacity* of any chosen center. Now, we also want to assign each point to a chosen center, such that no center is assigned more than U points. Here, we also require that, if a point p is assigned to a center c, then it is also covered by the ball placed at c. In a generalization called the non-uniform version of the problems, different centers may have different capacities.

The capacitated sum of radii problem has not yet been considered in the literature. Known techniques do not seem to extend to the capacitated sum of radii problem. Firstly, it can be easily shown that the standard Linear Programming (LP) relaxation has large integrality gap. Furthermore, it is not clear whether it is possible to strengthen this LP by imposing additional constraints implied by the problem structure, as done in [2] for capacitated facility location. Another piece of evidence is a hardness result from [3] for a closely related problem, which can be modified to rule out an $o(\log n)$-approximation in polynomial time, if we want to minimize the sum of α-th powers of radii, where $\alpha > 1$. While this hardness result does not rule out an $O(1)$ approximation in polynomial time for $\alpha = 1$, it does tell us that such a result would need to exploit rather special properties that hold in the $\alpha = 1$ case.

Before describing our results, we review the work on the capacitated versions of the k-center and k-median problems, and look at possible approaches that have been successful for these problems.

Related Work

The capacitated versions of the k-center and k-median problems can be defined analogously. These problems have received a great attention from the researchers. Constant approximations for the capacitated k-center are known [18], even for the non-uniform version. On the other hand, obtaining a constant approximation for the capacitated k-median problem has been a long-standing open problem in approximation algorithms. Researchers have explored different approaches for tackling this problem, in particular, relaxing some of the requirements of traditional approximation algorithms. One such relaxation allows for bi-criteria approximations – constant approximations have been obtained by violating the capacities by a small factor ([7, 10, 6, 13]), or by violating the k-constraint (the number of medians) by a constant factor ([20, 21]).

Yet another recent approach relaxes the requirement that the algorithm run in polynomial time. An algorithm with the running time of $f(p) \cdot n^{O(1)}$ is known as a Fixed Parameter Tractable (FPT) algorithm, where p is a parameter of the problem. Note that, although the running time may depend exponentially (or worse) on the parameter p, the dependence on n, the input size, is strictly polynomial. In the context of capacitated clustering, such an FPT algorithm parameterized by k, the number of clusters, may be acceptable, if k is a small constant. Adamczyk et al. [1] give a $(7+\epsilon)$-approximation for the (uniform/non-uniform) capacitated k-median problem in $k^{O(k)} \cdot n^{O(1)}$ time. They use an approximate solution for the *uncapacitated* k-median problem to convert the given instance into a simpler instance that has more structure, at an expense of a small constant factor loss in the approximation guarantee. Then, they obtain a near-optimal solution for this simpler instance in FPT time. Cohen-Addad and Li [11] improved the approximation guarantee to $3+\epsilon$, again using a similar FPT running time. Their algorithm is based on a coreset construction, and they obtain a constant approximation for this smaller coreset in FPT time. They also obtain an $(1+\epsilon)$-approximation in Euclidean metrics. These results are complemented by Adamczyk et al. [1], who observe that one cannot hope to obtain an exact FPT algorithm even for the *uncapacitated* k-median problem. Parameterized algorithms and complexity is a large and active domain of research, and we direct the reader to a textbook such as [12] for a more detailed background.

1.2 Our Results and Techniques

Our main result is a 28-approximation for the uniform capacitated sum of radii problem, that runs in $2^{O(k^2)} \cdot n^{O(1)}$ time. This result is in a similar vein as the aforementioned results ([1, 11]) for the capacitated k-median problem. Adapting techniques they develop for capacitated k-median, we can obtain an FPT approximation for capacitated sum of radii in metrics of constant doubling dimension. However, for general metrics, we have not been able to adapt their approach. Therefore, we develop a novel algorithm, which we discuss at a high level below.

Fix an optimal solution to the problem. First we discretize the optimal solution by rounding the radii up to a power of $1+\epsilon$ for a fixed $\epsilon > 0$, and now suppose that this discretized solution exactly k_i balls of radius r_i, where each r_i is a power of $(1+\epsilon)$. We show that this first step can be implemented in FPT time. Therefore, we can assume henceforth that we know the "radius profile" of the optimal solution. Our main algorithm proceeds in multiple *levels*. Roughly speaking, the goal of the algorithm at a particular level i is to guess the approximate locations of the k_i optimal balls of radius r_i. However, the size of the search space for this guessing is too large to ultimately obtain an FPT algorithm.

Therefore, we employ a certain *greedy* strategy to guess some balls not chosen in the optimal solution. This allows us to bound the size of the search space, while simultaneously allowing us to argue that an appropriate capacity reassignment is possible if our algorithm misses the approximate location of an optimal ball.

Sum of Radii with a Matroid Constraint

Although the high-level idea of our algorithm is relatively simple, the technical arguments to establish that such a capacity reassignment is possible are quite sophisticated and involved. Therefore, we first consider a related, but simpler, problem as a warm-up. The natural candidate is the uncapacitated version (for which constant approximations are known in polynomial time [8]), but we consider a more general version, which replaces the k-constraint by a more general matroid constraint.

A matroid $\mathcal{M}$, on the set of given points P, is the pair $(P, \mathcal{I})$, where $\mathcal{I}$ is a collection of subsets of P with the following properties: (i) $A \in \mathcal{I}$ implies that $\forall B \subseteq A$, $B \in \mathcal{I}$, (ii) If $A, B \in \mathcal{I}$ with $|B| < |A|$, then there exists a $p \in A \setminus B$, such that $B \cup \{p\} \in \mathcal{I}$. If a set C belongs to $\mathcal{I}$, then C is said to be *independent* in the corresponding matroid $\mathcal{M}$. One consequence of this definition is that all inclusion-wise maximal independent sets of a matroid $\mathcal{M}$ have equal size, and they are called the bases of $\mathcal{M}$.

In the Matroid Sum of Radii problem, a feasible solution consists of a set of centers C, and a radius assignment $r : C \to \mathbb{R}^+$ such that the resulting set of balls covers P. Furthermore, we also require that C be an independent set according to a given matroid $\mathcal{M}$. The objective of the problem is to minimize the sum of radii. We assume that we have an oracle access to an algorithm $\mathcal{A}_{\mathcal{M}}$ that answers in polynomial time, whether a candidate set of centers is independent in the matroid $\mathcal{M}$. For many "natural" matroids, the definition of an independent set is simple, and thus the oracle can be simulated in a straightforward manner.

We give a $(9+\epsilon)$-approximation for this problem in $b^{O(b)} \cdot n^{O(1)}$ time, where b is the size of a basis of the matroid $\mathcal{M}$. At a high level, our strategy is similar to our main result for the capacitated sum of radii problem. However, unlike the capacitated version, here we can always find the approximate locations of the optimal centers, which simplifies the algorithm and its analysis. Although this result is not the main contribution of our work, it provides a good vantage point to understand our result for capacitated sum-of-radii clustering.

We note that constant approximations are known for the matroid versions of k-center and k-median [9, 19]. These problems were originally motivated from the so-called red-blue median problem [16], where the centers come in one of the two *types*: red and blue, and we are required to select at most k_r red centers and k_b blue centers to minimize the k-median objective. The matroid formalization captures this scenario as well as its generalization for arbitrary number of types. In particular, the special case of one color corresponds to the k-constraint in the uncapacitated setting.

2 Sum of Radii with a Matroid Constraint

In this problem, we are given a finite metric space (P, d), and a matroid $\mathcal{M} = (P, \mathcal{I})$ on the set of points P. We want to place a set of balls to cover the points in P, while minimizing the sum of radii of the balls. Furthermore, it is required that the set of centers is an independent set in the given matroid.

Formally, we want to find a set of centers that forms an independent set, i.e., $C \in \mathcal{I}$ and assign radii $r : C \to \mathbb{R}^+$, such that $P \subseteq \bigcup_{c \in C} B(c, r(c))$. The objective is to minimize $\sum_{c \in C} r_c$, over all such feasible solutions.

Fix an optimal solution (C^*, r^*), and let $k = |C^*|$. Note that C^* is an independent set in $\mathcal{M}$. First, we guess the value of k by iteratively trying $k' = 1, 2, \ldots, b$, and returning the solution with smallest cost. Here, b denotes the size of any base in $\mathcal{M}$. For a particular value k', the algorithm runs in $k'^{O(k')} \cdot n^{O(1)}$ time. Note that the overall running time of this algorithm is $\sum_{k'=1}^{b} k'^{O(k')} \cdot n^{O(1)} = b^{O(b)} n^{O(1)}$. From now on, we will focus on the iteration where $k' = k$.

Before discussing our main algorithm, we discuss K-Center(U, r), which is an important subroutine. Here, $U \subseteq P$ is a set of points to be covered, and $r \geq 0$ denotes the target radius. It is a simple iterative procedure that selects an as-yet uncovered point p, and marks all points in its $2r$-neighborhood as covered. It also adds p to the set of centers Q, and this iterative procedure continues until all points in U are marked as covered. This is a well-known 2-approximation algorithm for the k-center problem, and is summarized in the following lemma.

▶ **Lemma 1.** *Let $U \subseteq P$ be a subset of points, and suppose there exists a set $C \subseteq P$ of k centers such that $\max_{u \in U} d(u, C) \leq r$. Then, the set of centers Q returned by* K-Center(U, r') *for any value $r' \geq r$ satisfies: (i) $|Q| \leq k$, and (ii) $\max_{u \in U} d(u, Q) \leq 2r'$.*

Algorithm 1 K-Center(U, r).

```
Initially, all points in U are marked as uncovered, Q ← ∅
while there exists an uncovered point in U do
    Let p ∈ U be an uncovered point, add p to Q
    Mark all points in B(p, 2r) as covered
return Q
```

Preprocessing

First, we discretize the possible choices of radii in the following way. Let $\epsilon > 0$ be a constant, and let R denote the smallest power of $1 + \epsilon$ larger than the maximum radius of any optimal ball – note that we can "guess" the value of R in polynomial time. Furthermore, for any ball with radius smaller than $\frac{\epsilon R}{k}$, we round its radius up to $\frac{\epsilon R}{k}$, and the total increase over at most k such balls is at most ϵR, which is at most ϵ times the optimal cost. Now, we round up radii of all balls to the next larger power of $(1 + \epsilon)$. Note that the resulting solution is within a factor of $(1 + \epsilon)^2$ from the cost of the optimal solution. Furthermore, there are $t = \log_{1+\epsilon} \frac{R}{\frac{R\epsilon}{k}} = O(\frac{1}{\epsilon} \log \frac{k}{\epsilon})$ distinct values of radii. Since ϵ is fixed, $t = O(\log k)$. From now onwards, we will slightly abuse the terminology, and use the terms "optimal solution", "optimal ball" etc. to refer to the corresponding entities in the optimal solution modified in this manner.

Define $r_1 = R, r_2 = \frac{R}{1+\epsilon}, \ldots, r_t = \frac{R}{(1+\epsilon)^{t-1}}$, where $r_{t+1} < \frac{R\epsilon}{k} \leq r_t$. Suppose for every $1 \leq i \leq t$, the optimal solution uses exactly $0 \leq k_i \leq k$ balls of radius r_i. Note that $\sum_{i=1}^{t} k_i = k$. Let $\mathcal{O} = \bigcup_{i=1}^{t} \mathcal{O}_i$ be the set of balls in the optimal solution, where $\mathcal{O}_i \subseteq \mathcal{O}_i$ is the subset of balls of radius r_i. Let $C^* \subseteq P$ denote the set of centers, and let $C^*_i \subseteq C^*$ denote the set of centers of the balls in $\mathcal{O}_i$. For a particular value of i, we define $\mathcal{O}_{<i} = \bigcup_{j=1}^{i-1} \mathcal{O}_j$, and the subsets $\mathcal{O}_{\leq i}, \mathcal{O}_{>i}, \mathcal{O}_{\geq i}$ (resp. $C^*_{\leq i}$ etc.) are defined similarly.

First, we guess the "radius profile" of the optimal solution. There are $O(\log k)$ classes of radii, and for each class r_i, $0 \leq k_i \leq k$. Therefore, the number of overall choices for the radius profile can be upper bounded by $k^{O(\log k)} \ll k^{O(k)}$.

Algorithm 2 MatroidSoR$(\mathcal{B}, i)$.

▷ $1 \le i \le t+1$ is the current level – we want to guess at most k_i centers for balls of radius $4r_i$
▷ $\mathcal{B}$ is the set of balls fixed at earlier levels 1 through $i-1$

$U_i \leftarrow P \setminus \left(\bigcup_{B\in\mathcal{B}} B\right)$ ▷ *Set of points not covered by balls in* $\mathcal{B}$
if $i = t+1$ and $U_i = \emptyset$ **then** ▷ *All points are covered at level* $\le t$
  $\mathcal{D} \leftarrow$ Disjointify$(\mathcal{B})$ ▷ *Procedure* Disjointify *is described before Observation 3 in text*
  for every non-empty subset $\mathcal{D}' \subseteq \mathcal{D}$ **do**
    if balls returned by MatroidIndependentSet$(\mathcal{D}', \mathcal{D})$ cover all points **then**
      output MatroidIndependentSet$(\mathcal{D}', \mathcal{D})$ and **halt**
else if $i = t+1$ and $U_i \neq \emptyset$ **then** ▷ *Not all points are covered by balls at level* $\le t$
  return ▷ $\mathcal{B}$ *is a wrong guess*
$P_i \leftarrow$ K-Center(U_i, r_i) ▷ P_i *is the potential set of centers at level* i
if $|P_i| > k$ **then** ▷ U_i *cannot be covered by at most* k *balls of radius* r_i
  return ▷ $\mathcal{B}$ *is a wrong guess*
else
  For every $C_i \subseteq P_i$ of size at most k_i, call MatroidSoR$(\mathcal{B} \cup \mathcal{B}(C_i), i+1)$
  ▷ $\mathcal{B}(C_i) \coloneqq \{B(c, 4r_i) : c \in C_i\}$

Algorithm

Having guessed the radius profile $(k_1, k_2, \ldots, k_t)$, our algorithm invokes MatroidSoR$(\emptyset, 1)$ (see Algorithm 2). The procedure MatroidSoR$(\mathcal{B}, i)$ is recursive, and proceeds in multiple *levels*. Fix $1 \le i \le t$, which denotes the current level. We are given a set of balls $\mathcal{B}$ selected at *higher* levels, i.e., levels 1 through $i-1$. For $1 \le j \le i-1$, we let C_j denote the set of centers of balls in $\mathcal{B}$ of level j. We know that $|C_j| \le k_j$, and each $c \in C_j$ has a ball of radius $4r_j$ around it. Now, we want to find a set of at most k_i centers to place balls of radius $4r_i$ at this level.

Let us now see how the algorithm MatroidSoR$(\mathcal{B}, i)$ places these balls at level i. We find U_i, the set of points not covered by any ball in $\mathcal{B}$. We then use algorithm K-Center(U_i, r_i) to find a solution to cover the points in U_i using balls of radius $2r_i$. We will later prove in Lemma 2 that if the set of balls $\mathcal{B}$ added to the solution so far is "correct" (formalized in the Lemma), then the solution P_i returned by the K-center algorithm contains at most k centers. Therefore, if $|P_i| > k$, we conclude that the set of balls $\mathcal{B}$ added to the solution so far is incorrect, and we stop.

Now, suppose $|P_i| \le k$. Then, we enumerate every subset $C_i \subseteq P_i$ of size at most k_i, and recurse on each subset. Note that the number of subsets can be upper bounded by $\sum_{i=0}^{k_i} \binom{k}{k_i} \le k^{O(k_i)}$. Assuming the set $\mathcal{B}$ is "correct", one of these $k^{O(k_i)}$ recursive calls is also "correct". Now we formalize this notion in the following Lemma.

▶ **Lemma 2.** *At any level $1 \le i \le t$, in one of the recursive calls to* MatroidSoR$(\mathcal{B}, i)$*, for any optimal center $c_j^* \in C_j^*$ with $1 \le j < i$, one of the following holds:*

1. *There exists $c \in C_\ell$ with $\ell \le j$, and $B(c_j^*, r_j) \subseteq B(c, 4r_\ell)$, OR*
2. *$B(c_j^*, r_j)$ is completely covered by balls in $\mathcal{B}$ of level 1 through $j-1$. In this case, there exists a center $c \in C_\ell$ with $\ell < j$, such that $d(c_j^*, c) \le 4r_\ell$.*

Proof. We prove this claim inductively.

Base case. This corresponds to $i = 2$. We want to show that, there exists a set $\mathcal{B}$ of balls chosen at level 1, such that, at the start of the algorithm MatroidSoR$(\mathcal{B}, 2)$, every ball $B(c_1^*, r_1) \in \mathcal{O}_1$ is contained in some ball in $\mathcal{B}$.

For the base case, consider the situation at the start of the algorithm, after we invoke MatroidSoR$(\emptyset, 1)$. Note that $U_1 = P$. Note that the optimal solution covers P using k balls of radius at most r_1. Consider the set P_1 of points returned by K-Center(U_1, r_1). Using Lemma 1, we have that $|P_1| \leq k$, and for any $c_1^* \in C_1^*$, there is some $\varphi(c_1^*) := c \in P_1$ with $d(c_1^*, c) \leq 2r_1$. Let $C_1 = \{\varphi(c_1^*) \mid c_1^* \in C_1^*\}$. Clearly, $|C_1| \leq |C_1^*| = k_1$, and for any $c_1^* \in C_1^*$, $B(c_1^*, r_1) \subseteq B(\varphi(c_1^*), 4r_1)$. Thus, the recursive call MatroidSoR$(\mathcal{B}(C_1), 2)$ satisfies the required properties.

Inductive hypothesis. Now we assume that the claim holds inductively at the start of iteration i, and prove that it also holds at level $i+1$ in one of the recursive calls. That is, fix a recursive call MatroidSoR$(\mathcal{B}, i)$, where $\mathcal{B}$ is a set of balls chosen at levels 1 through $i-1$, such that, any ball $B^* \in \mathcal{O}_{<i}$ is covered by a ball in $\mathcal{B}$, as guaranteed by the induction hypothesis. Now, let U_i^* be the set of points not covered by any such optimal ball (from $\mathcal{O}_{<i}$). Note that inductive hypothesis implies that $U_i \subseteq U_i^*$, which implies that U_i can be covered using at most k balls of radius at most r_i. Now, Lemma 1 implies that the set P_i of points returned by K-Center(U_i, r_i) has size at most k.

We will define a mapping $\varphi : C_i^* \to P_i \cup \{\perp\}$ that specifies a center in P_i whose ball covers $B(c_i^*, r_i)$, if any. Now, consider an optimal center $c_i^* \in C_i^*$, that has a ball $B^* = B(c_i^*, r_i)$ centered at it. We consider two different cases.

Case 1. If there exists a point $p \in B^* \cap U_i$, then using Lemma 1, there exists a center $c \in P_i$ returned by K-Center(U_i, r_i), such that $d(c, p) \leq 2r_i$. Since $d(c_i^*, c) \leq d(c_i^*, p) + d(p, c) \leq 3r_i$, $B(c_i^*, r_i) \subseteq B(c, 4r_i)$. In this case, we define $\varphi(c_i^*) = c$.

Case 2. Otherwise, $B^* \cap U_i = \emptyset$, which implies that all points in B^* are covered by the balls in $\mathcal{B}$ of levels 1 through $i-1$. In particular c_i^* is also covered by a ball $B(c, 4r_\ell)$, where $\ell < i$. In this case, we set $\varphi(c_i^*) = \perp$.

Note that the two cases correspond to the two criteria in the statement of the lemma. Furthermore, if $\varphi(c_i^*) = \perp$, then $B(c_i^*, r_i)$ is covered by one or more balls in $\mathcal{B}$ of levels 1 through $i-1$, i.e., we do not require a ball at level i to cover this ball. Otherwise, $\varphi(c_i^*) \in P_i$, and $B(c_i^*, r_i) \subseteq B(\varphi(c_i^*), 4r_i)$. Let $C_i := \{c \in P_i : \varphi^{-1}(c) \neq \emptyset\}$. Since $|C_i^*| = k_i$, $|C_i| \leq k_i$, and the recursive call corresponding to MatroidSoR$(\mathcal{B} \cup \mathcal{B}(C_i), i+1)$ satisfies the required properties, recalling that $\mathcal{B}(C_i) := \{B(c, 4r_i) : c \in C_i\}$. ◀

Now, let us discuss the algorithm at level $i = t+1$. Note that Lemma 2 implies that, all points must be covered at level $t+1$ in one of the recursive calls. Therefore, if $U_{t+1} \neq \emptyset$, then we conclude that the set of balls $\mathcal{B}$ is incorrect.

Henceforth, let $\mathcal{B}$ denote the set of balls at level $t+1$ guaranteed by Lemma 2, and focus on the call MatroidSoR$(\mathcal{B}, t+1)$. Note that $\mathcal{B}$ covers all points, which implies that U_{t+1} is empty. We now call the procedure Disjointify$(\mathcal{B})$, which we describe now. In this procedure, we assign each point in P to the largest ball in $\mathcal{B}$ that covers it, breaking ties arbitrarily. Let $D_j(c)$ denote the set of points assigned to a particular ball $B(c, 4r_j) \in \mathcal{B}$. Note that $D_j(c) \subseteq B(c, 4r_j)$; however the inclusion may be strict. In particular, it may be the case that $c \notin D_j(c)$, or $D_j(c)$ may even be empty.

Let $\mathcal{D}$ be the resulting collection of sets in $\mathcal{B}$ that are made disjoint in this manner. In order to distinguish the resulting disjoint sets from the original set of balls, we refer to them as *clusters*. The following observations follow from the definition of $\mathcal{B}$ and the description of Disjointify.

▶ **Observation 3.**

1. *The clusters in $\mathcal{D}$ partition P.*
2. *If an optimal center $c_i^* \in C_i^*$ is contained in a cluster $D_\ell(c) \in \mathcal{D}$, then $\ell \leq i$.*

Next, for every non-empty subset $\mathcal{D}' \subseteq \mathcal{D}$, we call MatroidIndependentSet$(\mathcal{D}', \mathcal{D})$. In this algorithm, we define a matroid $\mathcal{M}(\mathcal{D}', \mathcal{D})$ – see Algorithm 3 for the definition. It is easy to see that for any $\mathcal{D}' \subseteq \mathcal{D}$, that $\mathcal{M}(\mathcal{D}', \mathcal{D})$ is a (partition) matroid on P. We then find a common maximum independent set C in both matroids $\mathcal{M}$ and $\mathcal{M}(\mathcal{D}', \mathcal{D})$. Then, if $c \in C$ is contained in a cluster of level i in $\mathcal{D}$, then we place a ball of radius $9r_i$ around it. Next, we prove that, for at least one subset $\mathcal{D}' \subseteq \mathcal{D}$, the algorithm MatroidIndependentSet$(\mathcal{D}', \mathcal{D})$ finds a set of balls that covers the entire point set P.

Algorithm 3 MatroidIndependentSet$(\mathcal{D}', \mathcal{D})$.

1: Define a new matroid $\mathcal{M}(\mathcal{D}', \mathcal{D}) = (P, \mathcal{I}(\mathcal{D}'))$, where a set $C \subseteq P$ is independent in $\mathcal{I}(\mathcal{D}')$ iff it contains at most one point from each cluster in $\mathcal{D}'$, and no points from cluster in $\mathcal{D} \setminus \mathcal{D}'$
2: The weight of an independent set is equal to its size
3: Solve the maximum-weight matroid intersection problem for matroids $\mathcal{M}$ and $\mathcal{M}(\mathcal{D}', \mathcal{D})$ to find an independent set $C \subseteq P$
4: For every $c \in C$, place a ball of radius $9r_i$, if c is covered by a level i cluster in $\mathcal{D}$
5: **return** the resulting set of balls placed around each center in C

To this end, let $\mathcal{D}^* \subseteq \mathcal{D}$ denote the subset of clusters that contain at least one optimal center. This implies that clusters in $\mathcal{D} \setminus \mathcal{D}^*$ contain no optimal center. In the following claim, we focus on the call MatroidIndependentSet$(\mathcal{D}^*, \mathcal{D})$, and show that the set of balls found in this call covers P.

▶ **Lemma 4.** *The set of balls computed by* MatroidIndependentSet$(\mathcal{D}^*, \mathcal{D})$ *covers the entire set of points. Furthermore, the cost of this set of balls is upper bounded by* 9 *times the cost of* $\mathcal{B}$.

Proof. From every cluster in $\mathcal{D}^*$, pick an arbitrary optimal center, and let the resulting set be $\hat{C}^*$. Since $\hat{C}^* \subseteq C^*$, it is an independent set in $\mathcal{M}$. Furthermore, it contains exactly one point from each cluster in $\mathcal{D}^*$, and no point from any cluster in $\mathcal{D} \setminus \mathcal{D}^*$. Therefore, $\hat{C}^*$ is independent in the matroid $\mathcal{M}(\mathcal{D}^*, \mathcal{D})$.

Let C denote the maximum weight independent subset computed in MatroidIndependentSet$(\mathcal{D}^*, \mathcal{D})$. Thus $|C| \geq |\hat{C}^*|$. As C is independent in $\mathcal{M}(\mathcal{D}^*, \mathcal{D})$, this implies that C (like $\hat{C}^*$) contains exactly one point from each cluster in $\mathcal{D}^*$.

We prove that the set of balls, centered at C, computed at the end of MatroidIndependentSet$(\mathcal{D}^*, \mathcal{D})$, covers all points of P. Fix a point $p \in P$, and suppose it is covered by an optimal ball $B(c_j^*, r_j)$. From Observation 3, there exists a cluster $D_\ell(c) \in \mathcal{D}^*$ of level $\ell \leq j$ such that $c_j^* \in D_\ell(c)$. Therefore, $d(p, c) \leq r_j + 4r_\ell \leq 5r_\ell$. From the previous paragraph, there exists a center $c' \in C \cap D_\ell(c)$. Note that $d(c, c') \leq 4r_\ell$. This implies that, $d(p, c') \leq d(p, c) + d(c, c') \leq 5r_\ell + 4r_\ell = 9r_\ell$. Thus, p is covered by the ball of radius $9r_\ell$ centered at c'.

Finally, note that for every cluster in $\mathcal{D}^*$ of level i, we place at most one ball of radius $9r_i$. Therefore, the cost of the balls thus computed can be bounded by 9 times the cost of balls in $\mathcal{B}$. ◀

▶ **Theorem 5.** *For any fixed* $\epsilon \geq 0$*, there exists a* $(9 + O(\epsilon))$*-approximation algorithm to the Matroid Sum-of-radii problem that runs in* $b^{O(b)} \cdot n^{O(1)}$ *time, where* b *denotes the size of a base in the given matroid.*

Proof. We focus on a particular value of k', and show that the algorithm runs in $k'^{O(k')} \cdot n^{O(1)}$ time. Then, the running time guarantee follows, since $\sum_{k'=1}^{b} k'^{O(k')} = b^{O(b)}$, as previously discussed.

There are $k'^{O(\log k')}$ choices for guessing the "radius profile", and one of these choices corresponds to that of the modified optimal solution. Now fix this choice of the radius profile. At any level $1 \leq i \leq t$, there are at most $k'^{O(k_i)}$ recursive calls to the algorithm at level $i+1$. Therefore, the number of recursive calls at level $t+1$ can be bounded by $k'^{\sum_{i=1}^{t} O(k_i)} = k'^{O(k')}$. At level $t+1$, we call MatroidIndependentSet$(\mathcal{D}', \mathcal{D})$ for every non-empty $\mathcal{D}' \subseteq \mathcal{D}$. Thus, there are at most $2^{|\mathcal{D}|} \leq 2^{k'}$ calls to MatroidIndependentSet, and each matroid intersection problem can be solved in polynomial time, given access to the oracle for $\mathcal{M}$. Therefore, the algorithm terminates in $k'^{O(k')} \cdot n^{O(1)}$ time.

Now, consider the iteration when $k' = k$, and when we correctly guess the radius profile corresponding to the optimal solution. From Lemma 4, one of the calls to MatroidIndependentSet computes a solution that covers all the points, and the cost of this solution can be upper bound by 9 times the cost of the radius profile. Therefore, the cost of this solution can be upper bounded by $9 + O(\epsilon)$ times the cost of the *original* optimal solution. ◀

3 Uniform Capacitated Sum of Radii

Problem Definition

In this problem, we are given a finite metric space (P, d). We are also given a positive integer U, which denotes the *capacity*. We want to place a set of k balls $\mathcal{B}$, and assign each point of P to a ball containing it, such that no ball is assigned more than U points. Furthermore, we want to minimize the sum of radii of the balls in $\mathcal{B}$.

More formally, a feasible solution to the problem consists of a set of centers $C \subseteq P$ of size at most k, and a radius assignment $r : C \to \mathbb{R}^+$. Let $\mathcal{B}$ be the set of resulting balls. Note that the centers of balls in $\mathcal{B}$ are distinct. The solution also consists of an assignment $\mu : P \to \mathcal{B}$, such that p is contained in the ball $\mu(p)$, and $|\mu^{-1}(B)| \leq U$ for every $B \in \mathcal{B}$. Finally, the objective is find such a feasible solution that minimizes the sum of radii: $\sum_{B(c,r) \in \mathcal{B}} r$.

Notation

Let $P' \subseteq P$ be a subset of points, and let $\mathcal{B}'$ be some non-empty set of balls. Then, an assignment $\mu : P' \to \mathcal{B}'$ is said to be a *valid* assignment, if it satisfies the following two properties: (i) for every $p \in P', p \in \mu(p)$, and (ii) $|\mu^{-1}(B)| \leq U$ for any ball $B \in \mathcal{B}'$. In the following discussion, we will allow $\mathcal{B}'$ to contain concentric balls and even be a multi-set. But for simplicity of exposition, we will refer to a multi-set (resp. a multi-subset thereof) as simply a set (resp. a subset). Note that the definition of a valid assignment is consistent even if $\mathcal{B}$ is such a set of balls, by treating each copy of a ball in $\mathcal{B}$ as a distinct object.

Fix an optimal solution and the corresponding optimal assignment. We preprocess this solution in order to discretize the set of radii, exactly as done in the previous section. After this discretization, we assume that the solution uses exactly k_i balls of radius r_i, where $i \leq i \leq t$, and $t = O(\log k)$. Henceforth we will refer to the optimal solution modified in this manner. As in the previous section, let $\mathcal{O}$ denote the set of optimal balls, and $C^* \subseteq P$ be the set of optimal centers. The subsets $\mathcal{O}_i, \mathcal{O}_{\leq i}$ etc. of the optimal balls $\mathcal{O}$, and the subsets $C^*_i, C^*_{\leq i}$ etc. of the optimal centers are also defined as in the previous section. Furthermore, let $\mu_* : P \to \mathcal{O}$ be the optimal assignment. Note that μ_* is a valid assignment by definition.

Algorithm

Our algorithm invokes CapacitatedSoR$(\emptyset, 1)$. Now we describe CapacitatedSoR$(\mathcal{B}, i)$ (Algorithm 4), which is recursive and proceeds in multiple levels. At a particular level $1 \leq i \leq t$, we determine the set of balls of level i in the solution. At the start of the algorithm at level i, we are given a (multi-)set $\mathcal{B}$ of balls chosen earlier. $\mathcal{B}$ consists of balls of level 1 through $i-1$. Before we discuss the algorithm in iteration i, let us define some more notation.

Suppose a ball centered at c was added to $\mathcal{B}$ at level $j < i$ – its radius was $6r_j$ when it was added to $\mathcal{B}$. At every subsequent iteration $j+1 \leq \ell < i$, we expand its radius by an additive $2r_\ell$ factor. Thus, at the beginning of iteration i, the radius of this ball is $6r_j + \sum_{\ell=j+1}^{i-1} 2r_\ell$. Now, in iteration i, we will consider two versions of any ball in $\mathcal{B}$ – expanded and unexpanded. Consider a ball in $\mathcal{B}$, with center c, added during iteration $j < i$. At the beginning of iteration i, this ball has radius $6r_j + \sum_{\ell=j+1}^{i-1} 2r_\ell$; we refer to this as the unexpanded version. On the other hand, the expanded version has radius equal to $6r_j + \sum_{\ell=j+1}^{i} 2r_\ell$, which we denote by $E^i_j(c)$. Note that the expanded version $E^i_j(c)$ is larger than its corresponding version $B^i_j(c)$ by an additive $+2r_i$ factor. Therefore, if $B^* = B(c^*_i, r_i)$ has a non-empty intersection with $B^i_j(c)$, then $B^* \subseteq E^i_j(c)$.

Let $\mathcal{B}' \subseteq \mathcal{B}$ be any subset of balls chosen so far, and let $\overline{\mathcal{B}'} = \mathcal{B} \setminus \mathcal{B}'$. Let $E(\mathcal{B}')$ denote the set of expanded versions of balls in $\mathcal{B}'$. Finally, we define

$$\mathcal{I}(\mathcal{B}') \coloneqq \begin{cases} \left(\bigcap_{E \in E(\mathcal{B}')} E\right) \setminus \left(\bigcup_{B \in \overline{\mathcal{B}'}} B\right) & \text{if } \mathcal{B}' \neq \emptyset \\ P \setminus \left(\bigcup_{B \in \mathcal{B}} B\right) & \text{if } \mathcal{B}' = \emptyset \end{cases}$$

That is, if $\mathcal{B}'$ is non-empty, then $\mathcal{I}(\mathcal{B}')$ is exactly the set of points that belong to the common intersection of the expanded versions of balls in $\mathcal{B}'$, but not in any of the unexpanded versions of balls in $\overline{\mathcal{B}'}$. If $\mathcal{B}'$ is empty, then $\mathcal{I}(\mathcal{B}')$ is the set of points that does not belong to any unexpanded ball in $\mathcal{B}$. Note that if $\mathcal{B}' \subseteq \mathcal{B}$ is exactly the subset of balls that have non-empty intersection with an optimal ball $B(c^*_i, r_i)$, then $B(c^*_i, r_i) \subseteq \mathcal{I}(\mathcal{B}')$.

Let us return to the discussion of CapacitatedSoR$(\mathcal{B}, i)$. For each subset $\mathcal{B}' \subseteq \mathcal{B}$, we call Greedy$(\mathcal{B}', \mathcal{B}, r_i)$ (Algorithm 5). This Algorithm computes a set $P_i(\mathcal{B}')$ of at most $4k$ centers chosen in a certain "greedy" manner. This is the set of potential centers from the region $\mathcal{I}(\mathcal{B}')$ for placing balls of level i. The algorithm ensures that the distance between any two centers in $P_i(\mathcal{B})$ is greater than $4r_i$. Furthermore, if $P_i(\mathcal{B}') < 4k$, then each point in $\mathcal{I}(\mathcal{B}')$ is within distance $4r_i$ of some point in $P_i(\mathcal{B}')$. We repeat this process for every subset $\mathcal{B}'$ of $\mathcal{B}$ (including $\emptyset$).

We will first argue in Lemma 6 that, in some recursive call to the algorithm at level $i = t+1$, the set of balls $\mathcal{B}$ computed captures the optimal solution in an appropriate way. Having shown that this happens, the invocation of PostProcess$(\mathcal{B})$ (Algorithm 7) will appropriately modify the set of balls $\mathcal{B}$ and return a feasible solution. We will discuss the algorithm PostProcess and its analysis later.

▶ **Lemma 6.** *Fix a level $1 \leq i \leq t+1$. In one of the recursive calls to* CapacitatedSoR$(\mathcal{B}, i)$*, there exists an assignment that maps each point $p \in \mu_*^{-1}(\mathcal{O}_{<i})$ to a ball in $\mathcal{B}$ containing p, such that the number of points assigned to each ball does not exceed U.*

Proof. We prove this lemma inductively.

Algorithm 4 CapacitatedSoR$(\mathcal{B}, i)$.

if $i = t + 1$ **then**
  if PostProcess$(\mathcal{B}) \neq$ **fail then**
    return $\mathcal{B}(R)$ returned by PostProcess$(\mathcal{B})$ and **halt**
else
  For every $\mathcal{B}' \subseteq \mathcal{B}$, let $P_i(\mathcal{B}') \leftarrow$ Greedy$(\mathcal{B}', \mathcal{B}, r_i)$
  Let $P_i \leftarrow \bigcup_{\mathcal{B}' \subseteq \mathcal{B}} P_i(\mathcal{B}')$
  for every multi-subset $C_i \subseteq P_i$ of size at most k_i **do**
    Expand every ball in $\mathcal{B}$ by an additive $2r_i$ factor
    CapacitatedSoR$(\mathcal{B} \cup \mathcal{B}(C_i), i + 1)$ ▷ $\mathcal{B}(C_i) := \{B(c, 6r_i) : c \in C_i\}$

Algorithm 5 Greedy$(\mathcal{B}', \mathcal{B}, r)$.

Let $T \leftarrow \mathcal{I}(\mathcal{B}')$; start with all points of T as unmarked
$P(\mathcal{B}') \leftarrow \emptyset$
while $|P(\mathcal{B}')| < 4k$ and there is an unmarked point in T **do**
  $p \in T$ be an unmarked point with maximum number of unmarked points in $B(p, r) \cap T$
  Add p to $P(\mathcal{B}')$; mark all points in $B(p, 4r) \cap T$
return $P(\mathcal{B}')$

Base case

This corresponds to the start of the calls CapacitatedSoR$(\cdot, i)$, where $i = 2$. To this end, consider the invocation of the algorithm at the earlier level, i.e., CapacitatedSoR$(\emptyset, 1)$. Note that since $\mathcal{B} = \emptyset$, $\mathcal{B}' = \emptyset$, and there is only one call Greedy$(\emptyset, \emptyset, r_1)$. Note that the optimal solution uses $k_1 + k_2 + \ldots + k_t \leq k$ balls of radius at most r_1 in order to cover the entire set of points $P = \mathcal{I}(\mathcal{B}')$. Recall that the set of centers $P_1(\emptyset) = P_1$ returned by the Greedy algorithm has the property that any two centers in $P_1(\emptyset)$ are at least $4r_1$ away from each other. Therefore, $P_1(\emptyset)$ contains at most one point from each optimal ball, and thus $|P_1(\emptyset)| \leq k$. It follows that for any optimal center $c_1^* \in C_1^*$, there is a center $c \in P_1$, such that $d(c_1^*, c) \leq 4r_1$. That is, for every optimal ball $B^* = B(c_1^*, r_1) \in \mathcal{O}_1$, there exists $c \in P_1$, such that $B^* \subseteq B(c, 5r_1)$. We let $\varphi(B^*) := c$ (select a nearest c form c_1^* there are multiple such $c \in P_1$). Let $C_1 \subseteq P_1$ be the multi-set that is the image of the mapping $\varphi : \mathcal{O}_1 \to P_1$, where the multiplicity of each $c \in C_1$ is equal to $|\varphi^{-1}(c)|$. Therefore, for each $B^* \in \mathcal{O}_1$ the points in $\mu_*^{-1}(B^*)$ can be reassigned to a unique ball in $\mathcal{B}(C_1)$ centered at $\varphi(B^*) \in C_1$. This completes the proof for the base case.

Note that we were able to "guess" the locations of the optimal centers approximately in the base case. However, we cannot accomplish this in the subsequent levels, because some optimal balls may be contained in larger optimal balls. This is what complicates the algorithm and its analysis. Nevertheless, we will argue that we can find an appropriate set of substitute centers whenever necessary that will facilitate the reassignment process.

Inductive hypothesis

Suppose during some invocation of the algorithm CapacitatedSoR$(\mathcal{B}, i)$ at level i, we have a set of balls $\mathcal{B}$ of levels $1 \leq j < i$, such that the set of points $\mu_*^{-1}(\mathcal{O}_{<i})$ can be assigned to the balls in $\mathcal{B}$. Let us also suppose that we have a mapping $\varphi : \mathcal{O}_{<i} \to C_{<i}$, where $C_{<i}$ denotes the set of centers of balls in $\mathcal{B}$.

Inductive Step

We first sketch the high level idea. Here, we will extend φ to include $\mathcal{O}_i$, i.e., we will map every optimal ball in $\mathcal{O}_i$ to a center in P_i; where P_i is the set of centers computed in lines 5 and 6 of Algorithm 4. Now, let $C_i \subseteq P_i$ be the image of $\varphi(\mathcal{O}_i)$, where the multiplicty of each $c \in P_i$ is set to be $|\varphi^{-1}(c)|$. Note that $|C_i| = |\mathcal{O}_i| = k_i$. Having found such a mapping, we will consider an optimal ball $B_i^* = B(c_i^*, r_i) \in \mathcal{O}_i$, and reassign points in $\mu_*^{-1}(B_i^*)$ to the balls in $\mathcal{B} \cup \mathcal{B}(C_i)$. We will use the ball $\varphi(B_i^*)$ to show that this reassignment can be done without violating the capacities. Doing this for every optimal ball in $\mathcal{O}_i$, we will show that all points in $\mu_*^{-1}(\mathcal{O}_{\leq i})$ are assigned to balls in $\mathcal{B} \cup \mathcal{B}(C_i)$. Now we discuss the details of this inductive argument.

For any $B^* = B(c_i^*, r_i) \in \mathcal{O}_i$, if there is $c \in P_i$ such that $B^* \subseteq B(c, 6r_i)$, then we set $\varphi(B_i^*) = c$ (choosing a nearest such c from c^*, if there are multiple such $c^* \in P_i$). Let $\mathcal{O}_i^1 \subseteq \mathcal{O}_i$ be the subset that is mapped in such way, and let $C_i^1 \subseteq P_i$ be its image (with multiplicity $\varphi^{-1}(c)$ for every $c \in C_i^1$). Note that all the points assigned a ball $B_i^* \in \mathcal{O}_i^1$, are also contained in the ball $B(c, 6r_i)$, where $c = \varphi(B_i^*)$. Therefore, we can reassign points in $\mu_*^{-1}(B_i^*)$ to the ball $B(c, 6r_i)$.

Now, let $\mathcal{O}_i^2 \coloneqq \mathcal{O}_i \setminus \mathcal{O}_i^1$ be the set of optimal balls not mapped so far. We will map each ball in $\mathcal{O}_i^2$ to a unique center in $P_i \setminus C_i^1$, and use this mapping to compute the required reassignment. We describe the assignment in the following mapping procedure – note that this is used only in the analysis.

Algorithm 6 Mapping procedure.

1: Suppose all balls in $\mathcal{O}_i^2$ are unmapped at the beginning; let $C_i^2 \leftarrow \emptyset$
2: **for** each subset $\mathcal{B}' \subseteq \mathcal{B}$ in an arbitrary order **do**
3: Let $\mathcal{O}_i^2(\mathcal{B}') \subseteq \mathcal{O}_i^2$ be the subset of *unmapped* balls contained in $\mathcal{I}(\mathcal{B}')$
▷ *Unmapped balls in $\mathcal{O}_i^2$ are the balls that have not yet been mapped using φ in an earlier iteration.*
4: Let $F_i(\mathcal{B}') \subseteq P_i(\mathcal{B}')$ include every point that:
(i) belongs to C_i^1, or (ii) is chosen as a center of a ball in $\mathcal{B}$, or
(iii) is within $2r_i$ from some center in C_i^2, or
(iv) is within $r_i + r_j$ from some center in C_j^*, where $j \geq i$, or
(v) belongs to $C_{<i}^*$.
5: Extend φ to $\mathcal{O}_i^2(\mathcal{B}')$ by arbitrarily mapping each ball in $\mathcal{O}_i^2(\mathcal{B}')$ to a unique center in $P_i(\mathcal{B}') \setminus F_i(\mathcal{B}')$.
6: Let $C_i^2(\mathcal{B}')$ be the image of $\mathcal{O}_i^2(\mathcal{B}')$, under the above mapping φ.
7: Mark all balls in $\mathcal{O}_i^2(\mathcal{B}')$ as mapped, and add $C_i^2(\mathcal{B}')$ to C_i^2.

▷ Claim 7. For any $\mathcal{B}' \subseteq \mathcal{B}$, if $\mathcal{O}_i^2(\mathcal{B}') \neq \emptyset$, then $|\mathcal{O}_i^2(\mathcal{B}')| \leq |P_i(\mathcal{B}') \setminus F_i(\mathcal{B}')|$. That is, there are enough centers available in $P_i(\mathcal{B}') \setminus F_i(\mathcal{B}')$ to be mapped in Line 5.

Proof. Since $\mathcal{O}_i^2(\mathcal{B}') \neq \emptyset$, let $B(c_i^*, r_i) \in \mathcal{O}_i^2(\mathcal{B}')$. Note that $c_i^* \in \mathcal{I}(\mathcal{B}')$, and $B(c_i^*, r_i) \subseteq \mathcal{I}(\mathcal{B}')$.

Consider the call $\textsc{Greedy}(\mathcal{B}', \mathcal{B}, r_i)$. We first claim that the while loop ends with $|P_i(\mathcal{B}')| = 4k$. Suppose for the contradiction that the while loop ends because all points in $P_i(\mathcal{B}')$ are marked. Let c be the point added to $P_i(\mathcal{B}')$ when c_i^* is marked. Then, $d(c_i^*, c) \leq 4r_i$. Thus, $B(c_i^*, r_i) \subseteq B(c, 6r_i)$, which implies that $B(c_i^*, r_i) \in \mathcal{O}_i^1$. This is a contradiction, since $B(c_i^*, r_i) \in \mathcal{O}_i^2$.

Now we claim that $|F_i(\mathcal{B}')| \leq 3k$, by considering each of the five conditions ($F_i(\cdot)$ stands for centers *forbidden* due to one of the five conditions). Conditions (i) and (ii) include at most $\sum_{j=1}^{i-1} k_j$, and at most k_i points respectively. Therefore, k is an upper bound for points satisfying conditions (i) and (ii).

We now claim that k_i is also an upper bound for points satisfying condition (iii). To this end, we claim that for $c \in C_i^2$, there is at most one $c \in P_i(\mathcal{B}')$ such that $d(c, c') \leq 2r_i$. Suppose that there are two distinct such points $c_1, c_2 \in P_i(\mathcal{B}')$. Then, $d(c_1, c_2) \leq d(c, c_1) + d(c, c_2) \leq 4r_i$. This is a contradiction, since the distance between any two points in $P_i(\mathcal{B}')$ is greater than $4r_i$. Finally, since $|C_i^2| \leq k$, k is also an upper bound on the centers excluded due to condition (iii).

A similar proof also shows that for any fixed $c_j^* \in C_j^*$ with $j \geq i$, there is at most one $c \in P_i(\mathcal{B}')$ with $d(c_j^*, c) \leq r_i + r_j$. Therefore, $\sum_{j=i}^{t} k_j$ is an upper bound for points satisfying condition (iv). $\sum_{j=1}^{i-1} k_j$ is an upper bound on condition (v). Therefore, k is an upper bound on conditions (iv) and (v) together.

Putting everything together, $|F_i(\mathcal{B}')| \leq 3k$, which implies that $|P_i(\mathcal{B}') \setminus F_i(\mathcal{B}')| \geq k$. Therefore, each ball in $\mathcal{O}_i^2(\mathcal{B}')$ can be mapped to a unique point in $|P_i(\mathcal{B}')|$. ◁

The next claim is used later to argue that the reassignment can be done using the mapping φ constructed in this manner.

▷ Claim 8. Fix $\mathcal{B}' \subseteq \mathcal{B}$ and a ball $B^* = B(c_i^*, r_i) \in \mathcal{O}_i^2(\mathcal{B}')$. If $\varphi(B^*) = c$, then $|B(c, r_i) \cap \mathcal{I}(\mathcal{B}')| \geq |B^*| \geq |\mu_*^{-1}(B^*)|$.

Proof. We first claim that no point in B^* is marked in $\textsc{Greedy}(\mathcal{B}', \mathcal{B}, r_i)$. Otherwise, let $c' \in \mathcal{I}(B^*)$ be the point added to $P_i(\mathcal{B}')$ when a point $p \in B^*$ was marked. Then, $d(c', c_i^*) \leq d(c', p) + d(p, c_i^*) \leq 5r_i$, which implies that $B^* \subseteq B(c', 6r_i)$, which implies that $B(c_i^*, r_i) \in \mathcal{O}_i^1$. This is a contradiction, since $B(c_i^*, r_i) \in \mathcal{O}_i^2$. Therefore no point in $B^* \cap \mathcal{I}(\mathcal{B}') = B^*$ is marked until the end of the while loop.

Now, consider the beginning of the iteration when c was added to $P_i(\mathcal{B}')$. At this point, c_i^* is also a candidate. Since c is chosen over c_i^*, it implies that $|B(c, r_i) \cap \mathcal{I}(\mathcal{B}')| \geq |B^*| \geq |\mu_*^{-1}(B^*)|$, where the last inequality holds by definition. ◁

We use this claim to show that we can reassign points from $\mu_*^{-1}(\mathcal{O}_i^2)$ to balls in $\mathcal{B} \cup \mathcal{B}(C_i)$, where $C_i = C_i^1 \cup C_i^2$. Recall that we have already reassigned points $\mu_*^{-1}(\mathcal{O}_i^1)$ to C_i^1.

Now let us consider the optimal balls in $\mathcal{O}_i^2$ in the same order in which they were mapped in Algorithm 6. Consider an optimal ball $B^* = B(c_i^*, r_i)$, and suppose it was mapped in the iteration corresponding to $\mathcal{B}' \subseteq \mathcal{B}$. That is, $B^* \in \mathcal{O}_i^2(\mathcal{B}')$. Let $c = \varphi(B^*)$. Because of condition (v), there is no optimal center $c_j^* \in C_j^*$ with $j \geq i$, such that $B(c_j^*, r_j) \cap B(c, r_i) \neq \emptyset$. Therefore, all points in $B = B(c, r_i)$ are assigned to balls in $\mathcal{O}_{<i}$ in the optimal assignment μ_*. Furthermore, because of condition (iv), there is no other center $c' \in C_i^2$ within distance $2r_i$ from c, which implies that points in B have not been currently assigned to a ball in $\mathcal{B}(C_i)$. Therefore, by the inductive hypothesis, these points are assigned to balls in $\mathcal{B}$.

Now we reassign m points from the set $B \cap \mathcal{I}(\mathcal{B}')$ to the ball $B(c, 6r_i)$, where $m = \min\{U, |B \cap \mathcal{I}(\mathcal{B}')|\}$. These points are originally assigned to balls in $\mathcal{B}$. As they are contained in $\mathcal{I}(\mathcal{B}')$, no such point belongs to a ball in $\mathcal{B} \setminus \mathcal{B}'$, by the definition of $\mathcal{I}(\mathcal{B}')$. Thus, these points are assigned to balls in $\mathcal{B}'$. Their reassignment to $B(c, 6r_i)$ collectively frees up m units of capacity from balls in $\mathcal{B}'$. Note that B^* is also completely contained in $\mathcal{I}(\mathcal{B}')$, and $m \geq |\mu_*^{-1}(B^*)|$ by Claim 8. Therefore, we can use the freed capacity of balls in $\mathcal{B}'$ to assign points in $\mu_*^{-1}(B^*)$.

We perform this reassignment process for each ball in $\mathcal{O}_i^2$. Therefore, at the end, every point in $\mu_*^{-1}(\mathcal{O}_{\leq i})$ is assigned to a ball in $\mathcal{B} \cup \mathcal{B}(C_i)$. This finishes the proof of Lemma 6. ◀

Using Lemma 6 at level $t+1$, we know that there exists a recursive call to CapacitatedSoR$(\mathcal{B}, t+1)$, such that the set of points in $\mu_*^{-1}(\mathcal{O}) = P$ can be assigned to the balls in $\mathcal{B}$ without violating capacities. Fix such a recursive call. With this, let us define $C^1 = \bigcup_{i=1}^{t} C_i^1$, and $C^2 = \bigcup_{i=1}^{t} C_i^2$, and let $C = C^1 \cup C^2$. Note that there may be several concentric balls in $\mathcal{B}$. We want to move the concentric balls to "nearby" unique centers in order to obtain a feasible solution. The following observations, which follow from the description of the mapping procedure (see Algorithm 6), will aid us in doing this.

▶ **Observation 9.**

1. *For any $c \in C_i^1$, define $R^*(c)$ to be the set of optimal centers of the balls in $\varphi^{-1}(c)$.*
   (A) The sets $\{R^*(c)\}_{c \in C^1}$ are pairwise disjoint, i.e., for distinct $c_1, c_2 \in C^1$, we have that $R^*(c_1) \cap R^*(c_2) = \emptyset$.
   (B) $R^*(c) \subseteq B(c, 5r_i)$ for any $c \in C_i^1$.
2. *For any $c \in C^2$, define $R^*(c) \coloneqq \{c\}$*
   (A) $|\varphi^{-1}(c)| = 1$ for all $c \in C^2$
   (B) $C^2 \cap C^* = \emptyset$, and
   (C) $C^2 \cap C^1 = \emptyset$.
3. *Items 1 and 2 imply that the sets $\{R^*(c)\}_{c \in C}$ are pairwise disjoint.*

Proof. For item 1.A, note that $\varphi : \mathcal{O} \to C$ is a many-to-one function, and that every ball in $\mathcal{O}$ has a distinct center. Item 1.B follows from the definition of φ.

Claims in item 2 follow from the definition of set of *forbidden* centers $F_i(\mathcal{B}')$ in the mapping procedure (see line 4). ◀

Now we are ready to show that, when PostProcess$(\mathcal{B})$ (Algorithm 7) is called from CapacitatedSoR$(\mathcal{B}, t+1)$, where $\mathcal{B}$ is the set of balls guaranteed by Lemma 6, it successfully returns a feasible solution.

Algorithm 7 PostProcess$(\mathcal{B})$.

1: For every $1 \le i \le t$, and every $c \in C_i$, find a set $R(c) \subseteq B(c, 5r_i)$ where $|R(c)|$ equals the multiplicity of c in C_i and the sets $R(c)$ are pairwise disjoint for all $c \in C$
   This can be solved using a max-flow problem
2: **if** such a collection of sets $R(c)$ does not exist: **return fail**
3: Let $R = \bigcup_{c \in C} R(c)$, and let $\mathcal{B}(R) \coloneqq \{B(c, \alpha \cdot r_i) : c \in R\}$
   ▷ *α is defined below in the proof of Lemma 10*
4: Check whether there exists a feasible assignment from P to the balls in $\mathcal{B}(R)$
   This can be solved using a max-flow problem
5: **if** a feasible assignment exists: **return** $\mathcal{B}(R)$; **else**: **return fail**

▶ **Lemma 10.** PostProcess$(\mathcal{B})$ *succeeds in finding a set of balls $\mathcal{B}(R)$, and there is a feasible assignment $\mu' : P \to \mathcal{B}(\mathcal{R})$.*

Proof. From Lemma 6, there exists a feasible assignment $\mu : P \to \mathcal{B}$, however there may be concentric balls in the set $\mathcal{B}$. However, Observation 9 implies that the sets $R^*(c)$ are pairwise disjoint for $c \in C$, and that $R^*(c) \subseteq B(c, 5r_i)$ for a center $c \in C_i$. Therefore, in line 1 of Algorithm PostProcess$(\mathcal{B})$, we can successfully find the sets $R(c)$ as claimed. Note that $R(c) \subseteq B(c, 5r_i)$.

From Lemma 6, for any $B = B(c, 6r_i) \in \mathcal{B}$, $\mu^{-1}(B) \subseteq E_i^t(c)$. Note that the radius of the expanded version of the ball $E_i^t(c)$ is equal to $6r_i + \sum_{\ell=i+1}^{t} 2r_\ell \leq \alpha' \cdot r_i$, for some α'. Therefore, for any point $p \in \mu^{-1}(B)$, and any $c' \in R(c)$, we have that $d(p, c') \leq 5r_i + \alpha' r_i = (\alpha' + 5)r_i = \alpha r_i$ [1]. This implies that, in line 4 of the Algorithm 7, we can find such a feasible assignment μ'. ◀

▶ **Lemma 11.** CapacitatedSoR$(\emptyset, 1)$ *runs in* $2^{O(k^2)} \cdot n^{O(1)}$ *time.*

Proof. Fix a level $1 \leq i \leq t$, and consider CapacitatedSoR$(\mathcal{B}, i)$. At the beginning of the algorithm, $|\mathcal{B}| \leq k$, therefore the number of subsets can be upper bound by 2^k, which implies that $|P_i| \leq 4k \cdot 2^k$. Now, let $k_i' \leq k_i$ denote the size of the set C_i without multiplicities. Therefore, there are $\sum_{k_i'=1}^{k_i} \binom{4k \cdot 2^k}{k_i'} = \left(4k \cdot 2^k\right)^{O(k_i)} = 2^{O(k \cdot k_i)}$ number of choices for selecting the set C_i (without multiplicities). For a fixed choice of C_i, there are at most $(k_i)^{k_i'} = k^{O(k_i)}$ choices for placing one or more copies at each location in C_i. We make a recursive call for each such choice of the multi-set C_i. The overall number of recursive calls to level $i+1$ can be upper bounded by $k^{O(k_i)} \cdot 2^{O(k \cdot k_i)} = 2^{O(k \cdot k_i)}$.

Let $T(i)$ denote the running time of the algorithm at level i. Then, we have the following recurrence relation: $T(i) = 2^{O(k \cdot k_i)} \cdot T(i+1) + 2^{O(k \cdot k_i)} \cdot n^{O(1)}$.
Furthermore, $T(t+1) = n^{O(1)}$, since PostProcess runs in time polynomial in n. This recurrence solves to $T(1) = 2^{O(k^2)} \cdot n^{O(1)}$, where we use the fact that $\sum_{i=1}^{t} k_i = k$. ◀

▶ **Theorem 12.** *There exists a* 28*-approximation for the Capacitated Sum of Radii problem that runs in* $2^{O(k^2)} \cdot n^{O(1)}$ *time.s*

Proof. There are $O(n^2)$ choices for guessing the maximum radius, and $k^{O(\log k)}$ choices for guessing the radius profile of the optimal solution. Note that we lose a factor of $(1+\epsilon)^2$ in the latter step. Now, fix the correct value of maximum radius and the radius profile that corresponds to the modified optimal solution. By Lemma 11, the algorithm CapacitatedSoR$(\emptyset, 1)$ runs in $2^{O(k^2)} \cdot n^{O(1)}$ time for any fixed choice of the radius profile.

Furthermore, By Lemma 10, there exists a recurse call at level $t+1$, that returns a feasible solution. Finally, for any $1 \leq i \leq t$ we use k_i balls of radius αr_i in this solution, whereas the optimal solution uses k_i balls of radius r_i. Therefore, the approximation guarantee is at most $(1+\epsilon)^2 \cdot \alpha$, where α is as in Lemma 10. Choosing $\epsilon \approx 0.267$, the above quantity can be upper bounded by 28. ◀

4 Conclusion

We obtain constant approximations for the uniform capacitated sum of radii problem in FPT time. It is unclear whether a similar result can be obtained in polynomial time. Finally, obtaining a constant approximation for the matroid version of the problem in polynomial time remains open.

[1] It can be shown that $\alpha = \frac{2+11 \cdot \epsilon(1+\epsilon)}{\epsilon(1+\epsilon)}$.

References

1 Marek Adamczyk, Jaroslaw Byrka, Jan Marcinkowski, Syed M. Meesum, and Michal Wlodarczyk. Constant-factor FPT approximation for capacitated k-median. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.*, pages 1:1–1:14, 2019. doi:10.4230/LIPIcs.ESA.2019.1.

2 Hyung-Chan An, Mohit Singh, and Ola Svensson. Lp-based algorithms for capacitated facility location. *SIAM J. Comput.*, 46(1):272–306, 2017. doi:10.1137/151002320.

3 Sayan Bandyapadhyay, Santanu Bhowmick, Tanmay Inamdar, and Kasturi R. Varadarajan. Capacitated covering problems in geometric spaces. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, volume 99 of *LIPIcs*, pages 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.SoCG.2018.7.

4 Babak Behsaz and Mohammad R. Salavatipour. On minimum sum of radii and diameters clustering. *Algorithmica*, 73(1):143–165, 2015. doi:10.1007/s00453-014-9907-3.

5 Jaroslaw Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2):23:1–23:31, 2017.

6 Jaroslaw Byrka, Bartosz Rybicki, and Sumedha Uniyal. An approximation algorithm for uniform capacitated k-median problem with $1 + \epsilon$ capacity violation. In Quentin Louveaux and Martin Skutella, editors, *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, volume 9682 of *Lecture Notes in Computer Science*, pages 262–274. Springer, 2016. doi:10.1007/978-3-319-33461-5_22.

7 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002. doi:10.1006/jcss.2002.1882.

8 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *J. Comput. Syst. Sci.*, 68(2):417–441, 2004. doi:10.1016/j.jcss.2003.07.014.

9 Danny Z Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and knapsack center problems. *Algorithmica*, 75(1):27–52, 2016.

10 Julia Chuzhoy and Yuval Rabani. Approximating k-median with non-uniform capacities. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 952–958. SIAM, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070569.

11 Vincent Cohen-Addad and Jason Li. On the fixed-parameter tractability of capacitated clustering. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 41:1–41:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.41.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 H. Gökalp Demirci and Shi Li. Constant approximation for capacitated k-median with (1+epsilon)-capacity violation. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 73:1–73:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.73.

14 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi R. Varadarajan. On metric clustering to minimize the sum of radii. *Algorithmica*, 57(3):484–498, 2010. doi:10.1007/s00453-009-9282-7.

15 Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

16 MohammadTaghi Hajiaghayi, Rohit Khandekar, and Guy Kortsarz. Budgeted red-blue median and its generalizations. In *European Symposium on Algorithms*, pages 314–325. Springer, 2010.

17 Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.

18 Samir Khuller and Yoram J Sussmann. The capacitated k-center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000.

19 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 646–659, 2018.

20 Shi Li. Approximating capacitated k-median with $(1 + \epsilon)k$ open facilities. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 786–796, 2016. doi:10.1137/1.9781611974331.ch56.

21 Shi Li. On uniform capacitated k-median beyond the natural LP relaxation. *ACM Trans. Algorithms*, 13(2):22:1–22:18, 2017. doi:10.1145/2983633.

Optimal Polynomial-Time Compression for Boolean Max CSP

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands
b.m.p.jansen@tue.nl

Michał Włodarczyk
Eindhoven University of Technology, The Netherlands
m.wlodarczyk@tue.nl

Abstract

In the Boolean maximum constraint satisfaction problem – Max CSP(Γ) – one is given a collection of weighted applications of constraints from a finite constraint language Γ, over a common set of variables, and the goal is to assign Boolean values to the variables so that the total weight of satisfied constraints is maximized. There exists a concise dichotomy theorem providing a criterion on Γ for the problem to be polynomial-time solvable and stating that otherwise it becomes NP-hard. We study the NP-hard cases through the lens of kernelization and provide a complete characterization of Max CSP(Γ) with respect to the optimal compression size. Namely, we prove that Max CSP(Γ) parameterized by the number of variables n is either polynomial-time solvable, or there exists an integer $d \geq 2$ depending on Γ, such that:

1. An instance of Max CSP(Γ) can be compressed into an equivalent instance with $\mathcal{O}(n^d \log n)$ bits in polynomial time,
2. Max CSP(Γ) does not admit such a compression to $\mathcal{O}(n^{d-\varepsilon})$ bits unless NP $\subseteq$ co-NP/poly.

Our reductions are based on interpreting constraints as multilinear polynomials combined with the framework of constraint implementations. As another application of our reductions, we reveal tight connections between optimal running times for solving Max CSP(Γ). More precisely, we show that obtaining a running time of the form $\mathcal{O}(2^{(1-\varepsilon)n})$ for particular classes of Max CSPs is as hard as breaching this barrier for Max d-SAT for some d.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases constraint satisfaction problem, kernelization, exponential time algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.63

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.03443`.

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).
Michał Włodarczyk: The author was supported by the Foundation for Polish Science (FNP).

1 Introduction

Background and motivation. The framework of constraint satisfaction problems (CSPs) allows the computational complexity of a large class of problems to be studied through a common lens [11]. A typical instance of such a problem asks whether it is possible to assign each of the variables $x_1, \ldots, x_n$ a value from a finite domain D, such that a given list of constraint applications is satisfied. A constraint is applied to a fixed number of variables, and indicates which combinations of values are legal. In the Max CSP problem, the goal is to maximize the number of satisfied constraints. See Section 2 for formal definitions.

The investigation of CSPs has led to deep theorems characterizing the complexity of a CSP based on the type of constraints allowed in the instance [7, 24]. For example, the long-awaited CSP dichotomy theorem [6, 35] provides a criterion separating the NP-complete from the polynomial-time solvable CSPs; the work of Khanna, Sudan, Trevisan, and Williamson characterizes how well the maximization version of a Boolean CSP can be approximated [21] (see [13, 20] for larger domains; see [12, 27] for optimal approximation factors); and Cai and Chen [8] present a dichotomy that separates CSPs for which the number of complex-weighted solutions can be counted in polynomial time, from those where the problem is #P-hard.

In this work we analyze the complexity of constraint satisfaction in an algorithmic regime that is currently far from understood: polynomial-time compression and kernelization [15]. Here, the goal is to analyze how much (in terms of the number of variables n) an instance can be compressed by a polynomial-time algorithm without changing the answer, and to understand how the compressibility depends on the type of available constraints. A *compression* is a polynomial-time algorithm that reduces instances of one problem to equivalent, small instances of a potentially different problem; a *kernelization* compresses to an instance of the same problem (see Section 2.4). A kernelization of small size allows an instance to be stored, manipulated, and solved more efficiently. It is therefore of interest to find the smallest possible kernelizations. Since every kernelization yields a compression, one can prove lower bounds on the size of kernelizations by establishing lower bounds on compressions.

In recent years, there have been a number of advances in the understanding of compressibility of CSPs [9, 14, 18, 25]. A foundational result by Dell and van Melkebeek [14] states that for $d \geq 3$, CNF-SAT with clauses of size at most d (d-CNF-SAT) parameterized by the number of variables n admits no (polynomial-time) compression of size $\mathcal{O}(n^{d-\varepsilon})$ for any $\varepsilon > 0$, unless NP $\subseteq$ co-NP/poly (which is known to imply a collapse of the polynomial hierarchy [34]). As an instance of d-CNF-SAT can trivially be compressed to $\mathcal{O}(n^d)$ bits via a bitstring that encodes for each of the $\mathcal{O}(n^d)$ possible clauses whether or not it is present in the instance, the d-CNF-SAT problem does not admit any non-trivial compression. The situation is different for the related problem d-Not-All-Equal SAT (d-NAE-SAT), which is the variant where a clause is satisfied when its literals do not all evaluate to the same value. Jansen and Pieterse showed [17, 18] that for $d \geq 3$, the d-NAE-SAT problem has a compression of size $\mathcal{O}(n^{d-1} \log n)$, but not of size $\mathcal{O}(n^{d-1-\varepsilon})$ unless NP $\subseteq$ co-NP/poly. This example shows that the type of constraints affects the compressibility of a CSP.

The notion of a *constraint language* is used to rigorously analyze how the complexity of a CSP depends on the type of constraints. In this work, we will only consider CSPs over the Boolean domain: we work exclusively with Boolean constraints and constraint languages. A constraint is therefore a function of the form $f\colon \{0,1\}^k \to \{0,1\}$, where $k \geq 1$ is the *arity* of the constraint, also denoted as $\text{AR}(f)$. A constraint language Γ is a *finite* set of constraints. The input of the corresponding decision problem, denoted CSP(Γ), consists of a set of constraint applications of the form $f(x_{j_1}, \ldots, x_{j_{\text{AR}(f)}}) = 1$ over n common variables, where f is some constraint from Γ. The question is whether there is an assignment $\{x_1, \ldots, x_n\} \to \{0,1\}$ satisfying all the constraint applications.

In this terminology, Chen, Jansen, and Pieterse [9] characterized for all (Boolean) constraint languages Γ consisting of constraints of arity at most three, what the optimal compression size is for CSP(Γ). Lagerkvist and Wahlström [25] gave universal-algebraic conditions on Γ which ensure that CSP(Γ) has a compression of size $\mathcal{O}(n \log n)$, and a characterization is known of the constraint languages Γ_{sym} consisting entirely of *symmetric* functions for which CSP(Γ_{sym}) has a compression of near-linear size [9, §5]. Hence there is some understanding of the optimal compressibility of CSP(Γ).

However, when we move from the question of whether *all* constraints can be satisfied to the task of maximizing the number of satisfied constraints (Max CSP), the situation is much less understood. To the best of our knowledge, no non-trivial compressions are known for any Max CSP(Γ), and no compression lower bounds are known for Max CSP(Γ) other than those already implied from CSP(Γ). In this paper, we therefore analyze the compressibility of Max CSP(Γ).

Before presenting our results, we briefly summarize the main algorithmic approach for compressing CSP(Γ) and illustrate why it fails completely for Max CSP. Consider for example 3-NAE-SAT. The number of constraint applications in an n-variable instance of this problem can be reduced to $\mathcal{O}(n^2)$ without changing the solution space, which allows it to be encoded in $\mathcal{O}(n^2 \log n)$ bits. The *sparsification* to $\mathcal{O}(n^2)$ constraint applications is achieved by a linear-algebraic approach. Note that a not-all-equal constraint on variables $(x, y, z) \in \{0,1\}^3$ is satisfied if and only if $x + y + z - xy - xz - yz - 1 = 0$. Observe that if $p_1(x_1, \ldots, x_n) = 0, \ldots, p_m(x_1, \ldots, x_n) = 0$ are polynomial equalities which are satisfied by an assignment to $x_1, \ldots, x_n$, then also $\sum_{i=1}^{m} \alpha_i \cdot p_i(x_1, \ldots, x_n) = 0$ holds for any linear combination as determined by $\alpha_1, \ldots, \alpha_m$. To sparsify a 3-NAE-SAT instance with this insight, proceed as follows. Transform each constraint c_i into an equality $p_i(x_1, \ldots, x_n) = 0$ for a degree-2 polynomial p_i, substituting $1 - v$ for negated variables $\neg v$ in the constraint. This yields a system of equations of degree-2 polynomials in n variables, which have $\mathcal{O}(n^2)$ distinct monomials. The rank of a corresponding vector space is therefore $\mathcal{O}(n^2)$, which yields a basis of $\mathcal{O}(n^2)$ equalities such that all others can be expressed as their linear combinations. All constraints not corresponding to an element of this basis can be safely omitted from an instance of 3-NAE-SAT, since they will be automatically satisfied by any assignment that satisfies *all* basis constraints. This yields the claimed sparsification of $\mathcal{O}(n^2)$ constraints. Note, however, that this approach fails completely for the variant Max 3-NAE-SAT: if an assignment *does not* satisfy all constraints of the basis, this does not give any satisfaction guarantees on the linearly-dependent constraints. Hence the sparsification approach for CSP(Γ) is not applicable for Max CSP(Γ).

Our results. We provide a new route to compression for Max CSP(Γ), and prove that the resulting compressions are *essentially optimal* for all constraint languages Γ, assuming NP $\not\subseteq$ co-NP/poly. Our results characterize the optimal compressibility of all Boolean Max CSPs in terms of degrees of characteristic polynomials, and uncover a wide range of Max CSP(Γ) problems that admit a non-trivial compression. For a Boolean function $f\colon \{0,1\}^k \to \{0,1\}$, its *characteristic polynomial* is the *unique* k-variate multilinear polynomial $P_f(x)$ over $\mathbb{R}$ that agrees with f on all $x \in \{0,1\}^k$. The fact that this representation is unique is well-known (cf. [29]). For a constraint language Γ, define $\deg(\Gamma) = \max_{f \in \Gamma} \deg(P_f)$. We prove that $\deg(\Gamma)$ characterizes the compressibility of Max CSP(Γ).

To state our results precisely, we have to address a feature of the problem that is particular to the maximization variant: repetitions of constraint applications. While such repetitions are irrelevant in the CSP setting when all constraint applications have to be satisfied, they become relevant when maximizing the number of satisfied constraint applications. The standard approach in the Max CSP literature is therefore to give each constraint application a positive integer weight value [11, 21]. The decision problem Max CSP(Γ) then takes as input a system of Γ-constraint applications with weights from $\mathbb{N}$, and a threshold value t, and asks whether there is an assignment such that the weight of the satisfied constraint applications is at least t.

Let Γ be a (finite, Boolean) constraint language.[1] Our main positive result is the following.

▶ **Theorem 1.1.** *Max CSP(Γ) parameterized by the number of variables n, with positive integer weights bounded by $n^{\mathcal{O}(1)}$, admits a compression of size $\mathcal{O}\big(n^{\deg(\Gamma)} \log n\big)$.*

In fact, we are even able to reduce any instance of Max CSP(Γ) to an equivalent instance of the *same* problem, having $\mathcal{O}\big(n^{\deg(\Gamma)}\big)$ weighted constraint applications. We prove matching lower bounds whenever Max CSP(Γ) is NP-complete. It is known [10, 11, 21] that for inputs with positive integer weights, Max CSP(Γ) is polynomial-time solvable if Γ is 0-valid, 1-valid, or 2-monotone (see Section 2.1), and NP-complete otherwise.

▶ **Theorem 1.2.** *If Γ is not 0-valid, 1-valid, or 2-monotone, then assuming NP $\not\subseteq$ co-NP/poly, Max CSP(Γ) parameterized by the number of variables n, with positive integer weights bounded by $n^{\mathcal{O}(1)}$, does not admit a compression of size $\mathcal{O}\big(n^{\deg(\Gamma)-\varepsilon}\big)$ for any $\varepsilon > 0$.*

Our results uncover an interesting contrast in compressibility between decision CSPs and maximization CSPs. While both involve the analysis of the degrees of polynomials, the type of polynomials which is used differs, leading to differences in compressibility. For example, while d-NAE-SAT has a compression of size $\mathcal{O}(n^{d-1} \log n)$ for *all* $d \geq 3$, the corresponding Max d-NAE-SAT problem with weights of absolute value $n^{\mathcal{O}(1)}$ has a compression of size $\mathcal{O}(n^{d-1} \log n)$ for *odd* $d \geq 3$, but no compression of size $\mathcal{O}(n^{d-\varepsilon})$ for *even* d. Another example is d-Exact SAT, where we require exactly one literal in each clause to be true. Whereas d-Exact SAT admits a compression of size $\mathcal{O}(n \log n)$ for every fixed d [9], we show that Max d-Exact SAT cannot be compressed to $\mathcal{O}(n^{d-\varepsilon})$ bits.

Techniques. On a high level, our results are obtained by combining two ingredients: (1) a characterization of the complexity of a constraint language as $\deg(\Gamma)$, via the degree of the characteristic polynomials, and (2) reductions between different problems Max CSP(Γ) and Max CSP(Γ') by implementing constraints of one language by combinations of constraints from the other. While both ingredients have been used in isolation [10, 11, 21, 26, 33], their combination is novel and is the key to understanding compressibility. To comprehend how characteristic polynomials help to compress an instance of Max CSP(Γ), observe that since the characteristic polynomial gives 1 when a constraint is satisfied and 0 otherwise, the total value of satisfied constraint applications can be written as a weighted sum of applications of characteristic polynomials. If $\deg(\Gamma) = k$, then this weighted sum contains $\mathcal{O}(n^k)$ distinct monomials. An instance can therefore be compressed by expanding this weighted sum, and storing the coefficient of each monomial. If all weights in the input instance are bounded by $n^{\mathcal{O}(1)}$, each coefficient will have value $n^{\mathcal{O}(1)}$ and can therefore be encoded in $\mathcal{O}(\log n)$ bits.

Our lower bounds are obtained by parameterized reductions between Max CSPs in which the number of variables does not grow significantly. By a careful analysis of the terms of the characteristic polynomial, we show that if $\deg(\Gamma) = \deg(\Gamma')$, then constraint applications from Γ can effectively be simulated by combinations of constraints from Γ'. Here, we use the framework of *implementations* from an earlier work [21]. Since the characteristic polynomial of d-CNF clauses has degree d, this yields a reduction from d-CNF-SAT to Max CSP(Γ) for $\deg(\Gamma) = d$ that preserves the asymptotic size of the variable set, therefore transferring the cited lower bound for d-CNF-SAT [14] to Max CSP(Γ). A similar reduction is also used for our positive results, to turn the monomial-based compression sketched above into a self-reduction which outputs an instance of the original problem.

[1] While some recent work on sparsification for CSPs allows infinite constraint languages Γ [18], they are not interesting from our perspective as $\deg(\Gamma) = +\infty$.

Consequences for exponential-time algorithms. The framework we develop for parameterized reductions among Max CSPs also has consequences for exponential-time algorithms, which we believe to be of independent interest. The *Max 3-SAT Hypothesis* [26] states that Max 3-CNF-SAT with n variables cannot be solved in time $\mathcal{O}(2^{(1-\varepsilon)n})$ for any $\varepsilon > 0$ (cf. [1, 5]). Our reductions imply that this hypothesis is *equivalent* to the version where Max 3-CNF-SAT is replaced by Max CSP(Γ) for any constraint language Γ with $\deg(\Gamma) = 3$ in which negated literals can be expressed (§2.2). In particular, the Max 3-SAT hypothesis is equivalent to the statement that Max E3-Lin cannot be solved in time $\mathcal{O}(2^{(1-\varepsilon)n})$. What is more, for any $k \geq 2$, our reductions uncover an equivalence class of NP-hard problems whose optimal exponential-time running times coincide with the one for Max k-SAT.

Related work. Representations of Boolean functions by characteristic polynomials have been studied frequently in the literature [3, 4, 26, 28, 29, 31, 32] revealing, e.g., a relation between the degree of the representation and the decision tree complexity [29]. Algorithms for CSPs via their characteristic polynomials were first given by Williams [33]. He used the split-and-list technique to give accelerated exponential-time algorithms for Max 2-SAT and Max CSP(Γ) for $\deg(\Gamma) = 2$. In recent work, Lincoln, Williams, and Vassilevska Williams [26] give an exponential-time split-and-list reduction from Max CSP(Γ) for $\deg(\Gamma) = k$ to detecting an ℓ-hyperclique in a k-uniform hypergraph, for $\ell > k$, in support of the (k, ℓ)-Hyperclique Hypothesis, which states that detecting such a hyperclique in an n-vertex input requires time $n^{\ell - o(1)}$ on a Word-RAM with $\mathcal{O}(\log n)$-bit words. If this hypothesis fails for some k and ℓ, their reduction implies that each Max CSP(Γ) problem with $\deg(\Gamma) = k$ can be solved in time $\mathcal{O}(2^{(1-\varepsilon)n})$ for some $\varepsilon > 0$. Their reductions run in exponential time and are very different from ours.

Alon et al. [2] used a different representation of Boolean functions as polynomials in the work on Max r-SAT parameterized above the guarantee. Here, the goal is to find an assignment satisfying at least $((2^r - 1)m + k)/2^r$ clauses, where m is the total number of clauses and k is the parameter. They have shown that the problem is FPT and admits a polynomial kernel.

Organization. We begin with Section 2 containing the necessary definitions and properties of CSPs, including the implementation framework. In Section 3 we explain the idea of representing constraints by polynomials and provide an algebraic background for our reductions. It is followed by Section 4, where the notion of a reduction between constraint systems is formalized, and the main reductions are presented. It serves as a toolbox for proving the main results for compression (Section 5) and exponential-time algorithms (Section 6). The proofs of statements indicated with (★) can be found in the full version [19].

2 Preliminaries

For a set S and integer $d \geq 0$, let $\binom{S}{d}$ be the set of all size-d subsets of S. We use $[n]$ as a shorthand for $\{1, \ldots, n\}$. A k-ary constraint is a function $f\colon \{0,1\}^k \to \{0,1\}$. We refer to k as the arity of f, denoted $\textsc{ar}(f)$. We always assume that the domain is Boolean. A constraint f is satisfied by an input $s \in \{0,1\}^k$ if $f(s) = 1$. A constraint language (sometimes called constraint family) Γ is a finite collection of constraints $\{f_1, f_2, \ldots, f_\ell\}$, potentially with different arities. A *constraint application*, of a k-ary constraint f to a set of n Boolean variables, is a triple $\langle f, (i_1, i_2, \ldots i_k), w\rangle$, where the indices $i_j \in [n]$ select k of the n Boolean variables to whom the constraint is applied, and w is a weight, described formally below. The variables can be repeated in a single application.

▶ **Definition 2.1.** *A constraint system is a pair $CS(\Gamma, \mathbb{W})$, where Γ is a constraint language and $\mathbb{W}$ (for* weight range*) is either $\mathbb{Z}$ or $\mathbb{N}$. An instance (or formula) of $CS(\Gamma, \mathbb{W})$ is a set of constraint applications from Γ over a common set of variables, each application having a weight from $\mathbb{W}$.*

We denote the number of constraint applications in a formula Φ by $|\Phi|$ and the sum of absolute values of all weights in Φ by $||\Phi||$. For an assignment x, that is, a mapping from the set of variables to $\{0,1\}$, the integer $\Phi(x)$ is the sum of weights of the constraint applications satisfied by x.

In the decision problem Max CSP$(\Gamma, \mathbb{W}, c)$ we are given a formula Φ from CS$(\Gamma, \mathbb{W})$ over n variables such that $||\Phi|| \leq n^c$, together with the integer t, and we ask if there is an assignment x such that $\Phi(x) \geq t$. We specify the constant c to be accurate about the specific decision problems for which we show hardness results (the formal definitions of parameterized problems and compression are given in Section 2.4). When it does not lead to confusion, e.g., when some property holds for all c, we refer to this family of problems shortly as Max CSP$(\Gamma, \mathbb{W})$. Whenever we use the $\mathcal{O}$-notation, we do it with respect to a fixed problem, that is, we treat Γ and c as constants. The most commonly studied case is expressed by $\mathbb{W} = \mathbb{N}$ [13, 21, 27], where the weights can be interpreted as repetitions of constraint applications. It is important to make this distinction because it can be the case that Max CSP$(\Gamma, \mathbb{N})$ is polynomially solvable whereas Max CSP$(\Gamma, \mathbb{Z})$ is NP-hard [20]. Although our main reduction framework works for $\mathbb{W} = \mathbb{Z}$, we are able to transfer the compression lower bounds to the case $\mathbb{W} = \mathbb{N}$ as long as Max CSP$(\Gamma, \mathbb{N})$ is NP-hard.

Another decision problem that is related to constraint systems is Exact CSP$(\Gamma, \mathbb{W})$, where we ask whether there is an assignment for which the satisfied weights sum up exactly to a given integer [26, 33]. Even though we focus on the maximization variant, we formulate our reductions so that they could be employed for other problems over constraint systems or larger weight domains.

2.1 Types of constraints

We start by formally defining the most important constraints and constraint properties. They allow us to formulate the dichotomy theorem for Max CSP. We use the Boolean notation for negation, i.e., $\neg x = 1 - x$ for $x \in \{0,1\}$.

- A constraint is trivial if it is either always 1 or always 0 regardless of the arguments.
- The unary constraints T and F are given by $T(x) = x$ and $F(x) = \neg x$.
- OR_k and AND_k are k-ary constraints, such that $\mathrm{OR}_k(x_1, \ldots, x_k) = \bigvee_{i=1}^{k} x_i$ and analogously $\mathrm{AND}_k(x_1, \ldots, x_k) = \bigwedge_{i=1}^{k} x_i$. The Not-All-Equal constraint is defined as $\mathrm{NAE}_k(x_1, \ldots, x_k) = \mathrm{OR}_k(x_1, \ldots, x_k) \wedge \mathrm{OR}_k(\neg x_1, \ldots, \neg x_k)$.
- XOR_k is a k-ary constraint defined as $\mathrm{XOR}_k(x_1, \ldots, x_k) = x_1 + \ldots + x_k \bmod 2$. We abbreviate $\mathrm{XOR} = \mathrm{XOR}_2$.
- A constraint f is 0-valid (resp. 1-valid) if $f(0, 0, \ldots, 0) = 1$ (resp. $f(1, 1, \ldots, 1) = 1$).
- A constraint f is 2-monotone if $f(x_1, x_2, \ldots, x_k) = (x_{i_1} \wedge x_{i_2} \wedge \cdots \wedge x_{i_p}) \vee (\neg x_{j_1} \wedge \neg x_{j_2} \wedge \cdots \wedge \neg x_{j_q})$, for some $p, q \geq 0$, $(p, q) \neq (0, 0)$, i.e., f is equivalent to a DNF-formula with at most two terms: one containing only positive literals and the other containing only negative literals.
- A constraint f is C-closed (complementation-closed) if for every assignment $x \in \{0,1\}^{\mathrm{AR}(f)}$, $f(x) = f(\bar{x})$, where $\bar{x}$ stands for the bit-wise complement of x.
- A constraint f is symmetric if for any two assignments $x_1, x_2 \in \{0,1\}^{\mathrm{AR}(f)}$ having the same number of ones, it holds that $f(x_1) = f(x_2)$.

A constraint language Γ is called 0-valid, 1-valid, 2-monotone, C-closed, or symmetric, if all non-trivial constraints in Γ satisfy the respective property. We call Γ *non-trivial* if it contains at least one non-trivial constraint. This regime is convenient for formulating the fundamental dichotomy theorem for Boolean Max CSP. For our purposes it is only important that APX-hardness entails NP-hardness.

▶ **Theorem 2.2** ([21, Theorem 2.11], cf. [10]). *Max CSP*$(\Gamma, \mathbb{N})$ *is solvable in polynomial time if* Γ *is either 0-valid, 1-valid, or 2-monotone. Otherwise, the problem is APX-hard.*

2.2 Closures of constraint languages

In some CSPs we are allowed to write constraint applications containing constants or negations of variables, which makes them more convenient to process. We formalize these properties with the notion of a *language closure*.

▶ **Definition 2.3.** *Let f be a k-ary constraint and let g be a d-ary constraint. We say that g is expressible by f with constants if the identity $g(x_1, x_2, \ldots, x_d) = f(\xi_1, \xi_2, \ldots, \xi_k)$ holds for a tuple $(\xi_1, \xi_2, \ldots, \xi_k)$, where each ξ_j is either a variable x_i for some $i \in [d]$ or one of the constants 0, 1.*

We say that g is expressible by f with literals if such an identity holds for a tuple $(\xi_1, \xi_2, \ldots, \xi_k)$, where each ξ_j is a literal: either a variable x_i or its negation $\neg x_i$ for $i \in [d]$.

For a constraint language Γ we introduce the following closures:

- the language $\Gamma^{T,F}$ contains all functions expressible by $f \in \Gamma$ with constants,
- the language Γ^{LIT} contains all functions expressible by $f \in \Gamma$ with literals,
- the language Γ^{NEG} is the *negation-wise closure* of Γ, i.e., $\Gamma^{NEG} = \bigcup_{f \in \Gamma} \{f, \neg f\}$.

It is easy to see that the closures satisfy $(\Gamma^{T,F})^{T,F} = \Gamma^{T,F}$, $(\Gamma^{LIT})^{LIT} = \Gamma^{LIT}$, $(\Gamma^{NEG})^{NEG} = \Gamma^{NEG}$. We will be particularly interested in those constraint languages in which negated literals can be expressed, as in, e.g., d-CNF-SAT or d-NAE-SAT. These are the languages that satisfy $\Gamma = \Gamma^{LIT}$. Below we present examples on how to express important CSPs using our definitions.

- d-CNF-SAT $= \mathrm{CSP}(\Gamma_{d\text{-SAT}})$ for $\Gamma_{d\text{-SAT}} = \{\mathrm{OR}_d\}^{LIT}$,
- d-NAE-SAT $= \mathrm{CSP}(\{\mathrm{NAE}_d\}^{LIT})$,
- Max Ed-Lin $=$ Max CSP$(\{\mathrm{XOR}_d\}^{NEG}, \mathbb{N})$,
- Max Cut $=$ Max CSP$(\{\mathrm{XOR}\}, \mathbb{N})$,
- Max DiCut $=$ Max CSP$(\{f\}, \mathbb{N})$ for $f(x_1, x_2) = x_1 \wedge \neg x_2$.

2.3 Constraint implementations

We describe the technique behind Theorem 2.2 [21]. The idea is to *implement* a constraint f by a collection of other constraints, so that satisfying f is equivalent to maximizing the number of satisfied constraints in that collection. It allows us to express formulas from Max CSP$(\Gamma_1, \mathbb{N})$ by those from Max CSP$(\Gamma_2, \mathbb{N})$, as long as constraints in Γ_1 can be implemented by those in Γ_2.

The caveat is that each implementation may introduce new auxiliary variables whereas for our purposes we need reductions that increase the number of variables only by a multiplicative constant. Therefore the reductions by Khanna et al. [21] do not transfer compressibility bounds; we will use the implementations in a different way. On the other hand, our reductions do not preserve approximation factors.

▶ **Definition 2.4** ([21, Definition 3.1]). *A collection of unit-weighted constraint applications $C_1, C_2, \dots C_m$ over a set of variables $x = \{x_1, x_2, \dots, x_p\}$ called primary variables and $y = \{y_1, y_2, \dots, y_q\}$ called auxiliary variables, is an α-implementation of a constraint $f(x)$ for a positive integer α if the following conditions hold.*

1. *Any assignment to x and y satisfies at most α constraint applications from $C_1, C_2, \dots C_m$.*
2. *$\forall x$ such that $f(x) = 1$, $\exists y$ such that exactly α constraint applications are satisfied.*
3. *$\forall x, y$ such that $f(x) = 0$, at most $\alpha\ -1$ constraint applications are satisfied.*

An α-implementation is called strict *if for every assignment of the primary variables x such that $f(x) = 0$, there exists an assignment of the auxiliary variables y such that exactly $\alpha\ -1$ constraint applications are satisfied.*

We say that a constraint language Γ implements a constraint f if there exists an α-implementation of f using constraints of Γ for some constant α. We use $\Gamma \implies f$ to denote that Γ implements f and $\Gamma \stackrel{s}{\implies} f$ when Γ strictly implements f. The above notation is also extended to allow the target to be a family of constraints. The following lemma encapsulates a toolbox of implementations which we will rely on.

▶ **Lemma 2.5.** *If Γ is a non-trivial constraint language such that*

1. *Γ is C-closed and not 0-valid (or equivalently not 1-valid), then $\Gamma \implies XOR$,*
2. *Γ is neither 0-valid, 1-valid, nor C-closed, then $\Gamma \implies \{T, F\}$,*
3. *Γ is neither 0-valid, 1-valid, nor 2-monotone, then $\Gamma \implies XOR$.*

In order to prove it, we need to refer to several statements from [21], beginning from the transitivity of strict implementations. Then we restate three lemmas that imply points (1, 2) directly, and point (3) is obtained via transitivity.

▶ **Lemma 2.6** ([21, Lemma 3.5]). *If $\Gamma_1 \stackrel{s}{\implies} \Gamma_2$ and $\Gamma_2 \stackrel{s}{\implies} \Gamma_3$, then $\Gamma_1 \stackrel{s}{\implies} \Gamma_3$.*

▶ **Lemma 2.7** ([21, Lemma 4.5]). *Let f be a non-trivial constraint which is C-closed and is not 0-valid (or equivalently not 1-valid). Then $\{f\} \stackrel{s}{\implies} XOR$.*

▶ **Lemma 2.8** ([21, Lemma 4.6]). *Let f_0, f_1, and g be non-trivial constraints, possibly identical, which are not 0-valid, not 1-valid, and not C-closed, respectively. Then $\{f_0, f_1, g\} \stackrel{s}{\implies} \{T, F\}$.*

▶ **Lemma 2.9** ([21, Lemma 4.11]). *Let f be a constraint which is not 2-monotone. Then $\{f, T, F\} \stackrel{s}{\implies} XOR$.*

Proof of Lemma 2.5. Claims (1, 2) follow from Lemmas 2.7 and 2.8, respectively. To see claim (3), first observe that if Γ is C-closed, then we can again use Lemma 2.7. Otherwise, by Lemma 2.8 we have $\Gamma \stackrel{s}{\implies} \{T, F\}$. Next, we take advantage of transitivity (Lemma 2.6) and implement XOR with Lemma 2.9. ◀

2.4 Parameterized complexity

A *parameterized problem* is a decision problem in which every input has an associated positive integer *parameter* that captures its complexity in some well-defined way. In our study of CSPs we use the number of variables as the parameter, but other choices have been considered [16, 22, 23]. For a parameterized problem $A \subseteq \Sigma^* \times \mathbb{N}$, a decision problem $B \subseteq \Sigma^*$, and a function $f\colon \mathbb{N} \to \mathbb{N}$, a *compression* of A into B of size f is an algorithm that, on input $(x, k) \in \Sigma^* \times \mathbb{N}$, takes time polynomial in $|x| + k$ and outputs an instance $y \in \Sigma^*$ such that $(x, k) \in A$ if and only if $y \in B$, and such that $|y| \leq f(k)$. A *kernelization* algorithm of size f for problem A reduces any instance (x, k) to an $f(k)$-sized equivalent instance of the *same problem* in polynomial time.

3 Characteristic polynomials

In this section we provide the technique necessary for expressing one constraint system by another without introducing too many auxiliary variables. This insight is based on interpreting constraints as multilinear polynomials.

▶ **Definition 3.1.** *For a k-ary constraint $f\colon \{0,1\}^k \to \{0,1\}$ its characteristic polynomial P_f is the unique k-ary multilinear polynomial over $\mathbb{R}$ satisfying $f(x) = P_f(x)$ for any $x \in \{0,1\}^k$.*

It is easy to construct P_f. First define $P_s(x_1, x_2, \ldots, x_k) = \prod_{i=1}^{k} R_i^s(x_i)$ for a vector $s \in \{0,1\}^k$, where $R_i^s(x) = x$ if $s_i = 1$ and $R_i^s(x) = 1 - x$ otherwise. Formally, P_s is given by the sequence of coefficients obtained by expanding all parentheses; they are all integers. It holds that $P_s(s) = 1$, while $P_s(x) = 0$ for any $x \neq s$. For a constraint f its characteristic polynomial is given as $P_f(x_1, x_2, \ldots, x_k) = \sum_{s:\, f(s)=1} P_s(x_1, x_2, \ldots, x_k)$. It is known that no other multilinear polynomial can take identical values on $\{0,1\}^k$ [29, 33]. This also means we can interchangeably analyze polynomials as formal objects and as functions on $\{0,1\}^k$.

▶ **Observation 3.2.** *The coefficients of any characteristic polynomial P_f are integers.*

Let $\deg(f) = \deg(P_f)$ and $\deg(\Gamma) = \max_{f\in\Gamma} \deg(f)$. For a k-ary constraint f we refer to the coefficient at the unique k-ary monomial in P_f as the leading coefficient. The leading coefficient is non-zero if and only if $\deg(P_f) = k$. If g is expressible by f with literals, then we can obtain P_g from P_f by replacing each literal with negation $\neg x_i$ by $1 - x_i$ and expanding the parentheses within monomials. If g is expressible by f with constants, then we just substitute 0 or 1 for particular variables and remove monomials containing 0. These transformations imply $\deg(\Gamma^{T,F}) = \deg(\Gamma^{LIT}) = \deg(\Gamma^{NEG}) = \deg(\Gamma)$.

As an example, consider Max 3-NAE-SAT. The function $\mathrm{NAE}_3(x_1, x_2, x_3)$ has the degree-2 characteristic polynomial $x_1 + x_2 + x_3 - x_1x_2 - x_1x_3 - x_2x_3$, which allows us to construct a compression for Max 3-NAE-SAT of size $\mathcal{O}(n^2 \log n)$ by summing coefficients at all $\mathcal{O}(n^2)$ monomials. On the other hand, $\mathrm{OR}_2(x_1, x_2) = x_1 + x_2 - x_1x_2 = \mathrm{NAE}_3(x_1, x_2, 0)$, which indicates that solving Max 3-NAE-SAT should not be easier than Max 2-CNF-SAT. We will formalize these arguments in the next section.

We show that the set of characteristic polynomials of all constraints expressible by f with constants spans the linear space of multilinear polynomials over $\mathbb{Q}$ with degrees at most $\deg(f)$. We first prove that this set contains polynomials of all degrees up to $\deg(f)$ and then use them to express a basis of the linear space.

▶ **Lemma 3.3 (★).** *Let f be a k-ary constraint. For any $1 \leq d \leq \deg(f)$ there exists a d-ary constraint g expressible by f with constants, such that its characteristic polynomial P_g has degree exactly d, i.e., its leading coefficient is non-zero.*

▶ **Lemma 3.4.** *Let f be a non-trivial constraint and P be a multilinear polynomial over $\mathbb{Q}$ on ℓ variables of degree $0 \leq d \leq deg(f)$. There exists a sequence of constraint applications $\langle f_i, (j_i^1, \ldots, j_i^{AR(f_i)}), \alpha_i\rangle_{i=1}^{M}$ on ℓ variables, where each constraint f_i is expressible by f with constants, and $\alpha_i \in \mathbb{Q}$, such that the following polynomial identity holds.*

$$P(x_1, \ldots, x_\ell) = \sum_{i=1}^{M} \alpha_i \cdot P_{f_i}(x_{j_i^1}, \ldots, x_{j_i^{AR(f_i)}}).$$

Before proving this claim, let us demonstrate it on a less obvious example than the one from above with NAE_3 and OR_2. Let $P(x_1, x_2, x_3)$ be the characteristic polynomial of the constraint $\mathrm{OR}_3(x_1, x_2, \neg x_3)$, derived using the method described below Definition 3.1:

$$\begin{aligned} P(x_1, x_2, x_3) &= x_1x_2x_3 + (1-x_1)x_2x_3 + x_1(1-x_2)x_3 + x_1x_2(1-x_3) \\ &\quad + (1-x_1)x_2(1-x_3) + x_1(1-x_2)(1-x_3) + (1-x_1)(1-x_2)(1-x_3) \\ &= 1 - x_3 + x_1x_3 + x_2x_3 - x_1x_2x_3. \end{aligned}$$

We will represent it with characteristic polynomials from $\{\mathrm{EX}_3\}^{T,F}$, where $\mathrm{EX}_3(x_1, x_2, x_3) = 1$ if and only if exactly one variable is 1. Its characteristic polynomial Q is given as:

$$\begin{aligned} Q(x_1, x_2, x_3) &= x_1(1-x_2)(1-x_3) + (1-x_1)x_2(1-x_3) + (1-x_1)(1-x_2)x_3 \\ &= x_1 + x_2 + x_3 - 2x_1x_2 - 2x_1x_3 - 2x_2x_3 + 3x_1x_2x_3. \end{aligned}$$

We can express P as the following linear combination where, e.g., $Q(x_1, x_2, 0)$ is the characteristic polynomial for $\mathrm{EX}_3(x_1, x_2, 0)$, which is a binary constraint expressible by EX_3 with constants:

$$\begin{aligned} P(x_1, x_2, x_3) = &-\frac{1}{3}Q(x_1, x_2, x_3) + \frac{1}{3}Q(x_1, x_2, 0) - \frac{1}{6}Q(x_1, x_3, 0) - \frac{1}{6}Q(x_2, x_3, 0) \\ &+ \frac{1}{6}Q(x_1, 0, 0) + \frac{1}{6}Q(x_2, 0, 0) - \frac{1}{3}Q(x_3, 0, 0) + Q(1, 0, 0). \end{aligned}$$

Proof of Lemma 3.4. We proceed by induction over the degree of P. Since f is non-trivial, it admits a satisfying assignment s_T. If P is constant, then $P(x) = \alpha \cdot f(s_T)$ for some $\alpha \in \mathbb{Q}$. Suppose now $d = \deg(P) \geq 1$. For each $S \in \binom{[\ell]}{d}$ let α_S denote the (potentially zero) coefficient in P at the monomial $\prod_{i \in S} x_i$. By Lemma 3.3 there is a d-ary constraint f_d, which is expressible by f with constants and $\deg(P_{f_d}) = d$. Let β_d denote the leading coefficient of P_{f_d}. The polynomial

$$P'(x_1, \ldots, x_\ell) = P(x_1, \ldots, x_\ell) - \sum_{\substack{\{i_1, \ldots, i_d\} = S \in \binom{[\ell]}{d} \\ i_1 < \ldots < i_d}} \frac{\alpha_S}{\beta_d} \cdot P_{f_d}(x_{i_1}, \ldots, x_{i_d})$$

has no monomials of degree d, since each term in the sum subtracts exactly one of them. The polynomial P' has degree at most $d-1$, so we can apply the induction hypothesis to it and represent P' as a linear combination of characteristic polynomials of constraints from $\{f\}^{T,F}$. We obtain the claim by adding these polynomials to the sum above. ◀

Since f_i and P_{f_i} coincide as functions on $\{0,1\}^{\mathrm{AR}(f_i)}$, Lemma 3.4 allows us to represent any constraint of degree at most $\deg(f)$ as a linear combination of constraints from $\{f\}^{T,F}$.

▶ **Proposition 3.5.** *Let g, f be constraints, such that f is non-trivial and $\deg(g) \leq \deg(f)$. There exists a sequence of constraint applications $\langle f_i, (j_i^1, \ldots, j_i^{\mathrm{AR}(f_i)}), \alpha_i \rangle_{i=1}^M$ on $\mathrm{AR}(g)$ variables, where each constraint f_i is expressible by f with constants, and $\alpha_i \in \mathbb{Q}$, such that $g(x_1, \ldots, x_{\mathrm{AR}(g)}) = \sum_{i=1}^M \alpha_i \cdot f_i(x_{j_i^1}, \ldots, x_{j_i^{\mathrm{AR}(f_i)}})$ for all Boolean assignments.*

This resembles the idea of implementation, where we additionally equip constraints with (potentially negative) rational weights, but does not require introducing new variables.

4 Reductions between constraint systems

We first formalize our notion of reduction. The objects we work with are constraint systems and the reductions between them imply analogous relations between the associated decision problems. The reduction is crafted in such a way that it preserves the numbers of variables and constraints up to a constant factor, and the total weight up to a polynomial factor.

▶ **Definition 4.1.** *A linear transformation from a constraint system $CS(\Gamma_1, \mathbb{W}_1)$ to another constraint system $CS(\Gamma_2, \mathbb{W}_2)$ is a polynomial-time procedure that given a formula Φ_1 of $CS(\Gamma_1, \mathbb{W}_1)$ over n_1 variables and integer t_1, returns a formula $\Phi_2 \in CS(\Gamma_2, \mathbb{W}_2)$ over n_2 variables and integer t_2, so that the following conditions hold:*

1. $n_2 = \mathcal{O}(n_1)$,
2. $|\Phi_2| = \mathcal{O}(|\Phi_1| + n_1)$,
3. $||\Phi_2|| \leq ||\Phi_1|| \cdot n_1^{\mathcal{O}(1)}$,
4. $\exists_x \Phi_1(x) = t_1 \iff \exists_y \Phi_2(y) = t_2$,
5. $\exists_x \Phi_1(x) \geq t_1 \iff \exists_y \Phi_2(y) \geq t_2$.

If there exists a linear transformation from $CS(\Gamma_1, \mathbb{W}_1)$ to $CS(\Gamma_2, \mathbb{W}_2)$, we write concisely $CS(\Gamma_1, \mathbb{W}_1) \leq_{LIN} CS(\Gamma_2, \mathbb{W}_2)$. If (1) can be replaced with the stronger condition $n_2 = n_1 + \mathcal{O}(1)$, we call the transformation additive and write $CS(\Gamma_1, \mathbb{W}_1) \leq_{ADD} CS(\Gamma_2, \mathbb{W}_2)$.

Before moving forward, let us explain the importance of linear and additive transformations. We formulate two claims, which follow from the properties in Definition 4.1.

▶ **Proposition 4.2.** *If $CS(\Gamma_1, \mathbb{W}_1) \leq_{LIN} CS(\Gamma_2, \mathbb{W}_2)$ and Max CSP$(\Gamma_2, \mathbb{W}_2, c)$ admits a compression of size $\mathcal{O}(n^d)$, then Max CSP$(\Gamma_1, \mathbb{W}_1, c - \mathcal{O}(1))$ also admits a compression of size $\mathcal{O}(n^d)$.*

▶ **Proposition 4.3.** *If $CS(\Gamma_1, \mathbb{W}_1) \leq_{ADD} CS(\Gamma_2, \mathbb{W}_2)$ and Max CSP$(\Gamma_2, \mathbb{W}_2, c)$ admits an algorithm with running time $T(n)$, then Max CSP$(\Gamma_1, \mathbb{W}_1, c - \mathcal{O}(1))$ admits an algorithm with running time $T(n + \mathcal{O}(1))$.*

In particular, additive transformations preserve running times of the form $2^{(1-\varepsilon)n} n^{\mathcal{O}(1)}$.

▶ **Lemma 4.4 (★).** *Linear transformations (resp. additive transformations) are transitive.*

We continue with two simple additive transformations, which will allow us to use negations of constraints as an alternative to setting negative weights.

▶ **Lemma 4.5 (★).** *For every constraint language Γ we have*

1. *CS$(\Gamma^{NEG}, \mathbb{Z}) \leq_{ADD}$ CS$(\Gamma, \mathbb{Z})$,*
2. *CS$(\Gamma^{NEG}, \mathbb{Z}) \leq_{ADD}$ CS$(\Gamma^{NEG}, \mathbb{N})$.*

The rest of this section contains four lemmas that form a chain of reductions from CS$(\Gamma_1, \mathbb{Z})$ to CS$(\Gamma_2, \mathbb{N})$, which is valid as long as $\deg(\Gamma_1) \leq \deg(\Gamma_2)$ and Max CSP$(\Gamma_2, \mathbb{N})$ is NP-hard. First, we translate Proposition 3.5 into the language of additive transformations. In the proof we justify that one can replace rational coefficients with integer ones.

▶ **Lemma 4.6 (★).** *Let Γ_1, Γ_2 be non-trivial constraint languages satisfying $\deg(\Gamma_1) \leq \deg(\Gamma_2)$. Then CS$(\Gamma_1, \mathbb{Z}) \leq_{ADD}$ CS$(\Gamma_2^{T,F}, \mathbb{Z})$.*

Now, we formalize an intuitive notation for combining formulas. Consider two formulas Φ_1, Φ_2 over the sets of variables V_1, V_2, respectively, which might have a non-empty intersection. We define the sum of these formulas, $\Phi_1 + \Phi_2$, over the set of variables $V_1 \cup V_2$ by

taking the union of their sets of constraint applications and merging pairs of applications that share the same constraint and the same tuple of variables, i.e., replacing the pair with a single application with a weight being the sum of the respective weights. For an integer α, the formula $\alpha \cdot \Phi$ has the same constraint applications as Φ, with weights multiplied by α.

▶ **Lemma 4.7.** *Let Γ be a non-trivial constraint language, which is neither 0-valid nor 1-valid. Then $CS(\Gamma^{T,F}, \mathbb{Z}) \leq_{ADD} CS(\Gamma, \mathbb{Z})$.*

Proof. Given a formula $\Phi \in \mathrm{CS}(\Gamma^{T,F}, \mathbb{Z})$, we add two auxiliary variables x_T, x_F and we translate each constraint application $\langle \hat{f}, (j_1, \ldots, j_k), w \rangle$, where $\hat{f}$ is expressible by $f \in \Gamma$ with constants, into an application of f by replacing constants 0, 1 with variables x_T, x_F. Let us refer to this formula as $\Phi_1 \in \mathrm{CS}(\Gamma, \mathbb{Z})$ and note that $|\Phi_1| = |\Phi|$ and $||\Phi_1|| = ||\Phi||$.

In the next step, we will use implementations to impose particular conditions on x_T, x_F. We refer to x_T, x_F and all the new variables introduced within the implementations as auxiliary. Let $a = \mathcal{O}(1)$ denote their number. In the new formula we assume that the first n variables $x_1, \ldots, x_n$ are the primary variables and $x_{n+1}, \ldots, x_{n+A}$ are auxiliary. For $x \in \{0,1\}^{n+A}$ let $x|_n$ stand for the projection on the first n coordinates.

Assume first that Γ is not C-closed. Then by Lemma 2.5, point (2), we know that Γ α_1-implements function T and α_2-implements function F for some integers α_1, α_2. Let $\alpha = \alpha_1 + \alpha_2$. We implement constraint applications $T(x_T)$ and $F(x_F)$, that is, we construct formulas $\Phi_T, \Phi_F \in \mathrm{CS}(\Gamma, \mathbb{N})$ over the set of auxiliary variables, such that satisfying α constraint applications in $\Phi_T + \Phi_F$ is only possible when $x_T = 1$ and $x_F = 0$. Let $W = 2 \cdot ||\Phi|| + 1$. We define $\Phi_2 = \Phi_1 + W \cdot \Phi_T + W \cdot \Phi_F$, that is, we copy all the constraint applications from Φ_1 and add the applications from $\Phi_T + \Phi_F$ with weights multiplied by W. Recall that we have $(-||\Phi||) \leq \Phi(x) \leq ||\Phi||$ for all x. By the definition of implementation, any assignment to Φ_2 which does not satisfy $x_T = 1$ or $x_F = 0$ has value at most $(\alpha - 1) \cdot W + ||\Phi|| < \alpha\, W - ||\Phi||$. If the assignment x satisfies $x_T = 1$, $x_F = 0$, it holds that $\Phi_2(x) = \alpha\, W + \Phi(x|_n) \geq \alpha\, W - ||\Phi||$.

Now, if Γ is C-closed, then by Lemma 2.5, point (1), Γ α-implements function XOR for some constant α. We implement $\mathrm{XOR}(x_T, x_F)$, that is, we construct formula $\Phi_{\mathrm{XOR}} \in \mathrm{CS}(\Gamma, \mathbb{N})$ over the set of auxiliary variables, such that satisfying α constraint applications in Φ_{XOR} is only possible when $\mathrm{XOR}(x_T, x_F) = 1$. As before, we define $\Phi_2 = \Phi_1 + W \cdot \Phi_{\mathrm{XOR}}$, where $W = 2 \cdot ||\Phi|| + 1$. Any assignment to Φ_2 which does not satisfy $\mathrm{XOR}(x_T, x_F)$ has value at most $(\alpha - 1) \cdot W + ||\Phi|| < \alpha\, W - ||\Phi||$. For an assignment x satisfying $x_T = 1$, $x_F = 0$, it holds that $\Phi_2(x) = \alpha\, W + \Phi(x|_n) \geq \alpha\, W - ||\Phi||$. It might also be the case that $x_T = 0$, $x_F = 1$. Then, by C-closedness we have $\Phi_2(x) = \Phi_2(\bar{x}) = \alpha\, W + \Phi(\bar{x}|_n) \geq \alpha\, W - ||\Phi||$, where $\bar{x}$ is the bit-wise complement of x.

We summarize the transformation properties for both considered cases: the new number of variables is $n + \mathcal{O}(1)$, $|\Phi_2| = |\Phi| + \mathcal{O}(1)$, and $||\Phi_2|| = ||\Phi|| \cdot \mathcal{O}(1)$. If $t < -||\Phi||$, then we know that all assignments x satisfy $\Phi(x) > t$ and in such case we could return an empty formula over a singleton variable set (so the only possible value is 0) and set threshold $t' = -1$: this is an equivalent instance. To see properties (4, 5) observe that, assuming $t \geq -||\Phi||$, $\Phi(x) = t$ (resp. $\Phi(x) \geq t$) holds for some assignment x iff. $\Phi_2(y) = t'$ (resp. $\Phi_2(y) \geq t'$) holds for some assignment y, where $t' = \alpha\, W + t$. ◀

▶ **Corollary 4.8.** *Let Γ_1, Γ_2 be non-trivial constraint languages such that $\deg(\Gamma_1) \leq \deg(\Gamma_2)$ and Γ_2 is neither 0-valid nor 1-valid. Then $CS(\Gamma_1, \mathbb{Z}) \leq_{ADD} CS(\Gamma_2, \mathbb{Z})$.*

So far we have established a relation between Γ_1 and Γ_2, which allows us to transform one constraint system to another by adding only a constant number of new variables. However, it works only when we allow negative weights. The next two lemmas explain how to get rid of them, so that the hardness results can be applied to the natural setting with only non-negative weights.

▶ **Lemma 4.9** (★). *If* Γ *is a non-trivial constraint language, then* $CS(\Gamma, \mathbb{Z}) \leq_{ADD} CS(\Gamma^{LIT}, \mathbb{N})$.

▶ **Lemma 4.10** (★). *Suppose non-trivial* Γ *is neither 0-valid, 1-valid, nor 2-monotone. Then* $CS(\Gamma^{LIT}, \mathbb{N}) \leq_{LIN} CS(\Gamma, \mathbb{N})$.

In the proof of Lemma 4.9 we increase weights of all the constraint applications at once, in such a way that it changes each value of $\Phi(x)$ by the same quantity. The proof of Lemma 4.10 is similar in spirit to that of Lemma 4.7, but this time the implementation framework is used to implement a negated copy of each variable. As all the transformation above are linear, by transitivity we can summarize them into the following statement connecting constraint languages of the same degree.

▶ **Corollary 4.11.** *Let* Γ_1, Γ_2 *be non-trivial constraint languages such that* $\deg(\Gamma_1) \leq \deg(\Gamma_2)$ *and* Γ_2 *is neither 0-valid, 1-valid, nor 2-monotone. Then* $CS(\Gamma_1, \mathbb{Z}) \leq_{LIN} CS(\Gamma_2, \mathbb{N})$.

5 Consequences for compression

Having an upper bound on $\deg(\Gamma)$ already provides compression for Max CSP$(\Gamma, \mathbb{Z}, c)$, since we can represent all constraint applications as polynomials, sum the coefficients at all $\mathcal{O}\left(n^{\deg(\Gamma)}\right)$ monomials, and store the weights in $\mathcal{O}(\log n)$ bits. Corollary 4.11 transforms the monomial-representation back into a small equivalent instance of Max CSP(Γ).

▶ **Theorem 5.1** (Formalization of Theorem 1.1). *Max CSP*$(\Gamma, \mathbb{N}, c)$ *parameterized by the number of variables* n *admits a compression of size* $\mathcal{O}(n^d \log n)$ *for all* c*, where* $d = \deg(\Gamma)$*. Furthermore, there is a polynomial-time algorithm that reduces any instance of Max CSP*$(\Gamma, \mathbb{N}, c)$ *to an equivalent instance of Max CSP*$(\Gamma, \mathbb{N}, c + \mathcal{O}(1))$ *of size* $\mathcal{O}(n^d \log n)$*. The analogous statements for Max CSP*$(\Gamma, \mathbb{Z}, c)$ *also hold.*

Proof. If Γ is either trivial, 0-valid, 1-valid, or 2-monotone, then Max CSP$(\Gamma, \mathbb{N}, c)$ can be solved in polynomial time, so the compression is trivial. Suppose that it does not have any of these properties. We will prove both claims by compressing a formula $\Phi_1 \in \text{CS}(\Gamma, \mathbb{Z})$ on n variables into a formula $\Phi_2 \in \text{CS}(\Gamma, \mathbb{N})$ satisfying $|\Phi_2| = \mathcal{O}(n^d)$.

First we interpret each constraint application in Φ_1 as a polynomial of degree at most d. After summing these terms, we obtain a polynomial P with $\mathcal{O}(n^d)$ monomials, each associated with an integer weight of absolute value bounded by $||\Phi_1||$. A monomial $\Pi_{i=1}^{k} x_i$ is the characteristic polynomial for the constraint $\text{AND}_k(x_1, \ldots, x_k)$, therefore P can be treated as a formula of $\text{CS}(\Gamma_{d\text{-AND}}, \mathbb{Z})$ for $\Gamma_{d\text{-AND}}$ being the language consisting of functions AND_k for all $k \leq d$. Since $\deg(\Gamma_{d\text{-AND}}) = d$, we can apply Corollary 4.11 to obtain an equivalent formula $\Phi_2 \in \text{CS}(\Gamma, \mathbb{N})$ on $\mathcal{O}(n)$ variables, such that $|\Phi_2| = |\Phi_1| \cdot \mathcal{O}(1) + \mathcal{O}(n) = \mathcal{O}(n^d)$ and $||\Phi_2|| = ||\Phi_1|| \cdot n^{\mathcal{O}(1)}$, so each weight can be stored in $\mathcal{O}(\log n)$ bits. ◀

The self-reduction in Theorem 5.1 is almost, but not quite, a kernelization: the formal decision problem we reduce to is not the same as the original one, due to the increase in weight values. Our lower bounds are based on reducing Max d-CNF-SAT to Max CSP$(\Gamma, \mathbb{N})$ for $\deg(\Gamma) = d$. For $d \geq 3$ it is known that even the non-maximization variant d-CNF-SAT does not admit a compression of size $\mathcal{O}\left(n^{d-\varepsilon}\right)$ [14]. However, the 2-CNF-SAT problem is solvable in polynomial time and only its maximization version becomes NP-hard. We first note that Max 2-CNF-SAT also cannot have any non-trivial compression.

▶ **Lemma 5.2** (★). *Max CSP*$(\Gamma_{2\text{-}SAT}, \mathbb{N}, 3)$ *does not admit a compression of size* $\mathcal{O}\left(n^{2-\varepsilon}\right)$ *for any* $\varepsilon > 0$*, unless NP* $\subseteq$ *co-NP/poly.*

▶ **Theorem 5.3** (Formalization of Theorem 1.2). *Let non-trivial Γ be neither 0-valid, 1-valid, nor 2-monotone, and let $d = \deg(\Gamma)$. Then there is a constant c such that* Max CSP$(\Gamma, \mathbb{N}, c)$ *does not admit a compression of size $\mathcal{O}\left(n^{d-\varepsilon}\right)$ for any $\varepsilon > 0$, unless NP $\subseteq$ co-NP/poly.*

Proof. First observe that a characteristic polynomial of a constraint with $d = 1$ is linear. Since this polynomial is 0/1-valued, it can depend only on one variable, which means that the constraint is 2-monotone. Hence we can assume that $d \geq 2$.

We know that there is a constant c_d so that Max CSP$(\Gamma_{d\text{-SAT}}, \mathbb{N}, c_d)$ does not admit compression of size $\mathcal{O}\left(n^{d-\varepsilon}\right)$ unless NP $\subseteq$ co-NP/poly: for $d = 2$ it is due to Lemma 5.2 and for $d \geq 3$ it follows from the lower bound for the non-maximization variant of d-CNF-SAT [14]. As noted in Proposition 4.2, if we had such a compression for Max CSP$(\Gamma, \mathbb{N}, c)$ with sufficiently large c, then by Corollary 4.11 it would transfer to Max CSP$(\Gamma_{d\text{-SAT}}, \mathbb{N}, c_d)$. ◀

Applications to specific CSPs. Having established both the lower and the upper bound, we can refer to $\deg(\Gamma)$ as the *optimal compression exponent* for Max CSP$(\Gamma, \mathbb{N})$. We are now equipped with a handy but powerful tool for determining the optimal compressibility of Max CSP$(\Gamma, \mathbb{N})$ as this task reduces to computing the degrees of characteristic polynomials in Γ.

As an example, we apply this technique to compute the optimal compression exponent to the following three problems, which are all NP-hard for $k \geq 2$:

- Max k-NAE-SAT = Max CSP$(\{\text{NAE}_k\}^{LIT}, \mathbb{N})$,
- Max Ek-Lin = Max CSP$(\{\text{XOR}\}^{NEG}, \mathbb{N})$,
- Max k-Exact-SAT = Max CSP$(\{\text{EX}_k\}^{LIT}, \mathbb{N})$, where $\text{EX}_k(x_1, \ldots, x_k) = 1$ if and only if there is exactly one 1 in $(x_1, \ldots, x_k)$.

Since $\deg(\Gamma^{LIT}) = \deg(\Gamma^{NEG}) = \deg(\Gamma)$ it suffices to analyze the characteristic polynomials for functions NAE_k, XOR_k, and EX_k. Let $e_i(x_1, \ldots, x_k)$ denote i-th elementary symmetric polynomial, i.e., the sum of all degree-i monomials on k variables.

$$\begin{aligned}
\text{NAE}_k(x_1, \ldots, x_k) &= \textstyle\sum_{i=1}^{k-1} (-1)^{i-1} e_i(x_1, \ldots, x_k) && \text{for odd } k,\\
\text{NAE}_k(x_1, \ldots, x_k) &= \textstyle\sum_{i=1}^{k} (-1)^{i-1} e_i(x_1, \ldots, x_k) && \text{for even } k,\\
\text{XOR}_k(x_1, \ldots, x_k) &= \textstyle\sum_{i=1}^{k} (-2)^{i-1} e_i(x_1, \ldots, x_k) && \text{for all } k,\\
\text{EX}_k(x_1, \ldots, x_k) &= \textstyle\sum_{i=1}^{k} i \cdot (-1)^{i-1} e_i(x_1, \ldots, x_k) && \text{for all } k.
\end{aligned}$$

It is easy to check these formulas using the binomial theorem. We present the argument for XOR_k as an example. Suppose the number of 1s in the vector $(x_1, \ldots, x_k)$ is ℓ. Then $e_i(x_1, \ldots, x_k)$ equals $\binom{\ell}{i}$ for $i \leq \ell$ and 0 for $i > \ell$. The formula for $\text{XOR}_k(x_1, \ldots, x_k)$ becomes $\sum_{i=1}^{\ell} (-2)^{i-1} \binom{\ell}{i} = (-\frac{1}{2}) \cdot \left(\sum_{i=0}^{\ell} (-2)^i \binom{\ell}{i} - 1 \right) = (-\frac{1}{2}) \cdot ((-1)^\ell - 1)$ which is 1 for odd ℓ and 0 for even ℓ, as expected. By these identities we deduce that the optimal compression exponent for Max k-NAE-SAT is k in the even case, $k-1$ in the odd case, and the optimal compression exponent for both Max Ek-Lin and Max k-Exact-SAT is k.

An example of a non-symmetric constraint with a non-trivial upper bound on its degree is f_k, with $\text{AR}(f_k) = 3^k$, defined recursively: $f_0(x) = x$, and

$$f_k(x_1, \ldots x_{3^k}) = \text{NAE}_3\big(f_{k-1}(x_1, \ldots x_{3^{k-1}}), f_{k-1}(x_{3^{k-1}+1}, \ldots x_{2 \cdot 3^{k-1}}), f_{k-1}(x_{2 \cdot 3^{k-1}+1}, \ldots x_{3^k})\big).$$

Because $\deg(\text{NAE}_3) = 2$, we have $\deg(f_k) = 2^k = \text{AR}(f_k)^{\log_3(2)}$ [29].

It is tempting to seek a concise characterization of Γ for which one can obtain a non-trivial bound on $\deg(\Gamma)$ and therefore a non-trivial compression for Max CSP$(\Gamma, \mathbb{N})$, where by non-trivial we mean a bound of the form $\deg(f) \leq \text{AR}(f) - 1$ for functions depending on all the coordinates. By generalizing the argument for NAE_k, it can be shown that $\deg(f) \leq$

$\text{AR}(f) - 1$ when the number of vectors that satisfy f and have an even number of 1s, is equal to the number of satisfying vectors with an odd number of 1s. Unfortunately, as far as we are aware no characterization is known describing all Γ with a non-trivial bound on $\deg(\Gamma)$, not even for symmetric polynomials induced by symmetric constraints. There exist some interesting partial results though, e.g., that if the number of variables k is a prime minus one then the degree is always k, and in general $\deg(f) \geq k - \mathcal{O}(k^{0.548})$ [32]. On the other hand, there are infinitely many symmetric functions for which $\deg(f) = \text{AR}(f) - 3$ [32]. When it comes to non-symmetric functions, there exist infinitely many examples with $\deg(f) \leq \log(\text{AR}(f))$ [30], which is asymptotically the lowest upper bound possible [29].

Compression lower bound for negative weights. For the sake of completeness, we show that Max CSP$(\Gamma, \mathbb{Z})$ admits an analogous classification as Max CSP$(\Gamma_{,}, \mathbb{N})$ that is, whenever Max CSP$(\Gamma, \mathbb{Z})$ is NP-hard, then the upper bound from Theorem 5.1 is essentially tight. The dichotomy theorem for $\mathbb{W} = \mathbb{Z}$ can be stated in a simpler manner, as the problem becomes NP-hard whenever $\deg(\Gamma) \geq 2$ [20] and the case $\deg(\Gamma) = 1$ reduces to linear function maximization. This dichotomy will follow also from the reduction below. First, we show that we can drop the assumption on the language being non 0-valid/1-valid in Corollary 4.8.

▶ **Lemma 5.4.** *Let Γ_1, Γ_2 be non-trivial constraint languages such that $\deg(\Gamma_1) \leq \deg(\Gamma_2)$. Then $CS(\Gamma_1, \mathbb{Z}) \leq_{ADD} CS(\Gamma_2, \mathbb{Z})$.*

Proof. Since f and $\neg f$ cannot be 0-valid (or 1-valid) at the same time, the language Γ_2^{NEG} is neither 0-valid nor 1-valid. As we also have $\deg(\Gamma_2) = \deg(\Gamma_2^{NEG})$, we can apply Corollary 4.8 to Γ_1 and Γ_2^{NEG}, obtaining $\text{CS}(\Gamma_1, \mathbb{Z}) \leq_{ADD} \text{CS}(\Gamma_2^{NEG}, \mathbb{Z})$. By Lemma 4.5, point (1), we get $\text{CS}(\Gamma_2^{NEG}, \mathbb{Z}) \leq_{ADD} \text{CS}(\Gamma_2, \mathbb{Z})$, which proves the claim. ◀

▶ **Theorem 5.5.** *Let non-trivial Γ be such that $d = \deg(\Gamma) \geq 2$. Then there is a constant c such that Max CSP$(\Gamma, \mathbb{Z}, c)$ does not admit a compression of size $\mathcal{O}\left(n^{d-\varepsilon}\right)$ for any $\varepsilon > 0$, unless NP $\subseteq$ co-NP/poly.*

Proof. From Lemma 5.4 we know that $\text{CS}(\Gamma_{d\text{-SAT}}, \mathbb{Z}) \leq_{ADD} \text{CS}(\Gamma, \mathbb{Z})$. Therefore an $\mathcal{O}\left(n^{d-\varepsilon}\right)$-size compression for Max CSP$(\Gamma, \mathbb{Z}, c)$ with sufficiently large c entails the same for Max CSP$(\Gamma_{d\text{-SAT}}, \mathbb{Z}, c - \mathcal{O}(1))$, which implies NP $\subseteq$ co-NP/poly (Lemma 5.2 for $d = 2$ and [14] for $d \geq 3$). ◀

6 Consequences for exponential-time algorithms

As mentioned before, our framework of reductions can be used to preserve the exponential running time as well. Namely, if $CS(\Gamma_1, \mathbb{W}_1) \leq_{ADD} CS(\Gamma_2, \mathbb{W}_2)$, then an algorithm for Max CSP$(\Gamma_2, \mathbb{W}_2, c)$ with running time $T(n)$ entails an algorithm for Max CSP$(\Gamma_2, \mathbb{W}_2, c - \mathcal{O}(1))$ with running time $T(n + \mathcal{O}(1))$. All the constructed transformations, except from $\text{CS}(\Gamma^{LIT}, \mathbb{N}) \leq_{LIN} \text{CS}(\Gamma, \mathbb{N})$ (Lemma 4.10), are additive and in particular they work as long as negative weights are allowed. Alternatively, we can take advantage of other properties of particular constraint languages to remove the negative weights.

▶ **Lemma 6.1.** *Let $d = \deg(\Gamma) \geq 2$. Then $CS(\Gamma_{d\text{-SAT}}, \mathbb{N}) \leq_{ADD} CS(\Gamma, \mathbb{Z}) \leq_{ADD} CS(\Gamma_{d\text{-SAT}}, \mathbb{N})$, that is, these constraint systems are equivalent with respect to the relation $\leq_{ADD}$.*

Proof. We take advantage of the fact that $\Gamma_{d\text{-SAT}}$ can express negated literals, i.e., $(\Gamma_{d\text{-SAT}})^{LIT} = \Gamma_{d\text{-SAT}}$. We have the following cycle of reductions.

$$\begin{aligned} \mathrm{CS}(\Gamma_{d\text{-SAT}}, \mathbb{Z}) &\leq_{ADD} \mathrm{CS}(\Gamma_{d\text{-SAT}}, \mathbb{N}) && \text{Lemma 4.9 for } (\Gamma_{d\text{-SAT}})^{LIT} = \Gamma_{d\text{-SAT}} \\ &\leq_{ADD} \mathrm{CS}(\Gamma, \mathbb{Z}) && \text{Lemma 5.4} \\ &\leq_{ADD} \mathrm{CS}(\Gamma_{d\text{-SAT}}, \mathbb{Z}) && \text{Lemma 5.4} \end{aligned}$$ ◀

▶ **Theorem 6.2.** *For each $d \geq 2$ and any constant $\alpha > 1$ either all the following problems admit an $\alpha^n n^{\mathcal{O}(1)}$ algorithm for all c, or none of them do:*

1. *Max CSP$(\Gamma_{d\text{-SAT}}, \mathbb{N}, c)$,*
2. *Max CSP$(\Gamma, \mathbb{Z}, c)$ for any Γ with $\deg(\Gamma) = d$,*
3. *Max CSP$(\Gamma, \mathbb{N}, c)$ for any Γ with $\deg(\Gamma) = d$ such that $\Gamma^{NEG} = \Gamma$ or $\Gamma^{LIT} = \Gamma$.*

Proof. As noted in Proposition 4.3, if $\mathrm{CS}(\Gamma_1, \mathbb{W}_1) \leq_{ADD} \mathrm{CS}(\Gamma_2, \mathbb{W}_2)$ and Max CSP$(\Gamma_2, \mathbb{W}_2, c)$ admits an algorithm with running time $\alpha^n n^{\mathcal{O}(1)}$, then the same holds for Max CSP$(\Gamma_1, \mathbb{W}_1, c - \mathcal{O}(1))$. The equivalency between (1) and (2) has been proven in Lemma 6.1. Since (2) is more general than (3), it suffices to reduce (2) into (3). Lemma 4.5, point (2), provides the reduction $\mathrm{CS}(\Gamma, \mathbb{Z}) \leq_{ADD} \mathrm{CS}(\Gamma^{NEG}, \mathbb{N})$ for the case $\Gamma^{NEG} = \Gamma$ and Lemma 4.9 provides the reduction $\mathrm{CS}(\Gamma, \mathbb{Z}) \leq_{ADD} \mathrm{CS}(\Gamma^{LIT}, \mathbb{N})$ for the case $\Gamma^{LIT} = \Gamma$. ◀

First corollary of this theorem is that problems of form Max CSP$(\Gamma, \mathbb{Z})$ are divided into equivalence classes with respect to the optimal running time. In particular, solving any Max CSP$(\Gamma, \mathbb{Z})$ with $\deg(\Gamma) \geq 3$ in time $\mathcal{O}(2^{(1-\varepsilon)n})$ for any $\varepsilon > 0$ contradicts the Max 3-SAT Hypothesis. Also, the hypothesis remains equivalent if we replace Max 3-CNF-SAT with Max 3-Lin SAT or Max 3-Exact SAT with only positive weights, because their constraint languages satisfy $\Gamma^{NEG} = \Gamma$ and $\Gamma^{LIT} = \Gamma$, respectively. Another corollary is that improving the running time $\mathcal{O}\big(2^{\frac{\omega}{3} n}\big)$ for Max Cut or Max DiCut with integer weights or Max 3-NAE-SAT with positive weights would imply an analogous breakthrough for Max 2-CNF-SAT.

Alman and Williams [1] have noted that it is not known how to improve the running time for Max 3-CNF-SAT, even for instances with a linear number of clauses. They have therefore formulated a stronger hypothesis. The Sparse Max 3-SAT Hypothesis states that there exists $c > 0$ such that Max 3-CNF-SAT with cn clauses does not admit an $\mathcal{O}(2^{(1-\varepsilon)n})$-time algorithm for any $\varepsilon > 0$. Similarly, their Sparse Max 2-SAT Hypothesis states that one cannot beat running time $\mathcal{O}\big(2^{\frac{\omega}{3} n}\big)$ for Max 2-CNF-SAT with cn clauses. Observe that our reductions preserve the property of having $\mathcal{O}(n)$ different constraint applications (condition (2) in Definition 4.1). Therefore as long as we allow negative weights at constraint applications, we can replace Max 2-CNF-SAT (resp. Max 3-CNF-SAT) in this hypothesis with Max CSP$(\Gamma, \mathbb{Z})$ for any constraint language Γ of degree 2 (resp. 3) to obtain an equivalent statement.

7 Conclusions and open problems

We have provided a complete characterization of the optimal compression for Max CSP(Γ) in the case of a Boolean domain. A natural question arises about larger domains. Our approach does not transfer even to the case with a domain of size 3, since there is no unique way to represent functions $\{0,1,2\}^k \to \{0,1\}$ as polynomials. One may consider, e.g., embedding to a Boolean domain or using non-multilinear polynomials, but it is not clear which approach leads to the optimal degree and how to find accompanying lower bounds.

On the exponential-time front, we have shown that Max d-CNF-SAT is as hard as any Max CSP of degree d as long as negative weights are allowed. Although we were able to get rid of the latter assumption in several cases, there is still a gap in this classification: does improving the running time for any degree-d Max CSP$(\Gamma, \mathbb{N})$ imply an improvement for Max d-CNF-SAT?

References

1. Josh Alman and Virginia Vassilevska Williams. OV Graphs Are (Probably) Hard Instances. In Thomas Vidick, editor, *Proceedings of the 11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 83:1–83:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2020.83.
2. Noga Alon, Gregory Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo. Solving MAX-r-SAT above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011.
3. David A. Mix Barrington, Richard Beigel, and Steven Rudich. Representing Boolean functions as polynomials modulo composite numbers. *Computational Complexity*, 4(4):367–382, 1994. doi:10.1007/BF01263424.
4. Richard Beigel. The polynomial method in circuit complexity. In *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*, pages 82–95. IEEE Computer Society, 1993. doi:10.1109/SCT.1993.336538.
5. Karl Bringmann, Nick Fischer, and Marvin Künnemann. A Fine-Grained Analogue of Schaefer's Theorem in P: Dichotomy of Exists^k-Forall-Quantified First-Order Graph Properties. In Amir Shpilka, editor, *Proceedings of the 34th Computational Complexity Conference, CCC 2019*, volume 137 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:27, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CCC.2019.31.
6. Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In Chris Umans, editor, *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.
7. Andrei A. Bulatov, Martin Grohe, Phokion G. Kolaitis, and Andrei A. Krokhin, editors. *The Constraint Satisfaction Problem: Complexity and Approximability, 25.10. - 30.10.2009*, volume 09441 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. URL: http://drops.dagstuhl.de/portals/09441/.
8. Jin-Yi Cai and Xi Chen. Complexity of counting CSP with complex weights. *J. ACM*, 64(3):19:1–19:39, 2017. doi:10.1145/2822891.
9. Hubie Chen, Bart M. P. Jansen, and Astrid Pieterse. Best-case and worst-case sparsifiability of Boolean CSPs. *Algorithmica*, 2020. Online first. doi:10.1007/s00453-019-00660-y.
10. Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. Syst. Sci.*, 51(3):511–522, 1995. doi:10.1006/jcss.1995.1087.
11. Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of Boolean constraint satisfaction problems*, volume 7 of *SIAM monographs on discrete mathematics and applications*. SIAM, 2001. doi:10.1137/1.9780898718546.
12. Víctor Dalmau, Andrei A. Krokhin, and Rajsekar Manokaran. Towards a characterization of constant-factor approximable finite-valued CSPs. *J. Comput. Syst. Sci.*, 97:14–27, 2018. doi:10.1016/j.jcss.2018.03.003.
13. Vladimir Deineko, Peter Jonsson, Mikael Klasson, and Andrei Krokhin. The approximability of MAX CSP with fixed-value constraints. *J. ACM*, 55(4), September 2008. doi:10.1145/1391289.1391290.
14. Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

15 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

16 Gregory Z. Gutin and Anders Yeo. Parameterized constraint satisfaction problems: a survey. In Andrei A. Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 179–203. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/DFU.Vol7.15301.7.

17 Bart M. P. Jansen and Astrid Pieterse. Sparsification upper and lower bounds for graph problems and not-all-equal SAT. *Algorithmica*, 79(1):3–28, 2017. doi:10.1007/s00453-016-0189-9.

18 Bart M. P. Jansen and Astrid Pieterse. Optimal sparsification for some binary CSPs using low-degree polynomials. *TOCT*, 11(4):28:1–28:26, 2019. doi:10.1145/3349618.

19 Bart M. P. Jansen and Michal Wlodarczyk. Optimal polynomial-time compression for Boolean Max CSP. *CoRR*, abs/2002.03443, 2020. arXiv:2002.03443.

20 Peter Jonsson and Andrei A. Krokhin. Maximum H-colourable subdigraphs and constraint optimization with arbitrary weights. *J. Comput. Syst. Sci.*, 73(5):691–702, 2007. doi:10.1016/j.jcss.2007.02.001.

21 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000. doi:10.1137/S0097539799349948.

22 Stefan Kratsch, Dániel Marx, and Magnus Wahlström. Parameterized Complexity and Kernelizability of Max Ones and Exact Ones Problems. *TOCT*, 8(1):1:1–1:28, 2016. doi:10.1145/2858787.

23 Stefan Kratsch and Magnus Wahlström. Preprocessing of Min Ones problems: A dichotomy. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Proceedings of the 37th International Colloquium on Automata, Languages and Programming, ICALP 2010*, volume 6198 of *Lecture Notes in Computer Science*, pages 653–665. Springer, 2010. doi:10.1007/978-3-642-14165-2_55.

24 Andrei A. Krokhin and Stanislav Zivny, editors. *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. URL: http://www.dagstuhl.de/dagpub/978-3-95977-003-3.

25 Victor Lagerkvist and Magnus Wahlström. Kernelization of constraint satisfaction problems: A study through universal algebra. In J. Christopher Beck, editor, *Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming, CP 2017*, volume 10416 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2017. doi:10.1007/978-3-319-66158-2_11.

26 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, SODA '18, pages 1236–1252, USA, 2018. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975031.80.

27 Konstantin Makarychev and Yury Makarychev. Approximation Algorithms for CSPs. In Andrei Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 287–325. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.287.

28 M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

29 Noam Nisan and Mario Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 4:301–313, 1994. doi:10.1007/BF01263419.

30 Hans Ulrich Simon. A tight $\Omega(\log\log n)$-bound on the time for parallel RAM's to compute nondegenerated boolean functions. *Information and Control*, 55(1-3):102–106, 1982. doi:10.1016/S0019-9958(82)90477-6.

31 Gábor Tardos and David A. Mix Barrington. A lower bound on the mod 6 degree of the OR function. *Computational Complexity*, 7(2):99–108, 1998. doi:10.1007/PL00001597.

32 Joachim von Zur Gathen and James R. Roche. Polynomials with two values. *Combinatorica*, 17(3):345–362, September 1997. doi:10.1007/BF01215917.

33 R. Ryan Williams. *Algorithms and Resource Requirements for Fundamental Problems*. PhD thesis, Carnegie Mellon University, USA, 2007. AAI3274191.

34 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983. doi:10.1016/0304-3975(83)90020-8.

35 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *Proceeings of the 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

A Linear Fixed Parameter Tractable Algorithm for Connected Pathwidth

Mamadou Moustapha Kanté
Université Clermont Auvergne, LIMOS, CNRS, Aubière, France
mamadou.kante@uca.fr

Christophe Paul
LIRMM, Université de Montpellier, CNRS, France
christophe.paul@lirmm.fr

Dimitrios M. Thilikos
LIRMM, Université de Montpellier, CNRS, France
sedthilk@thilikos.info

Abstract

The graph parameter of *pathwidth* can be seen as a measure of the topological resemblance of a graph to a path. A popular definition of pathwidth is given in terms of *node search* where we are given a system of tunnels (represented by a graph) that is contaminated by some infectious substance and we are looking for a search strategy that, at each step, either places a searcher on a vertex or removes a searcher from a vertex and where an edge is cleaned when both endpoints are simultaneously occupied by searchers. It was proved that the minimum number of searchers required for a successful cleaning strategy is equal to the pathwidth of the graph plus one. Two desired characteristics for a cleaning strategy is to be *monotone* (no recontamination occurs) and *connected* (clean territories always remain connected). Under these two demands, the number of searchers is equivalent to a variant of pathwidth called *connected pathwidth*. We prove that connected pathwidth is fixed parameter tractable, in particular we design a $2^{O(k^2)} \cdot n$ time algorithm that checks whether the connected pathwidth of G is at most k. This resolves an open question by [*Dereniowski, Osula, and Rzążewski, Finding small-width connected path-decompositions in polynomial time. Theor. Comput. Sci., 794:85–100, 2019*]. For our algorithm, we enrich the *typical sequence technique* that is able to deal with the connectivity demand. Typical sequences have been introduced in [*Bodlaender and Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms, 21(2):358–402, 1996*] for the design of linear parameterized algorithms for treewidth and pathwidth. While this technique has been later applied to other parameters, none of its advancements was able to deal with the connectivity demand, as it is a "global" demand that concerns an unbounded number of parts of the graph of unbounded size. The proposed extension is based on an encoding of the connectivity property that is quite versatile and may be adapted so to deliver linear parameterized algorithms for the connected variants of other width parameters as well. An immediate consequence of our result is a $2^{O(k^2)} \cdot n$ time algorithm for the monotone and connected version of the edge search number.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory; Theory of computation → Fixed parameter tractability

Keywords and phrases Graph decompositions, Parameterized algorithms, Typical sequences, Pathwidth, Graph searching

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.64

Related Version A full version of this extended abstract is available at [26], `https://arxiv.org/abs/2004.11937`.

Funding *Mamadou Moustapha Kanté*: DEMOGRAPH (ANR-16-CE40-0028) and ASSK (ANR-18-CE40-0025-01).
Christophe Paul: DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010).
Dimitrios M. Thilikos: DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010).

1 Introduction

Pathwidth. A *path-decomposition* of a graph $G = (V, E)$ is a sequence $\mathsf{Q} = \langle B_1, \ldots, B_q \rangle$ of vertex sets, called *bags* of Q, such that

1. $\bigcup_{i \in \{1,\ldots,q\}} B_i = V$,
2. every edge $e \in E$ is a subset of some member of Q, and
3. the *trace* of every vertex $v \in V$, that is the set $\{i \mid v \in B_i\}$, is a set of consecutive integers.

The *width* of a path-decomposition is $\max\{|B_i| - 1 \mid i \in \{1, \ldots, q\}\}$ and the *pathwidth* of a graph G, denoted by $\mathsf{pw}(G)$, is the minimum width of a path-decomposition of G.

The above definition appeared for the first time in [36]. Pathwidth can be seen as a measure of the topological resemblance of a graph to a path. Pathwidth, along with its tree-analogue *treewidth*, have been used as key combinatorial tools in the Graph Minors series of Robertson and Seymour [37] and they are omnipresent in both structural and algorithmic graph theory.

Deciding whether the pathwidth of a graph is at most k is an NP-complete problem [2]. This motivated the problem of the existence, or not, of a *parameterized algorithm* for this problem, and algorithm running in $f(k) \cdot n^{O(1)}$ time algorithm. An affirmative answer to this question was directly implied as a consequence of the algorithmic and combinatorial results of the Graph Minors series and the fact that, for every k, the class of graphs with pathwidth at most k is closed under taking of minors[1]. On the negative side, this implication was purely existential. The challenge of *constructing* an $f(k) \cdot n^{O(1)}$ time algorithm for pathwidth (as well as for treewidth) was a consequence of the classic result of Bodlaender and Kloks in [8] (see also [15, 30]). The main result in [8] implies a $2^{O(k^3)} \cdot n$ time algorithm. This was later improved to one running in $2^{O(k^2)} \cdot n$ time by Fürer in [20]).

Graph searching. In a *graph searching* game, the opponents are a group of *searchers* and an evading *fugitive*. The opponents move in turns in a graph. The objective of the searchers is to deploy a strategy of moves that leads to the capture of the fugitive. At each step of the *node searching game*, the searchers may either place a searcher at a vertex or remove a searcher from a vertex. The fugitive resides in the edges of the graph and is lucky, invisible, fast, and agile. The capture of the fugitive occurs when searchers occupy both endpoints of the edge where he currently resides. A *node searching strategy* is a sequence of moves of the searchers that can guarantee the eventual capture of the fugitive.[2] The cost of a searching strategy is the maximum number of searchers simultaneously present in the graph during the deployment of the strategy. The *node search number* of a graph G, denoted by $\mathsf{ns}(G)$, is defined as the minimum cost of a node searching strategy. Node searching was defined by Kirousis and Papadimitriou in [29] who proved that the game is equivalent to its monotone variant where search strategies are *monotone* in the sense that they prevent the fugitive from pervading again areas from where he had been expelled. This result along with the results in [27, 28, 31], imply that, for every graph G, $\mathsf{ns}(G) = \mathsf{pw}(G) + 1$.

The connectivity issue. In several applications of graph searching it is important to guarantee secure communication channels between the searchers so that they can safely exchange information. This issue was treated for the first time in the area of distributed

[1] A graph H is a *minor* of a graph G if H can be obtained by some subgraph of G by contracting edges.

[2] An equivalent setting of graph searching is to see G as a system of pipelines or corridors that is contaminated by some poisonous gas or some highly infectious substance. The searchers can be seen as cleaners that deploy a decontamination strategy [13, 19]. The fact that the fugitive is invisible, fast, lucky, and agile permits us to see him as being omnipresent in any edge that has not yet been cleaned.

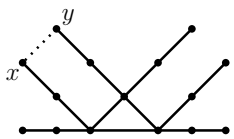


Figure 1 A graph G of connected pathwidth 2 with a subgraph of connected pathwidth 3.

computing, in particular in [4], where the authors considered the problem of capturing an intruder by mobile agents (acting for example as antivirus programs). As agents deploy their cleaning strategy, they must guarantee that, at each moment of the search, the cleaned territories remain connected, so to permit the safe exchange of information between the coordinating agents.

The systematic study of connected graph searching was initiated in [3, 5]. When, in node searching, we demand that the search strategies are monotone and connected, we define *monotone connected node search number*, denoted by $\mathsf{mcns}(G)$. The graph decomposition counterpart of this parameter was introduced by Dereniowski in [16]. He defined the *connected pathwidth* of a connected graph, denoted by $\mathsf{cpw}(G)$, by considering *connected* path-decompositions $\mathsf{Q} = \{B_1, \ldots, B_q\}$ where the following additional property is satisfied:

▶ For every $i \in \{1, \ldots, q\}$, the subgraph of G induced by $\bigcup_{h\in\{1,\ldots,i\}} B_h$ is *connected*.

As noticed in [16], for every connected graph G, $\mathsf{mcns}(G) = \mathsf{cpw}(G) + 1$ (see also [1]). Notice that the above demand results to a break of symmetry: the fact that $\langle B_1, \ldots, B_q\rangle$ is a connected path-decomposition does not imply that the same holds for $\langle B_q, \ldots, B_1\rangle$ (while this is always the case for conventional path-decompositions). This *sense of direction* seems to be the source of all combinatorial particularities (and challenges) of connected pathwidth.

Computing connected pathwidth. It is easy to see that checking whether $\mathsf{cpw}(G) \leq k$ is an NP-complete problem: if we define G^* as the graph obtained from G after adding a new vertex adjacent with all the vertices of G, then observe that $\mathsf{pw}(G) = \mathsf{cpw}(G^*) - 1$. This motivates the question on the parameterized complexity of the problem. The first progress in this direction was done recently in [17] by Dereniowski, Osula, and Rzążewski who gave an $f(k) \cdot n^{O(k^2)}$ time algorithm. In [17, Conjecture 1], they conjectured that there is a fixed parameter algorithm checking whether $\mathsf{cpw}(G) \leq k$. The general question on the parameterized complexity of the connected variants of graph search was raised as an open question by Fedor V. Fomin during the GRASTA 2017 workshop [18].

A somehow dissuasive fact towards a parameterized algorithm for connected pathwidth is that connected pathwdith is not closed under minors and therefore it does not fit in the powerful algorithmic framework of Graph Minors (which is the case with pathwidth). The removal of an edge may increase the parameter. For instance, the connected pathwidth of the graph in Figure 1 has connected pathwidth 2 while if we remove the edge $\{x, y\}$ its connected pathwidth increases to 3. On the positive side, connected pathwidth is closed under contractions (see e.g., [1]), i.e, its value does not increase when we contract edges and, moreover, the yes-instances of the problem have bounded pathwidth, therefore they also have bounded treewidth. Based on these observations, the existence of a parameterized algorithm would be implied if we can prove that, for any k, the set $\mathcal{Z}_k$ of contraction-minimal[3] graphs with connected pathwidth more than k is *finite*: as contraction containment can be expressed

[3] For instance, the graph $G \setminus \{x, y\}$ from Figure 1 belongs in $\mathcal{Z}_2$.

in MSO logic, one should just apply Courcelle's theorem [14] to check whether some graph of $\mathcal{Z}_k$ is a contraction of G. The hurdle in this direction is that we have no idea whether $\mathcal{Z}_k$ is finite or not. The alternative pathway is to try to devise a linear parameterized algorithm by applying the algorithmic techniques that are already known for pathwidth.

The typical sequence technique. The main result of [8] was an algorithm that, given a path-decomposition Q of G of width at most k and an integer w, outputs, if exists, a path-decomposition of G of width at most w, in $2^{O(k(w+\log k))} \cdot n$ time. In this algorithm Bodlaender and Kloks introduced the celebrated *typical sequence technique*, a refined dynamic programming technique that encodes partial path/tree decompositions as a system of suitably compressed sequences of integers, able to encode all possible path-decompositions of width at most w (see also [15, 30]). This technique was later extended/adapted for the design of parametrized algorithms for numerous graph parameters such as branchwidth [9], linear-width [10], cutwidth [39], carving-width [38], modified cutwidth, and others [6,7,40]. In [6] this technique was viewed as a result of *un-nondeterminization*: a stepwise evolution of a trivial hypothetical non-deterministic algorithm towards a deterministic parameterized algorithm. A considerable generalization of the characteristic sequence technique was proposed in the PhD thesis of Soares [32] where this technique was implemented under the powerful meta-algorithmic framework of *q-branched Φ-width*. Non-trivial extensions of the typical sequence technique where proposed for devising parameterized algorithms for parameters on matroids such as matroid pathwidth [23], matroid branchwidth [25], as well as all the parameters on graphs or hypergraphs that can be expressed by them. Very recently Bodlaender, Jaffke, and Telle in [7] suggested refinements of the typical sequence technique that enabled the polynomial time computation of several width parameters on directed graphs. Finally, Bojańczyk and Pilipczuk suggested an alternative approach to the typical sequence technique, based on MSO transductions between decompositions [11].

Unfortunately, the above mentioned state of the art on the typical sequence technique is unable to encompass connected pathwidth. A reason for this is that the connectivity demand is a "global property" applying to *every* prefix of the path-decomposition which correspond to an unbounded number of subgraphs of arbitrary size.

Our result. In this paper we resolve *affirmatively* the conjecture that checking whether $\mathsf{cpw}(G) \leq k$ is fixed parameter tractable. Our main result is the following.

▶ **Theorem 1.** *One may construct an algorithm that given an n-vertex connected graph G, a path-decomposition $\mathsf{Q} = \langle B_1, \ldots, B_q \rangle$ of G of width at most k and an integer w, outputs a connected path-decomposition of G of width at most w or reports correctly that such an algorithm does not exist in $2^{O(k(w+\log k))} \cdot n$ time.*

To design an algorithm checking whether $\mathsf{cpw}(G) \leq k$ we first use the algorithms of [8] and [20], to build, if exists, a path decomposition of G of width at most k, in $2^{O(k^2)} \cdot n$ time. In case of a negative answer we know than $\mathsf{cpw}(G) > k$, otherwise we apply the algorithm of Theorem 1. The overall running time is dominated by the algorithm of Fürer in [20] which is $2^{O(k^2)} \cdot n$.

Our techniques. We now give a brief description of our techniques by focusing on the novel issues that we introduce. This description demands some familiarity with the typical sequence technique. Otherwise, the reader can go directly to the next section.

Let $\mathsf{Q} = \langle B_1, \ldots, B_q \rangle$ be a (nice) path-decomposition of G of width at most k. For every $i \in [q]$, we let $\mathbf{G}_i = (G_i, B_i)$ be the boundaried graph where $G_i = G[\bigcup_{h \in \{1,\ldots,i\}} B_h]$. We follow standard dynamic programming over a path-decomposition that consists in computing a representation of the set of partial solutions associated to $\mathbf{G}_i$, which in our case are *connected* path-decompositions of $\mathbf{G}_i$ of width at most w. The challenge is how to handle in a compact way the connectivity requirement of a path-decomposition of a graph that can be of arbitrarily large size.

A connected path-decomposition $\mathsf{P} = \langle A_1, \ldots, A_\ell \rangle$ of $\mathbf{G}_i$ is represented by means of a $(\mathbf{G}_i, \mathsf{P})$-*encoding sequence* $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$. For every $j \in [\ell]$, the element s_j of the sequence S is a triple $(\mathbf{bd}(\mathsf{s}_j), \mathbf{cc}(\mathsf{s}_j), \mathbf{val}(\mathsf{s}_j))$ where: $\mathbf{bd}(\mathsf{s}_i) = A_j \cap B_i$; $\mathbf{val}(\mathsf{s}_j) = |A_j \setminus B_i|$; and $\mathbf{cc}(\mathsf{s}_j)$ is the projection of the connected components of $G_i^j = G_i[\bigcup_{h \in \{1,\ldots,j\}} A_h]$ onto the subset of boundary vertices $B_i \cap V(G_i^j)$. To compress a $(\mathbf{G}_i, \mathsf{P})$-*encoding sequence* S, we identify a subset $\mathbf{bp}(\mathsf{S})$ of indexes, called *breakpoints*, such that $j \in \mathbf{bp}(\mathsf{S})$ if $\mathbf{bd}(\mathsf{s}_{j-1}) \neq \mathbf{bd}(\mathsf{s}_j)$ (type-1) or $\mathbf{cc}(\mathsf{s}_{j-1}) \neq \mathbf{cc}(\mathsf{s}_j)$ (type-2) or j is an index belonging to a typical sequence of the integer sequence $\langle \mathbf{val}(\mathsf{s}_b), \ldots, \mathbf{val}(\mathsf{s}_{c-1}) \rangle$ where $b, c \in [\ell]$ are consecutive type-1 or 2- breakpoints. We define $\mathsf{rep}(\mathsf{S})$ as the induced subsequence $\mathsf{S}[\mathbf{bp}(\mathsf{S})]$.

The novelty in this representation is the $\mathbf{cc}(\cdot)$ component which is a near-partition of the subset $B_i \cap V(G_i^j)$ of boundary vertices. The critical observation is that for every $j \in [\ell - 1]$, $\mathbf{cc}(\mathsf{s}_{j+1})$ is coarser than $\mathbf{cc}(\mathsf{s}_j)$. This, together with the known results on typical sequences, allows us to prove that the size of $\mathsf{rep}(\mathsf{S})$ is $O(kw)$ and that the number of representative sequences is $2^{O(k(w+\log k))}$. Finally, as in the typical sequence technique, we define a domination relation over the set of representative sequences. The DP algorithm over the path-decomposition Q consists then in computing a domination set $\mathbf{D}_w(G_{i+1})$ of the representative sequences of $\mathbf{G}_{i+1}$ from a domination set $\mathbf{D}_w(G_i)$ of the representative sequences of $\mathbf{G}_i$.

The above scheme extends the current state of the art on typical sequences as it further incorporates the encoding of the connectivity property. While this is indeed a "global property", it appears that its evolution with respect to the bags of the decomposition can be controlled by the second component of our encoding and this is done in terms of a sequence of a gradually coarsening partitions. This establishes a dynamic programming framework that can potentially be applied on the connected versions of most of the parameters where the typical sequence technique was used so far. Moreover, it may be the starting point of the algorithmic study of parameters where other, alternative to connectivity, global properties are imposed to the corresponding decompositions.

Consequences in connected graph searching. The original version of graph searching was the *edge searching* variant, defined [4] by Parsons [33,34], where the only differences with node searching is that a searcher can additionally slide along an edge and sliding is the only way to clean an edge. The corresponding search number is called *edge search number* and is denoted by $\mathsf{es}(G)$. If we additionally demand that the searching strategy is connected and monotone, then we define the *monotone connected edge search number* denoted by $\mathsf{mces}(G)$. As proved

[4] An equivalent model was proposed independently by Petrov [35]. The models of Parsons and Petrov where different but also equivalent, as proved by Golovach in [21,22]. The model of Parsons was inspired by an earlier paper by Breisch [12], titled *"An intuitive approach to speleotopology"*, where the aim was to rescue an (unlucky) speleologist lost in a system of caves. Notice that "unluckiness" cancels the speleologist's will of being rescued as, from the searchers' point of view, it imposes on him/her the status of an "evading entity". As a matter of fact, the connectivity issue appears even in the first inspiring model of the search game. In a more realistic scenario, the searchers cannot "teleport" themselves to non-adjacent territories of the caves while this was indeed permitted in the original setting of Parsons.

in [29], $\mathsf{es}(G) = \mathsf{pw}(G_{\mathsf{v}})$, where G_{v} is the graph obtained if we subdivide twice each edge of G. Applying the same reduction as in [29] for the monotone and connected setting, one can prove that $\mathsf{mces}(G) = \mathsf{cpw}(G_{\mathsf{v}})$. As we already mentioned, $\mathsf{mcns}(G) = \mathsf{cpw}(G_{\mathsf{v}}) + 1$. These two reductions imply that the result of Theorem 1 holds also for mcns and mces, i.e., the search numbers for the monotone and connected versions of both node and edge searching.

2 Preliminaries and definitions

2.1 Basic concepts

Sets and near-partitions. For an integer ℓ, we denote by $[\ell]$ the set $\{1, \ldots, \ell\}$. Let S be a finite set. A *near-partition* Q of S is a family of subsets $\{X_1, \ldots, X_k\}$ (with $k \leq |S| + 1$) of subsets of S, called *blocks*, such that $\bigcup_{i \in [k]} X_i = S$ and for every $1 \leq i < j \leq k$, $X_i \cap X_j = \emptyset$. Observe that a near-partition may contain several copies of the empty set. A *partition* of S is a near-partition with the additional constraint that if it contains the empty set, then this is the unique block. Let Q be a near-partition of a set S and Q' be a near-partition of a set S' such that $S \subseteq S'$. We say that Q is *thinner* than Q', or that Q' is *coarser* than Q, which we denote $\mathsf{Q} \sqsubseteq \mathsf{Q}'$, if for every block X of Q, there exists a block X' of Q' such that $X \subseteq X'$. For a near-partition $\mathsf{Q} = \{X_1, \ldots, X_\ell\}$ of S and a subset $S' \subseteq S$, we define the *projection of* Q *onto* S' as the near-partition $\mathsf{Q}_{|S'} = \{X_1 \cap S', \ldots, X_\ell \cap S'\}$. Observe that if Q is a partition, then $\mathsf{Q}_{|S'}$ may not be a partition: if several blocks of Q are subsets of $S \setminus S'$, then $\mathsf{Q}_{|S'}$ contains several copies of the emptyset.

Sequences. Let S be a set. A *sequence* of elements of S, denoted by $\alpha = \langle a_1, \ldots, a_\ell \rangle$, is a subset of S equipped with a total ordering: for $1 \leqslant i < j \leqslant \ell$, a_i occurs before a_j in the sequence α. The *length* of a sequence is the number of elements that it contains. Let $X \subseteq [\ell]$ be a subset of indexes of α. We define the *subsequence* of α *induced* by X as the sequence $\alpha[X]$ on the subset $\{a_i \mid i \in X\}$ such that, for $i, j \in X$, a_i occurs before a_j in $\alpha[X]$ if and only if $i < j$.

The *duplication* of the element a_j, with $j \in [\ell]$, in the sequence $\alpha = \langle a, \ldots, a_\ell \rangle$ yields the sequence $\alpha' = \langle a_1, \ldots, a_j, a_j, \ldots, a_\ell \rangle$ of length $\ell + 1$. A sequence β is an *extension* of the sequence α if it is either α or it results from a series of duplications on α. We define the set of extensions of α as: $\mathsf{Ext}(\alpha) = \{\alpha^* \mid \alpha^* \text{ is an extension of } \alpha\}$.

Let $\alpha = \langle a_1, \ldots, a_\ell \rangle$ be a sequence and $\alpha^* = \langle a_1, \ldots, a_p \rangle$ be an extension of α. If $p \leq \ell + k$, then α^* results from a series of at most k duplications and we say that α^* is a $(\leq k)$-*extension* of α. With the definition of an extension, every element of α^* is a copy of some element of α. We define the *extension surjection* as a surjective function $\delta_{\alpha^* \to \alpha} : [p] \to [\ell]$ such that if $\delta_{\alpha^* \to \alpha}(j) = i$ then $a^*_j = a_i$. An extension surjection $\delta_{\alpha^* \to \alpha}$ is a certificate that $\alpha^* \in \mathsf{Ext}(\alpha)$. Finally, we observe that if $\alpha^* \in \mathsf{Ext}(\alpha)$, then α is an induced subsequence of α^*. Moreover, if $\alpha^* \in \mathsf{Ext}(\alpha)$ and $\beta \in \mathsf{Ext}(\alpha^*)$, then β is an extension of α.

Graphs and boundaried graphs. Given a graph $G = (V, E)$ and a vertex set $S \subseteq V(G)$, we denote by $G[S]$ the *subgraph* of G that is *induced by the vertices of* S, i.e., the graph $(S, \{e \in E \mid e \subseteq S\})$. Also, if $x \in V$, we define $G \setminus x = G[V \setminus \{x\}]$. The *neighborhood* of a vertex v in G is the set of vertices that are adjacent to v in G and is denoted by $N_G(v)$.

A *boundaried graph* is a pair $\mathbf{G} = (G, B)$ such that G is a graph over a vertex set V and $B \subseteq V$ is a subset of distinguished vertices, called *boundary vertices*. We say that a boundaried graph $\mathbf{G} = (G, B)$ is *connected* if either G is connected and $B = \emptyset$ or, in case $B \neq \emptyset$, every connected component C of G contains some boundary vertex, that is $C \cap B \neq \emptyset$.

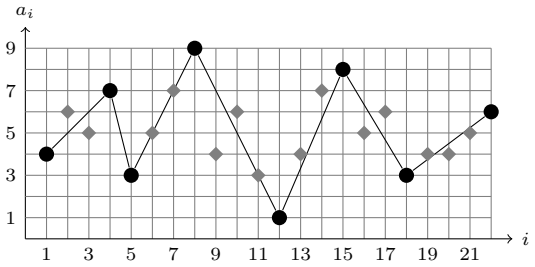


Figure 2 The black bullets forms the typical sequence $\mathsf{Tseq}(\alpha) = \langle 4,7,3,9,1,8,3,6\rangle$ of the sequence $\alpha = \langle 4,6,5,7,3,5,7,9,4,6,3,1,4,7,8,5,6,3,4,4,5,6\rangle$ of black and gray diamonds.

The definition of a connected path-decomposition also naturally extends to boundaried graphs as follows.

▶ **Definition 2.** *Let* $\mathsf{P} = \langle A_1, \ldots, A_\ell\rangle$ *be a path-decomposition of the boundaried graph* $\mathbf{G} = (G, B)$*. Then* P *is* connected *if, for every* $i \in [\ell]$*, the boundaried graph* $(G_i, V_i \cap B)$ *is connected, where* $V_i = \bigcup_{h\in[i]} A_h$ *and* $G_i = G[V_i]$*.*

2.2 Integer sequences

Let us recall the notion of *typical sequences* introduced by Bodlaender and Kloks [8] (see also [15, 30]).

▶ **Definition 3.** *Let* $\alpha = \langle a_1, \ldots, a_\ell\rangle$ *be an integer sequence. The* typical sequence $\mathsf{Tseq}(\alpha)$ *is obtained after iterating the following operations, until none is possible anymore:*

- *if for some* $i \in [\ell - 1]$*,* $a_i = a_{i+1}$*, then remove* a_{i+1} *from* α*;*
- *if there exists* $i, j \in [\ell]$ *such that* $i \leqslant j - 2$ *and* $\forall h$*,* $i < h < j$*,* $a_i \leq a_h \leq a_j$ *or* $\forall h$*,* $i < h < j$*,* $a_i \geq a_h \geq a_j$*, then remove the subsequence* $\langle a_{i+1}, \ldots, a_{j-1}\rangle$ *from* α*.*

As a typical sequence $\mathsf{Tseq}(\alpha) = \langle b_1, \ldots, b_i, \ldots, b_r\rangle$ is a subsequence of α, it follows that, for every $i \in [r]$, there exists $j_i \in [\ell]$ such that $b_i = a_{j_i}$. Herefater every such index j_i is called a *tip* of the sequence α.

If α and β are two integer sequences of same length ℓ, we say that $\alpha \leq \beta$ if for every $j \in [\ell]$, $a_j \leq b_j$.

▶ **Definition 4.** *Let* α *and* β *be two integer sequences. Then* $\alpha \preceq \beta$ *if there are* $\alpha^* \in \mathsf{Ext}(\alpha)$ *and* $\beta^* \in \mathsf{Ext}(\beta)$ *such that* $\alpha^* \leq \beta^*$*. Whenever* $\alpha \preceq \beta$ *and* $\beta \preceq \alpha$*, we say that* α *and* β *are* equivalent *which is denoted by* $\alpha \equiv \beta$*.*

We extend the definition of the $\leq$-relation and $\preceq$-relation on integer sequences to sequences of integer sequences. Let $\mathsf{P} = \langle \mathsf{L}_1, \ldots, , L_r\rangle$ and $\mathsf{Q} = \langle \mathsf{K}_1, \ldots, \mathsf{K}_r\rangle$ be two sequences of integer sequences such that for every $i \in [r]$, L_i and K_i have the same length. We say that $\mathsf{P} \leq \mathsf{Q}$ if for every $i \in [r]$, $\mathsf{L}_i \leq \mathsf{K}_i$. The set of *extensions* of P is $\mathsf{Ext}(\mathsf{P}) = \{\langle \mathsf{L}'_1, \ldots, \mathsf{L}'_r\rangle \mid i \in [r], \mathsf{L}'_i \in \mathsf{Ext}(\mathsf{L}_i)\}$. Finally we say that $\mathsf{P} \preceq \mathsf{Q}$ if there exist $\mathsf{P}' \in \mathsf{Ext}(\mathsf{P})$ and $\mathsf{Q}' \in \mathsf{Ext}(\mathsf{Q})$ such that $\mathsf{P}' \leq \mathsf{Q}'$. If $\mathsf{P} \preceq \mathsf{Q}$ and $\mathsf{Q} \preceq \mathsf{P}$, then we say that $\mathsf{P} \equiv \mathsf{Q}$. The relation $\equiv$ is an equivalence relation.

2.3 Boundaried sequences

We now define the main notion that will allow us to represent and manipulate (connected) path-decompositions of a boundaried graph $\mathbf{G} = (G, B)$ (see Subsection 3.1).

▶ **Definition 5** (B-boundaried sequence). *Let B be a finite set. A* B-boundaried sequence *is a sequence $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ such that for every $j \in [\ell]$, $\mathsf{s}_j = (\mathbf{bd}(\mathsf{s}_j), \mathbf{cc}(\mathsf{s}_j), \mathbf{val}(\mathsf{s}_j))$ is defined as follows:*

- *$\mathbf{bd}(\mathsf{s}_j) \subseteq B$ with the property that for every $x \in B$, the indices $j \in [l]$ such that $x \in \mathbf{bd}(\mathsf{s}_j)$ are consecutive;*
- *$\mathbf{cc}(\mathsf{s}_j)$ is a near-partition of $\bigcup_{i \leq j} \mathbf{bd}(\mathsf{s}_i) \subseteq B$ with the property that for every $j < \ell$, $\mathbf{cc}(\mathsf{s}_j) \sqsubseteq \mathbf{cc}(\mathsf{s}_{j+1})$;*
- *$\mathbf{val}(\mathsf{s}_j)$ is a positive integer.*

The width *of S is defined as $\mathsf{width}(\mathsf{S}) = \max_{j \in \ell}(|\mathbf{bd}(\mathsf{s}_j)| + \mathbf{val}(\mathsf{s}_j))$.*

▶ **Definition 6** (Connected B-boundaried sequence). *Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence for some finite set B. We say that S is* connected *if for every $i \in [\ell]$, $\mathbf{cc}(\mathsf{s}_i)$ is a partition of $\bigcup_{i \leq j} \mathbf{bd}(\mathsf{s}_i) \subseteq B$.*

Observe that if $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ is a connected B-boundaried sequence and if there exists some $i \in [\ell]$ such that $\mathbf{cc}(\mathsf{s}_i) = \{\emptyset\}$, then, for every $j \leq i$, $\mathbf{bd}(\mathsf{s}_j) = \emptyset$ and $\mathbf{cc}(\mathsf{s}_j) = \{\emptyset\}$.

▶ **Definition 7** (Breakpoints). *Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_j, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence for some finite set B. Then the index j, with $1 \leq j \leq \ell$, is a* breakpoint *of:*

- type-1 *if $j = 1$ or $\mathbf{bd}(\mathsf{s}_j) \neq \mathbf{bd}(\mathsf{s}_{j-1})$ or $j = \ell$;*
- type-2 *if it is not a type-1 breakpoint and $\mathbf{cc}(\mathsf{s}_j) \neq \mathbf{cc}(\mathsf{s}_{j-1})$;*
- type-3 *if it is not a type-1 nor a type-2 and j is a tip of the integer sequence $\langle \mathbf{val}(\mathsf{s}_{l_j}), \ldots, \mathbf{val}(\mathsf{s}_{r_j - 1}) \rangle$ where l_j and r_j are respectively the largest and smallest type-1 or type-2 breakpoints such that $l_j < j < r_j$.*

Let $\mathbf{bp}(\mathsf{S})$ be the set of breakpoints of S and $\mathbf{bp}_t(\mathsf{S})$ be the set of type-t breakpoints of S, for $t \in \{1, 2, 3\}$. The representative sequence $\mathsf{rep}(\mathsf{S})$ *of S is defined as $\mathsf{S}[\mathbf{bp}(\mathsf{S})]$.*

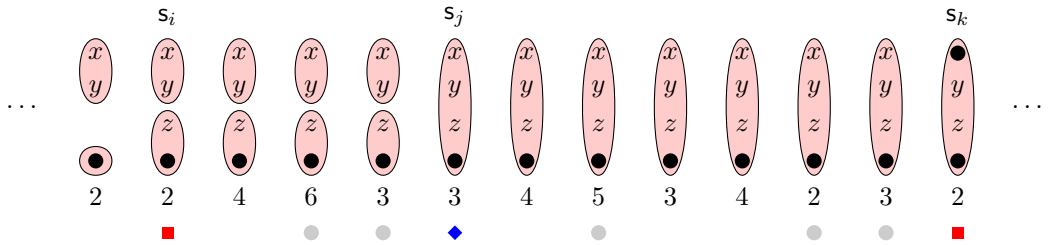


Figure 3 The part $\langle \mathsf{s}_{i-1}, \ldots \mathsf{s}_k \rangle$ of a B-boundaried sequence S for some set $B \supseteq \{x, y, z\}$. A bullet $\bullet$ at some index j represents an element of $\bigcup_{h<j} \mathbf{bd}(\mathsf{s}_h)$. Observe that at index k, x is indeed represented by a black bullet. For the index i, we have $\mathbf{bd}(\mathsf{s}_i) = \{x, y, z\}$, $\mathbf{cc}(\mathsf{s}_i) = \{\{x, y\}, \{z, \bullet\}\}$ and $\mathbf{val}(\mathsf{s}_i) = 2$. At every position j, only named elements belong to $\mathbf{bd}(\mathsf{s}_j)$. The red squares mark the type-1 breakpoints: at position i, element z is new, while at position k, element x is forgotten. The blue diamond at index j marks a type-2 breakpoint which corresponds to the merge of two parts of $\mathbf{cc}(\mathsf{s}_{i+4})$ into a single part. Finally, the grey bullets type-3 breakpoints corresponding to tips of the integer sequences $\langle \mathbf{val}(\mathsf{s}_i), \ldots \mathbf{val}(\mathsf{s}_{j-1}) \rangle$ and $\langle \mathbf{val}(\mathsf{s}_j), \ldots \mathbf{val}(\mathsf{s}_{k-1}) \rangle$.

Figure 3 illustrates the notions of B-boundaried sequence and breakpoints. Observe that $\mathsf{rep}(\mathsf{S})$ can be computed from the B-boundaried sequence S as in Definition 3 and is uniquely defined. The set of representative B-boundaried sequences of width at most w is defined as

$$\mathbf{Rep}_w(B) = \{\mathsf{rep}(\mathsf{S}) \mid \mathsf{S} \text{ is a } B\text{-boundaried sequence of width} \leq w\}.$$

▶ **Definition 8** (B-boundary model). *Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_j, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence. For every $j \in [\ell]$, we set $\dot{\mathsf{s}}_j = (\mathbf{bd}(\mathsf{s}_j), \mathbf{cc}(\mathsf{s}_j), \mathbf{t}(\mathsf{s}_j))$ with $\mathbf{t}(\mathsf{s}_j) = 1$ if $j \in \mathbf{bp}_1(\mathsf{S})$, $\mathbf{t}(\mathsf{s}_j) = 2$ if $j \in \mathbf{bp}_2(\mathsf{S})$ and $\mathbf{t}(\mathsf{s}_j) = 0$ otherwise. The B-*boundary model *of S, denoted by $\mathsf{model}(\mathsf{S})$, is the subsequence of $\dot{\mathsf{S}} = \langle \dot{\mathsf{s}}_1, \ldots, \dot{\mathsf{s}}_j, \ldots, \dot{\mathsf{s}}_\ell \rangle$ induced by $\mathbf{bp}_1(\mathsf{S}) \cup \mathbf{bp}_2(\mathsf{S})$.*

▶ **Lemma 9.** *Let* S *be a* B*-boundaried sequence. If* $\mathsf{S}^* \in \mathsf{Ext}(\mathsf{S})$*, then* $\mathsf{model}(S^*) = \mathsf{model}(S)$.

As in [8, 24], we will bound the number of representatives of B-boundaried sequences. For doing so, we bound the number of B-boundaried models and then use [8, Lemma 3.5] which gives an upper bound on the number of typical sequences. Taking into account the fact that there are $O(|B|)$ breakpoints and $|B|^{O(k)}$ different coarsening scenarios for the encoded near-partitions, we prove the following bound.

▶ **Lemma 10.** *Let* B *be a set of size* k*. Then,* $|\mathbf{Rep}_w(B)| = 2^{O(k(w+\log k)}$.

Notice that the notion of a B-boundary model corresponds to the one of *interval model* in [8]. Besides the B-boundary model of a sequence S, we introduce the *profile* of S, which corresponds to the concept of *list representation* in [8].

▶ **Definition 11** (Profile)**.** *Let* S *be a* B*-boundaried sequence of length* ℓ *and let* $1 = j_1 < \cdots < j_i < \cdots < j_r = \ell$ *be the subset of indices of* $[\ell]$ *that belong to* $\mathbf{bp}_1(\mathsf{S}) \cup \mathbf{bp}_2(\mathsf{S})$*. Then we set* $\mathsf{profile}(\mathsf{S}) = \langle \mathsf{L}_1, \ldots, \mathsf{L}_r \rangle$ *with, for* $i \in [r]$*,* $\mathsf{L}_j = \langle \mathbf{val}(\mathsf{s}_{j_i}), \ldots, \mathbf{val}(\mathsf{s}_{j_{i+1}-1}) \rangle$.

We introduce the domination relation over B-boundaried sequences. This allows us to compare B-boundaried sequences having the same model by means of their B-profiles.

▶ **Definition 12** (Domination relation)**.** *Let* $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_j, \ldots, \mathsf{s}_\ell \rangle$ *and* $\mathsf{T} = \langle \mathsf{t}_1, \ldots, \mathsf{t}_j, \ldots, \mathsf{t}_\ell \rangle$ *be two* B*-boundaried sequences such that* $\mathsf{model}(\mathsf{S}) = \mathsf{model}(\mathsf{T})$*. If* $\mathsf{profile}(\mathsf{S}) \leq \mathsf{profile}(\mathsf{T})$*, then we write* $\mathsf{S} \leq \mathsf{T}$*. And, we say that* S dominates T*, denoted by* $\mathsf{S} \preceq \mathsf{T}$*, if* $\mathsf{profile}(\mathsf{S}) \preceq \mathsf{profile}(\mathsf{T})$*. If we have* $\mathsf{profile}(\mathsf{S}) \preceq \mathsf{profile}(\mathsf{T})$ *and* $\mathsf{profile}(\mathsf{T}) \preceq \mathsf{profile}(\mathsf{S})$*, then we say that* S *and* T *are equivalent, which is denoted by* $\mathsf{S} \equiv \mathsf{T}$.

▶ **Lemma 13.** *Let* S *be a* B*-boundaried sequence. Then,*

1. $\mathsf{rep}(\mathsf{S}) \equiv \mathsf{S}$*,*
2. *if* $\mathsf{S}^* \in \mathit{Ext}(\mathsf{S})$*, then* $\mathsf{S}^* \equiv \mathsf{S}$*,*
3. $\mathsf{S} \preceq \mathsf{T}$ *if and only if* $\mathsf{rep}(\mathsf{S}) \preceq \mathsf{rep}(\mathsf{T})$*.*
4. *If* T *is a* B*-boundaried sequence such that* $\mathsf{S} \preceq \mathsf{T}$*, then there exist an extension* S^* *of* S *and an extension* T^* *of* T *such that* $\mathsf{S}^* \leq \mathsf{T}^*$*.*
5. *The relation* $\preceq$ *is transitive, and* $\equiv$ *is an equivalence relation (refering to boundary sequences).*

2.4 Operations on B-boundaried sequences

Given a finite set B, we define two operations on B-boundaried sequences that will be later used in the DP algorithm. The *projection* of a B-boundaried sequence S onto $B' \subsetneq B$ aims at changing the status of a boundary element from $B \setminus B'$ to the status of a non-boundary element. The second operation deals with the insertion in a B-boundaried sequence of a new boundary element x with respect to a subset $X \subseteq B$.

2.4.1 Projection of B-boundaried sequences

▶ **Definition 14** (Projection)**.** *Let* $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_i, \ldots, \mathsf{s}_\ell \rangle$ *be a* B*-boundaried sequence. For a subset* $B' \subseteq B$*, the* projection *of* S *onto* B' *is the* B'*-boundaried sequence* $\mathsf{S}_{|B'} = \langle \mathsf{s}_{1|B'}, \ldots, \mathsf{s}_{i|B'}, \ldots, \mathsf{s}_{\ell|B'} \rangle$ *such that for every* $i \in [\ell]$*:*

- $\mathbf{bd}(\mathsf{s}_{i|B'}) = \mathbf{bd}(\mathsf{s}_i) \cap B'$*;*
- $\mathbf{cc}(\mathsf{s}_{i|B'}) = \mathbf{cc}(\mathsf{s}_i)_{|B'}$*;*
- $\mathbf{val}(\mathsf{s}_{i|B'}) = \mathbf{val}(\mathsf{s}_i) + |\mathbf{bd}(\mathsf{s}_i) \setminus B'|$.

We observe that though the B-boundaried sequence S is connected, its projection $\mathsf{S}_{|B'}$ onto $B' \subseteq B$ may not be connected. This is the case if for some $j \in [\ell]$, the partition $\mathbf{cc}(\mathsf{s}_j)$ contains several blocks and at least one of them is a subset of $B \setminus B'$. Notice that the projection operation does not change the width of a sequence.

▶ **Lemma 15.** *Let B be a finite set and $B' \subsetneq B$. If S^* is an extension of a B-boundaried sequence S, then $\mathsf{S}^*_{|B'}$ is an extension of $\mathsf{S}_{|B'}$.*

▶ **Lemma 16.** *Let B be a finite set and $B' \subseteq B$. If S and T are B-boundaried sequences such that $\mathsf{S} \leq \mathsf{T}$, then $\mathsf{S}_{|B'} \leq \mathsf{T}_{|B'}$.*

Using Lemma 15 and Lemma 16, we prove the following:

▶ **Lemma 17.** *Let B be a finite set and $B' \subseteq B$. If S and T are B-boundaried sequences such that and $\mathsf{S} \preceq \mathsf{T}$, then $\mathsf{S}_{|B'} \preceq \mathsf{T}_{|B'}$.*

2.4.2 Insertion into a B-boundaried sequence

Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence and let X be a subset of B. An *insertion position* is a pair of indices (f_x, l_x) such that $1 \leq f_x \leq l_x \leq \ell$. An insertion position is *valid with respect to X in S* if $X \subseteq \bigcup_{f_x \leq j \leq l_x} \mathbf{bd}(\mathsf{s}_j)$.

▶ **Definition 18.** *Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence and (f_x, l_x) be a valid insertion position with respect to $X \subseteq B$. Then $\mathsf{S}^x = \mathsf{Ins}(\mathsf{S}, x, X, f_x, l_x) = \langle \mathsf{s}^x_1, \ldots, \mathsf{s}^x_\ell \rangle$ is the $(B \cup \{x\})$-boundaried sequence such that for every $j \in [\ell]$:*

- *if $j < f_x$, then $\mathbf{bd}(\mathsf{s}^x_j) = \mathbf{bd}(\mathsf{s}_j)$; $\mathbf{cc}(\mathsf{s}^x_j) = \mathbf{cc}(\mathsf{s}_j)$ and $\mathbf{val}(\mathsf{s}^x_j) = \mathbf{val}(\mathsf{s}_j)$.*
- *if $f_x \leq j \leq l_x$, then $\mathbf{bd}(\mathsf{s}^x_j) = \mathbf{bd}(\mathsf{s}_j) \cup \{x\}$; $\mathbf{cc}(\mathsf{s}^x_j)$ is obtained by adding a new block $\{x\}$ to $\mathbf{cc}(\mathsf{s}_j)$ and then merging that new block with all the blocks of $\mathbf{cc}(\mathsf{s}_j)$ that contain an element of X (if any); $\mathbf{val}(\mathsf{s}^x_j) = \mathbf{val}(\mathsf{s}_j)$.*
- *and otherwise, $\mathbf{bd}(\mathsf{s}^x_j) = \mathbf{bd}(\mathsf{s}_j)$; $\mathbf{cc}(\mathsf{s}^x_j)$ is obtained by adding a new block $\{x\}$ to $\mathbf{cc}(\mathsf{s}_j)$ and then merging that new block with all the blocks of $\mathbf{cc}(\mathsf{s}_j)$ that contain an element of X (if any); $\mathbf{val}(\mathsf{s}^x_j) = \mathbf{val}(\mathsf{s}_j)$.*

We can show that if T is an extension of S, then, to every valid insertion position (f_x, l_x) with respect to some subset $X \subseteq B$ in S, one can associate a valid insertion position (f^*_x, l^*_x) with respect to X in T. The reverse is not true as illustrated by Figure 4.

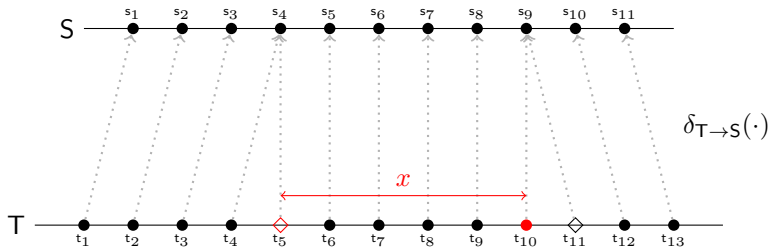


Figure 4 Let T be a 2-extension of the B-boundaried sequence S. Suppose that $(5, 10)$ is a valid insertion position with respect to some set $X \subseteq B$ in T. Observe that as $4 = \delta_{\mathsf{T}\to\mathsf{S}}(5)$ and $9 = \delta_{\mathsf{T}\to\mathsf{S}}(10)$, $(4, 9)$ is also a valid insertion position with respect to some for $X \subseteq B$ in S. However, $\mathsf{Ins}(\mathsf{T}, x, X, 5, 10)$ is not a 2-extension of $\mathsf{Ins}(\mathsf{S}, x, X, 4, 9)$.

However the next two lemmas provides an alternative by means of 2-extensions of S. The idea of the proof of Lemma 20 is illustrated by Figure 5.

▶ **Lemma 19.** *Let B and B' be finite sets with $B = B' \setminus \{x\}$ for some $x \in B'$. Let S and T be B-boundaried sequences such that $\mathsf{S} \leq \mathsf{T}$. If (f_x, l_x) is a valid insertion position with respect to a subset $X \subseteq B$ in T, then (f_x, l_x) is a valid insertion position with respect to X in S and $\mathsf{Ins}(\mathsf{S}, x, X, f_x, l_x) \leq \mathsf{Ins}(\mathsf{T}, x, X, f_x, l_x)$.*

▶ **Lemma 20.** *Let B and B' be finite sets with $B = B' \setminus \{x\}$ for some $x \in B'$. Let S and T be B-boundaried sequences such that $\mathsf{S} \preceq \mathsf{T}$. If (f_x^*, l_x^*) is a valid insertion position with respect to a subset $X \subseteq B$ in T, then there is a valid insertion position (f_x', l_x') in a (≤ 2)-extension R of S such that $\mathsf{Ins}(\mathsf{R}, x, X, f_x', l_x') \preceq \mathsf{Ins}(\mathsf{T}, x, X, f_x^*, l_x^*)$.*

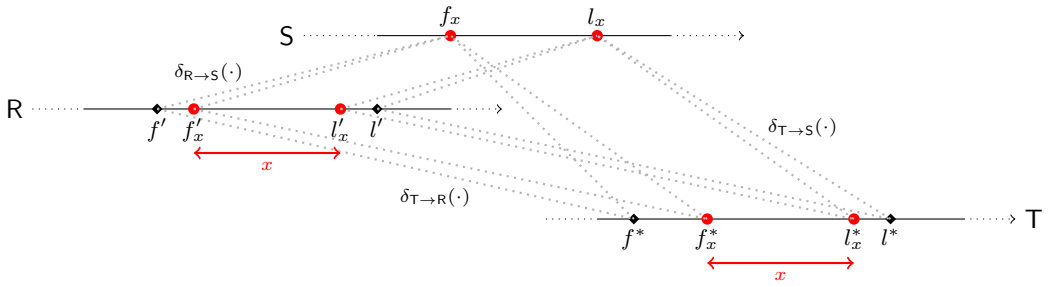


Figure 5 Let the B-boundaried sequence T be an extension of S that is certified by the surjective function $\delta_{\mathsf{T}\to\mathsf{S}}(\cdot)$ such that $f_x^* \neq f^*$, $l_x^* \neq l^*$ and $f^* = \min\{h \in [r] \mid \delta_{\mathsf{T}\to\mathsf{S}}(h) = f_x\}$, $l^* = \max\{h \in [r] \mid \delta_{\mathsf{T}\to\mathsf{S}}(h) = l_x\}$, $\delta_{\mathsf{T}\to\mathsf{S}}(f_x^*) = f_x$ and $\delta_{\mathsf{T}\to\mathsf{S}}(l_x^*) = l_x$. The B-boundaried sequence R is a 2-extension of S certified by the surjective function $\delta_{\mathsf{R}\to\mathsf{S}}(\cdot)$ such that $\delta_{\mathsf{R}\to\mathsf{S}}(f') = f_x$, $\delta_{\mathsf{R}\to\mathsf{S}}(f_x') = f_x$, $\delta_{\mathsf{R}\to\mathsf{S}}(l_x^*) = l_x$ and $\delta_{\mathsf{R}\to\mathsf{S}}(l') = l_x$. The fact that T is an extension of R can be certified by the surjective function $\delta_{\mathsf{T}\to\mathsf{R}}(\cdot)$ such that $\delta_{\mathsf{T}\to\mathsf{R}}(f^*) = f'$, $\delta_{\mathsf{T}\to\mathsf{R}}(f_x^*) = f_x'$, $\delta_{\mathsf{T}\to\mathsf{T}}(l_x^*) = l_x'$ and $\delta_{\mathsf{T}\to\mathsf{R}}(l^*) = l'$.

3 Computing the connected pathwidth

A (connected) path-decomposition $\mathsf{P} = \langle A_1, \ldots, A_\ell \rangle$ of a graph G is *nice* if $|A_1| = 1$ and $\forall i \in [p]$, $|A_{i-1} \vartriangle A_i| = 1$. A bag A_i, for $1 < 1 \leq \ell$, is called an *introduce* bag if $A_i \subsetneq A_{i-1}$ and a *forget* bag otherwise. As we will show, connected B-boundaried sequences are combinatorial objects designed in order to encode connected path-decompositions. Our algorithm is based on two routines. Forget Routine processes the forget bags by performing a projection operation on the B-boundaried sequences associated to those bags, while Insertion Routine handles the insertion bags by performing an insertion operation in the associated boundaried sequences.

3.1 Encoding a connected path-decomposition

▶ **Definition 21** ($(\mathbf{G}, \mathsf{P})$-encoding sequence). *Let $\mathsf{P} = \langle A_1, \ldots, A_\ell \rangle$ be a path-decomposition of the boundaried graph $\mathbf{G} = (G, B)$. A B-boundaried sequence $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_j, \ldots, \mathsf{s}_\ell \rangle$ is a $(\mathbf{G}, \mathsf{P})$-encoding sequence, if for every $j \in [\ell]$:*

- $\mathbf{bd}(\mathsf{s}_j) = A_j \cap B$*: the set of boundary vertices of (G, B) belonging to the bag A_j;*
- $\mathbf{cc}(\mathsf{s}_j) = \{V(C) \cap B \mid C$ *is a connected component of* $G_j\}$*;*
- $\mathbf{val}(\mathsf{s}_j) = |A_j \setminus B|$*: the number of non-boundary vertices in the bag A_j.*

It is worth to observe that $\mathbf{cc}(\mathsf{s}_j)$ is, in general, not a partition of A_j (see Figure 3). Also, notice that if G_j is connected and $B \cap V_j = \emptyset$, then $\mathbf{cc}(\mathsf{s}_j) = \{\emptyset\}$. Notice that if P is a connected path-decomposition, then S is a connected B-boundaried sequence.

▶ **Definition 22.** *Let* $\mathbf{G} = (G, B)$ *be a connected boundaried graph and* S *a* B*-boundaried sequence. We say that* S *is* realizable in $\mathbf{G}$ *if there is an extension* S^* *of* S *that is the* $(\mathbf{G}, \mathsf{P})$*-encoding sequence of some connected path-decomposition* P *of* $\mathbf{G}$.

Let us observe that if a B-boundaried sequence S is realizable, then S is connected. The set of *representative* B*-boundaried sequences of a connected boundaried graph* $\mathbf{G} = (G, B)$ *of width* $\leq w$ is defined as:

$$\mathbf{Rep}_w(\mathbf{G}) = \{\mathsf{rep}(\mathsf{S}) \mid \mathsf{S} \text{ of width} \leq w \text{ is realizable in } \mathbf{G} = (G, B)\}.$$

To compute the connected pathwidth of a graph, rather than computing $\mathbf{Rep}_w(\mathbf{G})$, we compute a subset $\mathbf{D}_w(\mathbf{G}) \subseteq \mathbf{Rep}_w(\mathbf{G})$, called *domination set*, such that for every representative B-boundaried sequence $\mathsf{S} \in \mathbf{Rep}_w(\mathbf{G})$, there exists a representative B-boundaried sequence $\mathsf{R} \in \mathbf{D}_w(\mathbf{G})$ where $\mathsf{R} \preceq \mathsf{S}$. Observe that a connected boundaried graph $\mathbf{G} = (G, B)$ has connected pathwidth at most w if and only if $\mathbf{D}_{w+1}(\mathbf{G}) \neq \emptyset$.

3.2 Forget Routine

Let $\mathbf{G} = (G, B)$ be a boundaried graph. If $x \in B$ is a boundary vertex, we denote by $B^{\overline{x}} = B \setminus \{x\}$. We define $\mathbf{G}^{\overline{x}} = (G, B^{\overline{x}})$, that is, while the graph G is left unchanged, we remove x from the set of boundary vertices. *Forget Routine* is described in Algorithm 1. Its correctness is proved in two steps. We first establish the *completeness* of the algorithm that is: for every connected path-decomposition P of $\mathbf{G}^{\overline{x}}$, there exists some B-boundaried sequence $\mathsf{S} \in \mathbf{D}_w(\mathbf{G})$ such that $\mathsf{rep}(\mathsf{S}_{|B\setminus\{x\}}) \preceq \mathsf{rep}(\mathsf{T})$ where T is the $(\mathbf{G}^{\overline{x}}, \mathsf{P})$-encoding sequence. For the soundness of the routine we prove that for every B-boundaried sequence $\mathsf{S} \in \mathbf{D}_w(\mathbf{G})$, $\mathsf{rep}(\mathsf{S}_{|B\setminus\{x\}}) \in \mathbf{D}_w(\mathbf{G}^{\overline{x}})$ if $\mathsf{S}_{|B\setminus\{x\}}$ is connected. The proofs of completeness and soundness rely both on Lemma 17. The time complexity is dominated by the bound on the number of representatives, given by Lemma 10.

Algorithm 1 Forget Routine.

Input: A boundaried graph $\mathbf{G} = (G, B)$, a vertex $x \in B$, and $\mathbf{D}_w(\mathbf{G})$.
Output: $\mathbf{D}_w(\mathbf{G}^{\overline{x}})$, a domination set of $\mathbf{Rep}_w(\mathbf{G}^{\overline{x}})$.

```
1 D_w(G^x̄) ← ∅;
2 foreach S ∈ D_w(G) do
3 |   if S_{|B\{x}} is connected, then add rep(S_{|B\{x}}) to D_w(G^x̄) ;
4 end
5 return D_w(G^x̄).
```

▶ **Theorem 23.** *Algorithm 1 computes* $\mathbf{D}_w(\mathbf{G}^{\overline{x}})$ *in* $2^{O(k(w+\log k))}$*-time, where* $k = |B|$.

3.3 Insertion Routine

Let $\mathbf{G} = (G, B)$ be a boundaried graph with $G = (V, E)$. For a subset $X \subseteq B$, we set $G^x = (V \cup \{x\}, E \cup \{xy \mid y \in X\})$ and $\mathbf{G}^x = (G^x, B^x)$ where $B^x = B \cup \{x\}$. Algorithm 2 is describing Insertion Routine (notations of Figure 5 are used in the pseudo-code). To prove its correctness, we proceed in two steps. We first establish the *completeness* of the algorithm: for every connected path-decomposition P^x of $\mathbf{G}^x$, the $(\mathbf{G}^x, \mathsf{P}^x)$-encoding sequence T^x is dominated by some B^x-boundaried sequence S^x that can be computed from a B-boundaried sequence S belonging to $\mathbf{D}_w(\mathbf{G})$. Then we argue about the *soundness* of Insertion Routine

Algorithm 2 Insertion Routine.

Input: A boundaried graph $\mathbf{G} = (G, B)$, a subset $X \subsetneq B$, and $\mathbf{D}_w(\mathbf{G})$.
Output: $\mathbf{D}_w(\mathbf{G}^x)$, a domination set of $\mathbf{Rep}_w(\mathbf{G}^x)$.

$\mathbf{D}_w(\mathbf{G}^x) \leftarrow \emptyset$;
foreach $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle \in \mathbf{D}_w(\mathbf{G})$ **do**
  foreach $f_x, l_x \in [\ell]$ *such that* $X \subseteq \bigcup_{f_x \leq j \leq l_x} \mathbf{bd}(\mathsf{s}_j)$ **do**
    foreach (≤ 2)-*extension* R *of* S *duplicating none, one or both of* s_{f_x} *and* s_{l_x} **do**
      let ℓ' be the length of R;
      set $f'_x = \max\{j \in [\ell'] \mid \delta_{\mathsf{R}\to\mathsf{S}}(j) = f_x\}$ and $l'_x = \min\{j \in [\ell'] \mid \delta_{\mathsf{R}\to\mathsf{S}}(j) = l_x\}$;
      set $\mathsf{S}^x = \mathsf{Ins}(\mathsf{R}, x, X, f'_x, l'_x)$;
      (observe that by construction (f'_x, l'_x) is valid with respect to X in R);
      if $\mathsf{width}(\mathsf{S}^x) \leq w$, **then** add $\mathsf{rep}(\mathsf{S}^x)$ to $\mathbf{D}_w(\mathbf{G}^x)$;
    end
  end
end
return $\mathbf{D}_w(\mathbf{G}^x)$.

that is: if S^x is generated from a B-boundaried $\mathsf{S} \in \mathbf{D}_w(\mathbf{G})$, then $\mathsf{rep}(\mathsf{S}^x)$ belongs to $\mathbf{D}_w(\mathbf{G}^x)$. The proofs of completeness and soundness rely both on Lemma 19, and Lemma 20. As for Forget Routine, the time complexity follows from Lemma 10.

▶ **Theorem 24.** *Algorithm 2 computes* $\mathbf{D}_w(\mathbf{G}^x)$ *in* $2^{O(k(w+\log k))}$*-time, where* $k = |B|$.

3.4 The dynamic programming algorithm

We are now in position to prove Theorem 1. We are given a nice path-decompositon $\mathsf{Q} = \langle B_1, \ldots, B_q \rangle$ of G of width at most k and for each $i \in [q]$, we consider the boundaried graphs $\mathbf{G}_i = (G[V_i], B_i)$, where $V_i = \bigcup_{1 \leq h \leq i} B_h$. We have a way to compute $\mathbf{D}_w(G_{i+1})$ from $\mathbf{D}_w(G_i)$, in $2^{O(k^2)} \cdot n$ time, using the algorithms of Theorem 23 or Theorem 24 depending on whether B_i is an insertion or a forget bag. We next describe the set $\mathbf{D}_{w+1}(\mathbf{G}_1)$. For this, we take the representative set $\mathbf{Rep}_{w+1}(\mathbf{G}_1)$ that consists for the following four possible connected B_1-boundaried sequences:

- $\mathsf{S}_1 = \langle (\{x\}, \{\{x\}\}, 0) \rangle$,
- $\mathsf{S}_2 = \langle (\emptyset, \{\emptyset\}, 0), (\{x\}, \{\{x\}\}, 0) \rangle$,
- $\mathsf{S}_3 = \langle (\emptyset, \{\emptyset\}, 0), (\{x\}, \{\{x\}\}, 0), (\emptyset, \{\{x\}\}, 0) \rangle$, and
- $\mathsf{S}_4 = \langle (\{x\}, \{\{x\}\}, 0), (\emptyset, \{\{x\}\}, 0) \rangle$.

As already noticed $\mathsf{cpw}(G) \leq k$ if and only if $\mathbf{D}_{w+1}(\mathbf{G}_q) \neq \emptyset$. This completes proof of the decision version of Theorem 1. In [8, Section 6] Bodlaender and Kloks explained how to turn their decision algorithm for pathwidth and treewidth to one that is able to construct, in case of a positive answer, the corresponding decomposition. It is straightforward to see that the modification of [8, Section 6] that transforms the decision algorithm for pathwidth to one that also constructs the corresponding path-decomposition also applies to our algorithm for connected pathwidth. This completes the proof of Theorem 1.

If we now use the result of Fürer [20] for constructing a path-decomposition of width at most k in $2^{O(k^2)} \cdot n$ time and taking into account that $\mathsf{pw}(G) \leq \mathsf{cpw}(G)$, we have the following.

▶ **Theorem 25.** *One may construct an algorithm that, given an* n*-connected graph* G *and a non-negative integer* k*, either outputs a connected path-decomposition of* G *of width at most* k *or correctly reports that such a decomposition does not exist in* $2^{O(k^2)} \cdot n$ *time.*

References

1 Isolde Adler, Christophe Paul, and Dimitrios M. Thilikos. Connected search for a lazy robber. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 150 of *Leibniz International Proceedings in Informatics*, pages 7:1–7:14, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.7.

2 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

3 Lali Barrière, Paola Flocchini, Fedor V. Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, and Dimitrios M. Thilikos. Connected graph searching. *Information and Computation*, 219:1–16, 2012. doi:10.1016/j.ic.2012.08.004.

4 Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In 14th Annual ACM Symposium on Parallel Algorithms and Architectures, *SPAA*, pages 200–209. ACM, 2002. doi:10.1145/564870.564906.

5 Lali Barrière, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M. Thilikos. Searching is not jumping. In 29th International Workshop on Graph-Theoretic Concepts in Computer Science, *WG*, volume 2880 of *Lecture Notes in Computer Science*, pages 34–45, 2003. doi:10.1007/978-3-540-39890-5_4.

6 Hans L. Bodlaender, Michael R. Fellows, and Dimitrios M. Thilikos. Derivation of algorithms for cutwidth and related graph layout parameters. *Journal of Computer and System Sciences*, 75(4):231–244, 2009. doi:10.1016/j.jcss.2008.10.003.

7 Hans L. Bodlaender, Lars Jaffke, and Jan Arne Telle. Typical sequences revisited - computing width parameters of graphs. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS*, volume 154 of *Leibniz International Proceedings in Informatics*, pages 57:1–57:16, 2020. doi:10.4230/LIPIcs.STACS.2020.57.

8 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.

9 Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In 24th International Colloquium on Automata, Languages and Programming, *ICALP*, volume 1256 of *Lecture Notes in Computer Science*, pages 627–637, 1997. doi:10.1007/3-540-63165-8_217.

10 Hans L. Bodlaender and Dimitrios M. Thilikos. Computing small search numbers in linear time. In First International Workshop on Parameterized and Exact Computation, *IWPEC*, volume 3162 of *Lecture Notes in Computer Science*, pages 37–48, 2004. doi:10.1007/978-3-540-28639-4_4.

11 Mikołaj Bojańczyk and Michal Pilipczuk. Optimizing tree decompositions in MSO. In 34th Symposium on Theoretical Aspects of Computer Science, *STACS*, volume 66 of *Leibniz International Proceedings in Informatics*, pages 15:1–15:13, 2017. doi:10.4230/LIPIcs.STACS.2017.15.

12 R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers (A publication of the Southwestern Region of the National Speleological Society)*, VI(5):72–78, 1967.

13 Gary Chartrand, Ping Zhang, Teresa W. Haynes, Michael A. Henning, Fred R. McMorris, and Robert C. Brigham. Graphical measurement. In *Handbook of Graph Theory*, pages 872–951. Chapman & Hall / Taylor & Francis, 2003. doi:10.1201/9780203490204.ch9.

14 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

15 Bruno Courcelle and Jens Lagergren. Equivalent definitions of recognizability for sets of graphs of bounded tree-width. *Mathematical Structure for Computer Sciecnce*, 6(2):141–165, 1996. doi:10.1017/S096012950000092X.

16 Dariusz Dereniowski. From pathwidth to connected pathwidth. *SIAM Journal on Discrete Mathematics*, 26(4):1709–1732, 2012. doi:10.1137/110826424.

17 Dariusz Dereniowski, Dorota Osula, and Paweł Rzążewski. Finding small-width connected path decompositions in polynomial time. *Theoretical Computer Science*, 794:85–100, 2019. doi:10.1016/j.tcs.2019.03.039.

18 Fedor V. Fomin. Complexity of connected search when the number of searchers is small. Open Problems of GRASTA 2017: The 6th Workshop on GRAph Searching, Theory and Applications, 2017.

19 Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008. doi:10.1016/j.tcs.2008.02.040.

20 Martin Fürer. Faster computation of path-width. In 27th International Workshop on Combinatorial Algorithms*, IWOCA*, Lecture Notes in Computer Science, pages 385–396, 2016. doi:10.1007/978-3-319-44543-4_30.

21 P. A. Golovach. Equivalence of two formalizations of a search problem on a graph (Russian). *Vestnik Leningrad. Univ. Mat. Mekh. Astronom.*, vyp. 1:10–14, 122, 1989. translation in Vestnik Leningrad Univ. Math. 22 (1989), no. 1, 13–19.

22 Petr A. Golovach (П. А. Головач). A topological invariant in pursuit problems, (Об одном топологическом инварианте в задачах преследования). *Differentsialânye Uravneniya (Differential Equations),* (Дифференц. уравнения), 25(6):923–929, 1989. URL: http://mi.mathnet.ru/de6861.

23 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Constructive algorithm for path-width of matroids. In 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA*, pages 1695–1704, 2016. doi:10.1137/1.9781611974331.ch116.

24 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. The "art of trellis decoding" is fixed-parameter tractable. *IEEE Transactions on Information Theory*, 63(11):7178–7205, 2017. doi:10.1109/TIT.2017.2740283.

25 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Finding branch-decompositions of matroids, hypergraphs, and more. In 45th International Colloquium on Automata, Languages, and Programming*, ICALP*, volume 107 of *Leibniz International Proceedings in Informatics*, pages 80:1–80:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.80.

26 Mamadou Moustapha Kanté, Christophe Paul, and Dimitrios M. Thilikos. A linear fixed parameter tractable algorithm for connected pathwidth. *CoRR*, abs/2004.11937, 2020. arXiv:2004.11937.

27 Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 1992. doi:10.1016/0020-0190(92)90234-M.

28 Lefteris M. Kirousis and Christos H. Papadimitriou. Interval graphs and seatching. *Discrete Mathematics*, 55(2):181–184, 1985. doi:10.1016/0012-365X(85)90046-9.

29 Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(3):205–218, 1986. doi:10.1016/0304-3975(86)90146-5.

30 Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In 18th International Colloquium on Automata, Languages and Programming*, ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 532–543, 1991. doi:10.1007/3-540-54233-7_161.

31 Rolf H. Möhring. Graph problems related to gate matrix layout and PLA folding. In *Computational graph theory*, volume 7 of *Comput. Suppl.*, pages 17–51. Springer, 1990.

32 Ronan Pardo Soares. *Pursuit-Evasion, Decompositions and Convexity on Graphs*. Theses, Université Nice Sophia Antipolis, November 2013. URL: https://tel.archives-ouvertes.fr/tel-00908227.

33 Torrence D. Parsons. Pursuit-evasion in a graph. In International Conference on Theory and applications of graphs, volume 642 of *Lecture Notes in Mathematics*, pages 426–441. Springer, 1978.

34 Torrence D. Parsons. The search number of a connected graph. In 9th Southeastern Conference on Combinatorics, Graph Theory and Computing, volume XXI of *Congress. Numer., XXI*, pages 549–554. Utilitas Math., 1978.

35 Nikolai N. Petrov (Н. Н. Петров). A problem of pursuit in the absence of information on the pursued, (Задачи преследования при отсутствии информации об убегающем). *Differentsial'nye Uravneniya (Differential Equations),* (Дифференц. уравнения), 18(8):1345–1352, 1468, 1982. URL: http://mi.mathnet.ru/de4628.

36 Neil Robertson and Paul D. Seymour. Graph minors. I. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

37 Neil Robertson and Paul D. Seymour. Graph minors. XX. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

38 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Constructive linear time algorithms for small cutwidth and carving-width. In 11th International Conference on Algorithms and Computation*, ISAAC*, volume 1969 of *Lecture Notes in Computer Science*, pages 192–203, 2000. doi:10.1007/3-540-40996-3_17.

39 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, 2005. doi:10.1016/j.jalgor.2004.12.001.

40 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth II: algorithms for partial w-trees of bounded degree. *Journal of Algorithms*, 56(1):25–49, 2005. doi:10.1016/j.jalgor.2004.12.003.

Exploiting c-Closure in Kernelization Algorithms for Graph Problems

Tomohiro Koana
Technische Universität Berlin, Algorithmics and Computational Complexity, Germany
tomohiro.koana@tu-berlin.de

Christian Komusiewicz
Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany
komusiewicz@informatik.uni-marburg.de

Frank Sommer
Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany
fsommer@informatik.uni-marburg.de

Abstract

A graph is c-closed if every pair of vertices with at least c common neighbors is adjacent. The c-closure of a graph G is the smallest number c such that G is c-closed. Fox et al. [SIAM J. Comput. '20] defined c-closure and investigated it in the context of clique enumeration. We show that c-closure can be applied in kernelization algorithms for several classic graph problems. We show that Dominating Set admits a kernel of size $k^{\mathcal{O}(c)}$, that Induced Matching admits a kernel with $\mathcal{O}(c^7k^8)$ vertices, and that Irredundant Set admits a kernel with $\mathcal{O}(c^{5/2}k^3)$ vertices. Our kernelization exploits the fact that c-closed graphs have polynomially-bounded Ramsey numbers, as we show.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Fixed-parameter tractability, kernelization, c-closure, Dominating Set, Induced Matching, Irredundant Set, Ramsey numbers

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.65

Related Version A continously updated version of the paper is available at `https://arxiv.org/abs/2005.03986`.

Funding *Tomohiro Koana*: Supported by the Deutsche Forschungsgemeinschaft (DFG), project FPTinP, NI 369/19.
Frank Sommer: Supported by the Deutsche Forschungsgemeinschaft (DFG), project MAGZ, KO 3669/4-1.

Acknowledgements This work was started at the research retreat of the TU Berlin Algorithms and Computational Complexity group held in September 2019 at Schloss Neuhausen (Prignitz).

1 Introduction

Parameterized complexity [10, 15] aims at understanding which properties of input data can be used in the design of efficient algorithms for problems that are hard in general. The properties of input data are encapsulated in the notion of a *parameter*, a numerical value that can be attributed to each input instance I. For a given hard problem and parameter k, the first aim is to find a *fixed-parameter algorithm*, an algorithm that solves the problem in $f(k) \cdot |I|^{\mathcal{O}(1)}$ time. Such an algorithm is efficient when f grows moderately and k takes on small values. A second aim is to provide a *kernelization*. This is an algorithm that given any instance (I, k) of a parameterized problem computes in polynomial time an equivalent instance of size $g(k)$. If g grows not too much and k takes on small values, then a kernelization

Table 1 A comparison of the c-closure with the number n of vertices, number m of edges, and the maximum degree Δ in social and biological networks.

Instance name	n	m	Δ	c
adjnoun-adjacency	112	425	49	14
arenas-jazz	198	2742	100	42
ca-netscience	379	914	34	5
bio-celegans	453	2025	237	26
bio-diseasome	516	1188	50	9
soc-wiki-Vote	889	2914	102	18
arenas-email	1133	5451	71	19
bio-yeast	1458	1948	56	8
ca-CSphd	1882	1740	46	3
soc-hamsterster	2426	16630	273	77
ca-GrQc	4158	13422	81	43
soc-advogato	5167	39432	807	218
bio-dmela	7393	25569	190	72
ca-HepPh	11204	117619	491	90
ca-AstroPh	17903	196972	504	61
soc-brightkite	56739	212945	1134	184

provably shrinks large input instances and thus gives a guarantee for the efficacy of data reduction rules. A central part of the design of good parameterized algorithms is thus the identification of suitable parameters.

A good parameter should have the following advantageous traits. Ideally, it should be easy to understand and compute.[1] It should take on small values in real-world input data. It should describe input properties that are not captured by other parameters. Finally, many problems should be amenable to parameterization using this parameter. In other words, the parameter should help when designing fixed-parameter algorithms or kernelizations.

Fox et al. [19] recently introduced the graph parameter *c-closure* which describes a structural feature of many real-world graphs: When two vertices have many common neighbors, it is likely that they are adjacent. More precisely, the c-closure of a graph is defined as follows.

▶ **Definition 1.1** ([19]). *A graph $G = (V, E)$ is* c-closed *if every pair of vertices $u \in V$ and $v \in V$ with at least c common neighbors is adjacent. The c-closure of a graph is the smallest number c such that G is c-closed.*

The parameter has many of the desirable traits mentioned above: It is easy to understand and easy to compute. Moreover, social networks are c-closed for relatively small values of c [19] (see also Table 1). In addition, the c-closure of a graph gives a new class of graphs which are not captured by other measures. This follows from the observation that every complete graph is 1-closed. Hence, a graph can have bounded c-closure but, for example, unbounded degeneracy and thus unbounded treewidth. Conversely, the graph consisting of two vertices u and v and many vertex-disjoint u-v-paths of length two is 2-degenerate,

[1] This cannot always be guaranteed. For example, the important parameter treewidth is hard to compute and not as easily understood as simpler parameters.

has treewidth two, and unbounded c-closure. Generally, one may observe that c-closure is different from many common parameterizations which measure, in different ways, the sparseness of the input graph. In this sense, the structure described by the c-closure is novel. The aim of this work is to show that c-closure also has the final, most important trait: it helps when designing fixed-parameter algorithms.

Fox et al. [19] applied c-closure to the enumeration of maximal cliques, showing that a c-closed graph may have at most $3^{(c-1)/3} \cdot n^2$ maximal cliques. In combination with known clique enumeration algorithms this implies that all maximal cliques of a graph can be enumerated in $\mathcal{O}^*(3^{c/3})$ time. We are not aware of any further fixed-parameter algorithms that make use of the c-closure parameter.

An easy example for how parameterization by c-closure helps can be seen for the Independent Set problem. In Independent Set we are given an undirected graph $G = (V, E)$ and an integer k and want to determine whether G contains a set of k vertices that are pairwise nonadjacent. Independent Set is W[1]-hard when parameterized by k [15, 10]. When one uses the maximum degree of G as an additional parameter, then Independent Set has a trivial kernelization: Any graph with at least $(\Delta + 1)k$ vertices has an independent set of size at least k. With the following data reduction rule, we can obtain a kernelization for the combination of c and k.

▶ **Reduction Rule 1.2.** *If G contains a vertex v of degree at least $(c-1)(k-1)+1$, then remove v from G.*

To see that Reduction Rule 1.2 is correct, we need to show that the resulting graph G' has an independent set I of size k if and only if the original graph G has one. The nontrivial direction to show is that if G has an independent set I of size k, then so does G'. Since this clearly holds for $v \notin I$, we assume that $v \in I$. To replace v by some other vertex, we make use of the c-closure: Every vertex u in $I \setminus \{v\}$ has at most $c-1$ neighbors in common with v since u and v are nonadjacent. Thus, at most $(c-1)(k-1)$ neighbors of v are also neighbors of some vertex in $I \setminus \{v\}$. Consequently, some neighbor w of v has no neighbors in $I \setminus \{v\}$ and, therefore, $(I \setminus \{v\}) \cup \{w\}$ is an independent set of size k in G'.

Applying Reduction Rule 1.2 exhaustively results in an instance with maximum degree less than ck which, due to the discussion above, directly gives the following.

▶ **Proposition 1.3.** Independent Set *admits a kernel with at most ck^2 vertices.*

Motivated by this simple result for a famous graph problem, we study how c-closure can be useful for further classic graph problems when they are parameterized by a combination of c and the solution size parameter k. We obtain the following positive results. In Section 4, we show that Dominating Set admits a kernel of size $k^{\mathcal{O}(c)}$ computable in $\mathcal{O}^*(2^c)$ time and show that this kernelization is asymptotically optimal with respect to the dependence of the exponent on c. Our results also hold for the more general Threshold Dominating Set problem where each vertex needs to be dominated r times. In Section 5, we show that Induced Matching admits a kernel with $\mathcal{O}(c^7k^8)$ vertices by means of LP relaxation of Vertex Cover. Finally in Section 6, we show that Irredundant Set admits a kernel with $\mathcal{O}(c^{5/2}k^3)$ vertices. All kernelizations exploit a bound on Ramsey numbers for c-closed graphs, which we prove in Section 3. This bound is – in contrast to Ramsey numbers of general graphs – polynomial in the size of a sought clique and independent set. We believe that this bound on the Ramsey numbers is of independent interest and that it provides a useful tool in the design of fixed-parameter algorithms for more problems on c-closed graphs.

2 Preliminaries

For $m \leq n \in \mathbb{N}$, we write $[m, n]$ to denote the set $\{m, m+1, \ldots, n\}$ and $[n]$ for $[1, n]$. For a graph G, we denote its vertex set and edge set by $V(G)$ and $E(G)$, respectively. Let $X, Y \subseteq V(G)$ be vertex subsets. We use $G[X]$ to denote the subgraph induced by X. We also use $G[X, Y] := (X \cup Y, \{xy \in E(G) \mid x \in X, y \in Y\})$ to denote the bipartite subgraph induced by X, Y for $X \cap Y = \emptyset$. We let $G - X$ denote the graph obtained by removing vertices in X. We denote by $N_G(X) := \{y \in V(G) \setminus X \mid xy \in E(G), x \in X\}$ and $N_G[X] := N_G(X) \cup X$, the open and closed neighborhood of X, respectively. For all these notations, when X is a singleton $\{x\}$ we may write x instead of $\{x\}$. Let $v \in V(G)$. We denote the degree of v by $\deg_G(v)$. We call v *isolated* if $\deg_G(v) = 0$ and *non-isolated* otherwise. We also say that v is a *leaf vertex* if $\deg_G(v) = 1$ and a *non-leaf vertex* if $\deg_G(v) \geq 2$. Moreover, we say that v is *simplicial* if $N_G(v)$ is a clique. The maximum and minimum degree of G are $\Delta_G := \max_{v \in V(G)} \deg_G(v)$ and $\delta_G := \min_{v \in V(G)} \deg_G(v)$, respectively. The degeneracy of G is $d_G := \max_{S \subseteq V(G)} \delta_{G[S]}$. We say that G is c-closed for $c = \max(\{0\} \cup \{|N_G(u) \cap N_G(v)| \mid uv \notin E(G)\}) + 1$. In particular, any cluster graph (a disjoint union of complete graphs) is 1-closed. We drop the subscript $\cdot_G$ when it is clear from context. A graph G has girth g if the shortest cycle in G has length g.

In this paper, we investigate the parameterized complexity of various problems whose input comprises of a graph G and an integer k. A problem is *fixed-parameter tractable* if it can be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time where $n := |V(G)|$ and f is some computable function. Instances (G, k) and (G', k') are *equivalent* if (G, k) is a Yes-instance if and only if (G', k') is a Yes-instance. A *kernelization algorithm* is a polynomial-time algorithm which transforms an instance (G, k) into an equivalent instance (G', k') such that $|V(G')| + k' \leq g(k)$, where g is some computable function. It is well-known that a problem is fixed-parameter tractable if and only if it admits a kernelization algorithm. Our kernelization algorithms consist of a sequence of *reduction rules*. Given an instance (G, k), a reduction rule computes an instance (G', k'). We will develop kernelization algorithms for c-closed graphs. For our purposes, we say that a reduction rule is *correct* if the input instance (G, k) for a c-closed graph G is equivalent to the resulting instance (G', k') and G' is also c-closed. For more information on parameterized complexity, we refer to the standard monographs [10, 15].

We will make use of the following observations throughout this work.

▶ **Observation 2.1.** *If G is c-closed, then $G - v$ is also c-closed for any $v \in V(G)$.*

▶ **Observation 2.2.** *Let C be a maximal clique in a c-closed graph G. Then $|C \cap N(v)| < c$ for every $v \in V(G) \setminus C$.*

▶ **Observation 2.3.** *Let G be a c-closed graph and let C be a*

- *clique of size at most $c - 1$ in G or*
- *a maximal clique in G.*

Then, the graph G' obtained by attaching a simplicial vertex v to C (that is, $N_{G'}(v) = C$) is c-closed.

Some proofs are deferred to a full version of this work.

3 On Ramsey Numbers of c-Closed Graphs

Ramsey's theorem states that there is a function R such that any graph G with at least $R(a, b)$ vertices contains a clique of size a or an independent set of size b, for any $a, b \in \mathbb{N}$. The numbers $R(a, b)$ are referred to as Ramsey numbers. It is known that $R(t, t) > 2^{t/2}$ for

any $t \geq 3$ [17, 23] and hence $R(t,t)$ grows exponentially with t. Here, we show that the Ramsey number $R(a,b)$ is actually polynomial in a and b in c-closed graphs. Let $R_c(a,b) := (c-1) \cdot \binom{b-1}{2} + (a-1)(b-1) + 1$.

▶ **Lemma 3.1.** *Any c-closed graph G on at least $R_c(a,b)$ vertices contains a clique of size a or an independent set of size b.*

Proof. Assume to the contrary that G has no clique of size a and no independent set of size b. Let $I = \{v_1, \ldots, v_{|I|}\}$ be a maximum independent set of G. Also let C_i be the set of vertices adjacent to v_i (including v_i) and nonadjacent to any other vertex in I (that is, $C_i = N[v_i] \setminus N(I \setminus \{v_i\})$) for each $i \in [|I|]$. Suppose that there exist $u \neq u' \in C_i$ with $uu' \notin E(G)$. Then, $(I \setminus \{v_i\}) \cup \{u, u'\}$ is an independent set of size $|I|+1$, which contradicts the choice of I. Hence, we see that C_i is a clique. Note that every vertex of G is adjacent to some vertex in I due to the maximality of I. It follows that

$$|V(G)| \leq \sum_{i \in [|I|]} |C_i| + \sum_{i<j \in [|I|]} |N(v_i) \cap N(v_j)|.$$

Note that $|C_i| \leq a-1$ for each $i \in [|I|]$ and $|N(v_i) \cap N(v_j)| \leq c-1$ for $i < j \in [|I|]$ by the c-closure of G. Since $|I| < b$, we have a contradiction on $|V(G)|$. ◀

The bound in Lemma 3.1 is essentially tight: Consider a graph G consisting of a disjoint union of $b-1$ complete graphs, each of order $a-1$. Note that G is c-closed for any $c \in \mathbb{N}$ and that G has no clique of size a or independent set of size b. Thus, we have a tight bound for $c = 1$. This example also suggests that the bound in Lemma 3.1 cannot be asymptotically improved for $a \geq cb$.

4 (Threshold) Dominating Set

In this section we show that Threshold Dominating Set admits a kernel with $k^{\mathcal{O}(cr)}$ vertices. The problem is defined as follows.

Threshold Dominating Set
Input: A graph G and $r, k \in \mathbb{N}$.
Question: Is there a vertex set $D \subseteq V(G)$ such that $|D| \leq k$ and each vertex $v \in V(G)$ is dominated by D at least r times, that is, $|N[v] \cap D| \geq r$?

Dominating Set is the special case of Threshold Dominating Set when $r = 1$.

Dominating Set is W[2]-hard when parameterized by k even in bipartite or split graphs [30]. Furthermore, Dominating Set was shown to remain NP-hard on graphs with girth at least t for any constant t [2]. Hence, Dominating Set is NP-hard even on 2-closed graphs.

There are several fixed-parameter tractability results in restricted graph classes: When the graph G contains no induced C_3 or C_4, Dominating Set admits a kernel of $\mathcal{O}(k^3)$ vertices and Threshold Dominating Set is fixed-parameter tractable [30]. Furthermore, Dominating Set in d-degenerate graphs can be solved in $k^{\mathcal{O}(dk)}n$ time [3]. This result was extended to an algorithm with running time $\mathcal{O}^*(k^{\mathcal{O}(dkr)})$ for Threshold Dominating Set in d-degenerate graphs [21].

When the graph G does not contain the complete bipartite graph $K_{i,j}$ for fixed $j \leq i$ as a (not necessarily induced) subgraph, Dominating Set admits a kernel of $\mathcal{O}((j+1)^{i+1}k^{i^2})$ vertices which can be computed in $\mathcal{O}(n^i)$ time [29]. Since d-degenerate graphs do not contain

a $K_{d+1,d+1}$ as a subgraph, Dominating Set admits a kernelization of $\mathcal{O}(k^{(d+1)^2})$ vertices computable in $\mathcal{O}^*(2^d)$ time [29]. This kernel size is essentially optimal since Dominating Set in d-degenerate graphs admits no kernel of size $\mathcal{O}(k^{(d-3)(d-1)-\epsilon})$ for any $\epsilon > 0$ unless NP $\subseteq$ coNP/poly [11]. When G does not contain the complete bipartite graph $K_{t,t}$ as a (not necessarily induced) subgraph, Dominating Set can be solved in $2^{\mathcal{O}(tk^2(4k)^t)}$ time [31]. Since each d-degenerate graph does not contain a $K_{d+1,d+1}$ as a subgraph, this extends the result of [3]. None of the above kernelizations and fixed-parameter algorithms implies a tractability result on c-closed graphs, since the respective structural restrictions on G all exclude cliques of some size. Moreover, since any graph without induced C_3 or C_4 is 2-closed, our results extend the kernelization algorithms for these graphs to a more general class of graphs.

To obtain a kernel for Threshold Dominating Set in c-closed graphs we first provide a kernelization for a more general, colored variant defined as follows. The input graph is a *bw-graph*, where the vertex set $V(G)$ is partitioned into black vertices B and white vertices W. We only require to dominate black vertices r times. The problem is defined as follows.

BW-Threshold Dominating Set
Input: A bw-graph G and $r, k \in \mathbb{N}$.
Question: Does G contain a *bw-threshold dominating set* $D \subseteq V(G)$, that is, a set such that $|N[v] \cap D| \geq r$ for each vertex $v \in B$, of size at most k?

Clearly, each instance (G, k) of Threshold Dominating Set is equivalent to the instance (G, k) of BW-Threshold Dominating Set where each vertex is black.

4.1 Polynomial Kernel in c-closed Graphs

We first develop a kernelization algorithm for BW-Threshold Dominating Set and then we will remove colors at the end. Before we present our reduction rules, we prove the following lemma, which will simplify some proofs later in this section.

▶ **Lemma 4.1.** *Let (G, k) be a Yes-instance of* BW-Threshold Dominating Set *and let v be a simplicial vertex with at least r neighbors. Then, there exists a bw-threshold dominating set D of size at most k such that $v \notin D$.*

Proof. Suppose that G has a bw-threshold dominating set D of size at most k. We are immediately done if $v \notin D$, so we can assume that $v \in D$. If $N[v] \subseteq D$, then $D \setminus \{v\}$ is a bw-threshold dominating set of size at most k. Otherwise, there is a vertex $u \in N(v) \setminus D$ and $(D \setminus \{v\}) \cup \{u\}$ is a bw-threshold dominating set of size at most k not containing v. ◀

We first aim to bound the number of black vertices. Our first reduction rule exploits the fact that any bw-threshold dominating set includes at least r vertices in C, where C is a maximal clique containing sufficiently many black vertices.

▶ **Reduction Rule 4.2.** *Let C be a maximal clique containing at least ck black vertices. Then,*
1. *add a vertex u and add an edge uv for each $v \in C$,*
2. *color u black, and*
3. *color all the vertices in C white.*

Note that Reduction Rule 4.2 does not add new maximal cliques. Hence, Reduction Rule 4.2 can be applied exhaustively in $\mathcal{O}^*(3^{c/3})$ time, because all maximal cliques can be enumerated in $\mathcal{O}^*(3^{c/3})$ time [19].

▶ **Lemma 4.3.** *Reduction Rule 4.2 is correct.*

Proof. Let D be a bw-threshold dominating set of G of size at most k. We claim that $|D \cap C| \geq r$. Assume to the contrary that $|D \cap C| \leq r-1$. By Observation 2.2, each vertex in $D \setminus C$ dominates at most $c-1$ vertices in C. Since C contains at least ck black vertices, there is a black vertex in C that is not dominated r times by D, a contradiction. Thus, $|D \cap C| \geq r$. Let G' be the graph obtained as a result of Reduction Rule 4.2. Since $uv \in E(G)$ for each $v \in C$, we see that $|N_{G'}(u) \cap D| \geq r$ and thus D is also a bw-threshold dominating set of the graph G'. The other direction of the equivalence follows from Lemma 4.1. Finally, note that Reduction Rule 4.2 maintains the c-closure by Observation 2.3. ◀

We will assume henceforth that Reduction Rule 4.2 has been applied exhaustively. Recall that each c-closed graph on at least $R_c(a,b) = (c-1)\binom{b-1}{2} + (a-1)(b-1) + 1$ vertices contains a clique of size a or an independent set of size b by Lemma 3.1. Since G does not contain any black clique of size at least ck, each subgraph of G with at least $\rho := R_c(ck, k+1)$ black vertices contains an independent set of at least $k+1$ black vertices. We take advantage of this observation in the following two reduction rules.

▶ **Reduction Rule 4.4.** *Suppose that $r \leq c-1$. We define Reduction Rule 4.4.i for each $i \in [1, c-r]$ as follows: Let C be a clique of size exactly $c-i$ and let $P := B \cap \{v \in V(G) \mid C \subseteq N(v)\}$ be the set of common black neighbors of C. If $|P| > k^{i-1}\rho$, then*

1. *add a vertex u and add an edge uv for each $v \in C$,*
2. *color u black, and*
3. *color all the vertices in C and P white.*

Apply Reduction Rule 4.4.i in increasing order of i.

Since $|P| > k^{i-1}\rho$, the common black neighbors P of C include an independent set I of size $k+1$. To apply Reduction Rule 4.4.i exhaustively, we consider each pair of vertices v, v' from V and then we consider each clique in the common neighborhood $N(v) \cap N(v')$. Since $|N(v) \cap N(v')| < c$, Reduction Rule 4.4 can be applied exhaustively in $\mathcal{O}^*(2^c)$ time.

▶ **Lemma 4.5.** *Reduction Rule 4.4 is correct.*

Proof. Let C be a clique of size exactly $c-i$ which has more than $k^{i-1}\rho$ common black neighbors P. We prove the following claim for increasing $i \in [1, c-r]$.

▷ Claim. If Reduction Rule 4.4.j has been applied exhaustively for each $j \in [i-1]$, then any bw-threshold dominating set D of size at most k includes at least r vertices of C.

Proof. Suppose that $i = 1$. We assume to the contrary that $|D \cap C| \leq r-1$. Recall that there is no clique of at least ck black vertices by Reduction Rule 4.2. Since $|P| > \rho$, we see from Lemma 3.1 that P contains an independent set I of at least $k+1$ vertices. By pigeon-hole principle, there exists a vertex $w \in D \setminus C$ which is adjacent to at least two vertices x and y in I. Hence, x and y have at least c common neighbors $C \cup \{w\}$, contradicting the c-closure of G. It follows that D contains at least r vertices of C.

Suppose that $i \in [2, c-r]$. Again we assume to the contrary that $|D \cap C| \leq r-1$. Since $|P| > k^{i-1}\rho$, there exists a vertex $w \in D \setminus C$ that dominates at least $k^{i-2}\rho$ vertices of P. Observe that w and each vertex in C have at least $k^{i-2}\rho > c$ common neighbors in P. Hence, we have $vw \in E(G)$ for each $v \in C$. Thus, $C \cup \{w\}$ is a clique of size $c-i+1$ with at least $k^{i-2}\rho$ common black neighbors. However, this contradicts the fact that Reduction Rule 4.4.$(i-1)$ has been applied exhaustively. Therefore, we obtain $|D \cap C| \geq r$. ◁

Let G' be the graph obtained as a result of Reduction Rule 4.4. By the above claim, any bw-threshold dominating set in G is also a bw-threshold dominating set in G'. The other direction follows from Lemma 4.1. Finally, note that G' is c-closed by Observation 2.3. ◀

▶ **Reduction Rule 4.6.** *Suppose that $r \geq c$. Let C be a clique of size exactly $c-1$ and let $P := B \cap \{v \in V(G) \mid C \subseteq N(v)\}$ be the set of common black neighbors of C. If $|P| > \rho$, then return No.*

▶ **Lemma 4.7.** *Reduction Rule 4.6 is correct.*

Proof. Suppose that G has a bw-threshold dominating set D of size at most k. We show that for each clique C of size $c-1$, there are at most ρ common black neighbors. Assume to the contrary that $|P| > \rho$ for $P := B \cap \{v \in V(G) \mid C \subseteq N(v)\}$. Then, there is an independent set $I \subseteq P$ of size $k+1$ by Lemma 3.1. Since $r \geq c$, there are two vertices $x, y \in I$ that are adjacent to a vertex $v \in D \setminus C$. Now, we have a contradiction to the c-closure of G, because x and y have $|C \cup \{v\}| = c$ neighbors. ◀

Note that Reduction Rule 4.6 also can be applied exhaustively in $\mathcal{O}^*(2^c)$ time. Hereafter, we will assume that Reduction Rules 4.4 and 4.6 have been applied exhaustively. In the next lemma, we show that the number of black neighbors is upper-bounded for each vertex.

▶ **Lemma 4.8.** *Each vertex has at most $k^{c-1}\rho$ black neighbors for any Yes-instance (G, k).*

Proof. First, suppose that $r \leq c-1$. To prove the lemma, we prove the following more general claim:

▷ Claim. Let $i \in [r]$ and let C be a clique of size exactly i with the set P of common black neighbors. Then, $|P| \leq k^{c-i}\rho$.

Proof. We prove the claim by induction on decreasing i. By Reduction Rule 4.4, the claim holds for the base case $i = r$. Suppose that $i < r$. Since $|C| = i \leq r-1$, for any bw-threshold dominating set D of size at most k, there is a vertex $v \in D \setminus C$ that dominates at least $|P|/k$ vertices of P. As $|P|/k > c$, the set $C \cup \{v\}$ is a clique with $|P|/k$ common black neighbors. By induction hypothesis, we obtain $|P|/k \leq k^{c-i-1}\rho$ and equivalently, $|P| \leq k^{c-i}\rho$. ◁

Observe that the lemma follows from the above claim for $i = 1$. Using Reduction Rule 4.6, the lemma can be proven analogously for the case $r \geq c$ as well. ◀

By Lemma 4.8, there are at most $k^c\rho$ black vertices for any Yes-instance (G, k):

▶ **Reduction Rule 4.9.** *If G contains more than $k^c\rho$ black vertices, then return No.*

To compute a kernel it remains to upper-bound the number of white vertices in G.

▶ **Reduction Rule 4.10.** *Let w be a white vertex in G. If there exist at least r vertices $v_1, \ldots, v_r$ such that $N(w) \cap B \subseteq N[v_i] \cap B$ for each $i \in [r]$, then remove w.*

It is easy to see that Reduction Rule 4.10 can be applied exhaustively in polynomial time.

▶ **Lemma 4.11.** *Reduction Rule 4.10 is correct.*

Proof. Let $G' := G - w$. Suppose that G has a bw-threshold dominating set D of size at most k. If $w \notin D$, then D is also a bw-threshold dominating set of G'. Hence, we can assume that $w \in D$. If $v_i \in D$ for all $i \in [r]$, then $D \setminus \{w\}$ is a bw-threshold dominating set for G and hence also for G'. Otherwise, there exists some $i \in [r]$ with $v_i \notin D$. Since $N(w) \cap B \subseteq N[v_i] \cap B$, the set $(D \setminus \{w\}) \cup \{v_i\}$ is a bw-threshold dominating set of size at most k of G and G'. The other direction follows trivially. Since Reduction Rule 4.10 only deletes white vertices the c-closure is maintained. ◀

In the following, we will assume that Reduction Rule 4.10 has been applied exhaustively. Now, we obtain a bound on the number of white vertices in G.

▶ **Lemma 4.12.** *The graph G contains $\mathcal{O}(c|B|^2 + |B|^{r-1})$ white vertices.*

Proof. Since Reduction Rule 4.10 has been applied exhaustively, G contains at most r white vertices w such that $N(w) \subseteq W$. Hence, it remains to bound the number of white vertices with at least one black neighbor. Observe that by the c-closure of G, there are $\mathcal{O}(c\,|B|^2)$ white vertices that are neighbors of two nonadjacent vertices $u, v \in B$.

Note that for all remaining white vertices w, the set $B_w := N(w) \cap B$ of black neighbors is a clique. Since Reduction Rule 4.10 has been applied exhaustively, we have $|B_w| < r$. Moreover, for each clique $C \subseteq B$ of size $i \in [r-1]$, there are at most $r - i$ white vertices with $B_w = C$. Thus, the number of white vertices w such that B_w is a clique is

$$\sum_{i=1}^{r-1} i\,|B|^{r-i} = \frac{|B|(|B|^r - 1)}{(|B|-1)^2} - \frac{|B|}{|B|-1} r \in \mathcal{O}(|B|^{r-1}).$$

Overall, there are $\mathcal{O}(c\,|B|^2 + |B|^{r-1})$ white vertices. ◀

Recall that there are $k^c\rho \in \mathcal{O}(ck^{c+2})$ black vertices by Reduction Rule 4.9. Hence, the overall number of vertices is $\mathcal{O}(c^3k^{2c+4} + c^{r-1}k^{(c+2)(r-1)})$, resulting in the following theorem:

▶ **Theorem 4.13.** BW-Threshold Dominating Set *has a kernel with $k^{\mathcal{O}(cr)}$ vertices computable in $\mathcal{O}^*(2^c)$ time.*

To obtain a kernel for Threshold Dominating Set, it remains to show that any BW-Threshold Dominating Set instance can be transformed into an equivalent instance of Threshold Dominating Set.

▶ **Theorem 4.14.** Threshold Dominating Set *has a kernel with $k^{\mathcal{O}(cr)}$ vertices computable in $\mathcal{O}^*(2^c)$ time.*

Proof. To obtain an $k^{\mathcal{O}(cr)}$-vertex kernel for Threshold Dominating Set, we first construct an equivalent instance (G, k) of BW-Threshold Dominating Set using Theorem 4.13. Then, we transform (G, k) into an equivalent instance (G', k') of Threshold Dominating Set in $k^{\mathcal{O}(cr)}$-closed graphs as follows.

We start with a copy of G. We add a clique $Q := \{w_1, \ldots, w_{r+1}\}$ of $r+1$ vertices. Then, for each white vertex w we add edges $ww_1, \ldots, ww_r$. Then, we remove all vertex colors. We call the resulting graph G'. Let $C = \{w_1, \ldots w_r\}$ and let $k' = k + r$. We show that (G, k) is a Yes-instance if and only if (G', k') is a Yes-instance.

Let D be a bw-threshold dominating set of G. By construction, $D \cup C$ is a threshold dominating set of size at most k' of G'. Conversely, suppose that G' has a threshold dominating set D' of size at most k'. By Lemma 4.1, we can assume that $w_{r+1} \notin D'$. Since $\deg_{G'}(w_{r+1}) = r$, it holds that $N_{G'}(w_{r+1}) = C \subseteq D'$. Hence, all white vertices of G are dominated r times by C in G'. Thus, $D := D' \setminus C$ is a bw-threshold dominating set of size at most k for G. ◀

Since the kernelization does not change the parameter r, it also gives a kernelization for Dominating Set.

▶ **Corollary 4.15.** Dominating Set *has a kernel with $k^{\mathcal{O}(c)}$ vertices which is computable in $\mathcal{O}^*(2^c)$ time.*

To complement this result, we show that there is no kernel for Dominating Set significantly smaller than that of Corollary 4.15 under a widely believed assumption.

▶ **Theorem 4.16.** *For $c \geq 3$,* Dominating Set *has no kernel of size $\mathcal{O}(k^{c-1-\epsilon})$ unless* coNP $\subseteq$ NP/poly.

Proof. We will show the theorem by a reduction from λ-Hitting Set.

λ-Hitting Set
Input: A set family $\mathcal{F}$ over an universe U, where each $S \in \mathcal{F}$ has size λ, and $k \in \mathbb{N}$.
Question: Is there a subset $X \subseteq U$ of size at most k such that for each $S \in \mathcal{F}$ we have $X \cap S \neq \emptyset$?

For any $\lambda \geq 2$, λ-Hitting Set does not have a kernel of size $\mathcal{O}(k^{\lambda-\epsilon})$ unless coNP $\subseteq$ NP/poly [13, 14]. Let $(U, \mathcal{F}, k)$ be an instance of λ-Hitting Set. We will construct a $\lambda+1$-closed graph G as follows: The vertex set $V(G)$ is $U \cup \mathcal{F}$. We add edges such that U forms a clique in G. We also add an edge between $u \in U$ and $S \in \mathcal{F}$ if and only if $u \in S$. Finally, we set $k' = k$. Since $\deg_G(S) = \lambda$ for each $S \in \mathcal{F}$ the graph G is $\lambda+1$-closed.

By construction, each hitting set X of size at most k is also a dominating set of size at most k of G. For the converse direction, we may assume by Lemma 4.1 that there is a dominating set D of size at most k for G not containing any vertex from $\mathcal{F}$. Thus, D is also a hitting set of $(U, \mathcal{F}, k)$.

Observe that our reduction preserves the parameter (that is, $k = k'$). Thus, it follows from the result of Hermelin and Wu [22] that if Dominating Set admits a kernel of size $\mathcal{O}(k^{\lambda-1-\epsilon})$ for some $\epsilon > 0$, then λ-Hitting Set admits a kernel of size $\mathcal{O}(k^{\lambda-\epsilon})$, implying that coNP $\subseteq$ NP/poly [13, 14]. ◀

We also obtain an algorithm for Threshold Dominating Set which is faster than brute-force search on the kernel of Theorem 4.14 and an improved kernel on bipartite graphs.

▶ **Theorem 4.17.** Threshold Dominating Set *can be solved in $\mathcal{O}^*(3^{c/3} + (ck)^{\mathcal{O}(rk)})$ time and* Dominating Set *can be solved in $\mathcal{O}^*((ck)^{\mathcal{O}(k)})$ time.*

▶ **Theorem 4.18.** Dominating Set *in bipartite graphs has a kernel with $\mathcal{O}(c^3k^4)$ vertices.*

5 Induced Matching

In this section, we develop kernelizations for Induced Matching in c-closed graphs.

Induced Matching
Input: A graph G and $k \in \mathbb{N}$.
Question: Is there a set M of at least k edges such that endpoints of distinct edges in M are pairwise nonadjacent?

Induced Matching is W[1]-hard when parameterized by k, even in bipartite graphs [26]. In terms of kernelizations, Induced Matching admits a kernel with $\mathcal{O}(\Delta^2 k)$ vertices [26] and $\mathcal{O}(k^d)$ vertices [18, 24]. The latter kernelization result is essentially tight: Unless coNP $\subseteq$ NP/poly, Induced Matching has no kernel of size $\mathcal{O}(k^{d-3-\varepsilon})$ for any $\varepsilon > 0$ [11]. Despite the lower bound in degenerate graphs, we discover in this section that Induced Matching in c-closed graphs has a polynomial kernel when parameterized by $k + c$.

5.1 Ramsey-like Bounds for Induced Matchings

Dabrowski et al. [12] derived fixed-parameter tractability for Induced Matching in $(K_a, K_{b,b})$-free graphs. At the heart of their algorithm lies a Ramsey-type result for induced matchings: For $a, b \in \mathbb{N}$, there exists an integer $Q_{a,b}$ such that any bipartite graph with a matching of size at least $Q_{a,b}$ contains a biclique $K_{a,a}$ or an induced matching of size b. In this subsection, we present analogous results for c-closed graphs where the number $Q_{a,b}$ is polynomial in a and b. We begin with two preliminary lemmas.

▶ **Lemma 5.1.** *Any graph G with a matching M of size at least $2\Delta b$ has an induced matching of size b.*

Proof. We prove by induction on b. The lemma clearly holds for the base case $b = 0$. For $b > 0$, let uv be a matched edge in M and let $G' := G - N[\{u, v\}]$. Since $|N[\{u, v\}]| \leq 2\Delta_G$, there is a matching of size at least $2\Delta_G b - 2\Delta_G \geq 2\Delta_{G'}(b-1)$ in G'. Consequently, there is an induced matching M' of size $b-1$ in G' by induction hypothesis. Thus, G has an induced matching $M' \cup \{uv\}$ of size b. ◀

▶ **Lemma 5.2.** *Suppose that G is a c-closed bipartite graph. If there are at least $2b$ vertices of degree at least cb, then G contains an induced matching of size at least b.*

Proof. Let A, B be a bipartition of G. Without loss of generality, assume that A contains a set A' of exactly b vertices of degree at least cb. Since G is c-closed, $|N(v) \cap N(v')| < c$ for all $v, v' \in A'$. It follows that each $v \in A'$ has a neighbor $u \in N(v)$ such that $u \notin N(v')$ for all $v' \in A' \setminus \{v\}$. Thus, G contains an induced matching of size b. ◀

In the following lemma, we obtain a Ramsey-type result for induced matchings in c-closed bipartite graphs.

▶ **Lemma 5.3.** *Let $Q_c(b) := 2cb^2 + 2b \in \mathcal{O}(cb^2)$. Let G be a c-closed bipartite graph. If G has a matching M of size at least $Q_c(b)$, then G contains an induced matching of size at least b.*

Proof. If there are at least $2b$ vertices of degree at least cb in G, then Lemma 5.2 yields an induced matching of size b. Thus, we can assume that $|S| < 2b$ for the set S of vertices of degree at least cb. Observe that $G - S$ has a matching of size $2cb^2$ and that $\Delta_{G-S} \leq cb$. Thus, $G - S$ has an induced matching of size b by Lemma 5.1. ◀

We extend Lemma 5.3 to non-bipartite c-closed graphs in the subsequent two lemmas. Recall that each c-closed graph G with at least $R_c(a, b) \in \mathcal{O}(cb^2 + ab)$ vertices contains a clique of a vertices or an independent set of b vertices by Lemma 3.1. Our proofs for Lemmas 5.4 and 5.5 put Lemmas 3.1 and 5.3 together.

▶ **Lemma 5.4.** *Let $Q'_c(a, b) := R_c(a, Q_c(b)) \in \mathcal{O}(cab^2 + c^3b^4)$. Any c-closed graph G with an independent set I of size at least $Q'_c(a, b)$ and a matching M saturating I contains a clique of size a or an induced matching of size b.*

Proof. Suppose that G contains no clique of size a. We show that there is an induced matching of size b in G. Let $H := V(M) \setminus I$ be the set of vertices matched to I in M. Since $|H| \geq R_c(a, Q_c(b))$, it follows from Lemma 3.1 that there is an independent set $H' \subseteq H$ of size at least $Q_c(b)$ in G'. Let $I' \subseteq I$ be the set of vertices matched to H' in M. Then, there is an induced matching of size at least b in $G[H' \cup I']$ by Lemma 5.3. Thus, G contains an induced matching of size b. ◀

▶ **Lemma 5.5.** *Let $Q''_c(a,b) := R_c(a, Q'_c(b)) \in \mathcal{O}(c^3a^2b^4 + c^7b^8)$. Any c-closed graph G with a matching M of size at least $Q''_c(a,b)$ contains a clique of size a or an induced matching of size b.*

Proof. Suppose that G contains no clique of size a. We will show that there is an induced matching of size b in G. Let I and H be disjoint vertex sets such that I and H consist of distinct endpoints of each edge in M. Since $|I| \geq R_c(a, Q'_c(b))$, it follows from Lemma 3.1 that there is an independent set $I' \subseteq I$ of size $Q'_c(b)$. Let $H' \subseteq H$ be the set of vertices matched to I' and let $G' := G[H' \cup I']$. Since I is an independent set of size at least $Q'_c(b)$, it follows from Lemma 5.3 that there is an induced matching M' of size at least b in G'. Consequently, G contains an induced matching of size b. ◀

5.2 Polynomial Kernel in c-closed Graphs

In this subsection, we prove that Induced Matching in c-closed graphs admits a kernel with $\mathcal{O}(c^7k^8)$ vertices. Our kernelization is based on Lemmas 5.4 and 5.5. To utilize these lemmas, we start with a reduction rule that destroys large cliques.

▶ **Reduction Rule 5.6.** *Let $v \in V(G)$ and let M_v be a maximum matching in $G[N(v)]$. If $|M_v| \geq 2ck$, then remove v.*

▶ **Lemma 5.7.** *Reduction Rule 5.6 is correct.*

Proof. Let $v \in V(G)$, let M_v be a maximum matching in $G[N_G(v)]$ of size at least $2ck$, and let $G' := G - v$. Suppose that G has an induced matching M of size at least k. We show that G' contains an induced matching of size at least k as well. We are done if M does not use v, because M is also an induced matching in G'. So we can assume that M uses v. Let $v_1v_2, \ldots, v_{2k-1}v_{2k}$ be k edges of M such that $v_{2k} = v$. By the definition of induced matching, $v_i \notin N_G(v)$ holds for each $i \in [2k-2]$. Thus, the c-closure of G yields that $|N_G(v) \cap N_G(v_i)| < c$ for each $i \in [2k-2]$. Since M_v is of size at least $2ck$, there is an edge e in M_v neither whose endpoint is adjacent to any vertex v_i for $i \in [2k-2]$. Hence, the edges $v_1v_2, \ldots, v_{2k-3}v_{2k-2}, e$ form an induced matching of size k in G'. The other direction follows trivially. Note that the c-closure is maintained by Observation 2.1. ◀

Henceforth, we assume that Reduction Rule 5.6 has been applied for each vertex. In the next lemma, we verify that there is no large clique.

▶ **Lemma 5.8.** *There is no clique of size $4ck + 1$ in G.*

Proof. Suppose that G contains a clique C of size at least $4ck + 1$ and let $v \in C$. Note that $C \subseteq N[v]$. Let M_v be a maximum matching in $G[N(v)]$. Also, let $N_v^1 \subseteq N(v)$ be the set of vertices incident with M_v and let $N_v^0 := N(v) \setminus N_v^1$. Since M_v is a maximum matching, N_v^0 is an independent set in $G[N(v)]$. Thus, C includes at most one vertex of N_v^0, that is, $|C \cap N_v^0| \leq 1$. Moreover, it follows from Reduction Rule 5.6 that $|M_v| \leq 2ck - 1$ and hence $|N_v^1| \leq 4ck - 2$. Now, we have a contradiction because $|C| = |C \cap N_v^0| + |C \cap N_v^1| + 1 \leq 4ck$. ◀

Once we show that the graph has a sufficiently large matching, Lemma 5.5 tells us that we can find a sufficiently large induced matching as well. Note, however, that a graph may not have a sufficiently large matching, even if it contains sufficiently many vertices (consider a star $K_{1,n}$ with n leaves). Our way around this obstruction is the LP (Linear Programming) relaxation of Vertex Cover (henceforth, we will abbreviate it as VCLP). It is well-known in the theory of kernelization that VCLP almost trivially yields a linear-vertex kernel for

Vertex Cover [7] due to the Nemhauser-Trotter theorem [28]. Here, we will exploit VCLP to ensure that after we apply some reduction rules, either the size of G is upper-bounded or the minimum vertex cover size (or equivalently the maximum matching size) of G is sufficiently large.

Recall that Vertex Cover can be formulated as an integer linear program as follows, using a variable x_v for each $v \in V(G)$:

$$\min \sum_{v \in V(G)} x_v \qquad \text{subject to} \quad \begin{aligned} x_u + x_v &\geq 1 \quad \forall uv \in E(G), \\ x_v &\in \{0,1\} \quad \forall v \in V(G). \end{aligned}$$

In VCLP, the last integral constraint is relaxed to $0 \leq x_v \leq 1$ for each $v \in V(G)$. It is known that VCLP admits a half-integral optimal solution (that is, $x_v \in \{0, 1/2, 1\}$ for each $v \in V(G)$) and such a solution can be computed in $\mathcal{O}(m\sqrt{n})$ time via a reduction to Maximum Matching (see, for instance, [4] or [10, Section 2.5]). Suppose that we have a half-integral optimal solution $(x_v)_{v \in V(G)}$. Let $V_0 := \{v \in V(G) \mid x_v = 0\}$, $V_1 := \{v \in V(G) \mid x_v = 1\}$, and $V_{1/2} := \{v \in V(G) \mid x_v = 1/2\}$.

We will bound the sizes of V_0, V_1, and $V_{1/2}$ in the upcoming rules. We begin with $V_{1/2}$. We use the bound Q''_c as specified in Lemma 5.5.

▶ **Reduction Rule 5.9.** *If $|V_{1/2}| \geq 3Q''_c(4ck+1, k)$, then return Yes.*

To show the correctness, we will use the fact that $VC + MM \geq 2LP$ for any graph G [20, Lemma 2.1]. Here, VC, MM, and LP refer to the minimum vertex cover size, the maximum matching size, and the optimal VCLP cost of G.

▶ **Lemma 5.10.** *Reduction Rule 5.9 is correct.*

Proof. Observe that the optimal cost of VCLP for $G[V_{1/2}]$ is $|V_{1/2}|/2$. Let X be a minimum vertex cover and M be a maximum matching in $G[V_{1/2}]$. Then, it follows that $|X| + |M| \geq |V_{1/2}|$ [20, Lemma 2.1]. Since $V(M)$ is a vertex cover in $G[V_{1/2}]$, we also have $2|M| \geq |X|$. Thus, $|M| \geq |V_{1/2}|/3 \geq Q''_c(4ck, k)$. Recall that there is no clique of size $4ck+1$ by Lemma 5.8. Hence, Lemma 5.5 yields that G contains an induced matching of size at least k. ◀

We next upper-bound the size of V_1. See Lemma 5.4 for the definition of Q'_c.

▶ **Reduction Rule 5.11.** *If $|V_1| \geq Q'_c(4ck+1, k)$, then return Yes.*

To prove the correctness of Reduction Rule 5.11, let us introduce the notion of *crowns* [9]. For a graph G, a crown is an ordered pair (I, H) of vertex sets of G with the following properties:

1. $I \neq \emptyset$ is an independent set in G,
2. $H = N(I)$, and
3. there is a matching saturating H in $G[H, I]$.

Crowns are closely related to VCLP – in fact, (V_0, V_1) is a crown [1, 8].

▶ **Lemma 5.12.** *Reduction Rule 5.11 is correct.*

Proof. Since (V_0, V_1) is a crown in G, there is a matching M saturating V_1 in $G[V_0, V_1]$. By definition, $I := V_0 \cap V(M)$ is an independent set of size $|V_1| \geq Q'_c(4ck+1, k)$ in G. Now, it follows from Lemma 5.4 that $G[I \cup V_1]$ contains an induced matching of size k. ◀

To deal with V_0, we introduce some additional rules which may add or remove vertices. Let us start with a simple rule. Basically, if there are multiple leaf vertices with the same neighborhood, then only one of them is relevant.

▶ **Reduction Rule 5.13.** *If $v_1 \in V_1$ has more than one leaf neighbor, then remove all but one of them.*

The correctness of Reduction Rule 5.13 is obvious and thus we omit the proof.

▶ **Reduction Rule 5.14.** *Let $v_0 \in V_0$ and let $v_1 \in V_1$. If $N_G[v_0] \subseteq N_G[v_1]$ and there is no leaf vertex attached to v_1, then attach a leaf vertex ℓ to v_1.*

▶ **Lemma 5.15.** *Reduction Rule 5.14 is correct.*

Proof. Let G' be the graph obtained by adding a leaf vertex ℓ to v_1. The forward direction is trivial. For the other direction, note that any induced matching M' in G' is an induced matching in G if M' does not include $v_1\ell$. Hence, it suffices to show that if there is an induced matching M' in G' such that $|M'| \geq k$ and $v_1\ell \in M'$, then there is an induced matching of size k in G as well. By the definition of induced matching, $M' \setminus \{v_1\ell\}$ includes no edge incident with a neighbor of v_1. Since $N_G[v_0] \subseteq N_G[v_1]$, the same holds for v_0. Thus, $(M' \setminus \{v_1\ell\}) \cup \{v_0v_1\}$ is an induced matching of size at least k in G.

For $c > 1$, Reduction Rule 5.14 maintains the c-closure by Observation 2.3. Note that Induced Matching can be solved in linear time when G is 1-closed: Since G is a disjoint union of complete graphs, (G, k) is a Yes-instance if and only if G contains at least k cliques of size at least two. ◀

▶ **Reduction Rule 5.16.** *Let $v_0 \in V_0$ be a non-leaf vertex. If each vertex $v_1 \in N_G(v_0)$ has a leaf neighbor, then remove v_0.*

▶ **Lemma 5.17.** *Reduction Rule 5.16 is correct.*

Proof. Let $G' = G - v_0$. Suppose that G has an induced matching M of size at least k. If M does not use v_0 we are done. So assume that M includes v_0v_1 for $v_1 \in N_G(v_0)$. Since there is a leaf vertex ℓ attached to v_1, the set $(M \setminus \{v_0v_1\}) \cup \{v_1\ell\}$ is an induced matching of size at least k in G'. The other direction follows trivially. The c-closure is maintained by Observation 2.1. ◀

▶ **Theorem 5.18.** Induced Matching *has a kernel with $\mathcal{O}(c^7k^8)$ vertices.*

Proof. We apply Reduction Rules 5.6, 5.9, 5.11, 5.13, 5.14 and 5.16 exhaustively. We also remove all isolated vertices. It is easy to verify that all these rules can be exhaustively applied in polynomial time.

Note that $|V_{1/2}| \in \mathcal{O}(c^7k^8)$ and $|V_1| \in \mathcal{O}(c^3k^4)$ by Reduction Rules 5.9 and 5.11. We show that $|V_0| \in \mathcal{O}(c|V_1|^2) = \mathcal{O}(c^7k^8)$. Note that there are at most $|V_1|$ leaf vertices in V_0 by Reduction Rule 5.13. All other vertices in V_0 are adjacent to at least two nonadjacent vertices in V_1: If there exists a vertex $v_0 \in V_0$ such that $N_G(v_0)$ is a clique of size at least two, then Reduction Rule 5.14 adds a leaf vertex to each vertex in $N_G(v_0)$ and Reduction Rule 5.16 removes v_0. Since G is c-closed, there are $c\binom{|V_1|}{2}$ non-leaf vertices in V_0. It follows that $|V_0| < |V_1| + c\binom{|V_1|}{2} \in \mathcal{O}(c^7k^8)$. ◀

We also obtain smaller kernels in bipartite graphs. Our kernelization is based on the following lemma, proven by a meet-in-the-middle approach on vertex degrees. Interestingly, this lemma will also play a central role in the kernelization for Irredundant Set in Section 6.

▶ **Lemma 5.19.** *Any bipartite graph G with at least $6\Delta^{3/2}b + 2\Delta b$ non-isolated vertices has an induced matching of size b.*

▶ **Theorem 5.20.** Induced Matching *in bipartite graphs has a kernel with $\mathcal{O}(\Delta^{3/2}k)$ vertices and a kernel with $\mathcal{O}(c^{3/2}k^{5/2})$ vertices.*

6 Irredundant Set

A vertex set $S \subseteq V(G)$ is *irredundant* if there is a private neighbor for each vertex v in S. Here, a *private neighbor* of $v \in S$ is a vertex $v' \in N[v]$ (possibly $v' = v$) such that $v' \notin N(u)$ for each $u \in S \setminus \{v\}$.

Irredundant Set
Input: A graph G and $k \in \mathbb{N}$.
Question: Is there an irredundant set S of at least k vertices in G?

Irredundant Set is W[1]-hard in general [16] but it admits a kernel with at most $(d+1)k$ vertices in d-degenerate graphs. This is because any d-degenerate graph on at least $(d+1)k$ vertices contains an independent set and thus an irredundant set of at least k vertices. In this section, we show that Irredundant Set admits a kernel with $\mathcal{O}(c^{5/2}k^3)$ vertices. Our kernelization relies on the Ramsey bound (Lemma 3.1) and the bound on induced matchings (Lemma 5.19). We show that the following reduction rule suffices to obtain a polynomial kernel.

▶ **Reduction Rule 6.1.** *If $u, v \in V(G)$ are simplicial vertices such that $N_G[u] = N_G[v]$, then remove v.*

▶ **Lemma 6.2.** *Reduction Rule 6.1 is correct.*

Proof. Let $u, v \in V(G)$ be vertices such that $N_G[u] = N_G[v]$. Let G' be the graph obtained by removing v as specified in Reduction Rule 6.1. Suppose that (G, k) is a Yes-instance with a solution S. It must hold that $u \notin S$ or $v \notin S$ by the definition of irredundant sets. Without loss of generality, assume that $v \notin S$. If v is a private neighbor of $w \in S$ (possibly $w = u$), then u is also a private neighbor of w. Thus, (G', k) is also a Yes-instance. The other direction follows trivially. The c-closure is maintained by Observation 2.1. ◀

We prove that Reduction Rule 6.1 yields a kernelization of the claimed size.

▶ **Theorem 6.3.** Irredundant Set *in c-closed graphs has a kernel with $\mathcal{O}(c^{5/2}k^3)$ vertices.*

Proof. We assume that Reduction Rule 6.1 has been applied exhaustively.

To simplify notation, let $\alpha' := 6c^{3/2}k + 2ck + 1 \in \mathcal{O}(c^{3/2}k)$ and $\alpha := R_c(\alpha', k) \in \mathcal{O}(c^{3/2}k^2)$. We claim that any instance (G, k) with at least $R_c(c\alpha + 1, k) \in \mathcal{O}(c^{5/2}k^3)$ vertices is a Yes-instance. By Lemma 3.1, G has a clique of size $c\alpha + 1$ or an independent set of size k. Since any independent set is also an irredundant set, (G, k) is a Yes-instance when G contains an independent set of size k. Thus, we assume that there is no independent set of size k in G.

It remains to show that if G has a maximal clique C of size greater than $c\alpha$, then (G, k) is a Yes-instance. Let $C' = \{v \in C \mid N_G(v) \setminus C \neq \emptyset\}$ be the set of vertices in C that have at least one neighbor outside C. There exists at most one vertex v with $N_G[v] = C$ by Reduction Rule 6.1 and thus $|C'| \geq |C| - 1 \geq c\alpha$. Let $G' = G - (C \setminus C')$. That is, G' is a graph obtained by removing a vertex adjacent to all vertices in C, if such a vertex exists. For each $i \in [\alpha]$, we will choose vertices $x_i \in C'$ and $y_i \in N_{G'}(C')$ as follows: Let x_i be an arbitrary vertex in $C' \setminus \bigcup_{j \in [i-1]} N_{G'}(y_j)$ and let y_i be an arbitrary vertex in $N_{G'}(x_i)$. Note that $C' \setminus \bigcup_{j \in [i-1]} N_{G'}(y_j) \neq \emptyset$ for each $i \in [\alpha]$, because $|C'| \geq c\alpha$ and y_j has less than c neighbors in C' for all $j \in [i-1]$ by Observation 2.2.

Since G has no independent set of size k, Lemma 3.1 gives us a clique of size α' among $y_1, \ldots, y_\alpha$. Without loss of generality, let $Y = \{y_1, \ldots, y_{\alpha'}\}$ be a clique of size α' and let $X = \{x_1, \ldots, x_{\alpha'}\}$. For $X' = X \setminus \{x_1\}$ and $Y' = Y \setminus \{y_1\}$, we prove that the bipartite graph $G[X', Y']$ has an induced matching of size k, using Lemma 5.19. First we show that $\Delta_{G[X',Y']} < c$. All vertices in Y' have less than c neighbors in X' by Observation 2.2.

By the choice of x_i and y_i, we have $x_i \notin N_G(y_1)$ for all $i \in [2, \alpha']$. It follows from the c-closure of G that x_i has less than c neighbors in Y' for each $i \in [2, \alpha']$. Thus, we have $\Delta_{G[X',Y']} < c$. Note that we choose x_i and y_i such that there is an edge $x_i y_i \in E(G)$ for each $i \in [2, \alpha']$. So $G[X', Y']$ has no isolated vertices. Therefore, it follows from Lemma 5.19 that there is an induced matching $\{x_{i_1}y_{i_1}, \ldots, x_{i_k}y_{i_k}\}$ of size k in $G[X', Y']$. Now, the set $\{x_{i_1}, \ldots, x_{i_k}\}$ is an irredundant set in G, where y_{i_j} is a private neighbor of x_{i_j} for each $j \in [k]$. ◀

7 Conclusion

We have demonstrated that the c-closure of a graph can be exploited in the design of parameterized algorithms for well-studied graph problems. We believe that the c-closure could become a standard secondary parameter just as the maximum degree Δ or the degeneracy d of the input graph and that studying problems with respect to this parameter may often lead to useful tractability results. In essence, whenever one obtains a fixed-parameter algorithm that uses Δ as one of its parameters, one should ask whether Δ can be replaced by the c-closure of the input graph. As concrete applications of the c-closure parameterization, one could consider further graph problems that are hard with respect to the solution size. For example, is Perfect Code [6] fixed-parameter tractable with respect to $c + k$ where k is the size of the code and does it admit a polynomial kernelization for this parameter? Further problems to investigate could be r-Regular Induced Subgraph which is W[1]-hard when parameterized by the subgraph size [27] or cardinality constrained optimization problems in graphs such as computing a maximum cut where the number of vertices in one part is constrained to be k [5]. These problems are often fixed-parameter tractable for the combination of the cardinality constraint k and the maximum degree Δ [5, 25]. Which of these problems is also fixed-parameter tractable for the combination of k and c? For answering such questions, the Ramsey bound of Lemma 3.1 could prove useful.

References

1 Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theory of Computing Systems*, 41(3):411–430, 2007.

2 Vladimir E. Alekseev, Dmitry V. Korobitsyn, and Vadim V. Lozin. Boundary classes of graphs for the dominating set problem. *Discrete Mathematics*, 285(1-3):1–6, 2004.

3 Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009.

4 Reuven Bar-Yehuda and Shimon Even. A local-ratio theorem for approximating the weighted vertex cover problem. In *Proceedings of the 9th International Workshop Graph-Theoretic Concepts in Computer Science (WG '83)*, pages 17–28. Universitätsverlag Rudolf Trauner, Linz, 1983.

5 Leizhen Cai. Parameterized complexity of cardinality constrained optimization problems. *The Computer Journal*, 51(1):102–121, 2008.

6 Marco Cesati. Perfect code is W[1]-complete. *Information Processing Letters*, 81(3):163–168, 2002.

7 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001.

8 Miroslav Chlebík and Janka Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156(3):292–312, 2008.

9 Benny Chor, Mike Fellows, and David W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Proceedings of the 30th International Workshop Graph-Theoretic Concepts in Computer Science (WG '04)*, volume 3353 of *Lecture Notes in Computer Science*, pages 257–269. Springer, 2004.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

11 Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. *ACM Transactions on Algorithms*, 13(3):43:1–43:22, 2017.

12 Konrad Dabrowski, Marc Demange, and Vadim V. Lozin. New results on maximum induced matchings in bipartite graphs and beyond. *Theoretical Computer Science*, 478:33–40, 2013.

13 Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*, pages 68–81. SIAM, 2012.

14 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the ACM*, 61(4):23:1–23:27, 2014.

15 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

16 Rodney G. Downey, Michael R. Fellows, and Venkatesh Raman. The complexity of irredundant sets parameterized by size. *Discrete Applied Mathematics*, 100(3):155–167, 2000.

17 Paul Erdös. Some remarks on the theory of graphs. *Bulletin of the American Mathematical Society*, 53(4):292–294, 1947.

18 Rok Erman, Łukasz Kowalik, Matjaž Krnc, and Tomasz Waleń. Improved induced matchings in sparse graphs. *Discrete Applied Mathematics*, 158(18):1994–2003, 2010.

19 Jacob Fox, Tim Roughgarden, C. Seshadhri, Fan Wei, and Nicole Wein. Finding cliques in social networks: A new distribution-free model. *SIAM Journal on Computing*, 49(2):448–464, 2020.

20 Shivam Garg and Geevarghese Philip. Raising the bar for vertex cover: Fixed-parameter tractability above a higher guarantee. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*, pages 1152–1166. SIAM, 2016.

21 Petr A. Golovach and Yngve Villanger. Parameterized complexity for domination problems on degenerate graphs. In *Proceedings of the 34th International Workshop Graph-Theoretic Concepts in Computer Science (WG '08)*, volume 5344 of *Lecture Notes in Computer Science*, pages 195–205, 2008.

22 Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*, pages 104–113. SIAM, 2012.

23 Stasys Jukna. *Extremal Combinatorics - With Applications in Computer Science*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2011.

24 Iyad A. Kanj, Michael J. Pelsmajer, Marcus Schaefer, and Ge Xia. On the induced matching problem. *Journal of Computer and System Sciences*, 77(6):1058–1070, 2011.

25 Christian Komusiewicz and Manuel Sorge. An algorithmic framework for fixed-cardinality optimization in sparse graphs applied to dense subgraph problems. *Discrete Applied Mathematics*, 193:145–161, 2015.

26 Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem. *Discrete Applied Mathematics*, 157(4):715–727, 2009.

27 Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *Journal of Discrete Algorithms*, 7(2):181–190, 2009.

28 George L. Nemhauser and Leslie E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

29 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Transactions on Algorithms*, 9(1):11:1–11:23, 2012.

30 Venkatesh Raman and Saket Saurabh. Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008.

31 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in biclique-free graphs. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA '12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 802–812. Springer, 2012.

Many Visits TSP Revisited

Łukasz Kowalik
Institute of Informatics, University of Warsaw, Poland
https://www.mimuw.edu.pl/~kowalik/
kowalik@mimuw.edu.pl

Shaohua Li
Institute of Informatics, University of Warsaw, Poland
shaohua.li@mimuw.edu.pl

Wojciech Nadara
Institute of Informatics, University of Warsaw, Poland
w.nadara@mimuw.edu.pl

Marcin Smulewicz
Institute of Informatics, University of Warsaw, Poland
m.smulewicz@mimuw.edu.pl

Magnus Wahlström
Royal Holloway, University of London, UK
Magnus.Wahlstrom@rhul.ac.uk

Abstract

We study the Many Visits TSP problem, where given a number $k(v)$ for each of n cities and pairwise (possibly asymmetric) integer distances, one has to find an optimal tour that visits each city v exactly $k(v)$ times. The currently fastest algorithm is due to Berger, Kozma, Mnich and Vincze [SODA 2019, TALG 2020] and runs in time and space $\mathcal{O}^*(5^n)$. They also show a polynomial space algorithm running in time $\mathcal{O}(16^{n+o(n)})$. In this work, we show three main results:

- A randomized polynomial space algorithm in time $\mathcal{O}^*(2^n D)$, where D is the maximum distance between two cities. By using standard methods, this results in a $(1+\epsilon)$-approximation in time $\mathcal{O}^*(2^n \epsilon^{-1})$. Improving the constant 2 in these results would be a major breakthrough, as it would result in improving the $\mathcal{O}^*(2^n)$-time algorithm for Directed Hamiltonian Cycle, which is a 50 years old open problem.
- A tight analysis of Berger et al.'s exponential space algorithm, resulting in an $\mathcal{O}^*(4^n)$ running time bound.
- A new polynomial space algorithm, running in time $\mathcal{O}(7.88^n)$.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases many visits traveling salesman problem, exponential algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.66

Funding This workshop was supported by a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 714704 (PI: Marcin Pilipczuk).
Łukasz Kowalik: Supported by ERC Starting Grant TOTAL (Grant Agreement No 677651).
Shaohua Li: Supported by ERC Starting Grant CUTACOMBS (Grant Agreement No 714704).
Wojciech Nadara: Supported by ERC Starting Grant CUTACOMBS (Grant Agreement No 714704).
Marcin Smulewicz: Supported by ERC Starting Grant TOTAL (Grant Agreement No 677651).

Acknowledgements The research leading to the results presented in this paper was partially carried out during the Parameterized Algorithms Retreat of the University of Warsaw, PARUW 2020, held in Krynica-Zdrój in February 2020.

1 Introduction

In the Many Visits TSP (MVTSP) we are given a set V of n vertices, with pairwise distances (or costs) $d : V^2 \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$. We are also given a function $k : V \to \mathbb{Z}_+$. A valid tour of length ℓ is a sequence of vertices $(x_1, \ldots, x_\ell)$, where $\ell = \sum_{v \in V} k(v)$, such that each $v \in V$ appears in the sequence exactly $k(v)$ times. The cost of the tour is $\sum_{i=1}^{\ell-1} d(x_i, x_{i+1}) + d(x_\ell, x_1)$. Our goal is to find a valid tour with minimum cost.

Many Visits TSP is a natural generalization of the classical (asymmetric) Traveling Salesman Problem (TSP), which corresponds to the case when $k(v) = 1$ for every vertex v. Similarly as its special case, MVTSP arises with a variety of applications, including scheduling [24, 17, 6, 27, 12], computational geometry [20] and parameterized complexity [21].

1.1 Related work

The standard dynamic programming for TSP of Bellman [1], Held and Karp [16] running in time $\mathcal{O}^*(2^n)$ can be easily generalized to MVTSP resulting in an algorithm with the running time of $\mathcal{O}^*(\prod_{v \in V}(k(v)+1))$, as noted by Psaraftis [24]. A breakthrough came in the work of Cosmadakis and Papadimitriou [7] who presented an algorithm running in time $2^{O(n \log n)} + \mathcal{O}(n^3 \log \ell)$ and space $2^{O(n \log n)}$, thus essentially removing the dependence on the function k from the bound (the $\log \ell$ factor can be actually skipped if we support the original algorithm with a today's state-of-the-art minimum cost flow algorithm). This may be surprising since the *length* of the output sequence is ℓ. However, beginning from the work of Cosmadakis and Papadimitriou we consider MVTSP with compressed output, namely the output is a multiplicity function which encodes the number of times every edge is visited by the tour. By using a standard Eulerian tour algorithm we can compute an explicit tour from this output.

The crux of the approach of Cosmadakis and Papadimitriou [7] was an observation that every solution can be decomposed to a minimal connected spanning Eulerian subgraph (which enforces connectivity of the solution) and a subgraph satisfying appropriate degree constraints (which completes the tour so that the numbers of visits agree). Moreover, once we guess the degree sequence δ of the Eulerian subgraph, our task splits into two separate tasks: finding a cheapest minimal connected Eulerian subgraph consistent with δ (which is computationally hard) and finding a cheapest subgraph satisfying the degree constraints (which can by solved in polynomial time by a reduction to minimum cost flow).

Yet another breakthrough came only recently, namely Berger, Kozma, Mnich and Vincze [3, 2] improved the running time to $\mathcal{O}^*(5^n)$. Their main contribution is an idea that it is more convenient to use outbranchings (i.e. spanning trees oriented out of the root) to force connectivity of the solution. The result of Berger et al. is the first algorithm for MVTSP which is optimal assuming Exponential Time Hypothesis (ETH) [18], i.e., there is no algorithm in time $2^{o(n)}$, unless ETH fails. Moreover, by applying the divide and conquer approach of Gurevich and Shelah [15] they design a polynomial space algorithm, running in time $\mathcal{O}(16^{n+o(n)})$.

1.2 Our results

In this work, we take the next step in exploration of the Many Visits TSP problem: we aim at algorithms which are optimal at a more fine grained level, namely with running times of the form $\mathcal{O}(c^n)$, such that an improvement to $\mathcal{O}((c-\epsilon)^n)$ for any $\epsilon > 0$ meets a kind of natural barrier, for example contradicts the Strong Exponential Time Hypothesis (SETH) [19] or the Set Cover Conjecture (SCC) [8]. Our main result is the following theorem.

▶ **Theorem 1.1.** *There is a randomized algorithm that solves* Many Visits TSP *in time* $\mathcal{O}^*(2^n D)$ *and polynomial space, where* $D = \max\{d(u,v) : u,v \in V, d(u,v) \neq \infty\}$. *The algorithm returns a minimum weight solution with constant probability.*

The natural barrier in this case is connected with Directed Hamiltonicity, the problem of determining if a directed graph contains a Hamiltonian cycle. Indeed, this is a special case of Many Visits TSP with $D = 1$, so an improvement to $\mathcal{O}^*(1.99^n D)$ in Theorem 1.1 would result in an algorithm in time $\mathcal{O}^*(1.99^n)$ for Directed Hamiltonicity. While it is not known whether such an algorithm contradicts SETH or SCC, the question about its existence is a major open problem which in the last 58 years has seen some progress only for special graph classes, like bipartite graphs [4, 9].

At the technical level, Theorem 1.1 uses the so-called algebraic approach and relies on two key insights. The first one is to enforce connectivity not by guessing a spanning connected subgraph as in the previous works, but by applying the Cut and Count approach of Cygan et al [10]. The second insight is to satisfy the degree constraints using the Tutte matrix [26, 22].

By using standard rounding techniques, we are able to make the algorithm from Theorem 1.1 somewhat useful even if the maximum distance D is large. Namely, we prove the following.

▶ **Theorem 1.2.** *For any* $\epsilon > 0$ *there is a randomized* $(1+\epsilon)$*-approximation algorithm that solves* Many Visits TSP *in* $\mathcal{O}^*(2^n \epsilon^{-1})$ *time and polynomial space.*

In Theorems 1.1 and 1.2 the better exponential dependence in the running time was achieved at the cost of sacrificing an $\mathcal{O}(D)$ factor in the running time, or the optimality of the solution. What if we do not want to sacrifice anything? While we are not able to get a $\mathcal{O}^*(2^n)$ algorithm yet, we are able to report a progress compared to the algorithm of Berger et al. in time $\mathcal{O}^*(5^n)$. In fact we do not show a new algorithm but we provide a refined analysis of the previous one. The new analysis is tight (up to a polynomial factor).

▶ **Theorem 1.3.** *There is an algorithm that solves* Many Visits TSP *in time and space* $\mathcal{O}^*(4^n)$.

In short, Berger et al.'s polyspace $\mathcal{O}^*(16^{n+o(n)})$ time algorithm iterates through all $O(4^n)$ degree sequences of an outbranching, finds the cheapest outbranching for each sequence in time $O(4^{n+o(n)})$, and completes it to satisfy the degree constraints using a polynomial time flow computation. Note that it is hard to speed up the cheapest outbranching routine, because for the sequence of $n-1$ ones and one zero we get essentially the TSP, for which the best known polynomial space algorithm takes time $O(4^{n+o(n)})$ [15]. However, we are still able to get a significant speed up of their algorithm, roughly, by using a more powerful minimum cost flow network, which allows for computing the cheapest outbranchings in smaller subgraphs.

▶ **Theorem 1.4.** *There is an algorithm that solves* Many Visits TSP *in time* $\mathcal{O}^*(7.88^n)$ *and polynomial space.*

Organization of the paper. In Section 3 we show that, essentially, using a polynomial time preprocessing step we can reduce an instance of Many Visits TSP to an equivalent one but with demands k bounded by $O(n^2)$. This reduction is a crucial prerequisite for Section 4 where we prove Theorem 1.1. Next, in Section 5 we prove Theorem 1.3 and in Section 6 we prove Theorem 1.4. We note that in these two sections we do not need the reduction from Section 3, however, in practice, applying it should speed-up the flow computations used in both algorithms described there. Finally, in Section 7 we show Theorem 1.2 and we discuss further research in Section 8.

2 Preliminaries

We use Iverson bracket, i.e., if α is a logical proposition, then the expression $[\alpha]$ evaluates to 1 when α is true and 0 otherwise.

For two integer-valued functions f, g on the same domain D, we write $f \leq g$ when $f(x) \leq g(x)$ for every $x \in D$. Similarly, $f + g$ (resp. $f - g$) denote the pointwise sum (difference) of f and g. This generalizes to functions on different domains D_f, D_g by extending the functions to $D_f \cup D_g$ so that the values outside the original domain are 0.

For a cost function $d : V^2 \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$, and a multiplicity function $m : V^2 \to \mathbb{Z}_{\geq 0}$ we denote the cost of m as $d(m) = \sum_{u,v \in V^2} d(u, v)m(u, v)$.

Multisets. Recall that a *multiset* A can be identified by its *multiplicity* function $m_A : U \to \mathbb{Z}_{\geq 0}$, where U is a set. We write $e \in A$ when $e \in U$ and $m_A(e) > 0$. Consider two multisets A and B. We write $A \subseteq B$ when for every $e \in A$ we have $e \in B$ and $m_A(e) \leq m_B(e)$. Also, $A = B$ when $A \subseteq B$ and $B \subseteq A$. Assume w.l.o.g. that m_A and m_B have the same domain U. Operations on multisets are defined by the corresponding multiplicities as follows: for every $e \in U$, we have $m_{A \cup B}(e) = \max\{m_A(e), m_B(e)\}$, $m_{A \cap B}(e) = \min\{m_A(e), m_B(e)\}$, $m_{A \setminus B}(e) = \max\{m_A(e) - m_B(e), 0\}$, $m_{A \triangle B}(e) = m_{(A \setminus B) \cup (B \setminus A)} = |m_A(e) - m_B(e)|$. This notation extends to the situation when A or B is a set, by using the indicator function $m_A(e) = [e \in A]$.

Directed graphs. Directed graphs (also called digraphs) in this paper can have multiple edges and multiple loops, so sets $E(G)$ will in fact be multisets. We call a directed graph *simple* if it has no multiple edges or loops. We call it *weakly simple* if it has no multiple edges or multiple loops (but single loops are allowed). For a digraph G by $G^{\downarrow}$ we denote the *support* of G, i.e., the weakly simple digraph on the vertex set $V(G)$ such that $E(G^{\downarrow}) = \{(u, v) \mid G \text{ has an edge from } u \text{ to } v\}$.

Given a digraph $G = (V, E)$ we define its *multiplicity function* $m_G : V^2 \to \mathbb{Z}_{\geq 0}$ as the multiplicity function of its edge multiset, i.e., for any pair $u, v \in V$, we put $m_G(u, v) = m_E((u, v))$. Conversely, for a function $m : V^2 \to \mathbb{Z}_{\geq 0}$ we define the *thick graph* $G_m = (V, E)$ so that $m_G = m$. Abusing notation slightly, we will identify m and G_m, e.g., we can say that m is strongly connected, contains a subgraph, etc.

We call a directed graph *connected* if the underlying undirected graph is connected. Similarly, a *connected component* of a digraph G is a subgraph of G induced by a vertex set of a connected component of the underlying undirected graph.

For a graph G (directed or undirected) and a subset $X \subseteq V(G)$, by $G[X]$ we denote the subgraph induced by X.

Solutions. The following observation follows easily from known properties of Eulerian digraphs.

▶ **Observation 2.1.** *Many Visits TSP has a tour of cost c if and only if there is a multiplicity function $m_G : V^2 \to \mathbb{Z}_{\geq 0}$ of cost c such that (i) for every $v \in V$, $\sum_{w \in V} m(v, w) = \sum_{w \in V} m(w, v) = k(v)$ and (ii) m contains a spanning connected subgraph.*

Thanks to Observation 2.1, in the remainder of this paper we refer to multiplicity functions as solutions of MVTSP (and some related problems which we are going to define). By standard arguments, the multiplicity function can be transformed to a tour in time $\mathcal{O}(\ell)$. Moreover, Grigoriev and Van de Klundert [14] describe an algorithm which transforms it to a compressed representation of the tour in time $O(n^4 \log \ell)$.

Out-trees. An *out-tree* is the digraph obtained from a rooted tree by orienting all edges away from the root. If an out-tree T is a subgraph of a directed graph G and additionally T spans the whole vertex set $V(G)$ we call T an *outbranching*. The sequence $\{\mathsf{outdeg}_T(v)\}_{v\in V(T)}$ is called the *outdegree sequence* of T. Consider a set of vertices $X \subseteq V$, $|X| \geq 2$.

▶ **Lemma 2.2** (Berger et al. [3], Lemma 2.4). *A sequence of nonnegative integers $\{d_v\}_{v\in X}$ is an outdegree sequence of an out-tree spanning X and rooted at $r \in X$ if and only if (i) $d_r \geq 1$ and (ii) $\sum_{v\in X} d_v = |X| - 1$.*

A sequence $\{d_v\}_{v\in X}$ that satisfies (i) and (ii) will be called an *out-tree sequence rooted at r*, or *outbranching sequence rooted at r* when additionally $X = V$. A δ-out-tree means any subtree spanning X with outdegree sequence δ.

3 Reduction to small demands

Consider the following problem, for a family of simple digraphs $\mathcal{F}$.

Fixed Degree $\mathcal{F}$-Subgraph
Input: $d : V^2 \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$, $\mathsf{in}, \mathsf{out} : V \to \mathbb{Z}_{\geq 0}$
Question: Find a function $m : V^2 \to \mathbb{Z}_{\geq 0}$ such that
(i) G_m contains a member of $\mathcal{F}$ as a spanning subgraph,
(ii) for every $v \in V$ we have $\mathsf{in}(v) = \mathsf{indeg}_{G_m}(v)$ and $\mathsf{out}(v) = \mathsf{outdeg}_{G_m}(v)$, and
so as to minimize the value of $d(m) = \sum_{v,w\in V} d(v,w)m(v,w)$.

In this paper, we will consider two versions of the problem: when $\mathcal{F}$ is the family of all oriented trees, called Fixed Degree Connected Subgraph, and when $\mathcal{F}$ is the family of all out-trees with a fixed root r, called Fixed Degree Subgraph With Outbranching. The role of $\mathcal{F}$ is to force connectivity of the instance. Other choices for $\mathcal{F}$ can also be interesting, for example Cosmadakis and Papadimitriou [7] consider the family of minimal Eulerian digraphs.

The goal of this section is to show that, essentially, using a polynomial time preprocessing step we can reduce an instance of Fixed Degree $\mathcal{F}$-Subgraph to an equivalent one but with demands in, out bounded by $O(n^2)$.

When considering the instance of Fixed Degree $\mathcal{F}$-Subgraph we will use the notation $n = |V|$ and $\ell = \sum_{v\in V} \mathsf{in}(v)$. (Clearly, we can assume that also $\ell = \sum_{v\in V} \mathsf{out}(v)$, for otherwise there is no solution.)

Observe that if the image of d is $\{0, +\infty\}$ we get the natural unweighted version, where we are given a graph with edge set $d^{-1}(0)$ and the goal is to decide if one can choose multiplicities of the edges so that the resulting digraph contains a member of $\mathcal{F}$ and its in- and outdegrees match the demands of in and out.

The following observation follows by standard properties of Eulerian cycles in digraphs and the fact that every strongly connected graph contains an outbranching rooted at arbitrary vertex.

▶ **Observation 3.1.** *Many Visits TSP is a special case of both Fixed Degree Connected Subgraph and Fixed Degree Subgraph With Outbranching with $\mathsf{in}(v) = \mathsf{out}(v) = k(v)$ for every vertex $v \in V$.*

In the following lemma, we consider the relaxed problem Fixed Degree Subgraph, defined exactly as Fixed Degree $\mathcal{F}$-Subgraph, but dropping the constraint that solutions must contain a member of $\mathcal{F}$. In what follows, $s_n(\mathcal{F}) = \max_{G\in\mathcal{F},|V(G)|=n} |E(G)|$. (Note that in applications we consider in this work $\mathcal{F}$ is a family of oriented spanning trees, so $s_n(\mathcal{F}) = n - 1$.)

▶ **Lemma 3.2.** *Fix an input instance $d : V^2 \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$, $\mathsf{in}, \mathsf{out} : V^2 \to \mathbb{Z}_{\geq 0}$. For every optimal solution r of* Fixed Degree Subgraph *there is an optimal solution c' of* Fixed Degree $\mathcal{F}$-Subgraph *such that for every $u, v \in V$*

$$|r(u,v) - c'(u,v)| \leq s_{|V|}(\mathcal{F}).$$

Before we proceed to a formal proof of Lemma 3.2, let us sketch an intuition behind it. We pick an optimal solution c of Fixed Degree $\mathcal{F}$-Subgraph and let $B \in \mathcal{F}$ be a spanning subgraph in G_c. The symmetric difference between $E(G_r)$ and $E(G_c)$ can be decomposed into "alternating" cycles. It suffices to alternate $|E(B) \setminus E(G_r)| \leq s_n(\mathcal{F})$ of them to enforce containing B. If we alternated *all* the cycles, we would get the cost of exactly $d(c)$, but because of the optimality of r, alternating any of the cycles does not decrease the cost of the solution. Hence alternating a subset of them cannot make the cost bigger than $d(c)$.

Proof. Let c be an arbitrary optimal solution of Fixed Degree $\mathcal{F}$-Subgraph and let B be an arbitrary graph from $\mathcal{F}$ which is a spanning subgraph of G_c. Our plan is to build an optimal solution c' of Fixed Degree $\mathcal{F}$-Subgraph which contains B and does not differ too much from r.

Define multisets $A_c = E(G_c) \setminus E(G_r)$, $A_r = E(G_r) \setminus E(G_c)$ and $A = A_c \cup A_r = E(G_c) \triangle E(G_r)$. In what follows, by an *alternating cycle* we mean an even cardinality set of edges

$$\{(v_0, v_1), (v_2, v_1), (v_2, v_3), (v_4, v_3) \ldots, (v_{2\ell-2}, v_{2\ell-1}), (v_0, v_{2\ell-1})\},$$

where edges come alternately from A_c and A_r. Note that an alternating cycle is not really a directed cycle, it is just an orientation of a simple undirected cycle.

Note that for every vertex $v \in V$, among the edges in A that enter (resp. leave) v the number of edges from A_c is the same as the number of edges from A_r (counted with corresponding multiplicities), since both c and r satisfy the degree constraints for the same instance. It follows that A can be decomposed into a multiset $\mathcal{C}$ of alternating simple cycles, i.e.,

$$m_A = \sum_{C \in \mathcal{C}} m_C,$$

where $m_C : V^2 \to \mathbb{Z}_{\geq 0}$ and for each pair $u, v \in V$ we have $m_C(u,v) = [(u,v) \in C] \cdot m_{\mathcal{C}}(C)$. To clarify, we note that the sum above is over all cycles in $\mathcal{C}$, and not over all copies of cycles.

Denote $B^+ = E(B) \setminus E(G_r)$. Since $B^+ \subseteq A_c$, for each $e \in B^+$, there is at least one cycle in $\mathcal{C}$ that contains e. We choose an arbitrary such cycle and we denote it by C_e. (Note that it may happen that $C_e = C_{e'}$ for two different edges $e, e' \in B^+$.) Let $\mathcal{C}^+ = \{C_e \mid e \in B^+\}$. Then we define c', by putting for every $u, v \in V$

$$c'(u,v) = r(u,v) + (-1)^{[(u,v) \in A_r]} \sum_{C \in \mathcal{C}^+} [(u,v) \in C]. \tag{1}$$

In other words, c' is obtained from r by iterating over all cycles in $C \in \mathcal{C}^+$, and adding one copy of each edge of $C \cap A_c$ and removing one copy of each edge of $C \cap A_r$.

Let us show that $G_{c'}$ contains B. This is trivial for every $e \in B^+$. When $e \in E(B) \cap E(G_r)$, consider two cases. If $e \notin A_r$, then $c'(e) \geq r(e)$, so $e \in G_{c'}$. If $e \in A_r$, $m_A(e) = r(e) - c(e)$. Then $c'(e) = r(e) - \sum_{C \in \mathcal{C}^+} [(u,v) \in C] \geq r(e) - m_A(e) = c(e) \geq 1$, where the last inequality follows since $B \subseteq G_c$.

To see that c' satisfies the degree constraints, recall that r does so, and note that if in (1) we consider only the summands corresponding to a single cycle $C \in \mathcal{C}^+$, then for every vertex we either add one outgoing edge and remove one outgoing edge, or add one ingoing edge and remove one ingoing edge, or we do not change the set of edges incident to it.

For a cycle $C \in \mathcal{C}$ let $\delta(C) = d(A_c \cap C) - d(A_r \cap C)$. Observe that for every cycle $C \in \mathcal{C}$ we have $\delta(C) \geq 0$, for otherwise $E(G_r) \setminus (C \cap A_r) \cup (C \cap A_c)$ contradicts the optimality of r. It follows that

$$d(c') = d(r) + \sum_{C \in \mathcal{C}^+} \delta(C) \leq d(r) + \sum_{C \in \mathcal{C}} \delta(C) = d(c). \tag{2}$$

Hence, since c is optimal solution of Fixed Degree $\mathcal{F}$-Subgraph, we get that c' is optimal solution of Fixed Degree $\mathcal{F}$-Subgraph as well. Moreover, by (1), for every $u, v \in V$,

$$|c'(u,v) - r(u,v)| \leq |\mathcal{C}^+| \leq |B| \leq s_{|V|}(\mathcal{F}). \tag{3}$$

This ends the proof. ◀

As noted in [7, 2], Fixed Degree Subgraph can be solved by a reduction to minimum cost flow. By applying Orlin's algorithm [23] we get the following.

▶ **Observation 3.3** (Folklore, [7, 2]). *Fixed Degree Subgraph can be solved in time* $O(n^3 \log n)$.

▶ **Theorem 3.4** (Kernelization). *There is a polynomial time algorithm which, given an instance* $I = (d, \mathsf{in}, \mathsf{out})$ *of Fixed Degree* $\mathcal{F}$*-Subgraph, outputs an instance* $I' = (d, \mathsf{in}', \mathsf{out}')$ *of the same problem and a function* $f : V^2 \to \mathbb{Z}_{\geq 0}$ *such that*

(i) $\mathsf{in}'(v), \mathsf{out}'(v) = \mathcal{O}(n \cdot s_n(\mathcal{F}))$ *for every vertex* v,

(ii) *if* m^* *is an optimal solution for* I'*, then* $f + m^*$ *is an optimal solution for* I.

The algorithm does not need to know $\mathcal{F}$*, just the value of* $s_n(\mathcal{F})$.

Proof. Our algorithm begins by finding an optimal solution r of Fixed Degree Subgraph using Observation 3.3.

Define $f_0 : V^2 \to \mathbb{Z}_{\geq 0}$, where for every $v, w \in V$ we put $f_0(v,w) = \max\{r(v,w) - s_n(\mathcal{F}), 0\}$. By Lemma 3.2, there exists an optimal solution c' for instance I such that $c' \geq f_0$. Now define $f : V^2 \to \mathbb{Z}_{\geq 0}$, where for every $v, w \in V$ we put $f(v,w) = \max\{f_0(v,w) - 1, 0\}$. Finally, we put $\mathsf{in}'(v) = \mathsf{in}(v) - \sum_{w \in V} f(w,v)$ and $\mathsf{out}'(v) = \mathsf{out}(v) - \sum_{w \in V} f(v,w)$. The algorithm outputs $I' = (d, \mathsf{in}', \mathsf{out}')$ and f. In what follows, we show that the output has the desired properties.

For the property (i), consider any vertex $v \in V$ and observe that $\sum_{w \in V} f(v,w) \geq \sum_{w \in V}(f_0(v,w) - 1) \geq \sum_{w \in V}(r(v,w) - s_n(\mathcal{F}) - 1)$. Since r is a feasible solution of I, we have $\mathsf{out}(v) = \sum_{w \in V} r(v,w)$. It follows that $\mathsf{out}'(v) \leq n(1 + s_n(\mathcal{F})) = O(n \cdot s_n(\mathcal{F}))$ as required. The argument for $\mathsf{in}'(v)$ is symmetric.

Now we focus on (ii). Let m^* be an optimal solution for I'. It is easy to check that $f + m^*$ satisfies the degree constraints for the instance I. Also, since m^* contains a subgraph from $\mathcal{F}$, then $f + m^*$ contains the same subgraph. It follows that $f + m^*$ is a feasible solution of I. It suffices to show that $f + m^*$ is an optimal solution for I.

Denote $r = c' - f$. Consider any pair $v, w \in V$ such that $c'(v,w) \geq 1$. We claim that $f(v,w) \leq c'(v,w) - 1$. Indeed, if $f_0(v,w) = 0$ then $f(v,w) = 0 \leq c'(v,w) - 1$, and if $f_0(v,w) \geq 1$ then $f(v,w) = f_0(v,w) - 1 \leq c'(v,w) - 1$. It follows that $r(v,w) \geq 1$. In particular, since c' contains a subgraph from $\mathcal{F}$, then also r contains the same subgraph. It follows that r is a feasible solution for I' (the degree constraints are easy to check). Hence, $d(m^*) \leq d(r)$. It follows that $d(f + m^*) \leq d(r + f) = d(c')$, so $f + m^*$ is indeed an optimal solution for I. ◀

4 The small costs case in time $\mathcal{O}^*(2^n D)$

In this section we establish Theorem 1.1. We do it in a bottom-up fashion, starting with a simplified core problem, and next generalizing the solution in a few steps.

4.1 Unweighted decision version with small degree demands

Consider the following problem.

Decision Unweighted Fixed Degree Connected Subgraph
Input: a digraph $G = (V, E)$, $\mathsf{in}, \mathsf{out} : V \to \mathbb{Z}_{\geq 0}$
Question: Is there a function $m : V^2 \to \mathbb{Z}_{\geq 0}$ such that $G_m^{\downarrow}$ is a connected subgraph of G and for every $v \in V$ we have $\mathsf{in}(v) = \mathsf{indeg}_{G_m}(v)$ and $\mathsf{out}(v) = \mathsf{outdeg}_{G_m}(v)$?

Note that Decision Unweighted Fixed Degree Connected Subgraph generalizes the directed Hamiltonian cycle problem, which is known to be solvable in $\mathcal{O}^*(2^n)$ time and polynomial space. In this section we show that this running time can be obtained for the more general problem as well, though we need to allow some randomization.

▶ **Theorem 4.1.** *There is a randomized algorithm which solves an instance $I = (\mathsf{in}, \mathsf{out})$ of Decision Unweighted Fixed Degree Connected Subgraph in time $\mathcal{O}^*(2^n \operatorname{poly}(M))$ and polynomial space, where $M = \max_v \max\{\mathsf{in}(v), \mathsf{out}(v)\}$. The algorithm is Monte Carlo with one-sided error, i.e., the positive answer is always correct and the negative answer is correct with probability at least p, for any constant $p < 1$.*

Our strategy will be to reduce our problem to detecting a perfect matching in a bipartite graph with an additional connectivity constraint.

We define a bipartite graph $B_G = (O, I, E(B_G))$ as follows. Let $I = \{v_1^I, \ldots, v_{\mathsf{in}(v)}^I \mid v \in V(H)\}$, $O = \{v_1^O, \ldots, v_{\mathsf{out}(v)}^O \mid v \in V(H)\}$, and $E(B_G) = \{u_i^O v_j^I \mid (u, v) \in E(G)\}$.

▶ **Observation 4.2.** *$|I| = |O| = \mathcal{O}(nM)$ and $|E(B_G)| \leq E(G)M^2 = \mathcal{O}(n^2M^2)$.*

For an undirected graph H by $\mathcal{PM}(H)$ we denote the set of perfect matchings in H. We say that a matching M in B_G is *connected* when for every cut $(X, V \setminus X)$ with $\emptyset \neq X \subsetneq V$ the matching M contains an edge $u_i^O v_j^I$ such that $u \in X$ and $v \in V \setminus X$ or $v \in X$ and $u \in V \setminus X$.

For a matching M in B_G we define a *contraction* of M as function $m : V^2 \to \mathbb{Z}_{\geq 0}$ such that $m(u, v) = |\{u_i^O v_j^I \in M \mid i \in [\mathsf{out}(u)], j \in [\mathsf{in}(v)]\}|$. In other words G_m is obtained from M by (1) orienting every edge from O to I and (2) identifying all vertices in $\{v_1^I, \ldots, v_{\mathsf{in}(v)}^I\} \cup \{v_1^O, \ldots, v_{\mathsf{out}(v)}^O\}$ for every $v \in V$, and keeping the multiple edges and loops.

▶ **Lemma 4.3.** *$(G, \mathsf{in}, \mathsf{out})$ is a yes-instance of Decision Unweighted Fixed Degree Connected Subgraph iff graph B_G contains a connected perfect matching.*

Proof. Let M be a connected perfect matching in B_G and let m be its contraction. We claim that m is a solution of $(G, \mathsf{in}, \mathsf{out})$. By the definition of B_G, $G_m^{\downarrow}$ is a subgraph of G. Since M is connected, G_m is connected as well, and so is $G_m^{\downarrow}$. Moreover, since M is a perfect matching $\mathsf{in}(v) = \mathsf{indeg}_{G_m}(v)$ and $\mathsf{out}(v) = \mathsf{outdeg}_{G_m}(v)$ for every vertex v.

For the other direction, let m be a solution for $(G, \mathsf{in}, \mathsf{out})$. For every $v \in V$, there are exactly $\mathsf{out}(v)$ edges leaving v in G_m. Let us denote them $e_{v,1}^O, \ldots, e_{v,\mathsf{out}(v)}^O$. Similarly, let us denote all the edges entering v by $e_{v,1}^I, \ldots, e_{v,\mathsf{in}(v)}^I$. Then we define M as the set of edges of the form $u_i^O v_j^I$ such that G_m contains an edge $e = e_{u,i}^O = e_{v,j}^I$. The fact that M is a perfect matching is clear from the construction. Also, M is connected, for otherwise G_m is not connected. ◀

From now on, let $B = (O, I, E(B))$ be an arbitrary subgraph of B_G. Define the following multivariate polynomial over $\mathrm{GF}(2^t)$, for an integer t to be specified later.

$$R = \sum_{\substack{M \in \mathcal{PM}(B) \\ M \text{ is connected}}} \prod_{e \in M} x_e \tag{4}$$

▶ **Lemma 4.4.** *R is not the zero polynomial if and only if B contains a connected perfect matching.*

Proof. It is clear that if R is non-zero then B contains a connected perfect matching. For the reverse implication it suffices to notice that every summand in R has a different set of variables, so it does not cancel out with other summands over $\mathrm{GF}(2^t)$. ◀

Our strategy is to test whether R is non-zero by means of DeMillo–Lipton–Schwartz–Zippel Lemma, which we recall below.

▶ **Lemma 4.5** (DeMillo and Lipton [11], Schwartz [25], Zippel [28])**.** *Let $P(x_1, x_2, \ldots, x_m)$ be a nonzero polynomial of degree at most d over a field $\mathbb{F}$ and let S be a finite subset of $\mathbb{F}$. Then, the probability that P evaluates to zero on a random element $(a_1, a_2, \ldots, a_m) \in S^m$ is bounded by $d/|S|$.*

By Lemmas 4.4 and 4.5, the task reduces to *evaluating* R fast. To this end, we will define a different polynomial P which is easier to evaluate and turns out to be equal to R over $\mathrm{GF}(2^t)$.

Consider a subset $X \subseteq V$. Let $I_X = \{v_i^I \in I \mid v \in X, i = 1, \ldots, \mathsf{in}(v)\}$ and $O_X = \{v_i^O \in O \mid v \in X, i = 1, \ldots, \mathsf{out}(v)\}$. Abusing the notation slightly, we will denote $B[X] = B[I_X \cup O_X]$. Define the following polynomial.

$$P_X = \sum_{M \in \mathcal{PM}(B[X])} \prod_{e \in M} x_e \tag{5}$$

In what follows, v^* is an arbitrary but fixed vertex of V. Define yet another polynomial.

$$P = \sum_{\substack{X \subseteq V \\ v^* \in X}} P_X P_{V \setminus X}. \tag{6}$$

▶ **Lemma 4.6.** *$P = R$.*

Proof. For a matching M in B we say that a set $X \subseteq V$ is *consistent* with M when M does not contain an edge $u_i^O v_j^I$ such that $u \in X$ and $v \in V \setminus X$ or $v \in X$ and $u \in V \setminus X$. The family of all subsets of V that are consistent with M will be denoted by $\mathcal{C}(M)$. Then we can rewrite P as follows.

$$\begin{aligned}
P &= \sum_{\substack{X \subseteq V \\ v^* \in X}} \sum_{M_1 \in \mathcal{PM}(B[X])} \sum_{M_2 \in \mathcal{PM}(B[V \setminus X])} \prod_{e \in M_1 \cup M_2} x_e && \text{[definition]} \\
&= \sum_{M \in \mathcal{PM}(B)} \sum_{\substack{X \in \mathcal{C}(M) \\ v^* \in X}} \prod_{e \in M} x_e && \text{[group by } M = M_1 \uplus M_2\text{]} \\
&= \sum_{M \in \mathcal{PM}(B)} |\{X \in \mathcal{C}(M) \mid v^* \in X\}| \prod_{e \in M} x_e && \text{[trivial]}
\end{aligned}$$

Let us consider a perfect matching $M \in \mathcal{PM}(B)$ and the corresponding contraction m. Observe that the number of sets that are consistent with M and contain a vertex v^* is equal to $2^{\mathrm{cc}(M)-1}$, where $\mathrm{cc}(M)$ is the number of connected components of G_m. Indeed, when X is consistent with M, then for every connected component Q of G_m, either $V(Q) \subseteq X$ or $V(Q) \subseteq V \setminus X$. For the component that contains v^* the choice is fixed, while every choice for the remaining components defines a set consistent with M. It follows that when M is not connected $\mathrm{cc}(M) \geq 2$, and the value of $2^{\mathrm{cc}(M)-1}$ is equal to 0 in $\mathrm{GF}(2^t)$, so the corresponding summand vanishes. On the other hand, if M is connected, the corresponding summand equals just $\prod_{e \in M} x_e$ and it does not cancel out with another summand because the monomial has a unique set of variables. It follows that $P = R$. ◀

▶ **Lemma 4.7** (Tutte, Lovász [26, 22])**.** *For an arbitrary set $X \subseteq V$, the polynomial P_X can be evaluated using $\mathrm{poly}(n+M)$ field operations.*

Proof. Compute the determinant of the corresponding Tutte matrix of dimension $|O| \times |I|$. ◀

Let us now fix our field, namely $t = \lceil 1 + \log n + \log M \rceil$. Since arithmetic operations in $\mathrm{GF}(2^t)$ can be performed in time $\mathcal{O}(t \log^2 t) = \mathcal{O}(\log(n+M) \log^2 \log(n+m))$, by the definition of P and Lemma 4.7 we get the following corollary.

▶ **Corollary 4.8.** *P can be evaluated in time $2^n \,\mathrm{poly}(n+M)$.*

▶ **Lemma 4.9.** *There is a randomized algorithm which decides if B contains a connected perfect matching in time $\mathcal{O}^*(2^n \,\mathrm{poly}(M))$ and polynomial space, where $M = \max_v \max\{\mathsf{in}(v), \mathsf{out}(v)\}$. The algorithm is Monte Carlo with one-sided error, i.e., the positive answer is always correct and the negative answer is correct with probability at least p, for any constant $p < 1$.*

Proof. The algorithm evaluates polynomial P using Corollary 4.8 substituting a random element of $\mathrm{GF}(2^t)$ for each variable, and reports "yes" when the evaluation is nonzero and "no" otherwise. If it reported "yes", then P was a non-zero polynomial and by Lemma 4.4 the answer is correct. Assume it reported 'no' for a yes-instance. By Lemma 4.4 P is non-zero. Since $\deg P = |I| \leq nM$, by Lemma 4.5 the probability that P evaluated to 0 is bounded by $\deg P / 2^t \leq 1/2$ and we can make this probability arbitrarily small by repeating the whole algorithm a number of times, and reporting "yes" if at least one evaluation was nonzero. The claim follows. ◀

Theorem 4.1 follows immediately from Lemma 4.3 and Lemma 4.9 applied to B_G.

4.2 Finding the solution

▶ **Lemma 4.10.** *There is a randomized algorithm which, given a yes-instance of* Decision Unweighted Fixed Degree Connected Subgraph*, always returns the corresponding solution m in expected time $\mathcal{O}^*(2^n \,\mathrm{poly}(M))$. The time can be made deterministic at the cost of introducing arbitrarily small probability of failure.*

In order to prove Lemma 4.10 we cast the problem in the setting of *inclusion oracles* from the work of Björklund et al. [5]. Consider a universe U and an (unknown) family of *witnesses* $\mathcal{F} \subseteq 2^U$. An *inclusion oracle* is a procedure which, given a query set $Y \subseteq U$, answers (either YES or NO) whether there exists at least one witness $W \in \mathcal{F}$ such that $W \subseteq Y$. Björklund et al. prove the following.

▶ **Theorem 4.11** ([5]). *There exists an algorithm that extracts a witness of size k in $\mathcal{F}$ using in expectation at most $O(k \log |U|)$ queries to a randomized inclusion oracle that has no false positives but may output a false negative with probability at most $p \leq \frac{1}{4}$.*

Proof of Lemma 4.10. Let $U = E(B_G)$ and let $\mathcal{F}$ be the family of all connected perfect matchings in B_G. Note that $|U| = O(n^2M^2)$ and witnesses in $\mathcal{F}$ have all size $|I| = O(nM)$. Then, Lemma 4.9 provides a randomized inclusion oracle and we can apply Theorem 4.11. (If one insists on deterministic, and not expected, running time, it suffices to chose a sufficiently large constant r and stop the algorithm if it exceeds the expected running time at least r times – by Markov's inequality, this happens with probability at most $1/r$.) ◀

4.3 Proof of Theorem 1.1

In the lemma below we will adapt the construction from Section 4.1 to the weighted case in a standard way, by introducing a new variable tracking the weight.

▶ **Lemma 4.12.** *There is a randomized algorithm which solves an instance $I = (d, \mathsf{in}, \mathsf{out}, w)$ of Fixed Degree Connected Subgraph in time $\mathcal{O}^*(2^n D \operatorname{poly}(M))$ and polynomial space, where $M = \max_v \max\{\mathsf{in}(v), \mathsf{out}(v)\}$ and D is the maximum integer value of d. The algorithm returns a minimum weight solution with probability at least p, for any constant $p < 1$.*

Proof. Define $G = (V, E)$ where $E = \{(u, v) \in V^2 \mid d(u, v) \in \mathbb{Z}_{\geq 0}\}$. Let R' be the polynomial obtained from R by replacing every variable x_e for $e = u_i^O v_j^I \in E(B_G)$ by the product $x_e \cdot y^{d(u,v)}$, where y is a new variable. Proceed similarly with P, obtaining P'. By Lemma 4.4, $P' = R'$. Decompose R' as $R' = \sum_{i=0}^{|I|\cdot D} R'_i y^i$, where R'_i, for every $i = 0, \ldots, |I| \cdot D$, is a polynomial in variables $\{x_e\}_{e \in E(B_G)}$. The monomials in R'_i enumerate all matchings M such that the contraction m of M has weight $d(m) = i$. By the construction in the proof of Lemma 4.3 R'_i is non-zero if and only if instance I has a solution of weight i. Using Lagrange interpolation, we can recover the value of each R'_i for random values of the variables $\{x_e\}_{e \in E(B_G)}$ (the values are the same for all the polynomials). The interpolation algorithm requires $|I| \cdot D = \mathcal{O}(nMD)$ evaluations of R'. Since $R' = P'$, by Lemma 4.8 each of them takes $2^n \operatorname{poly}(n + M))$ time. Our algorithm reports the minimum w such that R'_w evaluated to a non-zero element of $\mathrm{GF}(2^t)$, or $+\infty$ if no such w exists. The solution of weight w is then found using Lemma 4.10. The event that the optimum value w^* is not reported means that R'_{w^*} is a non-zero polynomial that evaluated to 0 at the randomly chosen values. By Lemma 4.5 this happens with probability at most $\deg P / 2^t \leq 1/2$, and one can make this probability arbitrarily small by standard methods. ◀

Theorem 1.1 follows now immediately by applying Theorem 3.4 which reduces the general problem to the $M = \mathcal{O}(n^2)$ case and solving the resulting instance by Lemma 4.12. Theorem 1.1 says in particular that if finite weights are bounded by a polynomial in n then we can solve Many Visits TSP in time $\mathcal{O}^*(2^n)$ and polynomial space by a randomized algorithm with no false positives and with false negatives with arbitrarily small constant probability.

5 The general case

In this section we prove Theorem 1.3, i.e., we show an algorithm solving Many Visits TSP in time $\mathcal{O}^*(4^n)$. In fact, we do not introduce a new algorithm, but we consider an algorithm by Berger et al. (Algorithm 5 in [3]) and we provide a refined analysis, resulting in an improved running time bound $\mathcal{O}^*(4^n)$, which is tight up to a polynomial factor.

Let us recall the algorithm of Berger et al., in a slightly changed notation. In fact, they solve a slightly more general problem, namely Fixed Degree Subgraph With Outbranching. Let $I = (d, \mathsf{in}, \mathsf{out}, r)$ be an instance of this problem, i.e., we want to find a solution $m : V^2 \to \mathbb{Z}_{\geq 0}$ that satisfies the degree constraints specified by in and out and contains an outbranching rooted at r. In what follows we assume $V = \{1, \ldots, n\}$ and $r = 1$.

Consider an outbranching sequence $\{\delta_v\}_{v \in V}$ rooted at $r = 1$. In what follows, all outbranching sequences will be rooted at 1, so we skip specifying the root. Let T_δ be a minimum cost outbranching among all outbranchings with outdegree sequence δ and let r_δ be an optimum solution of Fixed Degree Subgraph for instance $(d, \mathsf{in}', \mathsf{out}')$ where $\mathsf{out}' = \mathsf{out} - \mathsf{outdeg}_T$ and $\mathsf{in}' = \mathsf{in} - \mathsf{indeg}_T$. Berger et al. note that then $m_\delta = m_{T_\delta} + r_\delta$ is a feasible solution for instance I of Fixed Degree Subgraph With Outbranching, and moreover it has minimum cost among all solutions that contain an outbranching with outdegree sequence δ. Since r_δ can be found in polynomial time by Observation 3.3, in order to solve instance I it suffices to find outbranchings T_δ for all outbranching sequences δ and return the solution m_δ of minimum cost. Hence, Theorem 1.3 boils down to proving the following lemma.

▶ **Lemma 5.1.** *There is an algorithm which, for every outbranching sequence δ, finds a minimum cost outbranching among all outbranchings with outdegree sequence δ and runs in time $\mathcal{O}^*(4^n)$.*

We prove Lemma 5.1 by using dynamic programming (DP). However, it will be convenient to present the DP as a recursive function BestOutbranching with two parameters, $S \subseteq V$ and $\{\delta_v\}_{v \in S}$ (see Algorithm 1). It is assumed that $1 \in S$. We will show that BestOutbranching(S, δ) returns a minimum cost out-tree among all out-trees with outdegree sequence δ that are rooted at 1 and span S. Our algorithm runs BestOutbranching for $S = V$ and all outbranching sequences $\delta : V \to \mathbb{Z}_{\geq 0}$. Whenever BestOutbranching returns a solution for an input (S, δ), it is memoized (say, in an efficient dictionary), so that when BestOutbranching is called with parameters (S, δ) again, the output can be retrieved in polynomial time.

Algorithm 1 A pseudocode of the algorithm from Lemma 5.1.

```
function BestOutbranching(S, δ)
    v_first ← min{v ∈ S | δ_v = 0}
    if |S| = 2 then return {(1, v_first)}.
    else
        minCost ← ∞
        for w ∈ S do
            if (δ_w ≥ 1 ∧ w ≠ 1) ∨ (δ_w ≥ 2 ∧ w = 1) then
                S' ← S \ {v_first}
                δ' ← δ|_{S'}
                δ'_w ← δ'_w − 1
                R_w ← BestOutbranching(S', δ') ∪ {(w, v_first)}
                if d(R_w) < minCost then
                    minCost ← d(R_w)
                    best ← R_w
        return best
```

Let us define $\mathsf{lastRmvd}(S) := \max(\{0, 1, 2, \ldots, n\} \setminus S)$ and $\mathsf{bad}(S, \delta) := \{v \in S \mid v < \mathsf{lastRmvd}(S) \wedge \delta_v = 0\}$. Let us call (S, δ) a *reachable state* if it meets the following conditions:

(i) $\delta_1 \geq 1$
(ii) $\sum_{v \in S} \delta_v = |S| - 1$
(iii) $|\mathsf{bad}(S, \delta)| \leq 1$

▶ **Lemma 5.2.** *If function* BestOutbranching *is given a reachable state as input then all recursively called* BestOutbranching *will also be given only reachable states.*

Proof. Let us fix a reachable state (S, δ) for $|S| > 2$ and consider the associated value v_{first} from the algorithm. Denote $S' = S \setminus \{v_{\mathsf{first}}\}$. Clearly, it suffices to show that all pairs (S', δ') created in the **for** loop are reachable states. First, let us argue that $\mathsf{bad}(S', \delta) = \emptyset$. There are two cases:

- Assume $|\mathsf{bad}(S, \delta)| = 0$. In this case $v_{\mathsf{first}} > \mathsf{lastRmvd}(S)$ so $\mathsf{lastRmvd}(S') = v_{\mathsf{first}}$. Then, $\mathsf{bad}(S', \delta) = \{v \in S' \mid v < \mathsf{lastRmvd}(S') \wedge \delta_v = 0\} = \{v \in S \mid v < v_{\mathsf{first}} \wedge \delta_v = 0\} = \emptyset$.
- Assume $|\mathsf{bad}(S, \delta)| = 1$. Then, (1) $\mathsf{lastRmvd}(S') = \mathsf{lastRmvd}(S)$ because $\mathsf{lastRmvd}(S) > v_{\mathsf{first}}$ and (2) $\mathsf{bad}(S, \delta) = \{v_{\mathsf{first}}\}$. It follows that $\mathsf{bad}(S', \delta) \overset{(1)}{=} \{v \in S' \mid v < \mathsf{lastRmvd}(S) \wedge \delta_v = 0\} = \mathsf{bad}(S, \delta) \setminus \{v_{\mathsf{first}}\} \overset{(2)}{=} \emptyset$.

Let us consider the recursive call of BestOutbranching for a particular w. The sequence $\delta'|_{S'}$ differs from δ only at w, so $\mathsf{bad}(S', \delta') \subseteq \{w\} \cup \mathsf{bad}(S', \delta) = \{w\}$. This means that condition (iii) from the definition of a reachable state holds for (S', δ'). Since (S, δ) is reachable, $\delta_1 \geq 1$. Then either $w \neq 1$ and $\delta'_1 = \delta_1 \geq 1$ or $w = 1$ and $\delta'_1 = \delta_1 - 1 \geq 1$, where the last inequality holds thanks to the condition in the **if** statement in Algorithm 1. In both cases, (i) holds for (S', δ'). Finally, (ii) is immediate by the definition of δ'. It follows that (S', δ') is a reachable state, as required. ◀

▶ **Lemma 5.3.** *If the function* BestOutbranching *is given a reachable state (S, δ), it returns a cheapest out-tree T rooted at vertex 1, spanning S and with outdegree sequence δ.*

Proof. We will use induction on $|S|$.

In the base case $|S| = 2$, there is only one outbranching spanning S rooted at 1, namely $\{(1, v_{\mathsf{first}})\}$ and it is indeed returned by the algorithm.

In the inductive step assume $|S| > 2$. By conditions (i) and (ii) in the definition of a reachable state and Lemma 2.2, there is at least one out-tree rooted at 1, spanning S, and with outdegree sequence δ. Let T be a cheapest out-tree among all such out-trees. Vertex v_{first} is a leaf of T, since $\delta_{v_{\mathsf{first}}} = 0$. At some point w in the **for** loop in Algorithm 1 is equal to the parent w^* of v_{first} in T. Then, $T \setminus \{(w^*, v_{\mathsf{first}})\}$ is an out-tree rooted at 1, spanning S', and with outdegree sequence δ'. Since (S', δ') is a reachable state by Lemma 5.2, by the inductive hypothesis we know that a cheapest such out-tree T' will be returned by BestOutbranching(S', δ'). In particular, it means that $d(T') \leq d(T \setminus \{(w^*, v_{\mathsf{first}})\})$. Denote $R_{w^*} = T' \cup \{(w^*, v_{\mathsf{first}})\}$. Then, $d(R_{w^*}) = d(T') + d(w^*, v_{\mathsf{first}}) \leq d(T \setminus \{(w^*, v_{\mathsf{first}})\}) + d(w^*, v_{\mathsf{first}}) = d(T)$. It follows that BestOutbranching returns a set of edges best of cost at most $d(T)$. However $\mathsf{best} = R_w$ for a vertex w and by applying the induction hypothesis it is easy to see that R_w is an out-tree rooted at 1, spanning S with outdegree sequence δ. The claim follows. ◀

▶ **Lemma 5.4.** *There are $\mathcal{O}^*(4^n)$ reachable states.*

Proof. Any sequence of n nonnegative integers that sums up to at most $n-1$ will be called an *extended sequence*. It is well known that there are exactly $\binom{2n-1}{n} < 2^{2n-1} = \mathcal{O}(4^n)$ such sequences. To see this consider sequences of $n-1$ balls and n barriers and bijectively map them to the sequences of n numbers by counting balls between barriers and discarding the balls after the last barrier.

Let us fix an extended sequence $\bar{\delta} = \{\bar{\delta}_v\}_{v \in V}$, and denote $\bar{s} := n - (1 + \sum_{i=1}^{n} \bar{\delta}_i)$. We claim that there are only $\mathcal{O}(n)$ reachable states (S, δ) such that $\bar{\delta}|_S = \delta$ and $\bar{\delta}|_{V \setminus S} = 0$. Consider any such pair (S, δ). Let $(v_1, v_2, \ldots, v_k)$ be the vertices of $\{v \in V \mid \bar{\delta}_v = 0\}$ sorted

in increasing order. By the definition of a reachable state we know that $|S| = 1 + \sum_{i=1}^{n} \bar{\delta}_i$, so $\bar{s} = |\{1, 2, \ldots, n\} \setminus S|$. By (ii), for at least one vertex $v \in S$ we have $\bar{\delta}_v = \delta_v = 0$, so $k \geq \bar{s} + 1$. Let us assume that $k \geq \bar{s} + 2$ and $\mathsf{lastRmvd}(S) \geq v_{\bar{s}+2}$. Then, $\{v_1, v_2, \ldots, v_{\bar{s}+1}\} \cap S \subseteq \mathsf{bad}(S, \delta)$. Since $v_{\bar{s}+2} \leq \mathsf{lastRmvd}(S) \notin S$, at most $\bar{s} - 1$ elements from $\{v_1, v_2, \ldots, v_{\bar{s}+1}\}$ are outside S, so $|\mathsf{bad}(S, \delta)| \geq (\bar{s} + 1) - (\bar{s} - 1) = 2$. This is a contradiction with (S, δ) being a reachable state, which proves that $k \leq \bar{s} + 1$ or $\mathsf{lastRmvd}(S) < v_{\bar{s}+2}$. In any case, $\{1, 2, \ldots, n\} \setminus S \subseteq \{v_1, \ldots, v_{\bar{s}+1}\}$. There are $\bar{s} + 1 = \mathcal{O}(n)$ ways to choose $\bar{s}$ elements to the set $\{1, 2, \ldots, n\} \setminus S$ from $\{v_1, \ldots, v_{\bar{s}+1}\}$, so equivalently there are $\mathcal{O}(n)$ sets S such that (S, δ) is a reachable state, $\bar{\delta}|_S = \delta$ and $\bar{\delta}|_{V \setminus S} = 0$.

Every reachable state (S, δ) has the corresponding extended sequence $\{\bar{\delta}\}_{v \in V}$ defined by $\bar{\delta}|_S = \delta$ and $\bar{\delta}|_{V \setminus S} = 0$. Since there are $\mathcal{O}(4^n)$ extended sequences, and each of them has $O(n)$ corresponding reachable states there are $\mathcal{O}(4^n) \cdot \mathcal{O}(n) = \mathcal{O}^*(4^n)$ reachable states in total. ◀

We are ready to prove Lemma 5.1. Recall that our algorithm runs BestOutbranching(V, δ) for all outbranching sequences δ and uses memoization to avoid repeated computation. We claim that for any outbranching sequence δ, the pair (V, δ) is a reachable state. Indeed, conditions (i) and (ii) hold since δ is an outbranching sequence. By definition, $\mathsf{lastRmvd}(V) = 0$, so $\mathsf{bad}(V, \delta) = \emptyset$ which implies (iii). Hence by Lemma 5.3 the algorithm is correct. By Lemma 5.2 the running time can be bounded by the number of reachable states times a polynomial, which is $\mathcal{O}^*(4^n)$ by Lemma 5.4. This ends the proof of Lemma 5.1 and hence also Theorem 1.3, as discussed in the beginning of this section.

6 Polynomial space

In this section we show Theorem 1.4, that is, we solve Many Visits TSP in $\mathcal{O}^*(7.88^n)$ time and polynomial space. Berger et al. [2] solved this problem in $\mathcal{O}(16^{n+o(n)})$ time and polynomial space, with the key ingredient being the following.

▶ **Lemma 6.1** (Berger et al. [2]). *There is a polynomial space algorithm running in time $\mathcal{O}(4^{n+o(n)})$ which, given an outdegree sequence $\{\delta_v\}_{v \in V}$, a cost function $d : V^2 \to \mathbb{Z}_{\geq 0}$, and a root $r \in V$ computes the cheapest outbranching rooted at r with the required outdegrees.*

More precisely, the $\mathcal{O}(16^{n+o(n)})$-time algorithm consists of the following steps:

- **(i)** Enumerate all $\mathcal{O}(4^n)$ outbranching sequences
- **(ii)** For each outbranching sequence compute the cheapest outbranching with required degrees using Lemma 6.1 in time $\mathcal{O}(4^{n+o(n)})$
- **(iii)** For each of these outbranchings complete it to a solution of the original Many Visits TSP instance with an optimal solution of Fixed Degree Subgraph on the residual degree sequences (in polynomial time, by Observation 3.3).

The intuition behind our approach is as follows. We iterate over all subsets of vertices R. Here, R represents our guess of the set of inner vertices of an outbranching in an optimal solution. Then we perform (i) and (ii) in the smaller subgraph induced by R. Finally, we replace (iii) by a more powerful flow-based algorithm which connects the vertices in $V \setminus R$ to R, and at the same time computes a feasible solution of Fixed Degree Subgraph on the residual degree sequences, so that the total cost is minimized. Let $r = |R|$. Clearly, when r is a small fraction of n, we get significant savings in the running time. The closer r/n is to 1 the smaller are the savings, but also the smaller is the number $\binom{n}{r}$ of sets R to examine.

In fact, the real algorithm is slightly more complicated. Namely, we fix an integer parameter K, and then R corresponds to the set of vertices left from an outbranching in an optimal solution after K iterations of removing all leaves. The running time of our algorithm depends on K, because the algorithm actually guesses the layers of leaves in each iteration. The space complexity is polynomial and does not depend on K. In the end of this section, we show that our running time bound is minimized when $K = 4$.

6.1 Our algorithm

Similarly as in Section 5, we solve the more general Fixed Degree Subgraph With Outbranching: for a given instance $I = (d, \mathsf{in}, \mathsf{out}, \mathsf{root})$ we want to find a solution $m : V^2 \to \mathbb{Z}_{\geq 0}$ that satisfies the degree constraints specified by in and out and contains an outbranching rooted at root.

Let T be an arbitrary outbranching. We define a sequence $L_1(T), L_2(T), \ldots$ of subsets of $V(T)$ as follows. For $i \geq 1$ let $L_i(T)$ be the set of leaves of $T \setminus (L_1(T) \cup L_2(T) \cup \ldots \cup L_{i-1}(T))$ if $|V(T) \setminus (L_1(T) \cup \ldots \cup L_{i-1}(T))| > 1$, and otherwise $L_i = \emptyset$. The sets $L_i(T)$ will be called *leaf layers*. Denote $R_i(T) = V \setminus (L_1(T) \cup \cdots \cup L_i(T))$ for any $i \geq 1$.

▶ **Lemma 6.2.** *For every $i \geq 1$ we have $\mathsf{root} \in R_i(T) \setminus L_{i+1}(T)$, $|L_i(T)| \geq |L_{i+1}(T)|$ and $|L_{i+1}| \leq \frac{n-|R_i(T)|}{i}$.*

Proof. In this proof we skip the "(T)" in L_i and R_i because there is no ambiguity. Assume $\mathsf{root} \in L_i$ for some $i \geq 1$. It means that root is a leaf in $T \setminus (L_1 \cup L_2 \cup \ldots \cup L_{i-1})$. Then $V \setminus (L_1 \cup L_2 \cup \ldots \cup L_{i-1}) = \{\mathsf{root}\}$ and $L_i = \emptyset$, a contradiction. Hence $\mathsf{root} \notin L_i$ for all $i \geq 1$, and in consequence $\mathsf{root} \in R_i$ for all $i \geq 1$. However, $\mathsf{root} \in R_{i+1}(T)$ implies that $\mathsf{root} \notin L_{i+1}(T)$, hence $\mathsf{root} \in R_i \setminus L_{i+1}$.

If $|V \setminus (L_1 \cup \ldots \cup L_i)| > 1$, then L_{i+1} is the set of leaves of the out-tree $T \setminus (L_1 \cup \ldots \cup L_i)$, which is contained in the set of parents of vertices in L_i. Since every vertex in L_i has exactly one parent, $|L_i| \geq |L_{i+1}|$. If $|V \setminus (L_1 \cup \ldots \cup L_i)| \leq 1$ then $L_{i+1} = \emptyset$ and clearly $|L_i| \geq |L_{i+1}| = 0$.

Finally, since for every $j < i$ we have $|L_j| \geq |L_i|$ we get $n - |R_i| = |L_1| + \ldots + |L_i| \geq i|L_i|$. It follows that $|L_{i+1}| \leq |L_i| \leq \frac{n-|R_i|}{i}$, as required. ◀

Pseudocode of our algorithm is presented as Algorithm 2.

Algorithm 2 A pseudocode of the algorithm from Section 6.1.

```
1:  function Solve(G, out, in, d, root)
2:      best ← ∞
3:      for R, L_{K+1}, δ do
4:          T_R ← cheapest δ-out-tree spanning R rooted at root (Lemma 6.1)
5:          out' ← out − outdeg_{T_R}
6:          in' ← in − indeg_{T_R}
7:          for L_1, ..., L_K do
8:              F ← CreateNetwork(G, R, out', in', d, L_1, ..., L_K)
9:              f ← MinCostMaxFlow(F)
10:             if |f| = Σ_{v∈V(G)} out'(v) and cost(f) + d(T_R) < best then
11:                 best ← cost(f) + d(T_R)
        return best
```

For clarity, in the pseudocode we skipped some constraints that we enforce on the sets L_i and sequence δ. We state them below.

(C1) $L_{K+1} \subseteq R \subseteq V, \mathsf{root} \in R \setminus L_{K+1}, |L_{K+1}| \leq \frac{n-|R|}{K}$

(C2) $\{\delta_v\}_{v \in R}$ is a rooted out-tree sequence, i.e., for all $v \in R$ we have $\delta_v \in \mathbb{Z}_{\geq 0}, \sum_{v \in R} \delta(v) = |R| - 1$; also $\delta_{\mathsf{root}} \geq 1$ if $|R| \geq 2$ and $\delta_{\mathsf{root}} = 0$ if $|R| = 1$.

(C3) for every $v \in L_{K+1}$ we have $\delta_v = 0$ and for every $v \in R \setminus (L_{K+1} \cup \{\mathsf{root}\})$ we have $\delta_v \geq 1$

(C4) $L_1 \uplus L_2 \uplus \cdots \uplus L_K = V \setminus R$

(C5) $|L_i| \geq |L_{i+1}|$ for $i = 1, \ldots, K$.

It is clear that all these possibilities can be enumerated in time proportional to their total number times $O(n)$.

Let us provide some further intuition about Algorithm 2. Consider an optimum solution m of I and any outbranching B in m rooted at root. In Algorithm 2, for any $i = 1, \ldots K+1$, the set L_i is a guess of the leaf layer $L_i(B)$, while R is a guess of $V \setminus (L_1(B) \cup \cdots \cup L_K(B))$. Finally, δ is a guess of the outdegree sequence of the out-tree $B[R]$.

In Line 8 we create a flow network, and in line 9 a minimum cost maximum flow is found in polynomial time. In the next section we discuss the flow network and properties of the flow.

6.2 The flow

In this section we consider a run of Algorithm 2, and in particular we assume that the variables $R, \delta, L_1, \ldots, L_{K+1}$ have been assigned accordingly. The function CreateNetwork in our algorithm builds a flow network $F = (V(F), E(F), \mathsf{cap}, \mathsf{cost})$, where $E(F)$ is a set of directed edges and cap and cost are functions from edges to integers denoting capacities and costs of corresponding edges. As usual, the function cost extends to flow functions in a natural way, i.e., $\mathsf{cost}(f) = \sum_{e \in E(F)} f(e)\mathsf{cost}(e)$. We let $V(F) = \{s, t\} \cup \{v^I, v^O \mid v \in V(G)\} \cup \{v^C \mid v \in V \setminus R\}$, where s and t denote the source and the sink of F.

We put following edges into $E(F)$:

- (s, v^O), where $\mathsf{cap}(s, v^O) = \mathsf{out}'(v), \mathsf{cost}(s, v^O) = 0$ for every $v \in V(G)$
- (v^I, t), where $\mathsf{cap}(v^I, t) = \mathsf{in}'(v), \mathsf{cost}(v^I, t) = 0$ for every $v \in R$
- (v^I, t), where $\mathsf{cap}(v^I, t) = \mathsf{in}'(v) - 1, \mathsf{cost}(v^I, t) = 0$ for every $v \notin R$
- (v^C, t), where $\mathsf{cap}(v^C, t) = 1, \mathsf{cost}(v^C, t) = 0$ for every $v \notin R$
- (u^O, v^I), where $\mathsf{cap}(u^O, v^I) = \infty, \mathsf{cost}(u^O, v^I) = d(u, v)$ for every $(u, v) \in E(G)$
- (u^O, v^C), where $\mathsf{cap}(u^O, v^C) = \infty, \mathsf{cost}(u^O, v^C) = d(u, v)$ for every $v \in L_i, u \in R \cup L_{i+1} \cup \ldots \cup L_K, (u, v) \in E(G)$.

We will say that F has a *full flow* if it has a flow f with value $|f| = \sum_{v \in V} \mathsf{out}'(v)$. By the construction of F, then all edges leaving source are saturated, i.e., carry flow equal to their capacity. Since $\sum_{v \in V} \mathsf{out}'(v) = \sum_{v \in V} \mathsf{in}'(v)$, also all edges that enter the sink are saturated.

Essentially, the network above results from extending the standard network used to get Observation 3.3 by vertices v^C. The flow between $\{v^O \mid v \in V\}$ and $\{v^I \mid v \in V\} \cup \{v^C \mid v \in V \setminus R\}$ represents the resulting solution. In a full flow the edges leaving v^C are saturated, so a unit of flow enters every vertex v^C, which results in connecting v in the solution to a higher layer or to R. Thanks to that the solution resulting from adding the out-tree T_R to the solution extracted from f contains an outbranching.

▶ **Lemma 6.3.** *If f is a full flow of minimum cost in F then there exists a solution of I with cost $\mathsf{cost}(f) + d(T_R)$. Moreover, the solution can be extracted from f in polynomial time.*

Proof. By standard arguments, since all capacities in F are integer, we infer that there is an integral flow of minimum cost (and it can be found in polynomial time), so we assume w.l.o.g. that f is integral.

Let $b : V^2 \to \{0, 1\}$ denote a function such that $b(u, v) = [(u, v) \in T_R]$. Now we construct a solution $m : V^2 \to \mathbb{Z}_{\geq 0}$ of I.

$$m(u,v) = \begin{cases} f(u^O, v^I) + b(u,v) & \text{if } v \in R \\ f(u^O, v^I) + f(u^O, v^C) & \text{if } v \notin R. \end{cases}$$

In other words, m describes how many times edge (u, v) was used by the out-tree T_R and flow f in total. Let us verify that m is a feasible solution for I. The degree constraints are easy to verify, so we are left with showing that m contains an outbranching rooted at root. To this end it suffices to show that every vertex v is reachable from root in G_m. Clearly, this holds for vertices in R, thanks to the out-tree T_R. Pick an arbitrary vertex $v \notin R$. Then $v \in L_i$ for some $i = 1, \ldots, K$. We know that $f(v^C, t) = 1$, so there exists u such that $f(u^O, v^C) = 1$. Therefore, v is connected in G_m to a vertex from $R \cup L_{i+1} \cup \ldots L_K$. Since v in G_m has an in-neighbor either in R or in a layer with a higher index, we can conclude that there is a path in G_m from R to v. Hence m indeed contains the required outbranching.

Finally, it can be easily checked that $d(m) = \mathsf{cost}(f) + d(T_R)$, what concludes this proof. ◀

Let m be a feasible solution for I. Let R, L_i for $i = 1, \ldots, K + 1$ be sets of vertices and δ an out-tree sequence on R, as in Algorithm 2. We say that m is *compliant* with R, $L_1, \ldots, L_{K+1}$ and δ when m contains an outbranching T rooted at root such that $R_K(T) = R$, $L_i(T) = L_i$ for $i = 1, \ldots, K + 1$ and δ is equal to the outdegree sequence of $T[R]$.

▶ **Lemma 6.4.** *Assume that there exists a solution m of I that is compliant with R, $L_1, \ldots, L_{K+1}$ and δ. Then F has a full flow f such that $\mathsf{cost}(f) + d(T_R) \leq d(m)$.*

Proof. Let T be an outbranching in m which certifies that m is compliant with $R, L_1, \ldots, L_{K+1}$ and δ. Let $p : V^2 \to \{0, 1\}$ be a function such that for every $u, v \in V$ we have $p(u, v) = [(u, v) \in T]$.

We set $f(s, u) = \mathsf{cap}(s, u)$ for all edges $(s, u) \in E(F)$ and $f(u, t) = \mathsf{cap}(u, t)$ for all edges $(u, t) \in E(F)$. If $v \in V \setminus R$ then we set $f(u^O, v^C) = p(u, v)$. For all $u, v \in V(G)$ we set $f(u^O, v^I) = m(u, v) - p(u, v)$. It can be easily checked that such function f is a full flow and $\mathsf{cost}(f) = d(m) - d(T[R])$. However, since $T[R]$ is a δ-out-tree rooted at root and T_R is a cheapest such out-tree, $d(T_R) \leq d(T[R])$. It follows that $\mathsf{cost}(f) \leq d(m) - d(T_R)$, so $\mathsf{cost}(f) + d(T_R) \leq d(m)$ as required. ◀

Consider a *minimum cost* full flow f' in F that is found by Algorithm 2 for a choice of $R, L_1, \ldots, L_{K+1}, \delta$. The claim above implies that $\mathsf{cost}(f') + d(T_R) \leq d(m)$. However, notice that we do not claim that $\mathsf{cost}(f')$ is the cost of optimal completion of T_R consistent with all guesses, as the intuitions we described earlier might suggest. It could be the case that in the solution resulting from f', a vertex which was guessed to belong to L_i does not have any out-neighbor that was guessed to belong to L_{i-1}, which would mean that this vertex should be in an earlier layer. However, that is not an issue for the extraction of the global optimum solution of I, because we may get only better solutions than the optimum completion for that particular guess.

6.3 Correctness

▶ **Lemma 6.5.** *Function* Solve *returns the cost of an optimal solution of* I.

Proof. From Lemma 6.3 we infer that Solve returns the cost of a feasible solution of I. It remains to show that it returns a value that is smaller or equal to the cost of an optimal solution of I. To this end, let m be an arbitrary optimal solution of I and let T be an arbitrary outbranching rooted at root in G_m. Let $R = R_K(T)$, $L_i = L_i(T)$ for $i = 1, \ldots, K+1$ and let δ be the outdegree sequence of $T[R]$.

Let us verify that $R, L_1, \ldots, L_{K+1}$ and δ satisfy constraints (C1)–(C5). We get (C1) and (C5) by Lemma 6.2. (C2) follows from the definition of δ. For (C3), consider two cases. If $|R| > 1$, then L_{K+1} is the set of leaves in R and hence indeed for every $v \in L_{K+1}$ we have $\delta_v = 0$ and for every $v \in R \setminus (L_{K+1} \cup \{\mathsf{root}\})$ we have $\delta_v \geq 1$. When $|R| \leq 1$, we have $L_{K+1} = \emptyset$ and since $\mathsf{root} \in R$ by Lemma 6.2, $R = \{\mathsf{root}\}$. Then both sets L_{K+1} and $R \setminus (L_{K+1} \cup \{\mathsf{root}\})$ are empty, so (C3) trivially holds. Finally, (C4) follows by the definition of leaf layers.

Since $R, L_1, \ldots, L_{K+1}$ and δ satisfy constraints (C1)–(C5), then Solve reaches this particular evaluation of the variables $R, L_1, \ldots, L_{K+1}$ and δ. Then, based on Lemma 6.4, the network F has a full flow f such that $\mathsf{cost}(f) + d(T_R) \leq d(m)$, and it follows that Solve returns a value $\mathsf{best} \leq \mathsf{cost}(f) + d(T_R) \leq d(m)$, as required. ◀

Obviously, Solve can be easily adapted to return a solution of I with the cost it returns, but we have not taken this into account in Algorithm 2 for the sake of its readability.

6.4 Running time

Having a correct algorithm solving Fixed Degree Subgraph With Outbranching in polynomial space, let us analyze its complexity depending on K.

Let us denote $r = |R|$ and $c = |L_{K+1}|$. Recall that $1 \leq r \leq n$ and $0 \leq c \leq \lfloor \frac{n-r}{K} \rfloor$.

If we fix r and c, then there are $\binom{n-1}{r-1}$ guesses for R (it has to contain root) and at most $\binom{r-1}{c}$ guesses for L_{K+1}. Let us bound the number of guesses for δ. By (C2) and (C3), $\sum_{v \in R} \delta_v = r - 1$, and $\delta_v = 0$ iff $v \in L_{K+1}$ so essentially we put $r-1$ balls into $r-c$ bins that must be nonempty, which is $\binom{r-2}{c-1}$ by standard combinatorics. In the special case $c = 0$ there is one choice for δ, where $\delta_{\mathsf{root}} = 0$.

In total, there are at most $\binom{n}{r}\binom{r}{c}^2$ guesses for all R, L_{K+1}, δ simultaneously. For each of these guesses, using Lemma 6.1 function Solve calculates an optimal δ-out-tree spanning R, which takes time $\mathcal{O}(4^{n+o(n)})$. It follows that that part takes time $\mathcal{O}^*(\sum_r \binom{n}{r} 4^{r+o(r)} \sum_c \binom{r}{c}^2)$. Then, Solve guesses a partition of $V \setminus R$ into $L_1, \ldots, L_K$ in at most K^{n-r} ways. For each such guess, Solve spends polynomial time, so that part takes $\mathcal{O}^*(\sum_r \binom{n}{r} K^{n-r} \sum_c \binom{r}{c}^2)$ time. Hence the total running time can be bounded by

$$\mathcal{O}^*\left(2^{o(n)} \sum_{r=1}^{n} \sum_{c=0}^{\lfloor \frac{n-r}{K} \rfloor} \underbrace{\binom{n}{r}(K^{n-r} + 4^r)\binom{r}{c}^2}_{\xi(r,c)}\right).$$

Since there are polynomially many guesses for r and c, we can actually replace sums with maxima in the expression above and focus on the expression $\xi(r,c) = \binom{n}{r}(K^{n-r} + 4^r)\binom{r}{c}^2$.

We will heavily use the well-known bound $\binom{n}{\alpha n} < 2^{h(\alpha)n}$, where $h(\alpha) = -\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha)$ is the binary entropy function (see e.g. [13]). For readability, let us denote $f(\alpha) = 2^{h(\alpha)}$ and let us point out that f is increasing on the interval $[0, \frac{1}{2}]$ and decreasing on the interval $[\frac{1}{2}, 1]$. Let us denote $\beta \coloneqq \frac{r}{n}$. We are going to distinguish two cases here.

1. $\frac{n-r}{K} \geq \frac{r}{2}$
   This inequality can be rephrased as $r \leq \frac{2}{K+2}n$, which is equivalent to $\beta \leq \frac{2}{K+2}$. We will use here a trivial bound $\binom{r}{c} \leq 2^r$. Then, $\xi(r,c) \leq f(\beta)^n((K^{1-\beta})^n + 4^{\beta n})4^{\beta n} = (f(\beta)K^{1-\beta}4^{\beta})^n + (f(\beta)4^{2\beta})^n$
2. $\frac{n-r}{K} < \frac{r}{2}$
   In that case we know that $\max_{c=0}^{\lfloor \frac{n-r}{K} \rfloor} \binom{r}{c}^2$ is attained when $c = \lfloor \frac{n-r}{K} \rfloor$ and for that particular value of c we can use the following bound.

$$\binom{r}{c}^2 = \left(\frac{r}{\lfloor \frac{n-r}{K} \rfloor} \cdot r\right)^2 = \mathcal{O}^*\left(f\left(\frac{\lfloor \frac{n-r}{K} \rfloor}{r}\right)^{2r}\right) = \mathcal{O}^*\left(f\left(\frac{\frac{n-r}{K}}{r}\right)^{2r}\right) =$$

$$= \mathcal{O}^*\left(f\left(\frac{1-\beta}{K\beta}\right)^{2\beta n}\right)$$

   In the third equality above we used fact that f is increasing on the interval $[0, \frac{1}{2}]$. To sum up, in this case,

$$\xi(r,c) = \mathcal{O}^*\left(\left(f(\beta)K^{1-\beta}f\left(\frac{1-\beta}{K\beta}\right)^{2\beta}\right)^n + \left(f(\beta)4^{\beta}f\left(\frac{1-\beta}{K\beta}\right)^{2\beta}\right)^n\right).$$

Our numerical analysis shows that it is optimal to choose $K = 4$. For that particular value of K, the first case applies if and only if $\beta \leq \frac{1}{3}$. Then,

$$\xi(r,c) = (4f(\beta))^n + (4^{2\beta}f(\beta))^n = \mathcal{O}((4f(\beta))^n) = \mathcal{O}((4f(\tfrac{1}{3}))^n) = \mathcal{O}(7.56^n).$$

Let us now investigate the second case, when $\beta > \frac{1}{3}$. For $\frac{1}{3} < \beta < \frac{1}{2}$ the summand $\left(f(\beta)4^{1-\beta}f\left(\frac{1-\beta}{4\beta}\right)^{2\beta}\right)^n$ dominates, and for $\beta \geq \frac{1}{2}$ the summand $\left(f(\beta)4^{\beta}f\left(\frac{1-\beta}{4\beta}\right)^{2\beta}\right)^n$ dominates. We have numerically verified that

$$f(\beta)4^{1-\beta}f\left(\frac{1-\beta}{4\beta}\right)^{2\beta} \leq 7.68 \text{ for } \beta \in (\tfrac{1}{3}, \tfrac{1}{2})$$

and

$$f(\beta)4^{\beta}f\left(\frac{1-\beta}{4\beta}\right)^{2\beta} \leq 7.871 \text{ for } \beta \in [\tfrac{1}{2}, 1].$$

Hence, we can conclude that for $K = 4$ and our algorithm runs in time $\mathcal{O}^*(7.871^{n+o(n)}) = \mathcal{O}(7.88^n)$ and in polynomial space. This concludes the proof of Theorem 1.4.

7 $(1+\epsilon)$-approximation

In this section we show theorem 1.2, i.e. we present an algorithm for Many Visits TSP which finds a $(1+\epsilon)$-approximation in $\mathcal{O}^*\left(\frac{2^n}{\epsilon}\right)$ time and polynomial space.

To achieve this we consider a more general problem, namely Fixed Degree Connected Subgraph. The main idea is to round weights of edges of the given instance, so that we can use the algorithm for polynomially bounded weights from Lemma 4.12 which is an analog of theorem 1.1 for Fixed Degree Connected Subgraph.

Let us first consider the case with degrees bounded by a polynomial.

▶ **Lemma 7.1.** *For given $\epsilon > 0$ and an instance $I = (d, \mathsf{in}, \mathsf{out})$ of* Fixed Degree Connected Subgraph *such that $\mathsf{in}(v), \mathsf{out}(v) \leq \mathcal{O}(n^2)$ for every vertex v there exists an algorithm finding a $(1+\epsilon)$-approximate solution in $\mathcal{O}^*\left(\frac{2^n}{\epsilon}\right)$ time and polynomial space.*

Proof. Let us denote the optimal solution for I by OPT. First, our algorithm guesses the most expensive edge used by OPT. Let us denote its cost by E, in particular

$$E \leq d(\mathrm{OPT}). \tag{1}$$

Let us denote by C the universal constant such that $\mathsf{in}(v), \mathsf{out}(v) \leq Cn^2$ for every vertex v and let us round d in the following way

$$d'(u,v) := \begin{cases} \left\lceil \frac{Cn^3}{\epsilon E} d(u,v) \right\rceil & \text{if } d(u,v) \leq E \\ \infty & \text{if } d(u,v) > E \end{cases} \tag{2}$$

If $d'(u,v)$ is finite then it is bounded by $\left\lceil \frac{Cn^3}{\epsilon E} E \right\rceil = \left\lceil \frac{Cn^3}{\epsilon} \right\rceil$. Our algorithm simply returns the optimal solution for instance $I' = (d', \mathsf{in}, \mathsf{out})$ which can be found in $\mathcal{O}^*\left(\frac{2^n}{\epsilon}\right)$ time using the algorithm from Lemma 4.12 with $D = \left\lceil \frac{Cn^3}{\epsilon} \right\rceil$. Let us denote this solution by ALG. Now we only need to prove that ALG is a $(1+\epsilon)$-approximation for the original instance I. We know that ALG is an optimal solution for I', in particular

$$d'(\mathrm{ALG}) \leq d'(\mathrm{OPT}). \tag{3}$$

For every v we have $\mathsf{out}(v) \leq Cn^2$, so

$$\sum_{(u,v) \in V^2} \mathrm{OPT}(u,v) = \sum_{u \in V} \mathsf{out}(u) \leq n \cdot Cn^2. \tag{4}$$

The following chain of inequalities finishes the proof.

$$d(\mathrm{ALG}) \overset{(2)}{\leq} \frac{\epsilon E}{Cn^3} d'(\mathrm{ALG}) \overset{(3)}{\leq} \frac{\epsilon E}{Cn^3} d'(\mathrm{OPT}) \overset{(2)}{\leq} \frac{\epsilon E}{Cn^3} \sum_{(u,v) \in V^2} \mathrm{OPT}(u,v) \left(\frac{d(u,v) Cn^3}{\epsilon E} + 1 \right) =$$

$$= d(\mathrm{OPT}) + \frac{\epsilon E}{Cn^3} \sum_{(u,v) \in V^2} \mathrm{OPT}(u,v) \overset{(4)}{\leq} d(\mathrm{OPT}) + \frac{\epsilon E}{Cn^3} Cn^3 \overset{(1)}{\leq} (1+\epsilon) d(\mathrm{OPT}) \qquad \blacktriangleleft$$

Now we can generalize the algorithm from Lemma 7.1 by using it as a black box for the general case.

▶ **Lemma 7.2.** *For a given $\epsilon > 0$ and an instance $I = (d, \mathsf{in}, \mathsf{out})$ of* Fixed Degree Connected Subgraph *there exists an algorithm finding a $(1+\epsilon)$-approximate solution in $\mathcal{O}^*\left(\frac{2^n}{\epsilon}\right)$ time and polynomial space.*

Proof. First let us use the algorithm from Theorem 3.4 which outputs an instance $I' = (d, \mathsf{in}', \mathsf{out}')$ of Fixed Degree Connected Subgraph and a function $f : V^2 \to \mathbb{Z}_{\geq 0}$. Let us denote the optimal solution for I' by OPT'. By Theorem 3.4 the optimal solution for I equals $\mathrm{OPT}' + f$. Moreover, we know that $\mathsf{in}', \mathsf{out}'(v) \leq \mathcal{O}(n^2)$ for every vertex v. In particular we can use algorithm from Lemma 7.1 to get a solution ALG' for instance I' such that $d(\mathrm{ALG}') \leq (1+\epsilon) d(\mathrm{OPT}')$. Our algorithm simply returns solution $\mathrm{ALG}' + f$, which is a solution for I because ALG is connected and f increases degrees exactly by difference between I and I'. To prove $\mathrm{ALG}' + f$ is $(1+\epsilon)$-approximation we just need to observe that

$$d(\mathrm{ALG}' + f) \leq (1+\epsilon) d(\mathrm{OPT}') + d(f) \leq (1+\epsilon) d(\mathrm{OPT}' + f). \qquad \blacktriangleleft$$

Fixed Degree Connected Subgraph is a generalization of Many Visits TSP so the algorithm from Lemma 7.2 proves Theorem 1.2.

8 Further Research

Since TSP is solvable in time $\mathcal{O}^*(2^n)$ and exponential space [1, 16] and time $\mathcal{O}(4^{n+o(n)})$ and polynomial space [15], the main remaining question is whether these bounds can be achieved for Many Visits TSP avoiding in the running time bound the linear dependence on maximum distance D. Another interesting goal is a deterministic version of Theorem 4.1.

References

1 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, January 1962. doi:10.1145/321105.321111.

2 André Berger, László Kozma, Matthias Mnich, and Roland Vincze. Time- and space-optimal algorithms for the many-visits TSP. *CoRR*, abs/1804.06361, 2018. arXiv:1804.06361.

3 André Berger, László Kozma, Matthias Mnich, and Roland Vincze. A time- and space-optimal algorithm for the many-visits TSP. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1770–1782. SIAM, 2019. doi:10.1137/1.9781611975482.106.

4 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed Hamiltonicity and out-branchings via generalized Laplacians. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 91:1–91:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.91.

5 Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Fast witness extraction using a decision oracle. In *ESA*, volume 8737 of *Lecture Notes in Computer Science*, pages 149–160. Springer, 2014.

6 Nadia Brauner, Yves Crama, Alexander Grigoriev, and Joris Van de Klundert. A framework for the complexity of high-multiplicity scheduling problems. *J. Comb. Optim.*, 9:313–323, May 2005. doi:10.1007/s10878-005-1414-7.

7 Stavros S. Cosmadakis and Christos H. Papadimitriou. The traveling salesman problem with many visits to few cities. *SIAM J. Comput.*, 13(1):99–108, 1984.

8 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3), May 2016. doi:10.1145/2925416.

9 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3), March 2018. doi:10.1145/3148227.

10 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159. IEEE Computer Society, 2011.

11 R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7:193–195, 1978.

12 Hamilton Emmons and Kamlesh Mathur. Lot sizing in a no-wait flow shop. *Operations Research Letters - ORL*, 17:159–164, May 1995. doi:10.1016/0167-6377(95)00008-8.

13 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

14 Alexander Grigoriev and Joris Van de Klundert. On the high multiplicity traveling salesman problem. *Discrete Optimization*, 3:50–62, March 2006. doi:10.1016/j.disopt.2005.11.002.

15 Yuri Gurevich and Saharon Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, June 1987. doi:10.1137/0216034.

16 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, page 71.201–71.204, New York, NY, USA, 1961. Association for Computing Machinery. doi:10.1145/800029.808532.

17 Dorit S. Hochbaum and Ron Shamir. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Oper. Res.*, 39:648–653, 1991.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

20 László Kozma and Tobias Mömke. Maximum scatter TSP in doubling metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, page 143–153, USA, 2017. Society for Industrial and Applied Mathematics.

21 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. In *Algorithms – ESA 2010*, volume 64, pages 549–560, September 2010. doi:10.1007/978-3-642-15775-2_47.

22 László Lovász. On determinants, matchings and random algorithms. In *FCT*, volume 79, pages 565–574, January 1979.

23 James B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.*, 41(2):338–350, 1993. doi:10.1287/opre.41.2.338.

24 Harilaos N. Psaraftis. A dynamic programming approach for sequencing groups of identical jobs. *Operations Research*, 28(6):1347–1359, 1980.

25 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

26 W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, s1-22(2):107–111, April 1947. doi:10.1112/jlms/s1-22.2.107.

27 Jack A. A. van der Veen and Shuzhong Zhang. Low-complexity algorithms for sequencing jobs with a fixed number of job-classes. *Comput. Oper. Res.*, 23(11):1059–1067, November 1996. doi:10.1016/0305-0548(96)00016-0.

28 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. International Symposium on Symbolic and Algebraic Computation*, volume 72 of *LNCS*, pages 216–226, 1979.

Light Euclidean Spanners with Steiner Points

Hung Le
University of Victoria, Canada
University of Massachusetts, Amherst, MA, USA
hungle@uvic.ca

Shay Solomon
Tel-Aviv University, Israel
solo.shay@gmail.com

Abstract

The FOCS'19 paper of Le and Solomon [59], culminating a long line of research on Euclidean spanners, proves that the lightness (normalized weight) of the greedy $(1+\epsilon)$-spanner in $\mathbb{R}^d$ is $\tilde{O}(\epsilon^{-d})$ for any $d = O(1)$ and any $\epsilon = \Omega(n^{-\frac{1}{d-1}})$ (where $\tilde{O}$ hides polylogarithmic factors of $\frac{1}{\epsilon}$), and also shows the existence of point sets in $\mathbb{R}^d$ for which any $(1+\epsilon)$-spanner must have lightness $\Omega(\epsilon^{-d})$.[1] Given this tight bound on the lightness, a natural arising question is whether a better lightness bound can be achieved using *Steiner points*.

Our first result is a construction of Steiner spanners in $\mathbb{R}^2$ with lightness $O(\epsilon^{-1}\log\Delta)$, where Δ is the spread of the point set.[2] In the regime of $\Delta \ll 2^{1/\epsilon}$, this provides an improvement over the lightness bound of [59]; this regime of parameters is of practical interest, as point sets arising in real-life applications (e.g., for various random distributions) have polynomially bounded spread, while in spanner applications ϵ often controls the precision, and it sometimes needs to be much smaller than $O(1/\log n)$. Moreover, for spread polynomially bounded in $1/\epsilon$, this upper bound provides a quadratic improvement over the non-Steiner bound of [59], We then demonstrate that such a light spanner can be constructed in $O_\epsilon(n)$ time for polynomially bounded spread, where O_ϵ hides a factor of $\text{poly}(\frac{1}{\epsilon})$. Finally, we extend the construction to higher dimensions, proving a lightness upper bound of $\tilde{O}(\epsilon^{-(d+1)/2} + \epsilon^{-2}\log\Delta)$ for any $3 \le d = O(1)$ and any $\epsilon = \Omega(n^{-\frac{1}{d-1}})$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Euclidean spanners, Steiner spanners, light spanners

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.67

Funding *Hung Le*: Supported by a PIMS postdoctoral fellowship and an NSERC grant.
Shay Solomon: Partially supported by the Israel Science Foundation grant No.1991/19 and by Len Blavatnik and the Blavatnik Family foundation.

Acknowledgements Shay Solomon is grateful to Michael Elkin, Ofer Neiman and Michiel Smid for fruitful discussions.

1 Introduction

A *t-spanner* for a set P of points in the d-dimensional Euclidean space $\mathbb{R}^d$ is a *geometric graph* that preserves all the pairwise Euclidean distances between points in P to within a factor of t, called the *stretch factor*; by geometric graph we mean a weighted graph in which the vertices correspond to points in $\mathbb{R}^d$ and the edge weights are the Euclidean distances between the corresponding points. The study of Euclidean spanners dates back to the seminal work of Chew [27, 28] from 1986, which presented a spanner with constant stretch and $O(n)$ edges for any set of n points in $\mathbb{R}^2$. In the three following decades, Euclidean spanners

[1] The lightness of a spanner is the ratio of its weight and the MST weight.
[2] The spread $\Delta = \Delta(P)$ of a point set P in $\mathbb{R}^d$ is the ratio of the largest to the smallest pairwise distance.

have evolved into an important subarea of Discrete and Computational Geometry, having found applications in many different areas, such as approximation algorithms [65], geometric distance oracles [42, 45, 44, 43], and network design [52, 60]; see the book by Narasimhan and Smid "Geometric Spanner Networks" [61] for an excellent account on Euclidean spanners and some of their applications. Numerous constructions of Euclidean spanners in two and higher dimensions were introduced over the years, such as Yao graphs [70], Θ-graphs [29, 54, 55, 66], the (path-)greedy spanner [4, 23, 61] and the gap-greedy spanner [67, 6] – unveiling an abundance of techniques, tools and insights along the way; refer to the book of [61] for more spanner constructions.

In addition to low stretch, many applications require that the spanner would be *sparse*, in the unweighted and/or weighted sense. The *sparsity* (respectively, *lightness*) of a spanner is the ratio of its size (respectively, weight) to the size (resp., weight) of a spanning tree (resp., MST), providing a normalized notion of size (resp., weight), which should ideally be $O(1)$. For any dimension $d = O(1)$, spanners with constant sparsity are known since the 80s [70, 29, 54, 55, 66, 4, 67, 6]; also, it is known since the early 90s that the greedy spanner has constant lightness [4]. The constant bounds on the sparsity and lightness depend on both ϵ and d. In some applications, ϵ must be a very small sub-constant parameter, so as to achieve the highest possible precision and minimize potential errors, and in some situations ϵ may be as small as n^{-c} for some constant $0 < c < 1$. Consequently, besides the theoretical appeal, achieving the precise dependencies on ϵ and d in the sparsity and lightness bounds is of practical importance.

Culminating a long line of work, in FOCS'19 Le and Solomon [59] showed that the precise dependencies on ϵ in the sparsity and lightness bounds are $\Theta(\epsilon^{1-d})$ and $\tilde{\Theta}(\epsilon^{-d})$, respectively, for any $d = O(1)$ and any $\epsilon = \Omega(n^{-\frac{1}{d-1}})$; throughout we shall use $\tilde{O}, \tilde{\Omega}, \tilde{\Theta}$ to hide polylogarithmic factors of $\frac{1}{\epsilon}$. The lower bounds of [59] are proved for the d-dimensional sphere, for $d = O(1)$. On the upper bound side, sparsity $O(\epsilon^{1-d})$ is achieved by a number of classic constructions such as Yao graphs [70], Θ-graphs [29, 54, 55, 66], the (path-)greedy spanner [4, 23, 61] and the gap-greedy spanner [67, 6], and the argument underlying all these upper bounds is basic and simple. On the other hand, constant lightness upper bound (regardless of the dependency on ϵ and d) is achieved only by the greedy algorithm [4, 31, 32, 65, 61, 41, 13, 59], and all known arguments for constant lightness are highly nontrivial, even for $d = 2$; in fact, the proofs in [4, 31, 32, 65] have missing details. The first complete proof was given in the book of [61], in a 60-page chapter, where it is shown that the greedy $(1+\epsilon)$-spanner has lightness $O(\epsilon^{-2d})$, which improved the dependencies on ϵ and d given in all previous work. In SODA'19, Borradaile, Le and Wulff-Nilsen [13] presented a shorter and arguably simpler alternative proof that applies to the wider family of *doubling metrics*, which improves the ϵ dependency provided in FOCS'15 by Gottlieb [41] for doubling metrics, but is inferior to the lightness bound of $O(\epsilon^{-2d})$ by [61].[3] Finally, the lightness bound of $\tilde{O}(\epsilon^{-d})$ of [59] was proved via a tour-de-force argument; interestingly, the proof for $d = 2$ is much more intricate than for higher dimensions $d \geq 3$.

Le and Solomon [59] showed that, counter-intuitively, one can use Steiner points to bypass the sparsity lower bound of Euclidean spanners. Specifically, in this way they reduced the sparsity upper bound almost quadratically to $\tilde{O}(\epsilon^{(1-d)/2})$, and also provided a matching

[3] The *doubling dimension* of a metric space (X, δ) is the smallest value d such that every ball B in the metric space can be covered by at most 2^d balls of half the radius of B. This notion generalizes the Euclidean dimension, since the doubling dimension of the Euclidean space $\mathbb{R}^d$ is $\Theta(d)$. A metric space is called *doubling* if its doubling dimension is constant.

lower bound for $d = 2$. Their lower bound for sparsity is derived from a lightness lower bound; specifically, they first prove that, for a set of points evenly spaced on the boundary of the unit square, with distances $\Theta(\sqrt{\epsilon})$ between neighboring points, any Steiner $(1+\epsilon)$-spanner must incur lightness $\tilde{\Omega}(\frac{1}{\epsilon})$, and then they translated the lightness lower bound into a sparsity lower bound of $\tilde{\Omega}(\sqrt{1/\epsilon})$. Whether or not Steiner points can be used to reduce the lightness remained open in [59], but this should not come as a surprise.[4] First, bounding the lightness of spanners is inherently more difficult than bounding their sparsity – this is true for both Euclidean spaces and doubling metrics, as well as other graph families including general weighted graphs [4, 31, 32, 65, 61, 41, 13, 59, 26, 39, 33]. Second, constructing Steiner trees and spanners *with asymptotically improved bounds* is inherently more difficult than constructing their non-Steiner counterparts [34, 35, 68, 57, 11, 58]. In our particular case, while the classic sparsity upper bound for non-Steiner Euclidean spanners is simple, its improved counterpart for Steiner spanners by [59] requires a number of nontrivial insights and is rather intricate. On the other hand, the lightness upper bound of [59] uses a tour-de-force argument, and as mentioned already this is true even for $d = 2$. Consequently, obtaining an improved lightness bound using Steiner points seems currently out of reach, at least until an inherently simpler proof to [59] for non-Steiner lightness is found (if one exists).

1.1 Our Contribution

In this paper, we explore the power of Steiner points in reducing lightness for Euclidean spaces of bounded spread Δ.[5] Point sets of bounded spread have been studied extensively for Euclidean spanners and related geometric objects [9, 53, 30, 51, 17, 16, 36, 64, 1, 2, 5, 49, 50, 19, 62, 63, 18, 69, 21]. The motivation for studying point sets of bounded spread is three-fold.

1. Such point sets arise naturally in practice, and are thus important in their own right; indeed, for many random distributions, the spread is polynomial in the number of points – in expectation and with high probability. In particular, it is known that for n-point sets drawn uniformly at random from the unit square, the expected spread is $\Theta(n)$, and the expected spread in the unit d-dimensional hypercube is $n^{2/d}$ for any $d = O(1)$. Researchers have studied random distributions of point sets, in part to explain the success of solving various geometric optimization problems in practice [9, 53, 30], and there are many results on spanners for random point sets [24, 7, 37, 10, 38, 14, 61, 69, 15, 21, 8, 56]. Of course, the family of bounded spread point sets is much wider than that of random point sets.
2. Euclidean spanners can be constructed in (deterministic) $O(n)$ time in such point sets [20], while there is no $o(n \log n)$-time algorithm for constructing Euclidean spanners in arbitrary point sets. (The result of [20] extends to other geometric objects, such as WSPD and compressed quadtrees, and some of the aforementioned references (e.g., [50, 17]) build on this result to achieve faster algorithms for other geometric problems for point sets of polynomially bounded spread.)
3. The case of bounded spread is sometimes used as a stepping stone towards the general case; see, e.g., [40, 47, 46, 21, 22, 3, 59].

[4] The lightness of Steiner spanners can be defined with respect to the SMT (Steiner minimum tree) weight, but we can also stick to the original definition, since the SMT and MST weights differ by a constant factor smaller than 2.

[5] The *spread* of a Euclidean space is the ratio of the maximum to minimum pairwise distances in it.

As mentioned above, Le and Solomon [59] showed that for a set of points evenly spaced on the boundary of the unit square, with distances $\Theta(\sqrt{\epsilon})$ between neighboring points, any Steiner $(1+\epsilon)$-spanner must incur lightness $\tilde{\Omega}(\frac{1}{\epsilon})$. Note that the spread of this point set is $\frac{1}{\sqrt{\epsilon}}$. The non-Steiner upper bound for $d=2$ by [59] is $\tilde{O}(\epsilon^{-2})$. A natural arising question is whether one can improve the lightness upper bound of $\tilde{O}(\epsilon^{-2})$ using Steiner points, ideally quaratically, for point sets of bounded spread. We answer this question in the affirmative.

▶ **Theorem 1.** *Any point set P in $\mathbb{R}^2$ of spread Δ admits a Steiner $(1+\epsilon)$-spanner of lightness $O(\frac{\log\Delta}{\epsilon})$.*

Recalling that the lower bound of $\tilde{\Omega}(\frac{1}{\epsilon})$ by [59] applies to a point set of spread $\frac{1}{\sqrt{\epsilon}}$, the lightness upper bound provided by Theorem 1 is therefore tight (up to polylogarithmic factors in $\frac{1}{\epsilon}$) for point sets of spread $\text{poly}(\frac{1}{\epsilon})$. Moreover, this upper bound improves the general upper bound of $\tilde{O}(\epsilon^{-2})$ from [59] in the regime $\log\Delta \ll \epsilon^{-1}$, i.e., when $\Delta \ll 2^{1/\epsilon}$. For spread polynomial in n, we get an improvement over [59] as long as $\epsilon \ll \frac{1}{\log n}$. Of course, the improvement gets more significant as ϵ decays – in the most extreme situation ϵ is inverse polynomial in n, and then the improvement over [59] is polynomial in n even when Δ is exponential in n.

Our second result is that a Steiner spanner with near-optimal lightness can be constructed in linear time, when the spread Δ is polynomial in n.

▶ **Theorem 2.** *For any point set P in $\mathbb{R}^2$ of spread $\Delta = O(n^c)$, for any $c = O(1)$, a Steiner $(1+\epsilon)$-spanner of lightness $O(\frac{\log\Delta}{\epsilon})$ can be constructed in $O_\epsilon(n)$ time, where $O_\epsilon(.)$ hides a factor of $\text{poly}(\frac{1}{\epsilon})$.*

Higher dimensions

The lower bound of [58] states that any (non-Steiner) $(1+\epsilon)$-spanner must incur lightness $\Omega(\epsilon^{-d})$, for any $d = O(1)$. We show that, similarly to the 2-dimensional case, one can improve the lightness almost quadratically using Steiner points, for point sets of bounded spread.

▶ **Theorem 3.** *For any $d \geq 3$, any point set P in $\mathbb{R}^d$ of spread Δ admits a Steiner $(1+\epsilon)$-spanner of lightness $\tilde{O}(\epsilon^{-(d+1)/2} + \epsilon^{-2}\log\Delta)$.*

Interestingly, the dependence on the spread in the lightness bound provided by Theorem 3 does not grow with the dimension, hence the improvement over the non-Steiner bound gets more significant as the dimension grows, provided of course that the spread is not too large.

1.2 Proof Overview

Proof of Theorem 1

We partition the set of pairs of points into $m = O(\log\Delta)$ subsets $\{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ where $\mathcal{P}_i$ contains pairs of distances in $[2^{i-1}, 2^i)$. The objective is to show that one can preserve distances between all pairs in $\mathcal{P}_i$ to within a factor of $(1+\epsilon)$ using a Steiner spanner S_i of weight $O(\frac{w(\mathsf{MST})}{\epsilon})$; by taking the union of all such spanners, we obtain a Steiner spanner with the required lightness.

Let $L_i = 2^i$. A natural idea to preserve distances in $\mathcal{P}_i$ is to (a) find an (ϵL_i)-net N_i[6] of P and (b) add to the spanner edges between any two net points p and q such that $(u,v) \in \mathcal{P}_i$ and there is a pair (u,v) in $\mathcal{P}_i$ such that u and v are covered by (i.e., within distance ϵL_i

[6] A subset of points $N \subseteq P$ is an r-net if every point in P is within distance r from (or *covered* by) some point in N and pairwise distances between points in N are larger than r.

from) p and q, respectively. The stretch will be in check because u and v are at distance roughly $O(\epsilon)||u,v||$ from their net points and thus, the *additive* stretch between u and v is $O(\epsilon)||u,v||$. For lightness, we can show that the number of net points $|N_i| = O(\frac{w(\mathsf{MST})}{\epsilon L_i})$, and using a (nontrivial) packing argument, there are about $\frac{N_i}{\epsilon}$ edges of length $O(L_i)$ added in step (b). Thus, the total weight of the spanner is $O(\frac{w(\mathsf{MST})}{\epsilon^2})$, which is bigger than our aimed lightness bound by a factor of $\frac{1}{\epsilon}$.

To shave the factor of $\frac{1}{\epsilon}$, we employ two ideas. First, we take N_i to be a $\sqrt{\epsilon}L_i$-net of P. In this way $|N_i| = O(\frac{w(\mathsf{MST})}{\sqrt{\epsilon}})$. By applying the 2-dimensional Steiner spanner construction of [59] as a blackbox, we obtain a Steiner spanner with only $\frac{|N_i|}{\sqrt{\epsilon}}$ edges of length $O(L_i)$ that approximates distances between points in N_i; we have thus reduced the weight bound to $O(\frac{w(\mathsf{MST})}{\epsilon})$, as required. The problem now is with the stretch guarantee: Preserving distances between the points in N_i is no longer sufficient. Indeed, for every pair $(u,v) \in \mathcal{P}_i$, the distance between u and v to the nearest net points is $O(\sqrt{\epsilon})||u,v||$, hence the resulting stretch is $(1+O(\sqrt{\epsilon}))||u,v||$. We overcome this hurdle by introducing a novel construction of *single-source spanners*, which generalize Steiner shallow-light trees of Solomon [68]. Specifically, we open the black-box of [59] and observe that every time we want to preserve the distance from (some) Steiner point s to a net-point $p \in N_i$, instead of connecting s to p by a straight line of weight $O(L_i)$, we can use a single-source spanner (rooted at s) of *weight* $O(L_i)$ to preserve distances (up to a $(1+\epsilon)$ factor) from s to *every point* within distances $O(\sqrt{\epsilon}L_i)$ from p. As a result, our Steiner spanner can preserve distances between any two points $(u,v) \in \mathcal{P}_i$ to within a factor of $(1+\epsilon)$, where $u \in B(p, O(\sqrt{\epsilon}L_i))$, $v \in B(q, O(\sqrt{\epsilon}L_i))$, and p, q are their nearest net points.

Proof of Theorem 3

By extending the construction in Theorem 1, we can construct a Steiner spanner with lightness $\tilde{O}(\epsilon^{-(d+1)/2}\log\Delta)$ as follows: for each $\mathcal{P}_i$, we construct an ϵL_i-net N_i and then apply the construction of [59] as a black box to obtain a Steiner spanner S_i for N_i with weight $\tilde{O}(\epsilon^{-(d-1)/2}|N_i|L_i)$. The stretch will be in check since N_i is an ϵL_i-net. Since $|N_i| = O(\frac{w(\mathsf{MST})}{\epsilon L_i})$, $w(S_i) = \tilde{O}(\epsilon^{-(d+1)/2}w(\mathsf{MST}))$. The union of Steiner spanners S_i for all $i \in [1,m]$ has weight $\tilde{O}(\epsilon^{-(d+1)/2}\log\Delta)w(\mathsf{MST})$. Note when $d \geq 3$, we can not take N_i as a $\sqrt{\epsilon}L_i$-net since the construction of single-source spanners with $O(L_i)$ weight in the proof of Theorem 1 only works when $d = 2$.

Most of our effort is to further refine the result in a way that $\log\Delta$ term is multiplied only by ϵ^{-2} and not by the term that depends on d. We first reduce to the problem of approximating distances between pairs (of endpoints) in a family of edge sets $\mathcal{E} = \{E_1, \ldots, E_m\}$ with $m = O(\log\Delta)$, where edges of E_i have length (roughly) in the interval $(\frac{1}{2\epsilon^i}, \frac{1}{\epsilon^i}]$ and edges in E_1 have length in $[1, \frac{1}{\epsilon})$. Let $L_i = \frac{1}{\epsilon^i}$. For a technical reason, we will subdivide edges of MST by using a set of Steiner points K so that each new edge has length in $(1/2, 1]$.

We construct a Steiner spanner for edges in $\mathcal{E}$ using a *charging cover tree* T: T has depth $m+1$, level i of T is associated with an ϵL_i-*cover*[7] of $P \cup K$, and leaves (at level 0) of T are points in $P \cup K$. For each cover N_i at level i of T, we construct a graph H_i where $V(H_i) = N_i$ and there is an edge (u,v) between $u, v \in N_i$ if there is a corresponding edge in E_i whose endpoints are covered by u and v, respectively. Graph H_i is used to distinguish between *low degree points*, whose degree in H_i is $O(\frac{1}{\epsilon})$, and *high degree points*, whose degree

[7] A subset of points $N \subseteq P$ is an r-cover if every point in P is within distance r from some point in N.

in H_i is $\Omega(\frac{1}{\epsilon})$. T will have two *charging properties*: (1) every point $p \in N_i$ has at least ϵL_i *uncharged descendants* and (2) at every level i, one can charge up to $\frac{\epsilon L_i}{2}$ uncharged descendants of high degree points. Note that, once an uncharged point is *charged* at level i, it will be marked as charged at higher levels; initially at level 0, every point is *uncharged*.

We then use a charging cover tree T to guide the Steiner spanner construction. Specifically, at level i, we add all edges incident to low degree points to the spanner and we can show that the total weight of all these edges over all levels is at most $O(\frac{w(\mathsf{MST})\log\Delta}{\epsilon^2})$. For high degree points, we apply the construction of Le and Solomon [59] to obtain a Steiner spanner S_i, and we charge the weight of S_i to $\frac{L_i\epsilon}{2}$ uncharged descendants of each high degree point. This charging is possible by the charging property (2) of T. We then show that each point in $K \cup P$ is charged a weight at most $\tilde{O}(\epsilon^{-(d+1)/2})$. Thus, the total weight of the Steiner spanners (for high degree points) at all levels is $\tilde{O}(\epsilon^{-(d+1)/2})w(\mathsf{MST})$.

Our construction of a charging cover tree is inspired by the construction of a hierarchy of clusters in the iterative clustering technique. The technique was initially developed by Chechick and Wulff-Nilsen [25] to construct light spanners for general graphs, and then was adapted to many other different settings [25, 12, 13, 59, 58]. Our construction is directly inspired by the construction of Borradaile et al. [13] in the doubling dimension setting. However, our construction is much simpler. Specifically, we are able to decouple the Steiner spanner construction from the charging cover tree construction. We refer readers to Section 5 for more details.

2 Preliminaries

Let P be a point set of n points in $\mathbb{R}^d$. We denote by $||p,q||$ the Euclidean distance between two points $p, q \in \mathbb{R}^d$. Let $B(p,r) = \{x \in \mathbb{R}^d, ||p,x|| \leq r\}$ be the ball of radius r centered at p. Given a point p and a set of point Q on the plane, we define the distance between p and Q, denoted by $d(p,Q)$, to be $\inf_{x\in Q} ||p,x||$.

An *r-cover* of P is a subset of points $N \subseteq P$ such that for every point $x \in P$, there is at least one point $p \in N$ such that $||p,x|| \leq r$; we say x is *covered* by p. When the value of r is clear from the context, we simply call N a cover of P. A subset of point $N \subseteq P$ is called an *r-net* if N is an r-cover of P and also an *r-packing* of P, i.e., for every two points $p \neq q \in N$, $||p,q|| > r$.

Let G be a graph with weight function w on the edges. We denote the vertex set and edge set of G by $V(G)$ and $E(G)$, respectively. Let $d_G(p,q)$ be the distance between two vertices p, q of G. We denote by $G[X]$ the subgraph induced by a subset of vertices X.

G is *geometric* in $\mathbb{R}^d$ if each vertex of G corresponds to a point $p \in \mathbb{R}^d$ and for every edge (p,q), $w(p,q) = ||p,q||$. In this case, we use points to refer to vertices of G. We say that a geometric graph G is a $(1+\epsilon)$-*spanner* of P if $V(G) = P$ and for every two points $p \neq q \in P$, $d_G(p,q) \leq (1+\epsilon)||p,q||$. We say that G is a Steiner $(1+\epsilon)$-*spanner* for P if $P \subseteq V(G)$ and for every two points $p \neq q \in P$, $d_G(p,q) \leq (1+\epsilon)||p,q||$. Points in $V(G) \setminus P$ are called *Steiner points*. Note that distances between Steiner points may not be preserved in a Steiner $(1+\epsilon)$-spanner.

3 Steiner Spanners on the Plane

We focus on constructing a Steiner spanner with good lightness; the fast construction is in Section 4. We will use the following geometric Steiner *shallow-light tree* (SLT) construction by Solomon [68].

▶ **Lemma 4.** *Let L be a line segment of length $\sqrt{\epsilon}$ and p be a point on the plane such that $d(p, L) = 1$. For any point set $X \in L$, there is a geometric graph H of weight $\Theta(1)$ such that $d_H(p, x) \leq (1 + \epsilon)||p, x||$ for any point $x \in X$.*

We will use *single-source spanners* (defined below) as a black box in our construction.

▶ **Definition 5** (Single-source spanners). *Given a point p (source), a set of points X on the plane and a connected geometric graph S_X spanning X, a* single source $(1 + \epsilon)$-spanner *w.r.t. (p, X, S_X) is a graph H such that for every $x \in X$: $||p, x|| \leq d_{H \cup S_X}(p, x) \leq (1 + \epsilon)||p, x||$.*

Our starting point is the construction of a single source spanner from a point p to point set X enclosed in a circle C of radius $\sqrt{\epsilon}$ such that $d(p, C) = 1$. We show that, if S_X approximately preserves the distances between pairs of points in X up to a $(1 + g\epsilon)$ factor for any constant g, it is possible to construct a single-source spanner with weight $O(1)$. It is not so hard to see that if Steiner points are not allowed, a lower bound of weight $\Omega(\frac{1}{\sqrt{\epsilon}})$ holds here.

▶ **Lemma 6.** *Let X be a set of points in a circle C of radius $\sqrt{\epsilon}$ on the plane and a point p of distance 1 from C. Let S_X be a $(1 + g\epsilon)$-spanner of X for any constant g. Then there is a single-source $(1 + 13\epsilon)$-spanner H w.r.t. (p, X, S_X) of weight $O(1)$ when $g \ll \frac{1}{\epsilon}$.*

Proof. Let c be a center of C. W.l.o.g, we assume that pc is parallel to y-axis. Let Q be the axis-aligned smallest square bounding C. Observe that the side length of Q is at mos $2\sqrt{\epsilon}$. Place a $\frac{2}{\sqrt{\epsilon}} \times \frac{2}{\sqrt{\epsilon}}$ grid W on Q, so that every cell of W is a square of side length ϵ. Observe that:

$$w(W) \leq \epsilon \cdot \frac{4}{\epsilon} = O(1)$$

We extend W to W_1 by connecting an (arbitrary) corner of each grid cell to an arbitrary point of X in the cell. Observe that: $w(W_1) = O(W) = O(1)$. Let P be the set of grid points on the side, say L, of Q that is closer to p (than the opposite side). We apply the construction in Lemma 4 to p and L to obtain a geometric graph K. Let $H = W_1 \cup K$. Since $w(K) = O(1)$ by Lemma 4, it holds that $w(H) = O(1)$.

It remains to show the stretch bound. Let x be any point of X and v be the point in the same cell with x that is connected to a corner, say z of grid W. We will show below that:

$$d_{W \cup K}(z, p) \leq (1 + 3\epsilon)||p, z|| \tag{1}$$

If Equation 1 holds, it would imply:

$$\begin{aligned} d_{S_X \cup H}(x, p) &\leq d_{S_X}(x, v) + d_H(v, p) \leq (1 + g\epsilon)\sqrt{2}\epsilon + d_H(v, p) \\ &\leq (1 + g\epsilon)\sqrt{2}\epsilon + d_H(z, v) + d_H(z, p) \overset{g \ll 1/\epsilon}{\leq} 2\sqrt{2}\epsilon + \sqrt{2}\epsilon + d_{W \cup K}(z, p) \\ &\overset{\text{Eq. 1}}{\leq} (1 + 3\epsilon)||p, z|| + 3\sqrt{2}\epsilon \overset{||x,z|| \leq \sqrt{2}\epsilon}{\leq} (1 + 3\epsilon)(||p, x|| + \sqrt{2}\epsilon) + 3\sqrt{2}\epsilon \\ &\leq (1 + 3\epsilon)||p, x|| + 7\sqrt{2}\epsilon \leq (1 + 13\epsilon)||p, x|| \qquad \text{since } ||p, x|| \geq 1 \end{aligned}$$

Thus, it remains to prove Equation 1. To this end, let y be the projection of z on L. (Point y is also a grid point; see Figure 1) Let y', z' be projections of y and z on the line containing pc, respectively. Let u be the intersection of pz and yy'. Observe that $||p, y|| \leq (1 + \epsilon) \leq (1 + \epsilon)||p, u||$. Thus,

$$||p, y|| + ||y, z|| \ \leq \ (1 + \epsilon)||p, u|| + ||y, z|| \leq (1 + \epsilon)||p, u|| + ||u, z|| \leq (1 + \epsilon)||p, z|| \tag{2}$$

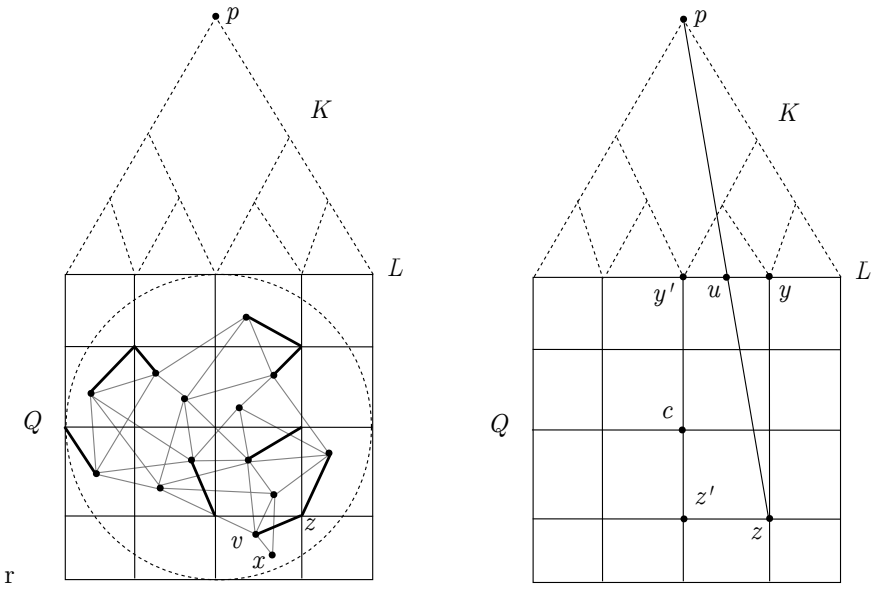


Figure 1 *(Left) A single source spanner from p to a set of points enclosed by a circle of radius $\sqrt{\epsilon}$. One point in each non-empty cell is connected to a corner by a thick edge. (Right) An illustration for analyzing the stretch of pz.*

Since $d_W(z,y) = ||z,y||$ and $d_K(y,p) = (1+\epsilon)||p,y||$ by Lemma 4, we have:

$$d_{W\cup K}(z,p) \;\leq\; (1+\epsilon)(||z,y|| + ||p,y||) \overset{Eq.\ 2}{\leq} (1+\epsilon)(1+\epsilon)||p,z|| \;\leq\; (1+3\epsilon)||p,z|| \quad (3)$$

which implies Equation 1. ◀

We obtain the following corollary of Lemma 6.

▶ **Corollary 7.** *Let X be a set of points in a circle C of radius $\sqrt{\epsilon}L$ on the plane and a point p of distance L/h from C for some constant $h \geq 1$. Let S_X be a $(1+g\epsilon)$-spanner of X for any constant g. Then there is a single-source $(1+13\epsilon)$-spanner H w.r.t. (p, X, S_X) of weight $O(hL)$ when $g \ll \frac{1}{\epsilon}$.*

Proof. We scale the space by L/h. In the scaled space, C has radius $h\sqrt{\epsilon}$ and $d(p,C) = 1$. Let $C_1, C_2, \ldots, C_m$, where $m = O(h^2)$, be circles of radius $\sqrt{\epsilon}$ covering C; such a set of circles can be constructed greedily. We apply Lemma 6 to p and each C_i to construct a single-source $(1+13\epsilon)$-spanner H_i from p to each C_i. The final spanner is $H = \cup_{i=1}^{m} H_i$ that has total weight $O(h^2)$ in the scaled metric. Thus, in the original metric, $w(H) = O(h^2 L/h) = O(hL)$. ◀

We are now ready to prove Theorem 1.

Proof of Theorem 1. Assume that the minimum pairwise distance is 1. Let $\mathcal{P} = \binom{P}{2}$ be all pairs of points. Partition $\mathcal{P}$ into $O(\log \Delta)$ sets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{\lceil \log \Delta \rceil}$ where $\mathcal{P}_i$ is the set of pairs (x,y) such that $||x,y|| \in [2^{i-1}, 2^i)$.

For a fixed i, we claim that there is a geometric graph H_i such that for every two distinct points $(x,y) \in \mathcal{P}_i$, $d_{H_1 \cup \ldots \cup H_i}(x,y) \leq (1+\epsilon)||x,y||$ and that $w(H_i) = O(\frac{1}{\epsilon})w(\mathsf{MST})$. Thus, $H_1 \cup \ldots \cup H_{\lceil \log \Delta \rceil}$ is a Steiner spanner with weight $O(\frac{\log \Delta}{\epsilon})w(\mathsf{MST})$.

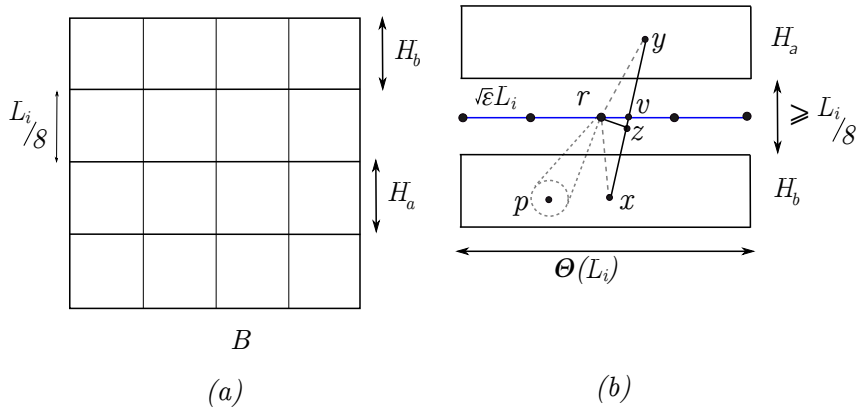


Figure 2 *(a) Square B is divided into $O(1)$ horizontal and vertical bands of width $L_i/8$ each. (b) The Steiner spanner construction for two non-adjacent horizontal bands H_a and H_b. The dashed cone represents a single-source spanner from r to circle $B(p, \sqrt{\epsilon}L_i)$.*

We now focus on constructing H_i. Let $S_{i-1} = H_1 \cup H_2 \ldots \cup H_{i-1}$. We will construct a spanner with stretch $(1 + c\epsilon)$ for some constant c. By induction, we can assume that:

$$d_{S_{i-1}}(p, q) \leq (1 + c\epsilon)||p, q|| \tag{4}$$

for any pair $(p, q) \in \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}$.

Let $L_i = 2^i$ and N_i be a $(\sqrt{\epsilon}L_i)$-net of P. For each point x, let $N_i(x)$ be the net point that *covers* x: the distance from x to $N_i(x)$ is at most $\sqrt{\epsilon}L_i$.

▷ Claim 8. $|N_i| = O(\frac{w(\mathsf{MST})}{L_i\sqrt{\epsilon}})$.

Proof. Consider the circle $B(p, \sqrt{\epsilon}L_i)$ centered at p; $B(p, \sqrt{\epsilon}L_i)$ contains a segment of length $\Omega(\sqrt{\epsilon}L_i)$ of the MST, which is not contained in any other circle. Thus, the claim holds. ◁

Next, we consider the smallest axis-aligned square Q bounding the point set. In the following, we divide Q into a set of (overlapping) sub-squares $\mathcal{B}$ of side length $\Theta(L_i)$ each. This way, for any pair $(x, y) \in \mathcal{P}_i$ (of distance at most L_i), there is a sub-square entirely containing $N_i(x), N_i(y)$, and the balls of radius $O(\sqrt{\epsilon}L_i)$ around the two net points.

Constructing $\mathcal{B}$. We first divide Q into subsquares of side length $5L_i$ each[8]. For each subsquare B, we extend its borders equally to four directions by an amount of $2L_i$ in each direction. After this extension, B has side length $9L_i$.

▷ Claim 9. Every point in Q belongs to at most 4 subsquares in $\mathcal{B}$. Furthermore, for each pair $(x, y) \in \mathcal{P}_i$, there is a subsquare $B \in \mathcal{B}$ such that $B(N_i(x), \sqrt{\epsilon}L_i), B(N_i(y), \sqrt{\epsilon}L_i)$ are entirely contained in B.

Proof. Let B be a subsquare in $\mathcal{B}$ containing one of the endpoints of (x, y), say x, before extension. Then, after extension, B will contain both x, y since $||x, y|| \leq L_i$, and furthermore, x and y are at least L_i away from the boundary since we extended B by $2L_i$ in each direction. Thus, points in $B(N_i(x), \sqrt{\epsilon}L_i)$ $(B(N_i(y), \sqrt{\epsilon}L_i))$ will be at most $2\sqrt{\epsilon}L_i < L_i$ from x (y) when $\epsilon \ll 1$. ◁

[8] We assume that the side length of Q is divisible by $5L_i$; otherwise, we can extend Q in such a way.

Consider a subsquare $B \in \mathcal{B}$. Let $N_B = N_i \cap B$. By abusing notation, we denote by $\mathcal{P}_i \cap B$ all the pairs in B of $\mathcal{P}_i$. We will show that:

▷ Claim 10. There is a Steiner spanner S_B of weight at most $O(|N_B|L_i/\sqrt{\epsilon})$ such that for any pair of points $(x, y) \in \mathcal{P}_i \cap B$, it holds that:

$$d_{S_B \cup S_{i-1}}(x, y) \leq (1 + c\epsilon)||x, y||$$

for some big enough constant c.

Proof. We divide B into $O(1)$ horizontal (vertical) *bands* of length (width) $L_i/8$ so that for any two points $x, y \in \mathcal{P}_i \cap B$, $N_i(x)$ and $N_i(y)$ are in two non-adjacent horizontal bands and/or vertical bands (see Figure 2). These bands exist since

$$||N_i(x), N_i(y)|| \ \geq \ ||x, y|| - 2\sqrt{\epsilon}L_i \ \geq \ L_i/2 - 2\sqrt{\epsilon}L_i \ \geq \ L_i/4$$

Now for each pair of non-adjacent horizontal bands H_a and H_b, $d(H_a, H_b) \geq \frac{L_i}{8}$. Draw a bisecting segment (touching two sides of B) between H_a and H_b and place $O(\frac{1}{\sqrt{\epsilon}})$ equally-spaced Steiner points, say R, on the bisecting line in a way that the distance between any two nearby Steiner points is $L_i\sqrt{\epsilon}$ (see Figure 2(a)). For each point $r \in R$, and each net point $p_i \in N_B \cap (H_a \cup H_b)$, we apply the construction of Corollary 7 to r, the set of endpoints of $\mathcal{P}_i$ inclosed in circle $B(p_i, \sqrt{\epsilon}L_i)$ and S_{i-1}; let $S_{a,b}(p_i)$ be the obtained geometric graph (see Figure 2(b)). Note that $d(r, B(p_i, \sqrt{\epsilon}L_i)) = \Omega(L_i)$. Thus, by Corollary 7, $w(S_{a,b}(p_i)) = O(L_i)$ and that:

$$d_{S_{i-1} \cup S_{a,b}(p_i)}(r, q) \leq (1 + 13\epsilon)||r, q|| \tag{5}$$

for any $q \in B(p_i, \sqrt{\epsilon}L_i) \cap P$. Let $S_{a,b}(r) = \cup_{p_i \in N_B \cap (H_a \cup H_b)} S_{a,b}(p_i)$. It holds that

$$w(S_{a,b}(r)) \leq O(L_i)|N_B \cap (H_a \cup H_b)|.$$

Let $S_{a,b} = \cup_{r \in R} S_{a,b}(r)$. Then, we have:

$$w(S_{a,b}) \leq O(L_i \cdot |R| \cdot |N_B \cap (H_a \cup H_b)|) = O(\frac{L_i}{\sqrt{\epsilon}}|N_B \cap (H_a \cup H_b)|) \tag{6}$$

We apply the same construction for every pair of non-adjacent vertical bands. We then let S_B be the union of all $S_{a,b}$ for every pair of non-adjacent horizontal/vertical bands H_a, H_b. It holds that:

$$w(S_B) = O(L_i \cdot |R| \cdot |N_B|) = O(\frac{L_i|N_B|}{\sqrt{\epsilon}}) \tag{7}$$

since there are only $O(1)$ pairs of bands. To bound the stretch, let (x, y) be a pair in $\mathcal{P}_i$ whose endpoints are in B. W.l.o.g, assume that H_a, H_b are two non-adjacent horizontal bands that contain $N_i(x)$ and $N_i(y)$, respectively. Let v be the intersection of segment xy and the bisecting line L of H_a, H_b (see Figure 2(b)). Let $r \in R$ be the closest Steiner point to v in L and z be the projection of r on xy. Observe that $||z, x||, ||z, y|| \geq \frac{L_i}{16} - ||z, v|| \geq L_i/16 - \sqrt{\epsilon}L_i \geq L_i/32$ when $\epsilon \ll 1$. We have:

$$\begin{aligned}
||r, x|| + ||r, y|| &= \sqrt{||x, z||^2 + ||r, z||^2} + \sqrt{||y, z||^2 + ||r, z||^2} \\
&\leq ||x, z||\sqrt{1 + \frac{\epsilon L_i^2}{||x, z||^2}} + ||y, z||\sqrt{1 + \frac{\epsilon L_i^2}{||y, z||^2}} \qquad \text{since } ||r, z|| \leq \sqrt{\epsilon}L_i \\
&\leq ||x, z||\sqrt{1 + 1024\epsilon} + ||y, z||\sqrt{1 + 1024\epsilon} \qquad \text{since } ||x, z||, ||y, z|| \geq L_i/32 \\
&\leq ||x, z||(1 + 512\epsilon) + ||y, z||(1 + 512\epsilon) = (1 + 512\epsilon)||x, y||
\end{aligned}$$

Thus, by Equation 5, we have:

$$d_{S_B \cup S_{i-1}}(x,y) \leq (1+13\epsilon)(||r,x||+||r,y||) = (1+13\epsilon)(1+512\epsilon)||x,y|| = (1+O(\epsilon))||x,y||$$

Thus, the stretch is $(1+c\epsilon)$ for a sufficiently big constant c. ◁

Let $H_i = \cup_{B\in\mathcal{B}} S_B$. Since each net point belongs to at most 4 subsquares in $\mathcal{B}$ by Claim 9, and $w(H_i) \leq O(\frac{|N_i|L_i}{\sqrt{\epsilon}})$ by Claim 10, it holds that:

$$w(H_i) = O(\frac{L_i}{\sqrt{\epsilon}}\frac{w(\mathsf{MST})}{L_i\sqrt{\epsilon}}) = O(\frac{w(\mathsf{MST})}{\epsilon}) \tag{8}$$

by Claim 8 as desired. ◀

4 A Linear Time Construction

In this section, we assume that $\Delta = O(n^c)$ for some constant c, and ϵ is a constant. We use the same model of computation used by Chan [20]: the real-RAM model with $\Theta(\log n)$ word size and floor function. We will use O_ϵ notation to hide a polynomial factor of $\frac{1}{\epsilon}$. Chan [20] showed that:

▶ **Theorem 11** (Step 4 in [20]). *Given a poin set $P \in \mathbb{R}^d$ with spread $\Delta = O(n^c)$ for* constant *d and c, a $(1+\epsilon)$-spanner of P can be constructed in $O_\epsilon(P)$ time.*

We will use a construction of an r-net for a point set P for any r in time $O(n)$. Such a construction was implicit in the work of Har-Peled [48] which was made explicit by Har-Peled and Raichel (Lemma 2.3 [50]).

▶ **Lemma 12** (Lemma 2.3 and Corollary 2.4 [50]). *Given $r \geq 1$ and an n-point set P in $\mathbb{R}^d$, an r-net N of P can be constructed in $O(n)$ time. Furthermore, for each net point p, one can compute all the points covered by p in total $O(n)$ time.*

We first show that the spanner in Corollary 7 can be implemented in $O_\epsilon(|X|)$ time.

▷ Claim 13. The single-source spanner H in Corollary 7 can be found in $O_\epsilon(|X|)$ time.

Proof. First, we observe that the single source spanner in Lemma 6 can be constructed in time $O_\epsilon(|X|)$. This is because the grid W has size $O_\epsilon(1)$ and the SLT tree from p to (a set of $O(\frac{1}{\sqrt{\epsilon}})$ grid points on) L can be constructed in $O_\epsilon(1)$ time. The single-source spanner in Corollary 7 uses a constant number of constructions construction in Lemma 6. Thus, the total running time is $O_\epsilon(|X|)$. ◁

Proof of Theorem 2. Our implementation will follow the construction in Section 3 ; we will reuse notation in that section as well. Let K be a $(1+\epsilon)$-spanner H for P constructed in time $O(n)$ by Theorem 11. In our fast construction algorithm, instead of considering all pairs of points $\mathcal{P} = \binom{P}{2}$, we only consider the pairs corresponding to edges of H; there are $O(n)$ such pairs. Our algorithm has four steps:

- **Step 1.** Partition pairs of endpoints in $E(H)$ into at most $O(\log(\Delta))$ sets $\mathcal{P}_1, \ldots, \mathcal{P}_{\lceil \log \Delta \rceil}$ where $\mathcal{P}_i$ is the set of pairs (x,y) such that $||x,y|| \in [2^{i-1}, 2^i)$. Let $L_i = 2^i$. This step can be implemented in time $O(|E(H)|) = O_\epsilon(n)$. The following steps are applied to each $i \in [1, \lceil \log \Delta \rceil]$. Let P_i be the set of endpoints of $\mathcal{P}_i$. We observe that:

$$\sum_{i=1}^{n} |P_i| \;\leq\; 2\sum_{i=1}^{n} |\mathcal{P}_i| \;=\; 2|E(H)| \;=\; O(n) \tag{9}$$

- **Step 2.** Construct a $(\sqrt{\epsilon}L_i)$-net N_i for P_i in $O(|P_i|)$ time using the algorithm in Lemma 12.
- **Step 3.** Compute a bounding square Q and divide it into (overlapping) subsquares of length $\Theta(L_i)$ each. Let $\mathcal{B}$ be the set of subsquares that contain at least one point participating in the pairs $\mathcal{P}_i$. Since there are only $O(|P_i|)$ non-empty subsquares, $\mathcal{B}$ can be computed in time $O(|P_i|)$ by iterating over each point and check (in $O(1)$ time) which subsquare the point falls into. Here we use the fact that each floor operation takes $O(1)$ time.
- **Step 4.** For each subsquare $B \in \mathcal{B}$, we divide it into $O(1)$ horizontal bands and vertical bands of length $L_i/8$. For each pair of non-adjacent (horizontal) bands H_a, H_b, construct a set of $O(\frac{1}{\sqrt{\epsilon}})$ Steiner points on the bisecting line between H_a and H_b as in Section 3. For each Steiner point r and each net point $p \in N_i \cap (H_a \cap H_b)$, we apply Corollary 7 to construct a single source spanner from r to a set of points $B(p, \sqrt{\epsilon}L_i) \cap P_i$; this step can be implemented in time $O_\epsilon(|B(p, \sqrt{\epsilon}L_i) \cap P_i|)$ by Claim 13. By Claim 9, the construction in this step can be implemented in $O_\epsilon(|P_i|)$ time. Our final spanner is the union of all single source spanners in all subsquares in $\mathcal{B}$.

The running time needed to implement Steps 2 to 4 is $O_\epsilon(\sum_{i=1} |P_i|) = O_\epsilon(n)$ by Equation 9. The same analysis in Section 3 gives $O(\frac{\log \Delta}{\epsilon})$ lightness. For stretch, we observe that the stretch of the spanner for each edge of H is $(1 + O(\epsilon))$. Thus, the stretch for every pair of points in P is $(1 + \epsilon)(1 + O(\epsilon)) = (1 + O(\epsilon))$. We can recover stretch $(1 + \epsilon')$ by setting $\epsilon' = \frac{\epsilon}{c}$ where c is the constant behind big-O. ◀

5 Steiner Spanners in High Dimension

In this section, we a light Steiner spanner for a point set $P \in \mathbb{R}^d$ with spread Δ as in Theorem 3. We rescale the metric so that every edge in $\binom{P}{2}$ has weight at least $\frac{1}{\epsilon}$. Let MST be the minimum spanning tree of P. We subdivide each MST edge of length > 1, by placing Steiner points greedily, in a way that each new edge has length *at least* $1/2$ *and at most* 1. Let K be the set of Steiner points. We observe that:

$$w(\mathsf{MST}) = \Theta(|P| + |K|) \tag{10}$$

Let $\delta > 1$ be some parameter and $L_i = \frac{\delta}{\epsilon^i}$. Let $\mathcal{E}_\delta = \{E_1, \ldots, E_m\}$ be the set of edges such that

$$E_i = \{e | e \in \binom{P}{2} \wedge w(e) \in (L_i/2, L_i]\} \tag{11}$$

where $m = \lceil \log_{\frac{1}{\epsilon}}(\Delta/(\epsilon\delta)) \rceil \leq \lceil \log_{\frac{1}{\epsilon}} \Delta \rceil + 1$. If an edge $e \in E_i$ for some $E_i \in \mathcal{E}_\delta$, we will abuse notation by saying that $e \in \mathcal{E}_\delta$. The main focus of this section is to show that:

▶ **Lemma 14.** *There is a Steiner spanner that preserves distances between the endpoints of edges in $\mathcal{E}_\delta$ with weight $\tilde{O}(\epsilon^{-(d+1)/2} + (\delta + \epsilon^{-2}) \log_{\frac{1}{\epsilon}} \Delta) w(\mathsf{MST})$.*

We will show below that Lemma 14 implies Theorem 3.

Proof of Theorem 3. We assume that $\frac{1}{\epsilon}$ is a power of 2. We partition the interval $[1, \epsilon)$ into $J = \log_2(\frac{1}{\epsilon})$ intervals $[1, 2), \ldots, [2^{J-1}, 2^J)$. For each fixed $j \in [1, J]$, let $\delta_i = 2^j$, and $\mathcal{E}_{\delta_j}$ be the set of edges with $\delta = \delta_j$ in the definition of $\mathcal{E}_\delta$. Recall that we scale the metric so that every edge in $\binom{P}{2}$ has weight at least $\frac{1}{\epsilon}$. Thus, $\binom{P}{2} = \cup_{j=1}^{J} \mathcal{E}_{\delta_j}$

Observe that $\delta_j \leq \frac{1}{\epsilon}$ for all $j \in [1, J]$. By Lemma 14, there exists a Steiner spanner S_j with weight $\tilde{O}(\epsilon^{-(d+1)/2} + (\epsilon^{-2} \log_{\frac{1}{\epsilon}} \Delta) w(\mathsf{MST})$ preseving distances between endpoints of edges in $\mathcal{E}_{\delta_j}$ up to a $(1 + \epsilon)$ factor. Then $S = \cup_{i=1}^{J} S_j$ is a Steiner $(1 + \epsilon)$-spanner with weight $\tilde{O}(\epsilon^{-(d+1)/2} + \epsilon^{-2} \log \Delta) w(\mathsf{MST})$. ◀

We now focus on constructing a Steiner spanner in Lemma 14. We will use the following Steiner spanner construction as a black box.

▶ **Theorem 15** (Theorem 1.3 [59]). *For a given point set P, there is a Steiner $(1+\epsilon)$-spanner, denoted by $\mathsf{STP}(P)$, with $\tilde{O}(\epsilon^{-(d-1)/2})|P|$ edges that preserves pairwise distances of points in P up to a $(1+\epsilon)$ factor.*

We will rely on a *cover tree* to construct a Steiner spanner. Let c be a sufficiently big constant chosen later ($c = 20$).

▶ **Definition 16** (Cover tree). *A cover T for point set $P \cup K$ with $(m+1)$ levels has each node associated with a point of P such that (a) level-0 of T is the point set $P \cup K$, (b) level-i of T is associated with a $(c\epsilon L_i)$-cover N_i of P and (c) $N_m \subseteq N_{m-1} \subseteq \ldots \subseteq N_0$.*

A point p may appear in many levels of a cover tree T. To avoid confusion, we denote by (p,i) the copy of p at level i, and we still call (p,i) a *point* of $P \cup K$. For each point (p,i), we denote by $\mathsf{child}(p,i)$ and $\mathsf{desc}(p,i)$ the set of children and descendants of (p,i) in T, respectively. Note that $\mathsf{desc}(p,i)$ includes (p,i).

We will construct the Steiner spanner level by level, starting from level 1. At every level, we will add a certain set of edges to E_{sp}. We then charge the weight of a subset of the edges to a subset of *uncharged points* of $P \cup K$; initially, every point of $P \cup K$ is *uncharged*. To decide which uncharged points we will charge to at level i, we consider a geometric graph H_i where $V(H_i) = N_i$ and there is an edge between two points $(p,i) \neq (q,i)$ (of weight $||p,q||$) in H_i if there exists at least one $e \in E_i$ between two descendants of (p,i) and (q,i), respectively. We say a cover point (p,i) has *high degree* if its degree in H_i is at least $\frac{4c}{\epsilon}$. At level i, we only charge to uncharged points which are descendants of high degree cover points. The intuition is that high cover points have many descendants. This leads to a notion of a *charging cover tree*. We call a cover tree a charging cover tree for $\mathcal{E}_\delta$ if for all level $i \geq 1$,

(1) Each point (p,i) has at least ϵL_i descendants that are uncharged at level less than i.

(2) Up to $\epsilon L_i/2$ uncharged descendant of each high-degree cover point (p,i) can be charged at level i. No descendant of low-degree points is charged at level i.

We show how to construct a charging cover tree in Appendix 5.4. We now show that given a charging cover tree, we can construct a Steiner spanner with the lightness bound in Lemma 14. Let T be such a charging cover tree. We define a set of edges E_T as follows:

$$E_T = \{(p,q) | (p \in \mathsf{child}(q,i) \vee q \in \mathsf{child}(p,i)) \text{ for some } i\} \tag{12}$$

We abuse notation by denoting E_T the graph induced by the set of edges in E_T.

▷ Claim 17. $w(E_T) = O(\delta + \epsilon^{-1} \log_{\frac{1}{\epsilon}} \Delta) w(\mathsf{MST}))$ and for any p and every $x \neq y \in \mathsf{desc}(p,i)$, $d_{E_T}(x,y) \leq 4c\epsilon L_i$.

Proof. Edges in E_T can be partitioned according to levels where an edge is at level i if it connects a point (p,i) and its parent. Observe that at level 0, the total edge weight is at most $c\delta$ times the number of points and hence, the total weight is $O(\delta|K \cup P|) = O(\delta w(\mathsf{MST}))$ by Equation 10. At higher level, we observe that the total weight of edges of E_T at level $i \geq 1$ is at most $L_i|N_i|$, and that $N_i \leq \frac{|K \cup P|}{|\epsilon L_i|}$ since each point (p,i) has $|\mathsf{desc}(p,i)| \geq \epsilon L_i$ by property (1) of the charging tree. Thus, the total weight of edges E_t at level at least 1 is at most:

$$\sum_{i=1}^{m} \frac{|K \cup P|}{\epsilon} = m \frac{|K \cup P|}{\epsilon} = O(\epsilon^{-1} \log_{\frac{1}{\epsilon}} \Delta |K \cup P|) = O(\epsilon^{-1} \log_{\frac{1}{\epsilon}} \Delta) w(\mathsf{MST})$$

This implies the weight bound of E_T. We now bound the distance between $x \neq y \in \mathsf{desc}(p,i)$. Let $x = v_0, v_1, \ldots, v_k = p$ be the (unique) path from x to p. By construction, $||v_{i-1}, v_i|| \leq \frac{v_i v_{i+1}}{\epsilon}$ for $i \in [1, k-1]$. This implies:

$$d_{E_T}(x,p) = ||p, v_{k-1}|| \sum_{i=0}^{k-1} \epsilon^i \leq \frac{||p, v_{k-1}||}{1-\epsilon} \leq 2||p, v_{k-1}|| = 2c\epsilon L_i$$

when $\epsilon \leq \frac{1}{2}$. Similarly, $d_{E_T}(y,p) \leq 2c\epsilon L_i$ and hence $d_{E_T}(x,y) \leq 4c\epsilon L_i$. ◁

Claim 17 implies that the descendants of any level i node in a charging cover tree form a subgraph of diameter at most $O(c\epsilon L_i)$. Let E_{sp} be the set of edges that will be our final spanner. Initially, $E_{sp} = E_T \cup \mathsf{MST}$. We will abuse notation by denoting E_{sp} the graph induced by edge set E_{sp}.

5.1 Spanner construction at level i

Recall that H_i is a geometric graph where $V(H_i) = N_i$ and there is an edge between two points $(p,i) \neq (q,i)$ in H_i if there exists at least one $e \in E_i$ between two descendants of (p,i) and (q,i), respectively. Recall that a high degree point (p,i) has at least $\frac{4c}{\epsilon}$ neighbors in H_i. We proceed in two steps.

- **Step 1.** For every low degee point (p,i), we add all incident edges of (p,i) in H_i to E_{sp}.
- **Step 2.** Let Q be the set of high degree points in N_i. We add to E_{sp} the set of edges of $\mathsf{STP}(Q)$. We take from each high degree cover point (p,i) exactly $\epsilon L_i/2$ uncharged descendants and let X be the set of these uncharged points. We charge the cost of $\mathsf{STP}(Q)$ equally to all points in X and mark them *charged*. This charging is possible by property (2) of T.

5.2 Bounding the stretch

We will show that the stretch is $(1 + (48c+1)\epsilon)$. We can recover stretch $(1+\epsilon')$ by setting $\epsilon' = \frac{\epsilon}{48c+1}$.

Observe by construction that for every edge $e \in H_i$, the stretch of e in E_{sp} is at most $(1+\epsilon)$. Recall that for every edge $(u,v) \in E_i$, there is an edge $(p,q) \in E(H_i)$ such that $u \in \mathsf{desc}(p,i), v \in \mathsf{desc}(q,i)$. By Claim 17, there is a path between u and v in E_{sp} of length at most $d_{E_{sp}}(p,q) + 8c\epsilon L_i$. By triangle inequality, $||p,q|| - 2c\epsilon L_i \leq ||u,v|| \leq ||p,q|| + 2c\epsilon L_i$. Thus, we have:

$$\begin{aligned}
\frac{d_{E_{sp}}(u,v)}{||u,v||} &\leq \frac{d_{E_{sp}}(p,q) + 8c\epsilon L_2}{||p,q|| - 2c\epsilon L_i} = \frac{d_{E_{sp}}(p,q)}{||p,q||} + \frac{(d_{E_{sp}}(p,q) + 4||p,q||)2c\epsilon L_i}{||p,q||(||p,q|| - 2c\epsilon L_i)} \\
&\leq \frac{d_{E_{sp}}(p,q)}{||p,q||} + \frac{12c\epsilon L_i}{||p,q|| - 2c\epsilon L_i} \qquad \text{since } d_{E_{sp}}(p,q) \leq 2||p,q|| \\
&\leq \frac{d_{E_{sp}}(p,q)}{||p,q||} + 48c\epsilon \leq 1 + (48c+1)\epsilon
\end{aligned}$$

The penultimate inequality follows from the fact that

$$||p,q|| - 2c\epsilon L_i \;\geq\; ||u,v|| - 4c\epsilon L_i \;\geq\; L_i/2 - 4c\epsilon L_i \;\geq\; L_i/4$$

when $\epsilon \ll \frac{1}{c}$.

5.3 Bounding $w(E_{sp})$

Observe that the total number of low degree points in N_i is at most $\frac{|K \cup P|}{\epsilon L_i}$ since each low degree point has at least ϵL_i (uncharged) descendants by property (1) of charging tree T. By triangle inequality, each edge of H_i has weight at most $L_i + 2c\epsilon L_i \leq 3L_i$ when $\epsilon < \frac{1}{c}$. Thus, the total weight of edges added to E_{sp} in Step 1 is bounded by:

$$\frac{3L_i \cdot 4c}{\epsilon} |\{p | p \in N_i \text{ and } p \text{ has low degree}\}| = O(\frac{L_i}{\epsilon} \frac{|K \cup P|}{\epsilon L_i}) = O(\frac{1}{\epsilon^2}) w(\mathsf{MST})$$

by Equation 10. Thus, the total weight of the edges added to E_{sp} in Step 1 over m levels is $O(\epsilon^{-2} \log_{\frac{1}{\epsilon}} \Delta) w(\mathsf{MST})$. Note here that $m \leq \lceil \log_{\frac{1}{\epsilon}} \Delta \rceil + 1$.

We now bound the total weight of the edges added to E_{sp} in Step 2 *over m levels*. Since each edge of H_i has weight at most $3L_i$, each edge of $\mathsf{STP}(Q)$ has weight a most $(1+\epsilon)3L_i \leq 6L_i$. By Theorem 15,

$$w(\mathsf{STP}(Q)) = \tilde{O}(\epsilon^{-(d-1)/2})|Q|6L_i = \tilde{O}(\epsilon^{-(d-1)/2+o(1)})|Q|L_i$$

Thus, in Step 2, each uncharged point is charged at most:

$$\frac{\tilde{O}(\epsilon^{-(d-1)/2})|Q|L_i}{|Q|\epsilon L_i/2} = \tilde{O}(\epsilon^{-(d+1)/2}) \tag{13}$$

This implies the total weight of the edges added to E_{sp} in Step 2 *over all levels* is $\tilde{O}(\epsilon^{-(d+1)/2})(|P \cup K|) = \tilde{O}(\epsilon^{-(d+1)/2}) w(\mathsf{MST})$. Together with Claim 17, we conclude that:

$$w(E_{sp}) = \tilde{O}(\epsilon^{-(d+1)/2} + (\delta + \epsilon^{-2}) \log_{\frac{1}{\epsilon}} \Delta) w(\mathsf{MST})$$

This completes the proof of Lemma 14.

5.4 Constructing a Charging Cover Tree

The main difficulty is to guarantee property (1) in constructing a charging cover tree; for property (2) at each level i, we simply charge to exactly $\epsilon L_i/2$ uncharged descendants of each high degree cover point.

A natural idea is to guarantee that each cover point has $\frac{1}{\epsilon}$ children. Then inductively, if each child of a cover point (p, i) has at least $\epsilon L_{i-1}/2$ uncharged descendants after the charging at level $i-1$, we can hope that p has at least $\frac{1}{\epsilon}\epsilon L_{i-1} = \epsilon L_i$ uncharged descendants. There are two issues with this idea: (a) if at least one child, say $(q, i-1)$, of (p, i) has high degree in the graph H_{i-1}, up to $\epsilon L_{i-1}/2$ uncharged points in $\mathsf{desc}(q, i-1)$ were charged at level $i-1$ by property (2), and thus, $(q, i-1)$ only contributes $\frac{\epsilon L_{i-1}}{2}$ uncharged descendants to (p, i); and (b) there may not be enough cover points at level $i-1$ close to p as these points and their descendants must be within distance $c\epsilon L_i$ from p.

In our construction, we resolve both issues by picking a cover point in a way that the total number of uncharged descendants of its children is at least ϵL_i. We do so by having a more accurate way to track the number of uncharged descendants of a cover point, instead of simply relying on the lower bound ϵL_i of uncharged descendants. Specifically, denote by $\mathsf{D}(X)$ the diameter of a point set X. We will construct a charging cover tree in a way that the following invariant is maintained *at all levels*.

Strong Charging Invariant: (SCI)
Each point (p, i) has at least $\max(\epsilon L_i, \mathsf{D}(\mathsf{desc}(p, i)) + 1)$ uncharged descendants (before the charging happened at level i).

Clearly, SCI implies property (1) of T. We begin by constructing the level-1 cover. Recall that MST edges have weight at most 1 and at least $\frac{1}{2}$, and that $\delta \geq 1$.

Level 1

We construct level-1 cover points N_1 by greedily breaking MST edges into subtrees of diameter at least δ and at most $3\delta + 2 \leq 5\delta$. Let X be such a subtree of MST with diameter d_X; X will have at least δ points since MST edge has weight at most 1. We pick any point, say $p \in X$, to be a level-1 cover point, and make other points in X become p's children; p will have at least δ children (uncharged at level 0). Recall that $\epsilon L_1 = \delta$. Since each MST edge has weight at most 1, the number of descendants of $(p, 1)$ is at least:

$$d_X + 1 \geq \max(\epsilon L_1, \mathsf{D}(\mathsf{desc}(p, 1)) + 1)$$

Thus, SCI holds for this level.

Level $i + 1$

Recall H_i is a graph with vertex set N_i. We construct a cover N_{i+1} in three steps A, B and C.

- **Step A.** For each high degree point p (with at least $\frac{4c}{\epsilon}$ unmarked neighbors in H_i), we pick p to N_{i+1} andmake its unmarked neighbors become its children. We then mark p and all of its neighbors. For each remaining unmarked high degree point x in H_i, at least one of its neighbors, say q, must be marked before. We make x become a child of q's parent.

The intuition of the construction in Step A is that $(p, i+1)$ picked at this step has at least $\frac{4c}{\epsilon}$ children. Since at least $(\epsilon L_i)/2$ descendants of each child of (p, i) remain uncharged after level i, the total number of uncharged descendants of $(p, i+1)$ is $\frac{4c}{\epsilon}\frac{\epsilon L_i}{2} = 2cL_i = 2c\epsilon L_{i+1} > \epsilon L_{i+1}$. Furthermore, since every high degree point is marked in this step, points in subsequent steps have low degree and hence no uncharged descendant of these points is charged at level i by property (2) of charging cover trees.

Let W be the set of remaining points in N_i. We construct a forest F from W as follows. The vertex set of F is W, and there is an edge between two vertices p, q of F if there is an MST edge, called the source of the edge, connecting a point in $\mathsf{desc}(p)$ and a point in $\mathsf{desc}(q)$. We set the weight of each edge in F to be the weight of the source edge. Note that F is not a geometric graph. We observe that:

▶ **Observation 18.** *For every connected component $C \in F$, there must be a point $p \in C$ and a point q marked in Step A such that there is an MST edge between a descendant of (p, i) and a descendant of (p, i), except when there is no point marked in Step A (and F is a tree in this case).*

Proof. The observation follows from the fact that MST spans $P \cup K$. ◀

We define the weight on each vertex p of F to be $w(p) = D(\mathsf{desc}(p, i))$, and the *vertex-weight* of a path P, denoted by $\mathsf{vw}(P)$, of F to be the total weight of vertices on the path. We define the *absolute weight* of P, denoted by $\mathsf{aw}(P)$, to be the total vertex and edge weight of P. Since each MST edge has weight at most 1 and each vertex has weight at least 1,

$$\mathsf{aw}(P) \leq 2\mathsf{vw}(P) \tag{14}$$

The *vertex-diameter* of a subtree C, denoted by $\mathsf{VD}(C)$, of F is defined to be the vertex-weight of the path of maximum vertex-weight in the subtree. The *absolute diameter* of C, denoted by $\mathsf{AD}(C)$, is defined similarly but w.r.t absolute weight.

- **Step B.** For each component C of F of vertex-diameter at least L_i, we greedily break C into sub-trees of vertex-diameter at least L_i and at most $3L_i$. For each subtree of C, choose an arbitrary point p to be a level-$(i+1)$ cover point and make other points become p's children.
- **Step C.** For each component C of F of vertex-diameter at most L_i, by Observation 18, there must be at least one MST connecting a point in $\mathsf{desc}(u,i)$ for some $u \in C$ to a point in $\mathsf{desc}(v,i)$ for some point v marked in Step 1. We make all points in C become children of v's parent.

The following claim implies that N_{i+1} is a $(c\epsilon L_{i+1})$-cover.

▷ Claim 19. For each point $(p, i+1)$, $\mathsf{D}(\mathsf{desc}(p,i+1)) \leq c\epsilon L_{i+1}$ for $c = 20$.

Proof. Note that for each point (q,i), $1 \leq \mathsf{D}(\mathsf{desc}(q,i)) \leq 2c\epsilon L_i$ since every point in $\mathsf{desc}(q,i)$ is within distance $c\epsilon L_i$ from q.

First, consider the case that $(p,i+1)$ is chosen in Step B. Then p and its children belong to a subtree C of F of vertex-diameter at most $3L_i$. by Equation 14, $\mathsf{AD}(C) \leq 2 \cdot 3L_i = 6L_i$. Thus, for a point $p \in N_{i+1}$ selected in Step B, $D(\mathsf{desc}(p,i+1)) \leq \mathsf{AD}(C) \leq 6L_i$.

We now consider the case where $(p,i+1)$ is chosen in Step A. (There is no cover point at level $(i+1)$ selected in Step C.) Recall that each edge of H_i has length at most $L_i + 2c\epsilon L_i$ by the triangle inequality. Observe that after Step 1, for any $(q,i) \in \mathsf{child}(p,i+1)$, the hop distance in H_i between (p,i) and (q,i) is at most 2, hence $||p,q|| \leq 2(L_i + 2c\epsilon L_i) = 2L_i + 4c\epsilon L_i$. That implies, after Step A,

$$||p,x|| \leq (2L_i + 4c\epsilon L_i) + c\epsilon L_i = 2L_i + 5c\epsilon L_i \quad \text{for any } x \in \mathsf{desc}(p,i+1) \tag{15}$$

In Step C, we add more points belonging to subtrees of F to $\mathsf{child}(p,i+1)$. Let C be any of these subtrees. Since $\mathsf{VD}(C) \leq L_i$, $\mathsf{AD}(C) \leq 2L_i$. By construction, there exists a point $(v,i) \in C$ and a point $(u,i) \in \mathsf{child}(p,i+1)$ such that there is an MST edge e connecting a point in $\mathsf{desc}(v,i)$ and a point in $\mathsf{desc}(u,i)$. Thus, the augmentation in Step C increases the distance from p to the furthest point of $\mathsf{desc}(p,i+1)$ by at most $w(e) + \mathsf{AD}(C) \leq 1 + 2L_i \leq 3L_i$. This implies:

$$\mathsf{D}(\mathsf{desc}(p,i+1)) \leq 2(2L_i + 5c\epsilon L_i + 3L_i) \leq 20L_i$$

when $\epsilon < \frac{1}{c}$. ◁

To complete the proof of Lemma 14, it remains to show SCI for level $(i+1)$.

▷ Claim 20. Each point $(p,i+1)$ has at least $\max(\epsilon L_{i+1}, \mathsf{D}(\mathsf{desc}(p,i+1)+1)$ uncharged descendants.

Proof. We first consider the case $(p,i+1)$ is picked in Step B. Let X be the subtree of F that p belongs to. Let P be a path of X of maximum absolute weight. By definition on absolute weight, $\mathsf{aw}(P) \geq \mathsf{D}(\mathsf{desc}(p,i+1))$. Since MST has length at most 1, we have:

$$\sum_{q \in P} \mathsf{D}(\mathsf{desc}(q,i)) + |E(P)| \geq \mathsf{aw}(P) \geq \mathsf{D}(\mathsf{desc}(p,i+1))$$

By SCI for level i, we conclude that the number of uncharged descendants of $(p,i+1)$ is at least:

$$\sum_{q \in P} (\mathsf{D}(\mathsf{desc}(q,i)) + 1) \geq (\sum_{q \in P} \mathsf{D}(\mathsf{desc}(q,i)) + |E(P)|) + 1 \geq \mathsf{D}(\mathsf{desc}(p,i+1))) + 1$$

To show that X has at least ϵL_{i+1} uncharged descendants, we observe by construction that X has a path Q with $\mathsf{vw}(Q) \geq L_i$. By definition of vertex-weight, $\mathsf{vw}(Q) = \sum_{q \in Q} \mathsf{D}(\mathsf{desc}(q, i))$. Thus, the total number of uncharged descendants of all $q \in Q$ by SCI is at least:

$$\sum_{q \in Q} (\mathsf{D}(\mathsf{desc}(q, i)) + 1) > \sum_{q \in Q} \mathsf{D}(\mathsf{desc}(q, i)) \geq L_i = \epsilon L_{i+1}$$

Thus, $(p, i+1)$ has at least $\max(\epsilon L_{i+1}, \mathsf{D}(\mathsf{desc}(p, i+1)) + 1)$ uncharged descendants.

It remains to consider the case $(p, i+1)$ is picked in Step A. By construction, $(p, i+1)$ has at least $\frac{4c}{\epsilon}$ children, and each has at least $\epsilon L_{i-1}/2$ uncharged descendants by property (2) of a charging cover tree. Note that $\mathsf{D}(\mathsf{desc}(p, i+1)) \leq c\epsilon L_{i+1} = cL_i$ by Claim 19. Thus, $(p, i+1)$ has at least:

$$\frac{4c}{\epsilon} \frac{\epsilon L_i}{2} = 2cL_i \geq \max(\epsilon L_{i+1}, \mathsf{D}(\mathsf{desc}(p, i+1))) + cL_i > \max(\epsilon L_{i+1}, \mathsf{D}(\mathsf{desc}(p, i+1)) + 1)$$

uncharged descendants as desired. ◁

References

1. M. A. Abam and S. Har-Peled. New constructions of sspds and their applications. In *Proceedings of the 26th Annual Symposium on Computational Geometry*, SoCG '10, page 192–200, 2010. doi:10.1145/1810959.1810993.
2. P. K. Agarwal, K. Fox, D. Panigrahi, K. R. Varadarajan, and A. Xiao. Faster algorithms for the geometric transportation problem. In *33rd International Symposium on Computational Geometry*, volume 77 of *SoCG '17*, pages 7:1–7:16, 2017. doi:10.4230/LIPIcs.SoCG.2017.7.
3. Stephen Alstrup, Søren Dahlgaard, Arnold Filtser, Morten Stöckel, and Christian Wulff-Nilsen. Constructing light spanners deterministically in near-linear time. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 4:1–4:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
4. I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete Computational Geometry*, 9(1):81–100, 1993. doi:10.1007/BF02189308.
5. B Aronov, M. Berg, O. Cheong, J. Gudmundsson, H. Haverkort, and A. Vigneron. Sparse geometric graphs with small dilation. In *International Symposium on Algorithms and Computation*, ISAAC '05, pages 50–59, 2005. doi:10.1007/11602613_7.
6. S. Arya and M. H. M. Smid. Efficient construction of a bounded degree spanner with low weight. *Algorithmica*, 17(1):33–54, 1997.
7. Sunil Arya, David M. Mount, and Michiel H. M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 703–712. IEEE Computer Society, 1994.
8. Gali Bar-On and Paz Carmi. \delta -greedy t-spanner. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, volume 10389 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2017.
9. J. Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55(4):299–327, 1959. doi:10.1017/s0305004100034095.
10. Marc Benkert, Alexander Wolff, Florian Widmann, and Takeshi Shirabe. The minimum manhattan network problem: Approximations and exact solutions. *Comput. Geom.*, 35(3):188–208, 2006.

11 G. Bodwin and V. V. Williams. Better distance preservers and additive spanners. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 855–872, 2016.

12 G. Borradaile, H. Le, and C. Wulff-Nilsen. Minor-free graphs have light spanners. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science*, FOCS '17, pages 767–778, 2017. doi:10.1109/FOCS.2017.76.

13 G. Borradaile, H. Le, and C. Wulff-Nilsen. Greedy spanners are optimal in doubling metrics. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 2371–2379, 2019. doi:10.1137/1.9781611975482.145.

14 Prosenjit Bose, Paz Carmi, Mathieu Couture, Michiel H. M. Smid, and Daming Xu. On a family of strong geometric spanners that admit local routing strategies. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Algorithms and Data Structures, 10th International Workshop, WADS 2007, Halifax, Canada, August 15-17, 2007, Proceedings*, volume 4619 of *Lecture Notes in Computer Science*, pages 300–311. Springer, 2007.

15 Prosenjit Bose, Luc Devroye, Maarten Löffler, Jack Snoeyink, and Vishal Verma. Almost all delaunay triangulations have stretch factor greater than pi/2. *Comput. Geom.*, 44(2):121–127, 2011.

16 K. Buchin. Delaunay triangulations in linear time? (part I). arXiv preprint, 2008. arXiv:arXiv:0812.0387.

17 K. Buchin and W. Mulzer. Linear-time delaunay triangulations simplified. In *25th European Workshop on Computational Geometry*, EuroCG '09, pages 235–238, 2009.

18 M. Bundefineddoiu, J. Chuzhoy, P. Indyk, and A. Sidiropoulos. Low-distortion embeddings of general metrics into the line. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, STOC '05, page 225–233, 2005. doi:10.1145/1060590.1060624.

19 T. Carpenter, F. V. Fomin, D. Lokshtanov, S. Saurabh, and A. Sidiropoulos. Algorithms for low-distortion embeddings into arbitrary 1-dimensional spaces. In *34th International Symposium on Computational Geometry*, SoCG '2018, pages 21:1–21:14, 2018. doi:10.4230/LIPIcs.SoCG.2018.21.

20 T. Chan. Well-separated pair decomposition in linear time? *Information Processing Letters*, 107(5):138–141, 2008. doi:10.1016/j.ipl.2008.02.008.

21 T.-H. H. Chan, M. Li, L. Ning, and S. Solomon. New doubling spanners: Better and simpler. *SIAM Journal on Computing*, 44(1):37–53, 2015. doi:10.1137/130930984.

22 T.-H. Hubert Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. *ACM Trans. Algorithms*, 12(4):55:1–55:22, 2016. Preliminary version appeared in SODA 2005.

23 B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, 1992.

24 Barun Chandra. Constructing sparse spanners for most graphs in higher dimensions. *Inf. Process. Lett.*, 51(6):289–294, 1994.

25 S. Chechik and C. Wulff-Nilsen. Near-optimal light spanners. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'16, pages 883–892, 2016.

26 Shiri Chechik and Christian Wulff-Nilsen. Near-optimal light spanners. *ACM Trans. Algorithms*, 14(3):33:1–33:15, 2018. preliminary version published in SODA 2016. doi:10.1145/3199607.

27 L. P. Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, pages 169–177, 1986.

28 L. P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.

29 K. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 56–65, 1987.

30 V. Cohen-Addad and C. Mathieu. Effectiveness of local search for geometric optimization. In *31st International Symposium on Computational Geometry*, volume 35 of *SoCG '2015*, pages 329–343, 2015. doi:10.4230/LIPIcs.SOCG.2015.329.

31 G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional euclidean space. In *Proceedings of the 9th Annual Symposium on Computational Geometry*, SCG '93, pages 53–62, 1993.

32 G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished euclidean graphs. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, pages 215–222, 1995.

33 Michael Elkin, Ofer Neiman, and Shay Solomon. Light spanners. *SIAM J. Discret. Math.*, 29(3):1312–1321, 2015. preliminary version published in ICALP 2014.

34 Michael Elkin and Shay Solomon. Narrow-shallow-low-light trees with and without steiner points. *SIAM J. Discret. Math.*, 25(1):181–210, 2011. preliminary version published in ESA 2009.

35 Michael Elkin and Shay Solomon. Steiner shallow-light trees are exponentially lighter than spanning ones. *SIAM J. Comput.*, 44(4):996–1025, 2015. preliminary version published in FOCS 2011. doi:10.1137/13094791X.

36 J. Erickson. Dense point sets have sparse delaunay triangulations or "... but not too nasty". *Discrete & Computational Geometry*, 33(1):83–115, 2004. doi:10.1007/s00454-004-1089-3.

37 Mohammad Farshi and Joachim Gudmundsson. Experimental study of geometric t-spanners. In Gerth Stølting Brodal and Stefano Leonardi, editors, *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*, volume 3669 of *Lecture Notes in Computer Science*, pages 556–567. Springer, 2005.

38 Mohammad Farshi and Joachim Gudmundsson. Experimental study of geometric t-spanners: A running time comparison. In Camil Demetrescu, editor, *Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proceedings*, volume 4525 of *Lecture Notes in Computer Science*, pages 270–284. Springer, 2007.

39 Arnold Filtser and Ofer Neiman. Light spanners for high dimensional norms via stochastic decompositions. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 29:1–29:15, 2018. doi:10.4230/LIPIcs.ESA.2018.29.

40 J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. *Computational Geometry*, 35(1):2–19, 2006. doi:10.1016/j.comgeo.2005.10.001.

41 Lee-Ad Gottlieb. A light metric spanner. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 759–772, 2015. doi:10.1109/FOCS.2015.52.

42 J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. Approximate distance oracles for geometric graphs. In *Proc. of 13th SODA*, pages 828–837, 2002.

43 J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. Approximate distance oracles for geometric spanners. *ACM Transactions on Algorithms*, 4(1), 2008.

44 J. Gudmundsson, G. Narasimhan, and M. H. M. Smid. Fast pruning of geometric spanners. In *Proc. of 22nd STACS*, pages 508–520, 2005.

45 Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Approximate distance oracles revisited. In *Proc. of 13th ISAAC*, pages 357–368, 2002.

46 Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Approximate distance oracles for geometric spanners. *ACM Trans. Algorithms*, 4(1):10:1–10:34, 2008.

47 Joachim Gudmundsson, Giri Narasimhan, and Michiel H. M. Smid. Fast pruning of geometric spanners. In Volker Diekert and Bruno Durand, editors, *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, volume 3404 of *Lecture Notes in Computer Science*, pages 508–520. Springer, 2005.

48 S. Har-Peled. Clustering motion. *Discrete and Computational Geometry*, 31(4):545–565, 2004. doi:10.1007/s00454-004-2822-7.

49 S. Har-peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011. doi:10.5555/2031416.

50 S. Har-Peled and B. Raichel. Net and prune: A linear time algorithm for euclidean distance problems. *Journal of the ACM*, 62(6):1–35, 2015. doi:10.1145/2831230.

51 S. Har-Peled and B. Sadri. How fast is the k-means method? In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, page 877–885, 2006.

52 Y. Hassin and D. Peleg. Sparse communication networks and efficient routing in the plane. In *Proc. of 19th PODC*, pages 41–50, 2000.

53 R. M. Karp. Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane. *Mathematics of Operations Research*, 2(3):209–224, 1977. doi:10.1287/moor.2.3.209.

54 J. M. Keil. Approximating the complete euclidean graph. In *Proceedings of the first Scandinavian Workshop on Algorithm Theory*, SWAT '88, pages 208–213, 1988.

55 J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete and Computational Geometry*, 7(1):13–28, 1992.

56 Michael Kerber and Arnur Nigmetov. Metric spaces with expensive distances. *CoRR*, abs/1901.08805, 2019. arXiv:1901.08805.

57 P. N. Klein. Subset spanner for planar graphs, with application to subset TSP. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, STOC '06, pages 749–756, 2006. doi:10.1145/1132516.1132620.

58 Hung Le. A PTAS for subset TSP in minor-free graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2279–2298, 2020. Full version: arxiv:1804.01588. doi:10.1137/1.9781611975994.140.

59 Hung Le and Shay Solomon. Truly optimal euclidean spanners. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1078–1100, 2019. Full version at arXiv:1904.12042. doi:10.1109/FOCS.2019.00069.

60 Y. Mansour and D. Peleg. An approximation algorithm for min-cost network design. *DIMACS Series in Discr. Math and TCS*, 53:97–106, 2000.

61 G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

62 A. Nayyeri and B. Raichel. Reality distortion: Exact and approximate algorithms for embedding into the line. In *Proceedings of the 56th Annual Symposium on Foundations of Computer Science*, FOCS '15, page 729–747, 2015. doi:10.1109/FOCS.2015.50.

63 A. Nayyeri and B. Raichel. A treehouse with custom windows: Minimum distortion embeddings into bounded treewidth graphs. In *Proceedings of the 28 th Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '17, page 724–736, 2017. doi:10.1137/1.9781611974782.46.

64 A. Nayyeri and B. Raichel. Viewing the rings of a tree: Minimum distortion embeddings into trees. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 2380–2399, 2019. doi:10.1137/1.9781611975482.146.

65 S. B. Rao and W. D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, STOC '98, pages 540–550, 1998. Full version at http://graphics.stanford.edu/courses/cs468-06-winter/Papers/rs-tsp.pdf. doi:10.1145/276698.276868.

66 J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proceedings of the 3rd Canadian Conference on Computational Geometry*, CCCG '91, page 207–210, 1991.

67 Jeffrey S. Salowe. On euclidean spanner graphs with small degree. In *Proceedings of the Eighth Annual Symposium on Computational Geometry, Berlin, Germany, June 10-12, 1992*, pages 186–191, 1992.

68 S. Solomon. Euclidean Steiner shallow-light trees. *Journal of Computational Geometry*, 6(2), 2015. preliminary version published in SoCG 2014. doi:10.20382/JOCG.V6I2A7.

69 S. Solomon and M. Elkin. Balancing degree, diameter and weight in euclidean spanners. In *Proceedings of the 18th Annual European Symposium on Algorithms*, ESA '10, pages 48–59, 2010. doi:10.1007/978-3-642-15775-2_5.

70 A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

Settling the Relationship Between Wilber's Bounds for Dynamic Optimality

Victor Lecomte
Columbia University, New York, NY, USA
vl2414@columbia.edu

Omri Weinstein
Columbia University, New York, NY, USA
omri@cs.columbia.edu

Abstract

In FOCS 1986, Wilber proposed two combinatorial lower bounds on the operational cost of any binary search tree (BST) for a given access sequence $X \in [n]^m$. Both bounds play a central role in the ongoing pursuit of the *dynamic optimality conjecture* (Sleator and Tarjan, 1985), but their relationship remained unknown for more than three decades. We show that Wilber's *Funnel bound* dominates his *Alternation bound* for all X, and give a tight $\Theta(\lg \lg n)$ separation for some X, answering Wilber's conjecture and an open problem of Iacono, Demaine et. al. The main ingredient of the proof is a new *symmetric* characterization of Wilber's Funnel bound, which proves that it is invariant under *rotations* of X. We use this characterization to provide initial indication that the Funnel bound matches the Independent Rectangle bound (Demaine et al., 2009), by proving that when the Funnel bound is constant, $\mathsf{IRB}_{\nearrow}$ is linear. To the best of our knowledge, our results provide the first progress on Wilber's conjecture that the Funnel bound is dynamically optimal (1986).

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases data structures, binary search trees, dynamic optimality, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.68

Related Version A full version of the paper is available on arXiv [10] at `https://arxiv.org/abs/1912.02858`. This conference version doesn't contain the last section, which relates the Funnel bound to the Independent Rectangle bound. The versions are otherwise identical.

Funding *Victor Lecomte*: Research supported by a fellowship of the Belgian American Educational Foundation.
Omri Weinstein: Research supported by NSF CAREER award CCF-1844887.

Acknowledgements We want to thank the anonymous reviewers for their enthusiastic feedback and the numerous typos they spotted.

1 Introduction

The *dynamic optimality conjecture* of Sleator and Tarjan [14] postulates the existence of an *instance optimal* binary search tree algorithm (BST), namely, an online self-adjusting BST whose running time[1] matches the best possible running time *in hindsight* for any fixed sequence of queries. More formally, letting $\mathcal{T}(X)$ denote the operational time of a BST algorithm $\mathcal{T}$ on a sequence $X = (x_1, \ldots, x_m) \in [n]^m$ of keys to be searched, the conjecture says that there is an online BST $\mathcal{T}$ such that $\forall X$, $\mathcal{T}(X) \leq O(\mathsf{OPT}(X))$, where $\mathsf{OPT}(X) := \min_{\mathcal{T}'} \mathcal{T}'(X)$ denotes the optimal offline cost for X. Such instance optimal algorithms are generally impossible, as an offline algorithm that sees the input X in advance

[1] i.e. the number of pointer movements and tree rotations performed by the BST

can simply "store the answers" and output them in $O(1)$ per operation, which is why worst-case analysis is the typical benchmark for online algorithms. Nevertheless, in the BST model, where the competing class of algorithms are self-adjusting binary search trees, instance optimality is an intriguing possibility. After 35 years of active research, two BST algorithms are still conjectured to be constant-competitive: The first one is the celebrated *splay tree* of [14], the second one is the more recent *GreedyFuture* algorithm [12, 5, 13]. However, optimality of both splay trees and GreedyFuture was proven only in special cases, and they are not known to be $o(\lg n)$-competitive for general access sequences X (note that every balanced BST is trivially $O(\lg n)$-competitive). The best provable result to date on the algorithmic side is an $O(\lg \lg n)$-competitive BST, the *Tango Tree* ([5] and its subsequent variants [15, 2]).

The ongoing pursuit of dynamically-optimal BSTs motivated the development of lower bounds on the cost of the offline solution $\mathsf{OPT}(X)$, attempting to capture the "correct" complexity measure of a fixed access sequence X in the BST model, and thereby providing a concrete benchmark for competitive analysis. Indeed, one defining feature of the dynamic optimality problem (and the reason why it is a viable possibility) is the existence of nontrivial lower bounds on $\mathsf{OPT}(X)$ for individual *fixed* access sequences X, as opposed to distributional lower bounds. [2] These lower bounds are all derived from a natural geometric interpretation of the access sequence $X = x_1, \ldots, x_m$ as a point set on the plane, mapping the i^{th} access x_i to point (x_i, i) ([5, 8], see Figure 1). The earliest lower bounds on $\mathsf{OPT}(X)$ were proposed in an influential paper of Wilber [16], and are the main subject of this paper.

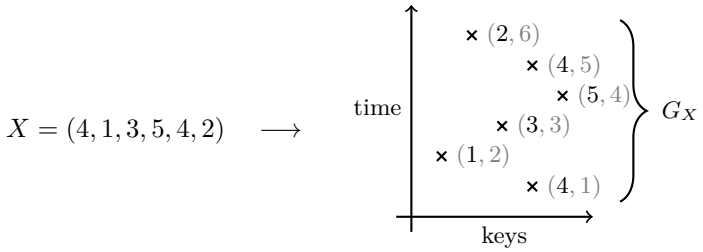


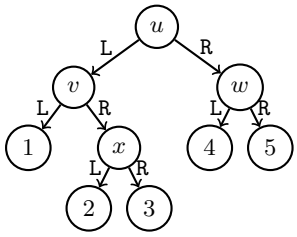
Figure 1 Transforming X into its geometric view G_X.

The Alternation bound

Wilber's first lower bound, the *Alternation bound* $\mathsf{Alt}_{\mathcal{T}}(X)$, counts the total number of left/right alternations obtained by searching the keys $X = (x_1, \ldots, x_m)$ on a *fixed* (static) binary search tree $\mathcal{T}$, where alternations are summed up over all nodes $v \in \mathcal{T}$ of the "reference tree" $\mathcal{T}$ (see Figure 2 and the formal definition in Section 2). Thus, the Alternation bound is actually a family of lower bounds, optimized by the choice of the reference tree $\mathcal{T}$, and we henceforth define $\mathsf{Alt}(X) \coloneqq \max_{\mathcal{T}} \mathsf{Alt}_{\mathcal{T}}(X)$. This lower bound played a key role in the design and analysis of Tango trees and their variants [6, 15], whose operational cost is in fact shown to be $O(\lg \lg n) \cdot \mathsf{Alt}_{\mathcal{T}}(X) \leq O(\lg \lg n) \cdot \mathsf{OPT}(X)$ (when setting the reference tree $\mathcal{T}$ to be the canonical balanced BST on $[n]$). Unfortunately, this bound is not tight, as we show that there are access sequences $\tilde{X}$ for which $\mathsf{Alt}_{\mathcal{T}}(\tilde{X}) \leq O(\mathsf{OPT}(\tilde{X})/\lg \lg n)$ *simultaneously* for all

[2] For example, Wilber's Alternation bound can be used to show that the "bit-reversal" access sequence obtained by reversing the binary representation of the monotone sequence $\{1, 2, 3, \ldots, n\}$ has cost $\Omega(\lg n)$ per operation [16].

choices of reference trees $\mathcal{T}$ (previously, this was known only for any *fixed* $\mathcal{T}$ [8]), and hence the combined bound $\mathsf{Alt}(X)$ does not capture dynamic optimality in general. Nevertheless, the algorithmic interpretation of the Alternation bound is an interesting proof-of-concept of how lower bounds can lead to new and interesting online BST algorithms.



reference tree $\mathcal{T}$

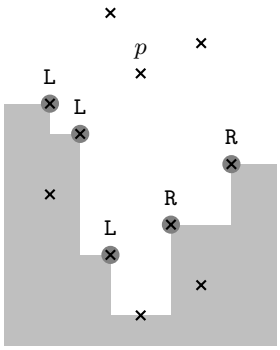
Node	Link used by each access	Group by letter	#
u	R, L, L, R, R, L	[R], [L, L], [R, R], [L]	4
v	L, R, R	[L], [R, R]	2
w	L, R, L	[L], [R], [L]	3
x	R, L	[R], [L]	2
Total			11

Figure 2 For access sequence $X = (4, 1, 3, 5, 4, 2)$ and reference tree $\mathcal{T}$, $\mathsf{Alt}_{\mathcal{T}}(X) = 11$.

The Funnel bound

The definition of Wilber's second bound, the *Funnel bound*, is less intuitive (and as such, was much less understood prior to this work). Let G_X be the set of m points in the plane given by the map $x_i \mapsto (x_i, i)$. The *funnel* of a point $p \in G_X$ is the set of "orthogonally visible" points below p, i.e. points q such that the axis-aligned rectangle with corners at p and q contains no other points (see Figure 3). For each p, look at the points in the funnel of p sorted by y coordinate, and count the number of alternations from the left to the right of P that occur. Call this $f(p)$; this is p's contribution to the lower bound. Summing this value for all $p \in G_X$ gives the lower bound $\mathsf{Funnel}(X) := \sum_{p \in G_X} f(p)$. An algorithmic view of this bound is as follows: consider the algorithm that simply brings each x_i to the root by a series of single rotations. Then $f(p)$ for $p = (x_i, i)$ is exactly the number of *turns* on the path from the root to x_i right before it is accessed [1, 8]. This view emphasizes the amortized nature of the funnel bound: at any point, there could be linearly many keys in the tree that are only *one* turn away from the root, so one can only hope to achieve this bound in some amortized fashion. This partially explains why Wilber's second bound has been so elusive to analyze (more on this interpretation can be found in the recent work of [11]).

Wilber conjectured that $\mathsf{Funnel}(X) \geq \Omega(\mathsf{Alt}(X))$ for every access sequence X, and that the Funnel bound is in fact *dynamically optimal*, i.e., that $\mathsf{Funnel}(X) = \Theta(\mathsf{OPT}(X))\ \forall X$. These conjectures were echoed multiple times in the long line of research spanning dynamic optimality (see e.g., [5, 8, 4, 9]). Very recently, Levy and Tarjan [11] gave a compelling intuitive explanation for why $\mathsf{Funnel}(X)$ is related to the amortized analysis of splay trees (see Section 4). Despite all this, the Funnel bound remained elusive and no progress was made on Wilber's conjectures for nearly 40 years (To the best of our knowledge, the only properties that were previously known about the Funnel bound is that it is optimal in the "key-independent" setting [7] and "approximately monotone" [11], both are prerequisites for dynamic optimality.)



the funnel of p has 5 points (highlighted)
Sorted by increasing y-coordinate: L, R, R, L, L.
This forms 3 groups [L], [R, R], [L, L], so $f(p) = 3$.

Figure 3 Computing $f(p)$ for $p = (4, 9)$ in the geometric view of $X = (4, 6, 3, 5, 1, 7, 2, 4, 6, 3)$. Notice how the funnel points form a staircase-like front on either side of p.

Our main contribution affirmatively answers Wilber's first question, and settles the relationship between the Alternation bound and the Funnel bound:

▶ **Theorem 1** (Funnel dominates Alt). *For every access sequence X without repeats*[3] *and for every tree $\mathcal{T}$, $\mathsf{Alt}_{\mathcal{T}}(X) \le O(\mathsf{Funnel}(X) + m)$.*

▶ **Theorem 2** (Tight separation). *There is an access sequence $\tilde{X}$ for which $\mathsf{Funnel}(\tilde{X}) \ge \Omega(\lg\lg n) \cdot (\mathsf{Alt}_{\mathcal{T}}(\tilde{X}) + m)$ simultaneously for all trees $\mathcal{T}$.*

The latter separation is tight up to constant factors, since Tango trees imply that $\mathsf{OPT}(X) \le O(\lg\lg n) \cdot \mathsf{Alt}(X)$. An interesting corollary of Theorem 2 is that the analysis of Tango trees cannot be improved by choosing *any* reference tree, answering an open question of Iacono [8]. (One attractive idea is to choose a *random* reference tree instead of the canonical balanced BST, but Theorem 2 shows that this will not help in general.)

A symmetric characterization of the Funnel bound

The geometric equivalence of dynamic optimality (through "arborally satisfied" rectangles [5]) makes it clear that $\mathsf{OPT}(X)$ is *invariant* under geometric transformations of the access sequence X. Indeed, a fundamental barrier in understanding the Funnel bound and its claim to optimality is that it was unclear whether Wilber's bounds were invariant under *rotations* of the access sequence X. Demaine et al. explicitly pointed out this challenge:

> "It is also unclear how [Wilber's] bounds are affected by 90-degree rotations of the point set representing the access sequence and, for the Funnel bound, by *flips*. Computer search reveals many examples where the bounds change slightly, and proving that they change by only a constant factor seems daunting." [5]

This shows that *exact* symmetry of $\mathsf{Funnel}(X)$ is hopeless, and can only hold in some "amortized" sense. Indeed, the heart of our paper, which is also a key ingredient in the proof of Theorem 1, is a new *symmetric* characterization of the Funnel bound, which proves

[3] As explained at the beginning of Section 2, it is fine for our purposes to focus on access sequences where each value appears only once.

that, up to a $\pm O(m)$ additive term, it is indeed invariant to rotations. More formally, we show that for any access sequence X, $\mathsf{Funnel}(X)$ is asymptotically equal to the number of occurrences in G_X of a configuration of 4 points that we call a **z-rectangle**[4] (see Figure 4).

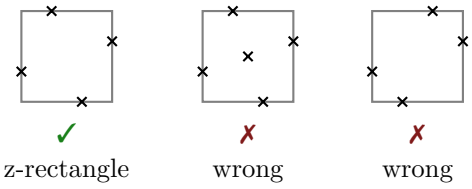


Figure 4 A z-rectangle is a configuration of 4 points. Its interior must be empty, and the relative order of the four points matters.

A crucial difference between z-rectangles and the notion of *independent rectangles* [5] is that the latter have to satisfy additional *independence* constaints across several rectangles, whereas z-rectangles have no "global" constraints whatsoever. In other words, z-rectangles are a *local* feature of the access sequence, in the sense that their existence and contribution to the lower bound are unaffected by other z-rectangles and by points outside of it. We believe this key property will make the analysis of online BST algorithms more tractable, as it gives a simpler competitive benchmark. We next describe an initial step in this direction.

Towards dynamic optimality of the Funnel Bound

One consequence of the simplicity of the z-rectangle characterization of the Funnel bound is that it makes it easier to compare it both to other BST lower bounds and to candidate algorithms for dynamic optimality. As a proof of concept, we show that when there is only a constant number of z-rectangle in G_X, then $\mathsf{IRB}_{\nearrow}(X)$ is linear, where $\mathsf{IRB}_{\nearrow}$ is one of the terms in the *Independent Rectangle* bound $\mathsf{IRB}(X) := \mathsf{IRB}_{\nearrow}(X) + \mathsf{IRB}_{\nwarrow}(X)$, which is known to dominate both of Wilber's bounds [5] (we define $\mathsf{IRB}_{\nearrow}(X)$ in the last section of the full version [10]). More formally,

▶ **Theorem 3.** *If G_X contains $O(1)$ z-rectangles, then $\mathsf{IRB}_{\nearrow}(X) \leq O(m)$.*

We remark that the proof of this theorem already introduces a nontrivial charging argument that could (hopefully) be generalized to prove that Funnel matches IRB, as conjectured by previous works [8].

Techniques

At a very high level, the main ideas in Theorem 1 are to use the self-reducible structure of the Alternation bound, and to show that interleaving two access sequences $X_{\mathtt{L}}$ and $X_{\mathtt{R}}$ on *disjoint ranges* is a super-additive operation, i.e., it increases the overall value of $\mathsf{Funnel}(X)$ to more than the sum of its parts $\mathsf{Funnel}(X_{\mathtt{L}}) + \mathsf{Funnel}(X_{\mathtt{R}})$. This argument involves both X and its *reverse* (flip), hence our new symmetric characterization of the Funnel bound (through z-rectangles) is key to the proof. The main idea behind Theorem 2 is to form hard sequences over geometrically-spaced sets of keys $\{i+1, i+2, i+4, i+8, \ldots\}$, each of which can "force" $\mathsf{Alt}_{\mathcal{T}}$ to pick a very lopsided reference tree $\mathcal{T}$. Those sequences can then be

[4] We thank an anonymous reviewer for informing us that z-rectangles have been discussed in the past under the name "pinwheel configuration", though (to the best of their knowledge) never in writing.

concatenated together so that the average value of $\mathsf{Alt}_{\mathcal{T}}$ is provably low whichever $\mathcal{T}$ was picked. Finally, the key idea in Theorem 3 is to study the consequences of the *absence* of z-rectangles on the combinatorial structure of point set G_X, and use this to bound the value of $\mathsf{IRB}_{\not\square}(X)$ by a charging argument.

Remark on independent work

In a concurrent and independent work, Chalermsook, Chuzhoy and Saranurak [3] obtain a (weaker) $\Theta(\lg\lg n/\lg\lg\lg n)$ separation between Alt and Funnel, in the same spirit as the tight separation we give in Theorem 2. Our works are otherwise unrelated.

2 Preliminaries

To make our definitions and proofs easier, we will work directly in the geometric representation of access sequences as (finite) sets of points in the plane $\mathbb{R}^2$.

▶ **Definition 4** (geometric view). *Any access sequence $X = (x_1, \ldots, x_m) \in [n]^m$ can be represented as the set of points $G_X = \{(x_i, i) \mid i \in [n]\}$, where the x-axis represents the key and the y-axis represents time (see Figure 1).*

By construction, in G_X, no two points share the same y-coordinate. We will say such a set has "distinct y-coordinates". In addition, we note that it is fine to restrict our attention to sequences X without repeated values.[5] The geometric view G_X of such sequences also has no two points with the same x-coordinate. We will say that such a set has "distinct x- and y-coordinates".

▶ **Definition 5** (x- and y-coordinates). *For a point $p \in \mathbb{R}^2$, we will denote its x- and y-coordinates as $p.x$ and $p.y$. Similarly, we define $P.x = \{p.x \mid p \in P\}$ and $P.y = \{p.y \mid p \in P\}$.*

We start by defining the *mixing value* of two sets: a notion of how much two sets of numbers are interleaved. It will be useful in defining both the Alternation bound and the Funnel bound. We define it in a few steps.

▶ **Definition 6** (mixing string). *Given two disjoint finite sets of real numbers L, R, let $\mathrm{mix}(L, R)$ be the string in $\{\mathtt{L}, \mathtt{R}\}^*$ that is obtained by taking the union $L \cup R$ in increasing order and replacing each element from L by $\mathtt{L}$ and each element from R by $\mathtt{R}$. For example, $\mathrm{mix}(\{2, 3, 8\}, \{1, 5\}) = \mathtt{RLLRL}$.*

▶ **Definition 7** (number of blocks). *Given a string $s \in \{\mathtt{L}, \mathtt{R}\}^*$, we define $\mathrm{blocks}(s)$ as the number of contiguous blocks of the same symbol in s. Formally,*

$$\mathrm{blocks}(s) := \begin{cases} 0 & \text{if } s \text{ is empty} \\ 1 + \#\{i \mid s_i \neq s_{i+1}\} & \text{otherwise.} \end{cases}$$

For example, $\mathrm{blocks}(\mathtt{LLLRLL}) = 3$. Note that if we insert characters into s, $\mathrm{blocks}(s)$ can only increase.

▶ **Definition 8** (mixing value). *Let $\mathrm{mixValue}(L, R) := \mathrm{blocks}(\mathrm{mix}(L, R))$ (see Figure 5).*

The mixing value has some convenient properties, which we will use later:

[5] Indeed, Appendix E in [4] gives a simple operation that transforms any sequence X into a sequence $\mathrm{split}(X)$ without repeats such that $\mathsf{OPT}(\mathrm{split}(X)) = \Theta(\mathsf{OPT}(X))$. Thus if we found a tight lower bound $L(X)$ for sequences without repeats, a tight lower bound for general X could be obtained as $L(\mathrm{split}(X))$.

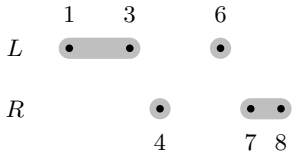


Figure 5 A visualization of $\text{mixValue}(\{1,3,6\},\{4,7,8\}) = 4$.

▶ **Fact 9** (properties of mixValue). *Function* $\text{mixValue}(L,R)$ *is:*

(a) *symmetric:* $\text{mixValue}(L,R) = \text{mixValue}(R,L)$*;*

(b) *monotone: if* $L_1 \subseteq L_2$ *and* $R_1 \subseteq R_2$*, then* $\text{mixValue}(L_1,R_1) \leq \text{mixValue}(L_2,R_2)$*;*

(c) *subadditive under concatenation: if* $L_1, R_1 \subseteq (-\infty, x]$ *and* $L_2, R_2 \subseteq [x, +\infty)$*, then* $\text{mixValue}(L_1 \cup L_2, R_1 \cup R_2) \leq \text{mixValue}(L_1,R_1) + \text{mixValue}(L_2,R_2)$.

Finally, $\text{mixValue}(L,R) \leq 2 \cdot \min(|L|,|R|) + 1$.

We now give precise definitions of Wilber's two bounds.[6]

▶ **Definition 10** (Alternation bound). *Let* P *be a point set with distinct* y*-coordinates, and let* $\mathcal{T}$ *be a binary tree in which leaves are labeled with elements of* $P.x$ *in increasing order, and each non-leaf node has two children.*

We define $\mathsf{Alt}_{\mathcal{T}}(P)$ *using the recursive structure of* $\mathcal{T}$*. If* $\mathcal{T}$ *is a single node, let* $\mathsf{Alt}_{\mathcal{T}}(P) := 0$*. Otherwise, let* $\mathcal{T}_{\mathtt{L}}$ *and* $\mathcal{T}_{\mathtt{R}}$ *be the left and right subtrees at the root. Partition* P *into two sets* $P_{\mathtt{L}} := \{p \in P \mid p.x \in \mathcal{T}_{\mathtt{L}}\}$ *and* $P_{\mathtt{R}} := \{p \in P \mid p.x \in \mathcal{T}_{\mathtt{R}}\}$*. Define quantity*

$$a(P,\mathcal{T}) := \text{mixValue}(P_{\mathtt{L}}.y, P_{\mathtt{R}}.y),$$

which describes how much $P_{\mathtt{L}}$ *and* $P_{\mathtt{R}}$ *are interleaved in time. Then*

$$\mathsf{Alt}_{\mathcal{T}}(P) := a(P,\mathcal{T}) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{L}}}(P_{\mathtt{L}}) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{R}}}(P_{\mathtt{R}}). \tag{1}$$

In addition, for an access sequence X*, let* $\mathsf{Alt}_{\mathcal{T}}(X) := \mathsf{Alt}_{\mathcal{T}}(G_X)$.

▶ **Definition 11** (axis-aligned rectangle delimited two points). *Given two points* p *and* q *with distinct* x*- and* y*- coordinates, let* $\square pq$ *be the smallest axis-aligned rectangle that contains both* p *and* q*. Formally,*

$$\square pq := [\min(p.x,q.x), \max(p.x,q.x)] \times [\min(p.y,q.y), \max(p.y,q.y)].$$

▶ **Definition 12** (empty rectangles). *Let* P *be a point set. Given* $p,q \in P$*, we say* $\square pq$ *is empty*[7] *in* P *if* $P \cap \square pq = \{p,q\}$ *(see Figure 6).*

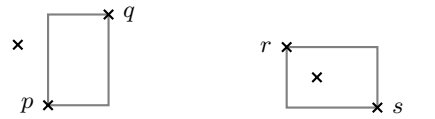
For the next definitions, it is helpful to refer back to Figure 3. In particular, $F_{\mathtt{L}}(P,p)$ and $F_{\mathtt{R}}(P,p)$ (the left and right funnel) correspond to the points marked with `L` and `R`.

▶ **Definition 13** (left and right funnel). *Let* P *be a point set. For each* $p \in P$*, we say that access* $q \in P$ *is in the left (resp. right) funnel of* p *within* P *if* q *is to the lower left (resp. lower right) of* p *and* $\square pq$ *is empty. Formally, let*

$$F_{\mathtt{L}}(P,p) := \{q \in P \mid q.y < p.y \wedge q.x < p.x \wedge P \cap \square pq = \{p,q\}\}$$

[6] These definitions may differ by a constant factor or an additive $\pm O(m)$ from the definitions the reader has seen before. We will ignore such differences, because the cost of a BST also varies by $\pm O(m)$ depending on the definition, and the interesting regime is when $\mathsf{OPT}(X) = \omega(m)$.

[7] This corresponds to the notion of "unsatisfied rectangle" in [5].



$\square pq$ is empty $\square rs$ is *not* empty

Figure 6 Some axis-aligned rectangles.

and

$$F_{\mathtt{R}}(P,p) \coloneqq \{q \in P \mid q.y < p.y \wedge q.x > p.x \wedge P \cap \square pq = \{p,q\}\}.$$

We will collectively call $F_{\mathtt{L}}(P,p) \cup F_{\mathtt{R}}(P,p)$ *the funnel of* p *within* P*.*

▶ **Definition 14** (Funnel bound)**.** *Let* P *be a point set with distinct* y*-coordinates. For each* $p \in P$*, define quantity*

$$f(P,p) \coloneqq \text{mixValue}(F_{\mathtt{L}}(P,p).y, F_{\mathtt{R}}(P,p).y),$$

which describes how much the left and right funnel of p *are interleaved in time. Then*

$$\mathsf{Funnel}(P) \coloneqq \sum_{p \in P} f(P,p).$$

In addition, for an access sequence X*, let* $\mathsf{Funnel}(X) \coloneqq \mathsf{Funnel}(G_X)$*.*

3 The Funnel bound dominates the Alternation bound

We prove that Funnel dominates Alt in two parts: in Section 3.1 we show that $\mathsf{Alt}(X)$ is dominated by the sum $\mathsf{Funnel}(X) + \mathsf{Funnel}(\overline{X})$, where $\overline{X}$ is the reverse of X, then in Section 3.2 we prove that $\mathsf{Funnel}(\overline{X}) \approx \mathsf{Funnel}(X)$ using our new characterization of Funnel by z-rectangles.

3.1 Upper-bounding the Alternation bound by a sum of two Funnel bounds

▶ **Definition 15** (time reversal)**.** *The time reversal of a point* $p \in \mathbb{R}^2$ *is* $\overline{p} \coloneqq (p.x, -p.y)$.[8] *The time reversal of a point set* P *is* $\overline{P} \coloneqq \{\overline{p} \mid p \in P\}$ *(see Figure 7).*

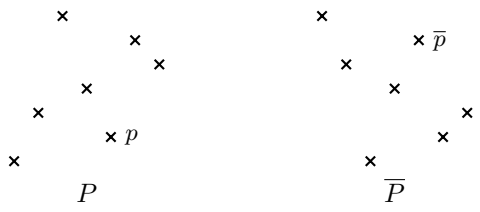


Figure 7 A point set and its time reversal.

We first prove the following lemma.

[8] The notation is inspired from the notion of complex conjugate, which is also a vertical flip.

▶ **Lemma 16.** *Let P be a point set with distinct y-coordinates, and let $\mathcal{T}$ be a tree that satisfies the conditions of Definition 10. Then* $\mathsf{Funnel}(P) + \mathsf{Funnel}(\overline{P}) \geq \mathsf{Alt}_{\mathcal{T}}(P)$.

Even though the formal proof of this lemma is a relatively involved case analysis, it is easy to understand geometrically. The key observation is the following. Consider two sequences $X_{\mathtt{L}}$ and $X_{\mathtt{R}}$ on disjoint ranges, and interleave to form a single sequence X. Then the more times we switch from elements of $X_{\mathtt{L}}$ to elements of $X_{\mathtt{R}}$, the bigger $\mathsf{Funnel}(X) + \mathsf{Funnel}(\overline{X})$ is going to be.

To see this, let's look at the geometric view of X (see Figure 8). Let p and q be two consecutive points on the $X_{\mathtt{L}}$ side that are separated by a streak of points from $X_{\mathtt{R}}$ (i.e. all accesses between p and q vertically are from $X_{\mathtt{R}}$). First, assume $p.x > q.x$. Then q is in the left funnel of p, and at least of the points on the $X_{\mathtt{R}}$ between p and q must be in the right funnel of p, which forms a completely new group of funnel points compared to what p had in $X_{\mathtt{L}}$. This means that the contribution of p to $\mathsf{Funnel}(X)$ is at least one higher than its contribution to $\mathsf{Funnel}(X_{\mathtt{L}})$.

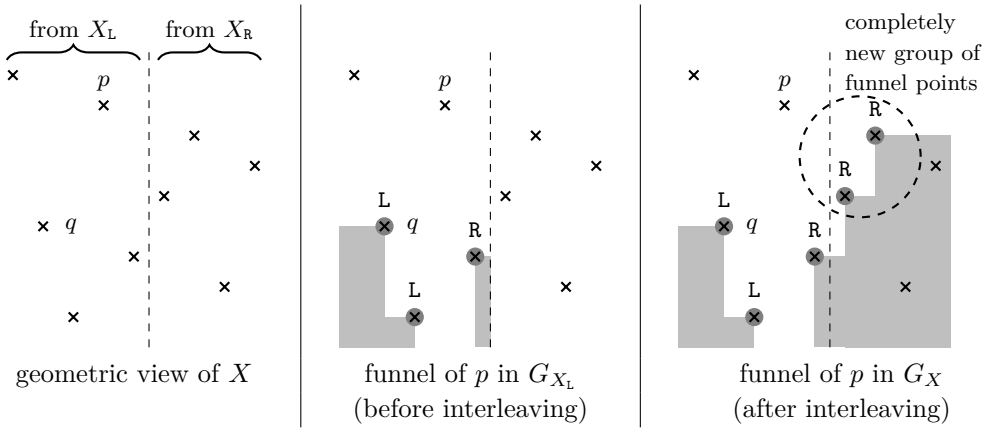


Figure 8 Interleaving sequences $X_{\mathtt{L}} = (3, 5, 2, 4, 1)$ and $X_{\mathtt{R}} = (8, 6, 9, 7)$ into $X = (3, 8, 5, 2, 6, 9, 7, 4, 1)$. The contribution of p to $\mathsf{Funnel}(X_{\mathtt{L}})$ is 3, while the contribution of p to $\mathsf{Funnel}(X)$ is 4.

What if $p.x < q.x$ instead? Then it turns out that an analogous argument can be made on q if we take the time reversal of X. That is, the contribution of $\overline{q}$ to $\mathsf{Funnel}(\overline{X})$ is at least one higher than its contribution to $\mathsf{Funnel}(\overline{X_{\mathtt{L}}})$. Indeed, if we flip the point set vertically, then p and q exchange roles, which means $p.x > q.x$ once again.

To conclude, it remains to observe that the $a(P, p)$ term in the recursive definition of $\mathsf{Alt}_{\mathcal{T}}(X)$ is precisely a measure of how much the subsequences $X_{\mathtt{L}}$ and $X_{\mathtt{R}}$ corresponding to the left and right subtree at the root of $\mathcal{T}$ are interleaved. So we can apply the argument above by induction to show that $\mathsf{Funnel}(X) + \mathsf{Funnel}(\overline{X}) \geq \mathsf{Alt}_{\mathcal{T}}(X)$. We now reluctantly move to the formal proof.

Proof of Lemma 16. We prove this by induction on $\mathcal{T}$. The base case is $\mathcal{T}$ made of a single node. In this case, $\mathsf{Alt}_{\mathcal{T}}(P) = 0$ by definition, so the inequality trivially holds.

Now consider a general tree $\mathcal{T}$, and define $\mathcal{T}_{\mathtt{L}}$, $\mathcal{T}_{\mathtt{R}}$, $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ as in Definition 10. Note that each leaf of $\mathcal{T}$ has a label in $P.x$ and $\mathcal{T}_{\mathtt{L}}$ and $\mathcal{T}_{\mathtt{R}}$ must each have at least one leaf, so $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ are not empty. Let's apply the induction hypothesis on $(P_{\mathtt{L}}, \mathcal{T}_{\mathtt{L}})$ and $(P_{\mathtt{R}}, \mathcal{T}_{\mathtt{R}})$. This means that

$$\mathsf{Funnel}(P_{\mathtt{L}}) + \mathsf{Funnel}(\overline{P_{\mathtt{L}}}) \geq \mathsf{Alt}_{\mathcal{T}_{\mathtt{L}}}(P_{\mathtt{L}})$$
$$\mathsf{Funnel}(P_{\mathtt{R}}) + \mathsf{Funnel}(\overline{P_{\mathtt{R}}}) \geq \mathsf{Alt}_{\mathcal{T}_{\mathtt{R}}}(P_{\mathtt{R}}).$$

Thus we find that

$$\begin{aligned}\mathsf{Alt}_{\mathcal{T}}(P) &= a(P,\mathcal{T}) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{L}}}(P_{\mathtt{L}}) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{R}}}(P_{\mathtt{R}}) && \text{(by definition)}\\ &\leq a(P,\mathcal{T}) + \mathsf{Funnel}(P_{\mathtt{L}}) + \mathsf{Funnel}(\overline{P_{\mathtt{L}}}) + \mathsf{Funnel}(P_{\mathtt{R}}) + \mathsf{Funnel}(\overline{P_{\mathtt{R}}}) && (2)\end{aligned}$$

▷ Claim 17. If $p \in P_{\mathtt{L}}$, then

$$f(P,p) \geq f(P_{\mathtt{L}},p) \quad \text{and} \quad f(\overline{P},\overline{p}) \geq f(\overline{P_{\mathtt{L}}},\overline{p});$$

and if $p \in P_{\mathtt{R}}$, then

$$f(P,p) \geq f(P_{\mathtt{R}},p) \quad \text{and} \quad f(\overline{P},\overline{p}) \geq f(\overline{P_{\mathtt{R}}},\overline{p}).$$

Proof. We will deal with the first case (the other three cases are symmetric). The key is that $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ operate on disjoint ranges of x-coordinates.

- The left funnel of p within $P_{\mathtt{L}}$ is identical to its left funnel within P, since all elements of $P_{\mathtt{R}}$ are to the right of p. Formally, $F_{\mathtt{L}}(P_{\mathtt{L}},p) = F_{\mathtt{L}}(P,p)$.
- All points q that were in the right funnel of p within $P_{\mathtt{L}}$ will still be part of the right funnel of p within P. Indeed, the only way for them to stop being funnel points would be to add accesses inside the rectangle delimited by p and q. This doesn't happen because all points in $P_{\mathtt{R}}$ are strictly to the right of all points in $P_{\mathtt{L}}$. Formally, $F_{\mathtt{R}}(P_{\mathtt{L}},p) \subseteq F_{\mathtt{R}}(P,p)$.

Therefore, $\mathrm{mix}(F_{\mathtt{L}}(P_{\mathtt{L}},p).y, F_{\mathtt{R}}(P_{\mathtt{L}},p).y)$ is a subsequence of $\mathrm{mix}(F_{\mathtt{L}}(P,p).y, F_{\mathtt{R}}(P,p).y)$, which means that

$$f(P_{\mathtt{L}},p) = \mathrm{blocks}(\mathrm{mix}(F_{\mathtt{L}}(P_{\mathtt{L}},p), F_{\mathtt{R}}(P_{\mathtt{L}},p))) \leq \mathrm{blocks}(\mathrm{mix}(F_{\mathtt{L}}(P,p), F_{\mathtt{R}}(P,p))) = f(P,p).$$

◁

Summing up $f(P,p)$ and $f(\overline{P},\overline{p})$ over all points $p \in P$, we obtain

$$\begin{aligned}\mathsf{Funnel}(P) &= \sum_{p\in P} f(P,p) \geq \sum_{p\in P_{\mathtt{L}}} f(P_{\mathtt{L}},p) + \sum_{p\in P_{\mathtt{R}}} f(P_{\mathtt{R}},p) = \mathsf{Funnel}(P_{\mathtt{L}}) + \mathsf{Funnel}(P_{\mathtt{R}})\\ \mathsf{Funnel}(\overline{P}) &= \sum_{p\in P} f(\overline{P},\overline{p}) \geq \sum_{p\in P_{\mathtt{L}}} f(\overline{P_{\mathtt{L}}},\overline{p}) + \sum_{p\in P_{\mathtt{R}}} f(\overline{P_{\mathtt{R}}},\overline{p}) = \mathsf{Funnel}(\overline{P_{\mathtt{L}}}) + \mathsf{Funnel}(\overline{P_{\mathtt{R}}}).\end{aligned} \tag{3}$$

This, combined with (2), gives

$$\begin{aligned}\mathsf{Funnel}(P) + \mathsf{Funnel}(\overline{P}) &\geq \mathsf{Funnel}(P_{\mathtt{L}}) + \mathsf{Funnel}(P_{\mathtt{R}}) + \mathsf{Funnel}(\overline{P_{\mathtt{L}}}) + \mathsf{Funnel}(\overline{P_{\mathtt{R}}})\\ &\geq \mathsf{Alt}_{\mathcal{T}}(P) - a(P,\mathcal{T})\end{aligned}$$

This falls $a(P,\mathcal{T})$ short of our goal (which makes sense, since we haven't used the interleaving of $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ yet). To fix this, we will show the following claim.

▷ Claim 18. Consider the following properties defined over a point $p \in P$:

(a) $p \in \mathcal{T}_{\mathtt{L}}$ and $f(P,p) \geq f(P_{\mathtt{L}},p) + 1$;
(b) $p \in \mathcal{T}_{\mathtt{L}}$ and $f(\overline{P},\overline{p}) \geq f(\overline{P_{\mathtt{L}}},\overline{p}) + 1$;
(c) $p \in \mathcal{T}_{\mathtt{R}}$ and $f(P,p) \geq f(P_{\mathtt{R}},p) + 1$;
(d) $p \in \mathcal{T}_{\mathtt{R}}$ and $f(\overline{P},\overline{p}) \geq f(\overline{P_{\mathtt{R}}},\overline{p}) + 1$.

The sum of the number of points in P having each property (a)–(d) is at least $a(P, \mathcal{T})$.

Proof. Let's number the points of P by increasing y-coordinate (i.e. in chronological order) as $p_1, \dots, p_m$. Recall that $a(P, \mathcal{T}) = \text{mixValue}(P_{\mathtt{L}}.y, P_{\mathtt{R}}.y)$. Also, $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ are non-empty, so $a(P, \mathcal{T}) \geq 2$. This means that as we go through the points $p_1, \dots, p_m$, we switch $a(P, \mathcal{T}) - 1 \geq 1$ times between points of $P_{\mathtt{L}}$ and points of $P_{\mathtt{R}}$.

Therefore, there are exactly $a(P, \mathcal{T}) - 2$ pairs of indices (i, j) with $i + 1 < j$ such that

- case 1: $p_i, p_j \in P_{\mathtt{L}}$ but $p_{i+1}, \dots, p_{j-1} \in P_{\mathtt{R}}$, or
- case 2: $p_i, p_j \in P_{\mathtt{R}}$ but $p_{i+1}, \dots, p_{j-1} \in P_{\mathtt{L}}$,

which "straddle accesses of the opposite side". Also, there is an index $i^* > 1$ (the "first element of the side that starts appearing later") such that

- case 3: $p_{i^*} \in P_{\mathtt{L}}$ but $p_1, \dots, p_{i^*-1} \in P_{\mathtt{R}}$, or
- case 4: $p_{i^*} \in P_{\mathtt{R}}$ but $p_1, \dots, p_{i^*-1} \in P_{\mathtt{L}}$

and similarly, there is an index $j^* < m$ (the "last element of the side that finishes appearing earlier") such that

- case 5: $p_{j^*} \in P_{\mathtt{L}}$ but $p_{j^*+1}, \dots, p_m \in P_{\mathtt{R}}$, or
- case 6: $p_{j^*} \in P_{\mathtt{R}}$ but $p_{j^*+1}, \dots, p_m \in P_{\mathtt{L}}$.

This makes for a total of $a(P, \mathcal{T}) - 2 + 1 + 1 = a(P, \mathcal{T})$ occurrences of one of the six cases. We will show that each of them leads to a point p satisfying one of the properties (a)–(d). More precisely, we claim that:

- case 1 implies p_j has property (a) or p_i has property (b);
- case 2 implies p_j has property (c) or p_i has property (d);
- case 3 implies p_{i^*} has property (a);
- case 4 implies p_{i^*} has property (c);
- case 5 implies p_{j^*} has property (b);
- case 6 implies p_{j^*} has property (d).

We will show this for case 1 and case 3. The other four cases are analogous. To treat case 1, let's separate into more cases.[9]

- If $p_i.x < p_j.x$, then p_i is in the left funnel of p_j within both P and $P_{\mathtt{L}}$. But within P, p_{j-1} would be an additional right funnel point. Since it has a higher index than p_i, this would add at least 1 to $f(P, p_j)$ compared to $f(P_{\mathtt{L}}, p_j)$. In other words, $f(P, p_j) \geq f(P_{\mathtt{L}}, p_j) + 1$ (scenario (a)).
- If $p_i.x > p_j.x$, then we can use the same argument as above on $\overline{P}$ and $\overline{P_{\mathtt{L}}}$ by swapping i and j, obtaining $f(\overline{P}, \overline{p_i}) \geq f(P_{\mathtt{L}}, \overline{p_i}) + 1$ (scenario (b)).
- If $p_i.x = p_j.x$, then both funnels of p_j within $P_{\mathtt{L}}$ are completely empty, which means that $f(P_{\mathtt{L}}, p_j) = 0$, while the right funnel of p_j in P would contain at least p_{j-1}. Therefore, $f(P, p_j) = 1 \geq f(P_{\mathtt{L}}, p_j) + 1$ (scenario (a)).

To treat case 3, it suffices to observe that both funnels of p_{i^*} within $P_{\mathtt{L}}$ would be completely empty (for lack of lower points), so $f(P_{\mathtt{L}}, p_{i^*}) = 0$, while in P the right funnel of x_{i^*} would contain at least p_{i^*-1}. Therefore, $f(P, p_{i^*}) \geq 1 = f(P_{\mathtt{L}}, p_{i^*}) + 1$ (scenario (a)). ◁

Now, if we sum up $f(P, p)$ and $f(\overline{P}, \overline{p})$ over all points p as we did in (3), but this time also apply Claim 18, we obtain that

$$\mathsf{Funnel}(P) + \mathsf{Funnel}(\overline{P}) \geq \mathsf{Funnel}(P_{\mathtt{L}}) + \mathsf{Funnel}(P_{\mathtt{R}}) + \mathsf{Funnel}(\overline{P_{\mathtt{L}}}) + \mathsf{Funnel}(\overline{P_{\mathtt{R}}}) + a(P, \mathcal{T}).$$

Combined with (2), this gives the desired result and concludes the inductive step. ◀

[9] We wish we were joking.

3.2 Characterizing the Funnel bound using z-rectangles

Lemma 16 asserts that all possible Alternation bounds for all choices of reference trees $\mathcal{T}$, are simultaneously upper-bounded by the sum of two specific Funnel bounds. While this is already a nontrivial bound, $\mathsf{Funnel}(P)$ and $\mathsf{Funnel}(\overline{P})$ could in principle be wildly different, and it is therefore more compelling to show that the *single* quantity $\mathsf{Funnel}(P)$ already provides an upper bound. (It is curious that the symmetry properties of the Funnel bound, which are a necessary precondition for dynamic optimality, already enter the picture in determining the relationship between Wilber's bounds.)

To achieve this, we need to think about how geometric transformations affect the value of the Funnel bound. It is clear from the definition that $\mathsf{Funnel}(P)$ is unaffected by a horizontal flip. Indeed, the left funnel would become the right funnel and vice versa, so this wouldn't affect the number of times we switch between the two: the quantity $f(P, p)$ would remain the same for each p (see Figure 9).

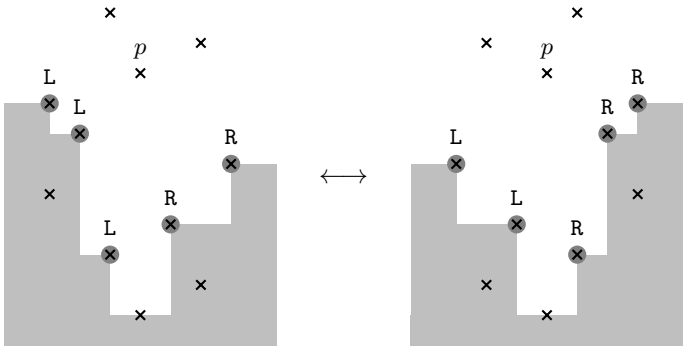
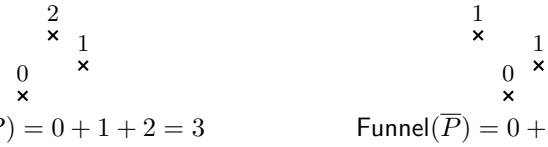


Figure 9 Flipping the geometric view horizontally conserves the contribution $f(P, p)$ of each point: the only change is that the labels of the funnel points flip between `L` and `R`.

On the other hand, it is far from obvious that the Funnel bound is unaffected by a vertical flip. Because of the time reversal, the notion of funnel changes completely. And indeed, the precise value will change, as is shown in Figure 10.



$\mathsf{Funnel}(P) = 0 + 1 + 2 = 3$ $\quad$ $\mathsf{Funnel}(\overline{P}) = 0 + 1 + 1 = 2$

Figure 10 A minimal example such that $\mathsf{Funnel}(P) \neq \mathsf{Funnel}(\overline{P})$ is $P = \{(1, 1), (3, 2), (2, 3)\}$. Each access p is labeled with its contribution $f(P, p)$ (left) or $f(\overline{P}, p)$ (right).

Nevertheless, we will show that for any point set P with distinct x- and y-coordinates, $\mathsf{Funnel}(P)$ and $\mathsf{Funnel}(\overline{P})$ are equal up to an additive $O(m)$. We do this by introducing a new characterization of the Funnel bound that is naturally invariant under *90° rotations* of the point set. This new characterization is the number of *z-rectangles*.

▶ **Definition 19** (z-rectangle). *Let P be a point set. We call tuple $(p, q, r, s) \in P^4$ a z-rectangle of P if the following conditions hold:*

(a) $q.x < p.x < r.x < s.x$;

(b) $r.y < q.y < s.y < p.y$;

(c) $P \cap [q.x, s.x] \times [r.y, p.y] = \{p, q, r, s\}$.

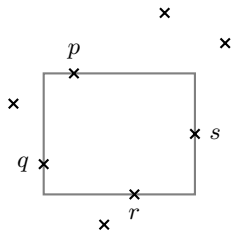


Figure 11 A z-rectangle. The relative order of points p, q, r, s horizontally and vertically matters.

In other words, a z-rectangle is a subsequence of 4 accesses with key values in relative order $3, 1, 4, 2$ and such that the axis-aligned rectangle that they span is empty (see Figure 11 for an example). We define the corresponding quantity, which we will prove is equivalent to the Funnel bound.

▶ **Definition 20** (z-rectangle bound). *For any point set P with distinct x- and y-coordinates,*[10] *let*

$$\mathsf{zRects}(P) := |\{(p, q, r, s) \mid (p, q, r, s) \text{ is a z-rectangle of } P\}|.$$

First, we formally state the rotation-invariance of z-rectangles.

▶ **Definition 21** (counter-clockwise 90° rotation). *For a point $p \in \mathbb{R}^2$, let $p^\perp := (-p.y, p.x)$. Analogously, for a point set P, let $P^\perp := \{p^\perp \mid p \in P\}$.*

▶ **Lemma 22.** *For any point set P, $\mathsf{zRects}(P) = \mathsf{zRects}(P^\perp)$.*

Proof. Each z-rectangle of P induces a z-rectangle in $P^\perp$ and vice-versa: z-rectangle (p, q, r, s) in P becomes z-rectangle $(s^\perp, p^\perp, q^\perp, r^\perp)$ in $P^\perp$ (the reader is encouraged to physically rotate the page containing figure 11 in order to convince themselves of this fact). Therefore, P and $P^\perp$ have the same number of z-rectangles. ◀

The relation between $\mathsf{Funnel}(P)$ and $\mathsf{zRects}(P)$ is proved in the following two lemmas.

▶ **Lemma 23.** $\mathsf{zRects}(P) \geq \mathsf{Funnel}(P)/2 - O(m)$.

▶ **Lemma 24.** $\mathsf{Funnel}(P) \geq 2 \cdot \mathsf{zRects}(P)$.

We will use the fact that P has distinct x- and y- coordinates.

Proof of Lemma 23. We will show that for each $p \in P$, the funnel of p induces at least $\lfloor f(P, p)/2 \rfloor - 1$ different z-rectangles of the form $(p, \cdot, \cdot, \cdot)$. Summing this up for each p then completes the proof.

Let's assume $f(P, p) \geq 4$; otherwise the claim holds vacuously. Let's number the points in $F_{\mathtt{L}}(P, p) \cup F_{\mathtt{R}}(P, p)$ (the funnel of p) by increasing y-coordinate as $a_1, a_2, \ldots, a_l$. Note that l may be greater than $f(P, p)$, because a sequence of funnel points that are all on the same side of p counts only for 1 in $f(P, p)$.

[10] If the x- and y-coordinates are not distinct, $\mathsf{zRects}(P)$ may give absurd results. For example, if we start with any P and add a duplicate point $(x, y + \epsilon)$ for every point (x, y) of P (with ϵ small enough), then $\mathsf{zRects}(P)$ will drop to 0.

We will call $(i, j) \in [l]^2$ a *left-straddling pair* if $i + 1 < j$, $a_i.x > p.x$ and $a_j.x > p.x$, but for all $i < k < j$, $a_k.x < p.x$. That is, a_i and a_j are to the right of p but all funnel points between them in order of height are to the left of p. Because funnel points alternate $f(P, p) - 1$ times between the left and the right of p, there must be at least $\lfloor f(P, p)/2 \rfloor - 1$ left-straddling pairs.

We claim that if (i, j) is a left-straddling pair, then (p, a_{i+1}, a_i, a_j) is a z-rectangle. Since all left-straddling pairs have distinct i, this produces $\lfloor f(P, p)/2 \rfloor - 1$ distinct z-rectangles.

First, we verify that p, a_{i+1}, a_i, a_j have the correct relative positions. The order in y-coordinate is correct by definition of the numbering $a_1, \ldots, a_k$. For the order in x-coordinates, we know that a_{i+1} is to the left of p and a_i, a_j are to its right, so we only need to verify that $a_i.x < a_j.x$. This is true because a_i is in the funnel of p, so $\square pa_i$ must be empty. If $a_i.x > a_j.x$, then a_j would be in $\square pa_i$.

What we still need to prove is that rectangle $[a_{i+1}.x, a_j.x] \times [a_i.y, p.y]$ is empty (except for points p, a_{i+1}, a_i, a_j themselves). First, since a_i, a_{i+1} and a_j in the funnel of p, we know that $\square pa_i$, $\square pa_{i+1}$ and $\square pa_j$ are empty. This covers the zones pictured in Figure 12.

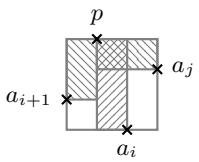


Figure 12 Proposed z-rectangle (p, a_{i+1}, a_i, a_j) with empty rectangles $\square pa_i$, $\square pa_{i+1}$ and $\square pa_j$ highlighted. If in addition we can prove that $\square a_i a_{i+1}$ and $\square a_i a_j$ are empty, then this is a valid z-rectangle.

Finally, we will prove that $\square a_i a_{i+1}$ and $\square a_i a_j$ are empty, which covers the missing parts.

- Assume $\square a_i a_{i+1}$ is not empty, and let b be the highest point of P in it (except for a_{i+1}). We have already shown that $\square pa_i$ and $\square pa_{i+1}$ are empty, so $\square pb$ must be empty. This means that b must be in the funnel of p. But $a_i.y < b.y < a_{i+1}.y$, so this contradicts the numbering by increasing y-coordinate.
- Assume $\square a_i a_j$ is not empty, and let b be the highest point of P in it (except for a_j). We have already shown that $\square pa_i$ and $\square pa_j$ are empty, so $\square pb$ must be empty. This means that b must be in the (right) funnel of p. But this contradicts our assumption that all funnel points between a_i and a_j in y-coordinate must be to the left of p.

Since points p, a_{i+1}, a_i, a_j and $[a_{i+1}.x, a_j.x] \times [a_i.y, p.y]$ is empty, (p, a_{i+1}, a_i, a_j) is a z-rectangle. This completes the proof of Lemma 23. ◀

Proof of Lemma 24. Essentially, the reason why this is true is because all z-rectangles must be exactly of the form described in the previous proof. We will prove something slightly weaker which still reaches the desired result. We will group the z-rectangles by their top point and show that if P has k rectangles of the form $(p, \cdot, \cdot, \cdot)$, then $f(P, p) \geq 2k$.

Fix p, and sort the k z-rectangles by the increasing y-coordinate of their bottom point r. Name their points (p, q_1, r_1, s_1) to (p, q_k, r_k, s_k). First, we will show that there can be no ties. Indeed, if $r_i.y = r_j.y$ then $r_i = r_j$. Also, when the p and r (top and bottom) points of a z-rectangle are fixed, then the other two points q and s are uniquely determined as the rightmost point in $(-\infty, p.x] \times [r.x, p.x]$ and the leftmost point in $[r.x, \infty) \times [r.x, p.x]$, respectively.

We will now prove that

$$q_1.y < s_1.y < q_2.y < s_2.y < \cdots < q_k.y < s_k.y. \tag{4}$$

The $q_i.y < s_i.y$ inequalities are true by the definition of a z-rectangle, so we only need to prove $s_i.y < q_{i+1}.y$. To do this, consider two consecutive z-rectangles (p, q_i, r_i, s_i) and $(p, q_{i+1}, r_{i+1}, s_{i+1})$ (see Figure 13). Since $r_i.y < r_{i+1}.y$, s_i can't be strictly to the right of r_{i+1}, because otherwise r_{i+1} would be inside z-rectangle (p, q_i, r_i, s_i). In turn, this means that s_i can't be strictly higher than r_{i+1} because otherwise it would be inside $\square pr_{i+1}$. Therefore, we have $s_i.y \leq r_{i+1}.y < q_{i+1}.y$.

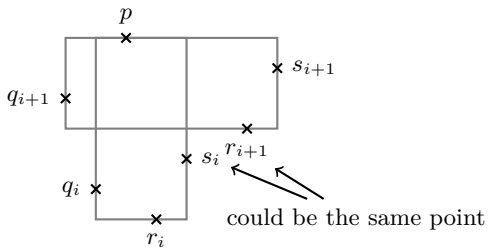


Figure 13 The only possible relative position of two z-rectangle with the same top point p.

Points $q_1, s_1, \ldots, q_k, s_k$ are all in the funnel of p by the definition of z-rectangle. Therefore, Equation (4) reveals $2k$ funnel points that alternate from the left to the right side of p with increasing y-coordinates. Thus $\mathrm{mix}(F_{\mathtt{L}}(P,p).y, F_{\mathtt{R}}(P,p).y)$ contains a subsequence $\mathtt{LRLR}\cdots\mathtt{LR}$ of length $2k$, and

$$f(P,p) = \mathrm{blocks}(\mathrm{mix}(F_{\mathtt{L}}(P,p).y, F_{\mathtt{R}}(P,p).y)) \geq \mathrm{blocks}(\underbrace{\mathtt{LRLR}\cdots\mathtt{LR}}_{\text{length } 2k}) = 2k.$$

Summing this up for each p completes the proof. ◀

▶ **Corollary 25.** $\mathsf{Funnel}(P) \geq \mathsf{Funnel}(\overline{P}) - O(m)$.

Proof. By the left-right symmetry of $\mathsf{Funnel}(\cdot)$, we know that $\mathsf{Funnel}(\overline{P}) = \mathsf{Funnel}(P^{\perp\perp})$, where $P^{\perp\perp}$ is P rotated by 180°. Therefore,

$$\begin{aligned}
\mathsf{Funnel}(P) &\geq 2 \cdot \mathsf{zRects}(P) && \text{(Lemma 24)}\\
&= 2 \cdot \mathsf{zRects}(P^{\perp\perp}) && \text{(Lemma 22)}\\
&\geq \mathsf{Funnel}(P^{\perp\perp}) - O(m) && \text{(Lemma 23)}\\
&= \mathsf{Funnel}(\overline{P}) - O(m). && \blacktriangleleft
\end{aligned}$$

We can now finally prove Theorem 1.

Proof of Theorem 1. By Lemma 16, $\mathsf{Alt}_{\mathcal{T}}(P) \leq \mathsf{Funnel}(P) + \mathsf{Funnel}(\overline{P})$. Combining this with Corollary 25, we obtain $\mathsf{Alt}_{\mathcal{T}}(P) \leq \mathsf{Funnel}(P) + (\mathsf{Funnel}(P) + O(m)) \leq O(\mathsf{Funnel}(P) + m)$. ◀

4 Separation between the Alternation bound and the Funnel bound

We will now define an access sequence $\tilde{X}$ such that the Alternation bound is too low for all reference trees $\mathcal{T}$ simultaneously. More precisely, we will define an access sequence $\tilde{X} \in [n]^m$ such that $\mathsf{Alt}_{\mathcal{T}}(\tilde{X}) = O(m)$ for all trees $\mathcal{T}$ while on the other hand $\mathsf{OPT}(\tilde{X})$ and $\mathsf{Funnel}(\tilde{X})$ are $\Theta(m \lg\lg n)$. This $\lg\lg n$ factor is the biggest possible separation: indeed, Tango trees show that for a balanced tree $\mathcal{T}$, $\mathsf{Alt}_{\mathcal{T}}(X)$ is always within $O(\lg\lg n)$ of $\mathsf{OPT}(X)$.

To define $\tilde{X}$, we will need the notion of a *bit-reversal* sequence. This is a permutation that in a sense looks "maximally shuffled" to a binary search tree.

▶ **Definition 26.** *Let k be a positive integer and let $K = 2^k$. Then let* $\text{bitReversal}^k \in \{0, \cdots, K-1\}^K$ *be the sequence where* bitReversal^k_i *is the number obtained by taking the binary representation of $i-1$, padding it with leading zeroes to reach length k, flipping it, then converting this back to a number.*

It is easiest to understand through an example. Take $k = 2$, then bitReversal^2 is obtained this way:

$$(0,1,2,3) \xrightarrow{\text{to binary}} (00,01,10,11) \xrightarrow{\text{flip}} (00,10,01,11) \xrightarrow{\text{from binary}} (0,2,1,3).$$

The reason why we use this sequence is the following well-known fact.

▶ **Fact 27.** *Let $\mathcal{T}$ be the complete binary tree of height k which has K leaves labeled 0 through $K-1$. Then $\mathsf{Alt}_{\mathcal{T}}(\text{bitReversal}^k) = kK = K \lg K$.*

Proof. Because of the way bitReversal^k is defined, for each node $u \in \mathcal{T}$, the keys that are accessed below u as the sequence is processed constantly alternate from u's left subtree to u's right subtree. So the contribution of u is exactly the number of keys of its subtree. This way, every key is counted once at each of the $k = \lg K$ levels, so the total is $K \lg K$. ◀

We can now define our access sequence as follows. Let $n \coloneqq 2^K = 2^{2^k}$, and let

$$S_i \coloneqq (i + 2^{\text{bitReversal}^k_1}, i + 2^{\text{bitReversal}^k_2}, \ldots, i + 2^{\text{bitReversal}^k_K}).$$

Then, denoting concatenation by $\circ$, we define

$$\tilde{X} \coloneqq \underbrace{S_0 \circ \cdots \circ S_0}_{n \text{ times}} \circ \underbrace{S_1 \circ \cdots \circ S_1}_{n \text{ times}} \circ \cdots \circ \underbrace{S_{n/2} \circ \cdots \circ S_{n/2}}_{n \text{ times}}.$$

The range of $\tilde{X}$ is $[n]$ and its length is $m = (\frac{n}{2}+1) \cdot n \cdot K = \Theta(n^2 \lg n)$. See Figure 14 for an example with $k = 2$. We will prove that for all $\mathcal{T}$, $\mathsf{Alt}_{\mathcal{T}}(\tilde{X}) \le O(m)$ while on the other hand $\mathsf{Funnel}(\tilde{X}) \ge \Omega(m \lg \lg n)$.

▶ **Lemma 28.** *For any $\mathcal{T}$, $\mathsf{Alt}_{\mathcal{T}}(\tilde{X}) \le O(m)$.*

▶ **Lemma 29.** $\mathsf{Funnel}(\tilde{X}) \ge \Omega(m \lg \lg n)$.

The combination of Lemma 28 and Lemma 29 shows the separation claimed in Theorem 2. Before we move to the proofs of those lemmas, let's go over some intuition for the proof of Lemma 28, which is the more complicated one.

First, note that the only reason we use bitReversal^k in $\tilde{X}$ is to make $\mathsf{Funnel}(\tilde{X})$ large. Replacing bitReversal^k by any other permutation of $\{0, \ldots, K-1\}$ would not affect the proof of Lemma 28 in any way because that proof only looks at the *set* of keys that are hit by each of the parts $S_0, \ldots, S_{n/2}$.

The general intuition of the proof of Lemma 28 is that while one tree could give a high lower bound for *one* of the sequences S_i, no tree can give a high lower bound *on average* over all S_i. The reason is that, given the geometric spacing of each S_i, any way to split an interval of keys into two will typically (on average over i) leave almost all the keys of S_i in either the left or the right part (Claim 31). Therefore, it is impossible to split the keys into subtrees in a way that would ensure a high number of alternations.

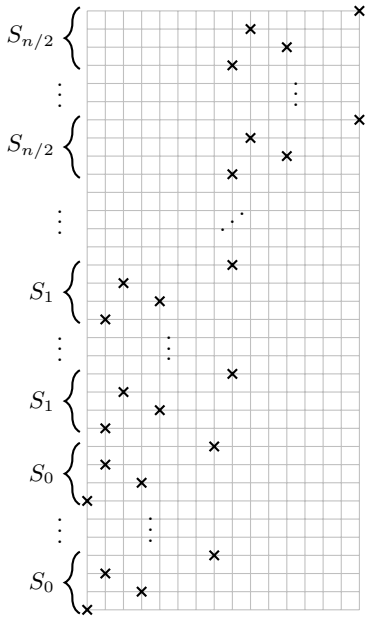


Figure 14 A schematic view of sequence $\tilde{X}$ for $k = 2$. Each part S_i is made of $K = 2^k = 4$ accesses. There are $n = 2^K = 16$ distinct keys and the length of $\tilde{X}$ is $m = (16/2 + 1)nK = 576$.

Proof of Lemma 28. The first step of the proof is to decompose $\tilde{X}$ into substrings $S_0 \circ \cdots \circ S_0$ through $S_{n/2} \circ \cdots \circ S_{n/2}$, and then bound the sum of their Alternation bounds. Let's denote those substrings as $S_0 * n$, $S_1 * n$, …, $S_{n/2} * n$. Because of the subadditivity of mixValue under concatenation (Fact 9), we have

$$\mathsf{Alt}_{\mathcal{T}}(\tilde{X}) \leq \sum_{i=0}^{n/2} \mathsf{Alt}_{\mathcal{T}}(S_i * n). \tag{5}$$

Note that we don't want to decompose $\tilde{X}$ down to the S_i's themselves: every time we split it, our analysis loses up to an additive $O(n)$ in precision. Intuitively, this $O(n)$ is due to a "warmup" cost which we might or might not incur at the beginning of each substring, depending on which parts of the tree were last visited. With our decomposition into n substrings, that's an extra $O(n^2)$ cost, which is okay since it is small compared to the total length of the sequence $\Theta(n^2 \log n)$. In fact, this is precisely why we repeated each S_i several times: if we had defined $\tilde{X}$ as $S_0 \circ S_1 \circ \cdots \circ S_{n/2}$ instead, this $O(n^2)$ would have been large compared to the length of the sequence $\Theta(n \log n)$.[11]

We will upper-bound the sum $\sum_i \mathsf{Alt}_{\mathcal{T}}(S_i * n)$ by induction on the recursive definition of $\mathsf{Alt}_{\mathcal{T}}(\cdot)$. Concretely, let $\mathcal{T}^*$ be a subtree of $\mathcal{T}$, and let $\mathcal{T}_{\mathtt{L}}{}^*, \mathcal{T}_{\mathtt{R}}{}^*$ be the left and right subtrees of $\mathcal{T}^*$. Let s, $s_{\mathtt{L}}$ and $s_{\mathtt{R}}$ be the number of keys in $\mathcal{T}^*$, $\mathcal{T}_{\mathtt{L}}{}^*$ and $\mathcal{T}_{\mathtt{R}}{}^*$ (note that $s = s_{\mathtt{L}} + s_{\mathtt{R}}$). For each i, let P_i^* be the subset of $P(S_i * n)$ corresponding to keys in $\mathcal{T}^*$, and let $P_{i,\mathtt{L}}^*, P_{i,\mathtt{R}}^*$ be the same for $\mathcal{T}_{\mathtt{L}}{}^*$ and $\mathcal{T}_{\mathtt{R}}{}^*$. We will prove the following claim by induction:

[11] The astute reader will notice that we could have repeated each S_i only $\Theta(n/\log n)$ times instead of n times. But we are not limited in terms of the length of $\tilde{X}$, so it was (notationally) simpler to repeat them n times.

▷ Claim 30. For some constant $C > 0$,

$$\sum_{i=0}^{n/2} \mathsf{Alt}_{\mathcal{T}^*}(P_i^*) \leq (s-1)(n/2+1) + 2Cns \lg s.$$

The base case is when $\mathcal{T}^*$ is a single node. Then $\mathsf{Alt}_{\mathcal{T}^*}(S_i * n) = 0$ for all i, while $s = 1$, so the result holds. To deal with the inductive step, we will need make a few tools first. By definition of the Alternation bound (Definition 10), for each i we have

$$\mathsf{Alt}_{\mathcal{T}^*}(P_i^*) = a(P_i^*, \mathcal{T}^*) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{L}}{}^*}(P_{i,\mathtt{L}}^*) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{R}}{}^*}(P_{i,\mathtt{R}}^*). \tag{6}$$

The challenging part is how to deal with $a(P_i^*, \mathcal{T}^*)$. By Fact 9, we have

$$a(P_i^*, \mathcal{T}^*) = \text{mixValue}(P_{i,\mathtt{L}}^*.y, P_{i,\mathtt{R}}^*.y) \leq 2 \cdot \min(|P_{i,\mathtt{L}}^*|, |P_{i,\mathtt{R}}^*|) + 1.$$

Summing this up over all i, we get

$$\sum_{i=0}^{n/2} a(P_i^*, \mathcal{T}^*) \leq \sum_{i=0}^{n/2} (2 \cdot \min(|P_{i,\mathtt{L}}^*|, |P_{i,\mathtt{R}}^*|) + 1) = (n/2+1) + 2 \cdot \sum_{i=0}^{n/2} \min(|P_{i,\mathtt{L}}^*|, |P_{i,\mathtt{R}}^*|). \tag{7}$$

▷ Claim 31. For some constant $C > 0$,

$$\sum_{i=0}^{n/2} \min(|P_{i,\mathtt{L}}^*|, |P_{i,\mathtt{R}}^*|) \leq Cn \cdot \begin{cases} s_{\mathtt{L}} \lg \frac{s}{s_{\mathtt{L}}} \text{ if } s_{\mathtt{L}} \leq s_{\mathtt{R}}, \text{ and} \\ s_{\mathtt{R}} \lg \frac{s}{s_{\mathtt{R}}} \text{ if } s_{\mathtt{R}} \leq s_{\mathtt{L}}. \end{cases}$$

This left-right symmetry is very surprising given that the sequences S_i themselves are *not* left-right symmetric. But it will be very convenient.

Proof. To simplify the notation, let's say that the keys in $\mathcal{T}_{\mathtt{L}}{}^*$ are in range $[a, b]$ and the keys in $\mathcal{T}_{\mathtt{R}}{}^*$ are in range $[b, c]$, for some real numbers a, b, c with $b - a = s_{\mathtt{L}}$ and $c - b = s_{\mathtt{R}}$.[12]

For each i, let $V_i = \{i + 2^0, \ldots, i + 2^{K-1}\}$ be the set of values that are hit by sequence S_i. Then $|P_{i,\mathtt{L}}^*|$ (resp. $|P_{i,\mathtt{L}}^*|$) is exactly n times the number of elements of V_i that are in $[a, b]$ (resp. $[b, c]$). Let's name this number of keys l_i (resp. r_i). We will instead prove that

$$\sum_{i=0}^{n/2} \min(l_i, r_i) \leq O\left(s_{\mathtt{L}} \lg \frac{s}{s_{\mathtt{L}}}\right) \text{ if } s_{\mathtt{L}} \leq s_{\mathtt{R}}, \text{ and} \tag{8}$$

$$\sum_{i=0}^{n/2} \min(l_i, r_i) \leq O\left(s_{\mathtt{R}} \lg \frac{s}{s_{\mathtt{R}}}\right) \text{ if } s_{\mathtt{R}} \leq s_{\mathtt{L}}. \tag{9}$$

Once this is proved, C can be set to the maximum of the two constants hidden inside the $O(\cdot)$s. Those constants might be different since the reasonings leading to (8) and (9) are completely different.

We first make a general observation. Look at set $V_i = \{i + 2^0, \ldots, i + 2^j, \ldots, \}$ in increasing order. Note that after $i + 2^j$, all further elements are spaced by at least 2^j. In order for $\min(l_i, r_i)$ to be non-zero, we need to have at least two elements of S_i in $[a, c]$: specifically, one in $[a, b]$ and one in $[b, c]$. But this means that $i + 2^{j+1} \in [a, c]$ isn't acceptable for $j > \lg s$:

[12] We can for example fix a to the first key of $\mathcal{T}_{\mathtt{L}}{}^*$ minus $\frac{1}{2}$, b to the last key of $\mathcal{T}_{\mathtt{L}}{}^*$ plus $\frac{1}{2}$, and c to the last key of $\mathcal{T}_{\mathtt{R}}{}^*$ plus $\frac{1}{2}$.

indeed, the closest other point in S_i is more than s away, so it must be outside of $[a,c]$. Therefore, in bounding $\sum \min(l_i, r_i)$, it is fine to imagine that the elements $i + 2^{j+1}$ for $j > \lg s$ simply do not exist.

Let us now prove (8). Assume $s_{\mathtt{L}} \leq s_{\mathtt{R}}$. We split into two cases:

- "Far" case: $i < a - s_{\mathtt{L}}$. Since i is further from $[a,b]$ than its size $s_{\mathtt{L}}$, this means that $[a,b]$ can only contain at most one point from S_i. So $l_i \leq 1$. Besides, that (potential) single point must have $j \leq 1 + \lg s$ (see above) and $j \geq \lg s_{\mathtt{L}}$ (because we have $i + 2^j \geq a$). And of course, we have in addition that $i + 2^j \in [a,b]$. Therefore, this limits the number of possible values of i to at most $s_{\mathtt{L}}(2 + \lg s - \lg s_{\mathtt{L}})$, and since $l_i \leq 1$, this also limits the total contribution to $\sum \min(l_i, r_i)$.
- "Close to right" case: $i \geq a - s_{\mathtt{L}}$. Then we also have $i \geq b - 2s_{\mathtt{L}}$. Since we need $l_i \neq 0$ to have some contribution, we must have $i < b$, so the total number of possible values of i is limited to $2s_{\mathtt{L}}$. Let's consider the values of j such that $i + 2^j$ can lie in $[b,c]$, the right part. We already know that $j \leq 1 + \lg s$, but we have no lower limit, as i could be very close to b. However, values of j much smaller than $\lg s_{\mathtt{L}}$ will be only for the few values of i close enough to b.

  More precisely, we study the contribution of each j to $\sum r_i$ into two groups:
  - $j \geq \lg s_{\mathtt{L}}$: there are $2 + \lg s - \lg s_{\mathtt{L}}$ such values j, and there are $2s_{\mathtt{L}}$ possible values of i, so the total contribution is at most $2s_{\mathtt{L}}(2 + \lg s - \lg s_{\mathtt{L}})$.
  - $j < \lg s_{\mathtt{L}}$: as j decreases, the number of acceptable values of i decreases exponentially. The number of values of i for which $i + 2^j \in [b,c]$ for $j \leq \lg s_{\mathtt{L}} - l$ is at most $s_{\mathtt{L}}/2^l$. Therefore, the overal contribution is at most $s_{\mathtt{L}} + s_{\mathtt{L}}/2 + \cdots \leq 2s_{\mathtt{L}}$.

All those quantities are upper bounded by $O(s_{\mathtt{L}}(1 + \lg(s/s_{\mathtt{L}})))$, which under the assumption $s_{\mathtt{L}} \leq s_{\mathtt{R}}$, is also bounded by $O(s_{\mathtt{L}} \lg(s/s_{\mathtt{L}}))$.

We now prove (9) in a very similar way. Assume $s_{\mathtt{L}} \leq s_{\mathtt{R}}$.

- "Far" case: $i < b - s_{\mathtt{R}}$. The argument is analogous to the "far" case for (8), but considering r_i this time. We obtain a contribution of at most $s_{\mathtt{R}}(2 + \lg s - \lg s_{\mathtt{R}})$.
- "Close to right" case: $i \geq b - s_{\mathtt{R}}$. The argument is analogous to the "close to right" case for (8), but with a distance of $s_{\mathtt{R}}$ instead of $2s_{\mathtt{L}}$ this time. We obtain contributions of at most $s_{\mathtt{R}}(2 + \lg s - \lg s_{\mathtt{R}})$ and $2s_{\mathtt{R}}$ for the two subcases.

All those quantities are upper bounded by $O(s_{\mathtt{R}}(1 + \lg(s/s_{\mathtt{R}})))$, which under the assumption $s_{\mathtt{R}} \leq s_{\mathtt{L}}$, is also bounded by $O(s_{\mathtt{R}} \lg(s/s_{\mathtt{R}}))$. ◁

We are now ready to finish the induction step.

Proof of Claim 30. We define C to be the same as in Claim 31. We have

$$\begin{aligned}
\sum_{i=0}^{n/2} \mathsf{Alt}_{\mathcal{T}^*}(P_i^*) &= \sum_{i=0}^{n/2} \left(a(P_i^*, \mathcal{T}^*) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{L}}^*}(P_{i,\mathtt{L}}^*) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{R}}^*}(P_{i,\mathtt{R}}^*)\right) && \text{(by (6))} \\
&\leq \left(\sum_{i=0}^{n/2} a(P_i^*, \mathcal{T}^*)\right) + (s_{\mathtt{L}} - 1)(n/2+1) + 2Cns_{\mathtt{L}} \lg s_{\mathtt{L}} + (s_{\mathtt{R}} - 1)(n/2+1) + 2Cns_{\mathtt{R}} \lg s_{\mathtt{R}} \\
& && \text{(inductive hypothesis)} \\
&\leq (n/2+1) + 2 \cdot \sum_{i=0}^{n/2} \min(|P_{i,\mathtt{L}}^*|, |P_{i,\mathtt{R}}^*|) \\
&\quad + (s_{\mathtt{L}} - 1)(n/2+1) + 2Cns_{\mathtt{L}} \lg s_{\mathtt{L}} + (s_{\mathtt{R}} - 1)(n/2+1) + 2Cns_{\mathtt{R}} \lg s_{\mathtt{R}} && \text{(by (7))} \\
&\leq (s-1)(n/2+1) + 2Cn(s_{\mathtt{L}} \lg s_{\mathtt{L}} + s_{\mathtt{R}} \lg s_{\mathtt{R}}) + 2 \cdot \sum_{i=0}^{n/2} \min(|P_{i,\mathtt{L}}^*|, |P_{i,\mathtt{R}}^*|) && (s = s_{\mathtt{L}} + s_{\mathtt{R}})
\end{aligned}$$

All we need to show is that

$$Cn(s_{\mathtt{L}} \lg s_{\mathtt{L}} + s_{\mathtt{R}} \lg s_{\mathtt{R}}) + \sum_{i=0}^{n/2} \min(|P^*_{i,\mathtt{L}}|, |P^*_{i,\mathtt{R}}|) \leq Cn(s \lg s).$$

Let's assume that $s_{\mathtt{L}} \leq s_{\mathtt{R}}$ (the other case is identical). Then by Claim 31,

$$\begin{aligned} Cn(s_{\mathtt{L}} \lg s_{\mathtt{L}} + s_{\mathtt{R}} \lg s_{\mathtt{R}}) + \sum_{i=0}^{n/2} \min(|P^*_{i,\mathtt{L}}|, |P^*_{i,\mathtt{R}}|) &\leq Cn(s_{\mathtt{L}} \lg s_{\mathtt{L}} + s_{\mathtt{R}} \lg s_{\mathtt{R}}) + Cns_{\mathtt{L}} \lg \frac{s}{s_{\mathtt{L}}} \\ &\leq Cn(s_{\mathtt{L}} \lg s + s_{\mathtt{R}} \lg s_{\mathtt{R}}) \\ &\leq Cn(s_{\mathtt{L}} \lg s + s_{\mathtt{R}} \lg s) \\ &= Cns \lg s. \end{aligned}$$

This completes the proof of Claim 30. ◁

Applying Claim 30 to the full tree $\mathcal{T}$, which has n keys, we get

$$\begin{aligned} \mathsf{Alt}_{\mathcal{T}}(\tilde{X}) &\leq \sum_{i=0}^{n/2} \mathsf{Alt}_{\mathcal{T}^*}(S_i * n) && \text{(by (5))} \\ &\leq (n-1)(n/2+1) + 2Cn^2 \lg n && \text{(Claim 30)} \\ &\leq O(n^2 \lg n) \\ &= O(m). \end{aligned}$$ ◀

We now move to the proof of Lemma 29, which is much simpler.

Proof of Lemma 29. From the definition of $\mathsf{Funnel}(\cdot)$ (Definition 14), it is easy to see that for any two sequences S and T, $\mathsf{Funnel}(S \circ T) \geq \mathsf{Funnel}(S) + \mathsf{Funnel}(T)$. Indeed concatenating S and T does not affect the funnel of each point in S, and can only add points to the funnel of each point in T. Therefore,

$$\mathsf{Funnel}(\tilde{X}) \geq n \sum_{i=0}^{n/2} \mathsf{Funnel}(S_i). \tag{10}$$

Since $\mathsf{Funnel}(\cdot)$ only depends on the relative order of the keys in the access sequence, not on their exact value, we have $\mathsf{Funnel}(S_i) = \mathsf{Funnel}(\text{bitReversal}^k)$ for each i. Besides, defining $\mathcal{T}$ to be the complete binary search tree of height k as in Fact 27, we have

$$\begin{aligned} \mathsf{Funnel}(\text{bitReversal}^k) &\geq \Omega(\mathsf{Alt}_{\mathcal{T}}(\text{bitReversal}^k)) - K && \text{(by Theorem 1)} \\ &\geq \Omega(K \lg K) - K && \text{(by Fact 27)} \\ &\geq \Omega(K \lg K). \end{aligned}$$

Combined with (10), this gives $\mathsf{Funnel}(\tilde{X}) \geq n \cdot (n/2+1) \cdot \Omega(K \lg K) \geq \Omega(m \lg K) = \Omega(m \lg \lg n)$. ◀

Due to space constraints, the last (short) section, which relates the Funnel bound to the Independent Rectangle bound, is deferred to the full version.

References

1. Brian Allen and Ian Munro. Self-organizing binary search trees. *J. ACM*, 25(4):526–535, October 1978. doi:10.1145/322092.322094.
2. Prosenjit Bose, Karim Douïeb, Vida Dujmovic, and Rolf Fagerberg. An $O(\log \log n)$-competitive binary search tree with optimal worst-case access times. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, pages 38–49, 2010. doi:10.1007/978-3-642-13731-0_5.
3. Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak. Pinning down the strong wilber 1 bound for binary search trees. *CoRR*, abs/1912.02900, 2019. arXiv:1912.02900.
4. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 410–423. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.32.
5. Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Patrascu. The geometry of binary search trees. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 496–505, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496825.
6. Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality - almost. *SIAM J. Comput.*, 37(1):240–251, 2007. doi:10.1137/S0097539705447347.
7. John Iacono. Key-independent optimality. *Algorithmica*, 42(1):3–10, 2005. doi:10.1007/s00453-004-1136-8.
8. John Iacono. In pursuit of the dynamic optimality conjecture. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 236–250, 2013. doi:10.1007/978-3-642-40273-9_16.
9. László Kozma and Thatchaphol Saranurak. Smooth heaps and a dual view of self-adjusting data structures. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 801–814, 2018. doi:10.1145/3188745.3188864.
10. Victor Lecomte and Omri Weinstein. Settling the relationship between wilber's bounds for dynamic optimality. *CoRR*, abs/1912.02858, 2019. arXiv:1912.02858.
11. Caleb C. Levy and Robert E. Tarjan. A new path from splay to dynamic optimality. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1311–1330. SIAM, 2019. doi:10.1137/1.9781611975482.80.
12. Joan Marie Lucas. *Canonical forms for competitive binary search tree algorithms*. Rutgers University, Department of Computer Science, Laboratory for Computer Science Research, 1988.
13. J. Ian Munro. On the competitiveness of linear search. In Mike Paterson, editor, *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345. Springer, 2000. doi:10.1007/3-540-45253-2_31.
14. Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985. doi:10.1145/3828.3835.
15. Chengwen Chris Wang, Jonathan Derryberry, and Daniel Dominic Sleator. O(log log n)-competitive dynamic binary search trees. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 374–383, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109600.
16. R. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM J. Comput.*, 18(1):56–67, February 1989. doi:10.1137/0218004.

On the Computational Complexity of Linear Discrepancy

Lily Li
Department of Computer Science, University of Toronto, Canada
xinyuan@cs.toronto.edu

Aleksandar Nikolov
Department of Computer Science, University of Toronto, Canada
anikolov@cs.toronto.edu

Abstract

Many problems in computer science and applied mathematics require rounding a vector $\mathbf{w}$ of fractional values lying in the interval $[0, 1]$ to a binary vector $\mathbf{x}$ so that, for a given matrix $\mathbf{A}$, $\mathbf{Ax}$ is as close to $\mathbf{Aw}$ as possible. For example, this problem arises in LP rounding algorithms used to approximate NP-hard optimization problems and in the design of uniformly distributed point sets for numerical integration. For a given matrix $\mathbf{A}$, the worst-case error over all choices of $\mathbf{w}$ incurred by the best possible rounding is measured by the linear discrepancy of $\mathbf{A}$, a quantity studied in discrepancy theory, and introduced by Lovasz, Spencer, and Vesztergombi (EJC, 1986).

We initiate the study of the computational complexity of linear discrepancy. Our investigation proceeds in two directions: (1) proving hardness results and (2) finding both exact and approximate algorithms to evaluate the linear discrepancy of certain matrices. For (1), we show that linear discrepancy is NP-hard. Thus we do not expect to find an efficient exact algorithm for the general case. Restricting our attention to matrices with a constant number of rows, we present a poly-time exact algorithm for matrices consisting of a single row and matrices with a constant number of rows and entries of bounded magnitude. We also present an exponential-time approximation algorithm for general matrices, and an algorithm that approximates linear discrepancy to within an exponential factor.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases discrepancy theory, linear discrepancy, rounding, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.69

Funding This research was supported by an NSERC Discovery Grant (application number RGPIN-2016-06333).

1 Notation

Below bold-face capital letters such as $\mathbf{A}$ denote matrices; $A_{i,j}$ is the entry in the i^{th} row and j^{th} column of the matrix $\mathbf{A}$. Bold-face lowercase letters such as $\mathbf{x}$ denote vectors; x_i is the i^{th} entry of the vector $\mathbf{x}$. We denote the all ones vector by $\mathbb{1}$. Whenever we write $i \in [n]$, we mean $i \in \{1, 2, ..., n\}$. For simplicity of exposition, assume that $\mathbf{A} \in \mathbb{Q}^{m \times n}$ in the following sections. We will explicitly indicate other uses of $\mathbf{A}$.

2 Introduction

A number of questions in mathematics and computer science can be reduced to the following basic rounding question: given a vector $\mathbf{w} \in [0, 1]^n$, and an $m \times n$ matrix $\mathbf{A}$, find an integer vector $\mathbf{x} \in [0, 1]^n$ such that $\mathbf{Ax}$ is as close as possible to $\mathbf{Aw}$. For example, many NP-hard optimization problems can be modeled as an integer program

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \{0,1\}^n \end{aligned}$$

This integer program can be relaxed to a linear program by replacing the integer variables $\mathbf{x} \in \{0,1\}^n$ with real-valued variables $\mathbf{w} \in [0,1]^n$. A powerful method in approximation algorithms is to solve this linear programming relaxation to get an optimal $\mathbf{w}$, and then round $\mathbf{w}$ to an integer solution $\mathbf{x}$ which is feasible (i.e., $\mathbf{A}\mathbf{x} \geq \mathbf{b}$), and has objective value not much bigger than $\mathbf{c}^\top \mathbf{w}$. Often, a useful intermediate step is to guarantee that $\mathbf{x}$ is approximately optimal, i.e., that the coordinates of $\mathbf{b} - \mathbf{A}\mathbf{x}$ are bounded from above. This approximately feasible solution can then, hopefully, be turned into a truly feasible one with a small loss in the objective value. This method was used, for example, by Rothvoss [18], and Rothvoss and Hoberg [12] to give the best known approximation algorithm for the bin packing problem.

Another example is provided by the problem of constructing uniformly distributed points, or, more generally, points that are well-distributed with respect to some measure. Variants of this problem date back to work by Weyl, van der Corput, van Aardenne-Ehrenfest, and Roth, and have important applications to such fields as numerical integration; see the book of Matoušek [15] for references and an introduction to the area. In the classical setting, the problem is to find, for any positive integer n, a set of n points P in $[0,1]^d$, so as to minimize the quantity

$$\sup_{R \in \mathcal{R}_d} ||R \cap P| - n\lambda_d(R)|,$$

where $\mathcal{R}_d$ is the set of all axis-aligned boxes contained in $[0,1]^d$, and λ_d is the Lebesgue measure on $\mathbb{R}^d$. The quantity above is known as the (unnormalized) discrepancy of P. Note that if we sample a random point uniformly from P, then it would land in R with probability $\frac{|R \cap P|}{n}$; on the other hand, if we sample a random point uniformly from $[0,1]^d$, then it would land in R with probability $\lambda_d(R)$. The problem of minimizing the discrepancy of P is then equivalent to finding a distribution that is uniform over n points that "looks the same" as the continuous uniform distribution to all boxes R.

The discrepancy minimization problem can be modeled by the rounding problem with which we started our discussion. To that end, we can discretize the domain $[0,1]^d$ to a finite set X of size N, and let $\mathbf{A}$ be the incidence matrix of sets induced by axis-aligned boxes, i.e., each row of $\mathbf{A}$ is associated with a box R, and equals the indicator vector of $R \cap X$. If we also let $\mathbf{w} = \frac{n}{N}\mathbb{1}$, where $\mathbb{1}$ is the all-ones vector, then, for a sufficiently fine discretization, each coordinate of $\mathbf{A}\mathbf{w}$ is a close approximation of $n\lambda_d(R)$. The problem of finding an n-point set P of minimum discrepancy then becomes essentially equivalent to minimizing $\|\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{w}\|_\infty$ over $\mathbf{x} \in \{0,1\}^N$, where $\|\cdot\|_\infty$ is the standard ℓ_∞ norm. In particular, given $\mathbf{x}$, we can take P to consist of the points in X for which the corresponding coordinate in $\mathbf{x}$ is set to 1. Then, since $[0,1]^d \in \mathcal{R}_d$, we have $||P| - n| \leq \|\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{w}\|_\infty$, and we can remove or add at most $\|\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{w}\|_\infty$ to P to make it exactly of size n. The discrepancy of P is then bounded by $2\|\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{w}\|_\infty$ plus the additional error incurred by the discretization of $[0,1]^d$.

These two examples motivate the definition of *linear discrepancy*, initially introduced by Lovász, Spencer, and Vesztergombi [14]. The smallest possible error for rounding $\mathbf{w}$ with respect to $\mathbf{A}$ is

$$\operatorname{lindisc}(\mathbf{A}, \mathbf{w}) = \min_{\mathbf{x} \in \{0,1\}^n} \|\mathbf{A}(\mathbf{w} - \mathbf{x})\|_\infty.$$

This is the linear discrepancy of $\mathbf{A}$ with respect to $\mathbf{w}$. The linear discrepancy of $\mathbf{A}$ is defined as as the worse case over all $\mathbf{w} \in [0,1]^n$, i.e.,

$$\operatorname{lindisc}(\mathbf{A}) = \max_{\mathbf{w}\in[0,1]^n} \operatorname{lindisc}(\mathbf{A}, \mathbf{w}). \tag{1}$$

It will be useful to consider a maximizer of equation (1) i.e. $\mathbf{w}^* \in [0,1]^n$ such that $\operatorname{lindisc}(\mathbf{A}, \mathbf{w}^*) = \operatorname{lindisc}(\mathbf{A})$. We call $\mathbf{w}^*$ a *deep-hole* of $\mathbf{A}$. Every $\mathbf{A}$ has at least one deep-hole, since linear discrepancy is a continuous function over the compact set $[0,1]^n$.

The special case of $\operatorname{lindisc}(\mathbf{A}, \mathbf{w})$ when $\mathbf{w} = \frac{1}{2}\mathbb{1}$ is especially well studied. When $\mathbf{A}$ is the indicator matrix of a collection $\mathcal{S}$ of m subsets of a universe X, $\operatorname{lindisc}(\mathbf{A}, \frac{1}{2}\mathbb{1})$ measures to what extent it is possible to choose a subset S of X that contains approximately half the elements of each set. This is a rescaling of the well-known combinatorial discrepancy of $\mathcal{S}$, defined as

$$\operatorname{disc}(\mathcal{S}) = \min_{\chi: X\to\{-1,+1\}} \max_{S\in\mathcal{S}} \left|\sum_{s\in S} \chi(s)\right|$$

It is straightforward to check that, by a change of variables, $\operatorname{disc}(\mathcal{S}) = 2\cdot\operatorname{lindisc}(\mathbf{A}, \frac{1}{2}\mathbb{1})$ where, again, $\mathbf{A}$ is the incidence matrix of $\mathcal{S}$. This definition can be extended to arbitrary matrices $\mathbf{A}$ as $\operatorname{disc}(\mathbf{A}) = 2\cdot\operatorname{lindisc}(\mathbf{A}, \frac{1}{2}\mathbb{1})$. Combinatorial discrepancy has been widely studied in combinatorics and computer science, see [4, 8, 15].

Sometimes $\operatorname{disc}(\mathbf{A})$ can be small "by accident", thus it is useful to define a more robust discrepancy variant.[1] The hereditary discrepancy of $\mathbf{A}$ is the maximum discrepancy over all sub-matrices, i.e.,

$$\operatorname{herdisc}(\mathbf{A}) = \max_{\mathbf{B}} \operatorname{disc}(\mathbf{B}), \tag{2}$$

where $\mathbf{B}$ ranges over submatrices of $\mathbf{A}$.

A fundamental theorem by Lovász, Spencer, and Vesztergombi shows that linear discrepancy can be bounded above by twice the hereditary discrepancy.

▶ **Theorem 1** (Lovász et al. 1986, [14]). $\operatorname{lindisc}(\mathbf{A}) \le 2\cdot\operatorname{herdisc}(\mathbf{A})$.

A number of the applications of combinatorial discrepancy use this basic theorem. In particular, it is common to give an upper bound on the hereditary discrepancy, and from that deduce an upper bound on the linear discrepancy. For example, this strategy was used to give approximation algorithms for bin packing [18, 12], and broadcast scheduling [3], and to design point sets well distributed with respect to arbitrary Borel measures [17, 1]. However, linear discrepancy can be much smaller (by a factor of at least 2^n) than hereditary discrepancy,[2] so hereditary discrepancy lower bounds do not translate to linear discrepancy, and, in general, linear discrepancy lower bounds appear to be challenging. Arguably, a better understanding of linear discrepancy itself would allow proving more and tighter results, in comparison with going through hereditary discrepancy. For example, it is likely that new

[1] Consider any matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$ and let $\mathbf{A} \in \mathbb{R}^{m\times 2n}$ be the concatenation of two copies of $\mathbf{B}$ side by side. Regardless of the discrepancy of $\mathbf{B}$, $\operatorname{disc}(\mathbf{A}) = 0$ since there exists $\mathbf{x} \in \{-1,1\}^n$ such that $\|\mathbf{Ax}\|_\infty = 0$, namely

$$\mathbf{x}^\intercal = [\underbrace{-1,\ldots,-1}_{n},\underbrace{1,\ldots,1}_{n}].$$

[2] Whether this remains true if the matrix $\mathbf{A}$ has bounded entries is a tantalizing open question.

analytic tools to estimate linear discrepancy would allow progress on questions in geometric discrepancy theory, as well as questions about the integrality gaps of linear programming relaxations of important optimization problems, such as the bin packing problem.

A sequence of recent works has shed light on the computational complexity of combinatorial and hereditary discrepancy. It is now known that combinatorial discrepancy does not allow efficient approximation algorithms, even in a weak sense (assuming $\mathsf{P} \neq \mathsf{NP}$)[6], while hereditary discrepancy is NP-hard to approximate better than a factor of 2 [2], and can be approximated within poly-logarithmic factors [16]. Despite being the tool most directly relevant to many applications of discrepancy, however, essentially nothing is known about the computational complexity of linear discrepancy itself. In this paper, we initiate the study of linear discrepancy from a computational viewpoint, and give both the first hardness results, as well as the first exact and approximate algorithms for it.

Before stating our results, it is worth mentioning that linear discrepancy can also be seen as an analogue of the covering radius in lattice theory. Let $\Lambda \subset \mathbb{R}^n$ be a lattice, i.e. discrete additive subgroup of $\mathbb{R}^n$, and let us choose $\mathbf{b}_1, \ldots, \mathbf{b}_n$ to be a basis of Λ. Let $\mathbf{B}$ be a matrix with the $\mathbf{b_i}$ as its columns. The covering radius of Λ in the ℓ_p-norm is defined as

$$\rho(\Lambda) = \max_{\mathbf{y}\in\mathbb{R}^n} \min_{\mathbf{z}\in\Lambda} \|\mathbf{y}-\mathbf{z}\|_p = \max_{\mathbf{w}\in\mathbb{R}^n} \min_{\mathbf{x}\in\mathbb{Z}^n} \|\mathbf{B}\cdot(\mathbf{w}-\mathbf{x})\|_p = \max_{\mathbf{w}\in[0,1]^n} \min_{\mathbf{x}\in\mathbb{Z}^n} \|\mathbf{B}\cdot(\mathbf{w}-\mathbf{x})\|_p, \quad (3)$$

and is independent of the basis. This definition is equivalent to the the definition of $\mathrm{lindisc}(\mathbf{A})$, except that the minimum is over $\mathbb{Z}^n$ rather than $\{0,1\}^n$. Haviv and Regev showed that the covering radius problem (CRP) in the ℓ_p-norm is Π_2-hard to approximate within some fixed constant for all large enough p [11], and Guruswami, Micciancio, and Regev showed it can be approximated within a factor of $2^{O(n \log n/\log\log n)}$ for the case of $p = 2$ [10].

2.1 Our Results

Let us start with the simple observation that, when $\mathbf{A}$ is a single row matrix, deciding $\mathrm{lindisc}(\mathbf{A}, t\mathbb{1}) = 0$ is the NP-hard Subset Sum problem with target sum $t\sum_{j=1}^{n} A_{1,j}$, and is, therefore, NP-hard. This does not show, however, that computing $\mathrm{lindisc}(\mathbf{A})$ is NP-hard. In this work we show the following hardness result for linear discrepancy.

▶ **Theorem 2.** *The Linear Discrepancy problem of deciding, given an $m \times n$ matrix $\mathbf{A}$ with rational entries, and a rational number t, whether* $\mathrm{lindisc}(\mathbf{A}) \leq t$, *is NP-hard and is contained in the class Π_2.*

We present algorithms for computing linear discrepancy exactly when the matrix $\mathbf{A}$ has a constant number of rows. We start with a result for a single row matrix.

▶ **Theorem 3.** *For any matrix $\mathbf{A} \in \mathbb{Q}^{1\times n}$,* $\mathrm{lindisc}(\mathbf{A})$ *can be computed in time $O(n \log n)$.*

Note that this stands in contrast to the observation above that computing $\mathrm{lindisc}(\mathbf{A}, \mathbf{w})$ is hard even for a single-row matrix $\mathbf{A}$. In addition to the theorem above, we also give a corresponding rounding result, showing that any $\mathbf{w} \in \mathbb{Q}^n$ can be efficiently rounded to within error bounded by the linear discrepancy in the case of single row matrices.

▶ **Theorem 4.** *For any matrix $\mathbf{A} \in \mathbb{Q}^{1\times n}$ and any $\mathbf{w} \in ([0,1] \cap \mathbb{Q})^n$, we can find an $\mathbf{x} \in \{0,1\}^n$ such that $\|\mathbf{A}(\mathbf{w}-\mathbf{x})\|_\infty \leq \mathrm{lindisc}(\mathbf{A})$ in time $O(n \log n)$.*

This result stands in contrast with the hardness of the subset sum problem, which easily implies that it is NP-hard to round $\mathbf{w}$ to within error $\mathrm{lindisc}(\mathbf{A}, \mathbf{w})$ even when $\mathbf{A}$ is a single row matrix.

We can extend Theorem 3 to the case of matrices with a bounded number of rows, with the additional assumption that the entries of $\mathbf{A}$ are bounded. Removing this additional assumption is a fascinating open question.

▶ **Theorem 5.** *For any matrix* $\mathbf{A} \in \mathbb{Z}^{d\times n}$ *where* d *is some fixed constant and* $\max_{i,j} |A_{i,j}| \leq \delta$, $\text{lindisc}(\mathbf{A})$ *can be computed in time* $O\left(d(n\delta)^{d^2+d}\right)$.

We further present an approximation algorithm for linear discrepancy.

▶ **Theorem 6.** *For any matrix* $\mathbf{A} \in \mathbb{Q}^{m\times n}$, $\text{lindisc}(\mathbf{A})$ *can be approximated in polynomial time within a factor of* 2^{n+1}.

3 Hardness Result

In this section, we show that linear discrepancy (LDS) is NP-Hard by reducing from monotone not-all-equal 3-SAT (MNAE3SAT) [9] to each. The decision problem version of linear discrepancy we consider is defined below.

[MNAE3SAT] **Monotone Not-All-Equal 3-SAT**
Let U be a collection of variables $\{u_1, ..., u_n\}$ and $\mathcal{C}$ be a 3-CNF with clauses $\{C_1, ..., C_m\}$ such that $C_i = t_{i,1} \vee t_{i,2} \vee t_{i,3}$ for positive literals $t_{i,j}$.
Question: Does there exist a truth assignment $\tau : U \to \{\mathsf{T}, \mathsf{F}\}$ such that $\mathcal{C}$ is satisfied and each clause has at least one true and one false literal?

[LDS] **Linear Discrepancy**
Let $\mathbf{A} \in \mathbb{Q}^{m\times n}$ be a matrix and $t \geq 0$ a rational value.
Question: Is $\text{lindisc}(\mathbf{A}) \leq t$?

3.1 Linear Discrepancy

Before we show that linear discrepancy is hard, we will show that the value of $\text{lindisc}(\mathbf{A})$ can be expressed using a polynomial number of bits in the bit complexity of a matrix for rational matrices. Due to space constraints, the proof can be found in Section A.1.

▶ **Lemma 7.** *For any matrix* $\mathbf{A} \in \mathbb{Q}^{m\times n}$, *there exists a deep hole for* $\mathbf{A}$ *with bit complexity polynomial in* n *and the bit complexity of* $\mathbf{A}$, *and, therefore,* $\text{lindisc}(\mathbf{A})$ *can be written in number of bits polynimial in* n *and the bit complexity of* $\mathbf{A}$.

Proof of Theorem 2. Note first that the fact that LDS is contained in Π_2 is a straightforward consequence of Lemma 7: the "for-all" quantifier is over potential deep holes $\mathbf{w} \in [0, 1]^n$ of the appropriate polynomially bounded bit complexity, and the "exists" quantifier is over $\mathbf{x} \in \{0, 1\}^n$.

Next we prove hardness. Let 3-CNF $\mathcal{C}$ be a MNAE3SAT instance as described above. The corresponding LDS instance will be the incidence matrix $\mathbf{A} \in \{0, 1\}^{m\times n}$ of $\mathcal{C}$: column $\mathbf{a}_j$ of $\mathbf{A}$ corresponds to variable u_j and row $\mathbf{r}_i$ of $\mathbf{A}$ corresponds to clause C_i, and $A_{i,j} = 1$ if and only if variable u_j appears in clause C_i. Let the target t in the LDS problem be $\frac{3}{2} - \epsilon$ for $\epsilon > 0$ to be determined later.

Consider first that case that $\mathcal{C}$ is a NO-instance of MNAE3SAT i.e. for every truth assignments τ, there exists a clause C_i whose literals all get the same truth assignment. Each $\mathbf{x} \in \{0,1\}^n$ corresponds to a truth assignment. If $x_i = 1$ (resp. $x_i = 0$) then u_i is true (resp. u_i is false). Let C_j be the clause whose literals have the same truth value. Then

$$\text{lindisc}(\mathbf{A}) \geq \text{lindisc}(\mathbf{A}, (1/2) \cdot \mathbb{1}) \geq \left| \mathbf{r}_j \left(\frac{1}{2} \cdot \mathbb{1} - \mathbf{x} \right) \right| = \frac{3}{2} > \frac{3}{2} - \epsilon,$$

so $\mathbf{A}$ is a NO-instance of LDS.

Consider next the case that $\mathcal{C}$ is a YES-instance of MNAE3SAT, and let τ be a satisfying assignment. Suppose $\mathbf{w}^* \in [0,1]^n$ is a deep-hole of $\mathbf{A}$. If $w_i^* = \frac{1}{2}$ for all $i \in [n]$ then

$$\text{lindisc}(\mathbf{A}) = \text{lindisc}(\mathbf{A}, (1/2) \cdot \mathbb{1}) = \text{disc}(\mathbf{A}) \leq \left\| \mathbf{A} \left(\frac{1}{2} \cdot \mathbb{1} - \mathbf{x}^* \right) \right\|_\infty = \frac{1}{2}$$

since every clause has exactly two elements with the same truth value. Thus $\mathbf{A}$ is a YES-instance of LDS as long as we choose $\epsilon \leq 1$. Suppose then that $\mathbf{w}^* \neq \frac{1}{2}\mathbb{1}$, and let ϵ be a lower bound on the smallest non-zero gap between w_i^* and $1/2$ i.e. for all $w_i^* \neq \frac{1}{2}$,

$$\left| w_i^* - \frac{1}{2} \right| \geq \epsilon.$$

By Lemma 7, which implies that $\mathbf{w}^*$ has polynomial bit complexity, we know that we can choose such an ϵ of polynomial bit complexity. We will show that $\text{lindisc}(\mathbf{A}, \mathbf{w}^*) \leq \frac{3}{2} - \epsilon$ by constructing a colouring $\mathbf{x}^*$. Let

$$x_i^* = \begin{cases} \text{rd}(w_i^*) & \text{if } w_i^* \neq \frac{1}{2} \\ \tau(u_i) & \text{otherwise} \end{cases}$$

where $\text{rd}(w_i^*)$ is w_i^* rounded to the closest integer and u_i is the variable corresponding to column i. Let $\mathbf{r}$ be a row of matrix $\mathbf{A}$ with non-zero entries in columns i, j, and k. We bound the discrepancy of row $\mathbf{r}$ based on the number of rounded variables R_v among $\{x_i, x_j, x_k\}$.

$R_v = 0$: Since none of the variables are rounded, $w_i^* = w_j^* = w_k^* = \frac{1}{2}$ and

$$|\mathbf{r}(\mathbf{x}^* - \mathbf{w}^*)| = \left| \left(x_i^* - \frac{1}{2} \right) + \left(x_j^* - \frac{1}{2} \right) + \left(x_k^* - \frac{1}{2} \right) \right| = \frac{1}{2}$$

since τ is a satisfying assignment.

$R_v = 1$: W.l.o.g assume that that x_i^* is the rounded value and $w_j^* = w_k^* = \frac{1}{2}$. Then

$$|\mathbf{r}(\mathbf{x}^* - \mathbf{w}^*)| = \left| (x_i^* - w_i^*) + \left(x_j^* - \frac{1}{2} \right) + \left(x_k^* - \frac{1}{2} \right) \right| \leq \left(\frac{1}{2} - \epsilon \right) + 1 = \frac{3}{2} - \epsilon.$$

$R_v = 2$: W.l.o.g assume that x_i^* and x_j^* are the rounded values and $w_k^* = \frac{1}{2}$. Then

$$|\mathbf{r}(\mathbf{x}^* - \mathbf{w}^*)| = \left| (x_i^* - w_i^*) + (x_j^* - w_j^*) + \left(x_k^* - \frac{1}{2} \right) \right| \leq 2 \cdot \left(\frac{1}{2} - \epsilon \right) + \frac{1}{2} = \frac{3}{2} - 2\epsilon.$$

$R_v = 3$: All three values are rounded so

$$|\mathbf{r}(\mathbf{x}^* - \mathbf{w}^*)| = \left| (x_i^* - w_i^*) + (x_j^* - w_j^*) + (x_k^* - w_k^*) \right| \leq 3 \cdot \left(\frac{1}{2} - \epsilon \right) = \frac{3}{2} - 3\epsilon.$$

Since $\mathbf{r}$ was an arbitrary row of $\mathbf{A}$, $\text{lindisc}(\mathbf{A}) = \text{lindisc}(\mathbf{A}, \mathbf{w}^*) \leq \frac{3}{2} - \epsilon$ as required. This completes the reduction. ◀

4 Algorithms for Linear Discrepancy

In the following we consider restrictions and variants of linear discrepancy for which we are able to give poly-time algorithms. The first subsection considers matrices with a single row. The second subsection considers matrices $\mathbf{A} \in \mathbb{Z}^{d \times n}$ with constant d and entry of largest magnitude δ. In that case, we compute $\text{lindisc}(\mathbf{A})$ in time $O\left(d(2n\delta)^{d^2}\right)$. The third subsection presents a poly-time 2^n approximation to $\text{lindisc}(\mathbf{A})$ for $\mathbf{A} \in \mathbb{Q}^{m \times n}$.

4.1 Linear Discrepancy of a Row Matrix

We begin by developing some intuition for the linear discrepancy of a one-row matrix, $\mathbf{A} = [a_1, ..., a_n]$. For now, let us make the simplifying assumption that the entries of $\mathbf{A}$ are non-negative and sorted in decreasing order. Define the *subset sums* of $\mathbf{A}$ to be the multi-set $\mathcal{S}(\mathbf{A}) = \{s_1, ..., s_{2^n}\}$ where each $s_i = \mathbf{Ax}$ for exactly one $\mathbf{x} \in \{0,1\}^n$. Enumerate the element of $\mathcal{S}(\mathbf{A})$ in non-decreasing order, i.e. $s_i \leq s_{i+1}$. If $\ell_A = 2 \cdot \text{lindisc}(\mathbf{A})$, then ℓ_A is the width of the largest gap between consecutive entries in $\mathcal{S}(\mathbf{A})$.

Suppose $\mathbf{A}_i = [a_1, ..., a_i]$. Let us consider how $\mathcal{S}(\mathbf{A}_i)$ and $\text{lindisc}(A_i)$ change for the first couple of values of i. Clearly, $\mathcal{S}(\mathbf{A}_1) = [0, a_1]$ and $\text{lindisc}(\mathbf{A}_1) = \frac{a_1}{2}$. $\mathcal{S}(\mathbf{A}_2)$ is the disjoint union of $\mathcal{S}(\mathbf{A}_1)$ and $\mathcal{S}(\mathbf{A}_1)$ shifted to the right by a_2. Since $a_1 \geq a_2$, $\mathcal{S}(\mathbf{A}_2) = [0, a_2, a_1, a_1 + a_2]$ where the largest gap is of size $\max(a_2, a_1 - a_2)$. See Figure 1. In general, the entries of $\mathcal{S}(\mathbf{A}_i)$ consists of two copies of $\mathcal{S}(\mathbf{A}_{i-1})$ with one shifted to the right by a_i. The gaps in $\mathcal{S}(\mathbf{A}_i)$ are gaps previously in $\mathcal{S}(\mathbf{A}_{i-1})$ or between an element of $\mathcal{S}(\mathbf{A}_{i-1})$ and one in $\{a_i + s : s \in \mathcal{S}(\mathbf{A}_{i-1})\}$.

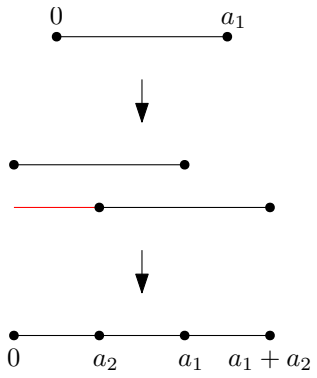


Figure 1 Obtaining $\mathcal{S}(\mathbf{A}_2)$ from $\mathcal{S}(\mathbf{A}_1)$ when $a_1 \geq a_2$.

A similar structure occurs for general matrices with real valued entries with two caveats: (1) the previous interval is shifted left or right depending on the sign of the current entry (negative and positive respectively) and (2) the smallest entry of $\mathcal{S}(\mathbf{A})$ is not zero but the sum of the negative entries in $\mathbf{A}$.

▶ **Lemma 8.** *Suppose $\mathbf{A}_{k-1} = [a_1, ..., a_{k-1}]$ with entries in $\mathbb{R}$ and $|a_i| \geq |a_{i+1}|$. Let the largest gap in $\mathcal{S}(\mathbf{A}_{k-1})$ be of size ℓ_{k-1}. Then, for $\mathbf{A}_k = [a_1, ..., a_{k-1}, a_k]$ where $|a_k| \leq |a_i|$ for all $i \in [k-1]$, the largest gap in $\mathcal{S}(\mathbf{A}_k)$ is of size $\max(|a_k|, \ell_{k-1} - |a_k|)$.*

Proof. Again, it is important to remember that the entries of $\mathcal{S}(\mathbf{A}_k)$ are exactly those in $\mathcal{S}(\mathbf{A}_{k-1})$ along with those in $\{a_k + s : s \in \mathcal{S}(\mathbf{A}_{k-1})\}$. Let $\ell = \max(|a_k|, \ell_{k-1} - |a_k|)$.

We first show that $2 \cdot \text{lindisc}(\mathbf{A}_k) \leq \ell$ by showing that gaps between consecutive entries in $\mathcal{S}(\mathbf{A}_k)$ have size at most ℓ. If (s_j, s_{j+1}) is a consecutive pair in $\mathcal{S}(\mathbf{A}_{k-1})$ such that $s_{j+1} - s_j > \ell$, then s_j and s_{j+1} are no longer consecutive in $\mathcal{S}(\mathbf{A}_k)$, since $s_j \leq s_j + a_k \leq s_{j+1}$

if $a_k > 0$ and $s_j \leq s_{j+1} + a_k \leq s_{j+1}$ if $a_k < 0$. See Figure 2. Then, the gap given by any such pair gets split into gaps of size at most $\max\{|a_k|, s_{j+1} - s_j - |a_k|\} \leq \ell$, where the inequality holds because $s_{j+1} - s_j \leq \ell_{k-1}$. It follows that the size of each gap in $\mathcal{S}(\mathbf{A}_k)$ is at most ℓ.

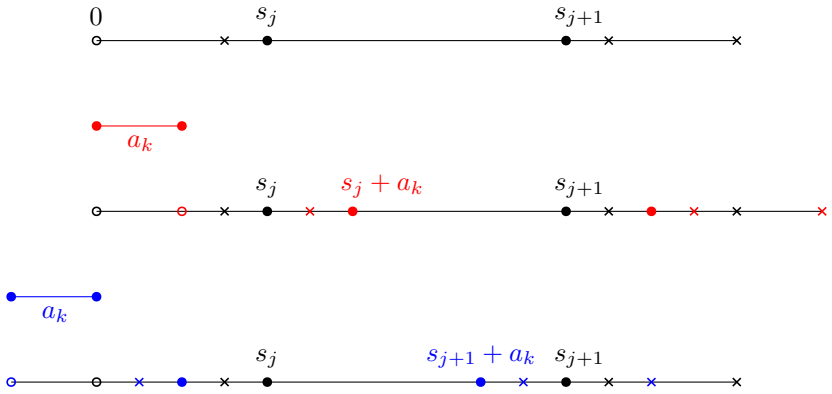


Figure 2 All consecutive pairs in $\mathcal{S}(\mathbf{A}_{k-1})$ of size greater than $|a_k|$ will be divided into two or more consecutive pairs in $\mathcal{S}(\mathbf{A}_k)$. The red interval indicates what happens when $a_k > 0$. The blue interval indicates what happens when $a_k < 0$.

Next we will show that $2 \cdot \text{lindisc}(\mathbf{A}_k) \geq \ell$ by producing a pair of consecutive entries in $\mathcal{S}(\mathbf{A}_k)$ which achieves gap ℓ. Suppose $\ell = |a_k|$. Recall that s_0 is the smallest subset sum of all entries in $\mathbf{A}_k$, which equals the sum of all negative entries in $\mathbf{A_k}$. Then it is easy to check that s_1 equals $s_0 + |a_k|$, where we recall that a_k is the entry in $\mathbf{A}_k$ with minimum absolute value. Therefore, $(s_0, s_0 + |a_k|)$ is a consecutive pair in $\mathcal{S}(\mathbf{A}_k)$. This means that if $\ell = |a_k|$, then we are done, as we have produced a pair with gap ℓ.

When $\ell = \ell_{k-1} - |a_k| > |a_k|$, we split our analysis into two cases: (1) $a_k > 0$ and (2) $a_k < 0$.

In the former case, let (s_{j^*}, s_{j^*+1}) be a consecutive pair in $\mathcal{S}(\mathbf{A}_{k-1})$ that achieves gap ℓ_{k-1} and suppose, towards a contradiction, that $s_{j^*} + a_k$ and s_{j^*+1} do not appear consecutively in $\mathcal{S}(\mathbf{A}_k)$. Then there must be some $s \in \mathcal{S}(\mathbf{A}_k)$ such that $s_{j^*} + a_k < s < s_{j^*+1}$. Note that s cannot be an element of $\mathcal{S}(\mathbf{A}_{k-1})$ since s_{j^*} and s_{j^*+1} are consecutive in $\mathcal{S}(\mathbf{A}_{k-1})$, so $s - a_k$ must be an element of $\mathcal{S}(\mathbf{A}_{k-1})$. However, since $s > s_{j^*} + a_k$, we have $s - a_k > s_{j^*}$. This is a contradiction since s_{j^*} and s_{j^*+1} are consecutive entries in $\mathcal{S}(\mathbf{A}_{k-1})$. See Figure 3. Thus $(s_{j^*} + a_k, s_{j^*+1})$ must be a consecutive pair in $\mathcal{S}(\mathbf{A}_k)$.

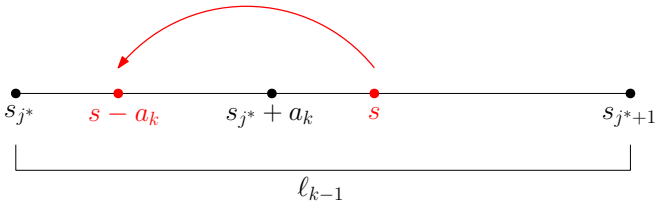


Figure 3 Suppose $a_k < \ell_{k-1} - a_k$ and there exists $s \in \mathcal{S}(\mathbf{A}_k)$ such that $s_{j^*} + a_k < s < s_{j^*+1}$.

The latter case, when $a_k < 0$, is similar. Again there exists a pair of consecutive entries (s_{j^*}, s_{j^*+1}) in $\mathcal{S}(\mathbf{A}_{k-1})$ which achieves gap ℓ_{k-1}. Suppose, towards contradiction, that s_{j^*} and $s_{j^*+1} - |a_k|$ do not appear consecutively in $\mathcal{S}(\mathbf{A}_k)$. Then there must be some $s \in \mathcal{S}(\mathbf{A}_k)$ such that $s_{j^*} < s < s_{j^*+1} - |a_k|$. Again, s cannot be an element of $\mathcal{S}(\mathbf{A}_{k-1})$ since s_{j^*} and s_{j^*+1} are consecutive in $\mathcal{S}(\mathbf{A}_{k-1})$, so $s + |a_k|$ must be an element of $\mathcal{S}(\mathbf{A}_{k-1})$. However since $s < s_{j^*+1} - |a_k|$, we have $s + |a_k| < s_{j^*+1}$. This is a contradiction since s_{j^*} and s_{j^*+1} are consecutive entries in $\mathcal{S}(\mathbf{A}_{k-1})$. ◀

Lemma 8 has the following curious corollary.

▶ **Corollary 9.** *Let* $\mathbf{A} = [a_1, ..., a_n]$ *and* $\mathbf{A}' = [|a_1|, ..., |a_n|]$. *Then* $\text{lindisc}(\mathbf{A}) = \text{lindisc}(\mathbf{A}')$.

Lemma 8 and Corollary 9 suggest an algorithm: replace the entries of $\mathbf{A}$ by their magnitudes. Sort $\mathbf{A}$. Consider each entry in turn and update the largest gap accordingly. See Algorithm 1.

Proof of Theorem 3. By Corollary 9 it is sufficient to consider row matrices with non-negative entries. Suppose that $\mathbf{A} = [a_1, ..., a_n]$ is such a matrix with entries sorted in decreasing order. Algorithm 1 correctly outputs the linear discrepancy for matrices with a single entry. Let $\mathbf{A}_i = [a_1, ..., a_i]$. Lemma 8 gives us a recursive method for computing the largest gap in $\mathcal{S}(\mathbf{A}_{i+1})$ from the largest gap in $\mathcal{S}(\mathbf{A}_i)$. Since $\text{lindisc}(\mathbf{A})$ is half the size of the largest gap in $\mathcal{S}(\mathbf{A})$, Algorithm 8 output $\text{lindisc}(\mathbf{A})$ as required. ◀

Algorithm 1 Linear discrepancy of row matrix.

Input: Matrix $\mathbf{A} \in \mathbb{Q}^{1\times n}$.
Output: $\text{lindisc}(\mathbf{A})$.
for i *from* 1 *to* n **do**
  $\mathbf{A}[i] \leftarrow |a_i|$
sort $\mathbf{A}$ in decreasing order
$\ell \leftarrow a_1$
for i *from* 2 *to* n **do**
  $\ell \leftarrow \max(a_i, \ell - a_i)$
return $\frac{\ell}{2}$

Thus, for any row matrix $\mathbf{A}$ with n elements, we can find $\text{lindisc}(\mathbf{A})$ in time $O(n \log n)$.

4.1.1 One Row Linear Discrepancy Rounding

Let $\text{lindisc}(\mathbf{A}) = \ell$. By the definition of linear discrepancy, for every $\mathbf{w} \in [0,1]^n$ there exists an $\mathbf{x} \in \{0,1\}^n$ such that $\|\mathbf{A}(\mathbf{w} - \mathbf{x})\|_\infty \leq \ell$. In-fact, if $\mathbf{w}$ is not a deep-hole, there exists an $\mathbf{x}$ which satisfies $\|\mathbf{A}(\mathbf{w} - \mathbf{x})\|_\infty < \ell$. However it is not obvious that finding such an $\mathbf{x}$ can be done efficiently i.e. in polynomial time with respect to the bit complexity of $\mathbf{A}$ and n. By reducing from the subset-sum problem, we observe that it is difficult to compute $\text{lindisc}(\mathbf{A}, \mathbf{w})$ let alone find an $\mathbf{x}$ which minimizes $\|\mathbf{A}(\mathbf{w} - \mathbf{x})\|_\infty \leq \ell$.

Proof of Theorem 4. To begin, let $\mathbf{A} = [a_1, ..., a_n]$ for positive a_i in non-increasing order. We will consider $\mathbf{A}$ with arbitrary entries at the end. Let $w = \mathbf{A}\mathbf{w}$. As before, let $\mathcal{S}(\mathbf{A}) = [s_0, ..., s_{2^n-1}]$ be the subset-sums of $\mathbf{A}$ where each $s_i = \mathbf{A}\mathbf{x}$ for an $\mathbf{x} \in \{0,1\}^n$ and $s_i \leq s_{i+1}$ for all i. Recall that $2 \cdot \text{lindisc}(\mathbf{A})$ is the largest gap between any two consecutive entries in $\mathcal{S}(\mathbf{A})$. Our algorithm will find a pair of subset sums containing w. If we can show that the size of the interval between these two subset sums is no more than the gap between some two consecutive entries in $\mathcal{S}(\mathbf{A})$, then the closest subset sum to w among these two will be within $\text{lindisc}(\mathbf{A})$ of w.

Just as in Algorithm 1, we refine the interval between two subset sums containing w by incrementally adding the entries of $\mathbf{A}$ in decreasing order. Initially our interval is $g_0 = [0, \sum_{i=1}^n a_i]$. We maintain the invariants: (1) $w \in g_i$ for all i, and (2) the end-points of g_i are subset sums.

Suppose $w \in g_i = [u, v]$ and we are considering a_i. If $u+a_i > w$ then set $v \leftarrow \min(v, u+a_i)$. Otherwise let $u \leftarrow u + a_i$. Algorithm 2 computes this interval and the associated vectors $\mathbf{u}$ and $\mathbf{v}$ representing its endpoints.

Consider the values of u and v at the end of the algorithm. We claim that the final interval $[u, v]$ is at most the width of some gap between two consecutive terms in $\mathcal{S}(\mathbf{A})$, the array of all subset sums of $\mathbf{A}$. Notice $u = a_1u_1 + \cdots + a_nu_n$ where $\mathbf{u} = [u_1, ..., u_n]$ is an endpoint of the interval once Algorithm 2 completes.

We partition $\mathbf{u}$ into maximal blocks where all entries in the same block have the same value i.e. $[u_1, u_2, ..., u_{\ell_1}], ..., [u_{\ell_r+1}, u_{\ell_r+2}, ..., u_n]$ such that $u_{\ell_i+1} = u_{\ell_i+2} = \cdots = u_{\ell_{i+1}}$ for $i = 0, 1, ..., r-1$ where $\ell_0 = 0$.

We claim that Algorithm 2 outputs an interval containing w whose width is at most the distance between some two consecutive entries in $\mathcal{S}(\mathbf{A})$. The proof is by induction on r, the number of blocks. In the base case, $r = 1$ and there is only one block. Thus $u = 0$ or $u = \sum a_i$. In the case where $u = 0$, we must have $a_i > w$ for all $i \in [n]$, and $v = a_n$. Thus $w \in [0, a_n]$ with consecutive elements 0 and a_n of $\mathcal{S}(\mathbf{A})$. In the latter case when $u = \sum a_i$, we can output $\mathbf{w}$ since it is already a subset sum.

Suppose next that the claim holds for all matrices where the algorithm outputs a vector $\mathbf{u}$ with k blocks, and we will show that it still holds for a matrix $\mathbf{A}$ whose output $\mathbf{u}$ has $k+1$ blocks. Let $\mathbf{u}' = [u_1, ..., u_{\ell_{k+1}}]$ and $\mathbf{v}' = [v_1, ..., v_{\ell_{k+1}}]$ be the final vectors after running the algorithm on $\mathbf{A}' = [a_1, ..., a_{\ell_{k+1}}]$. Further let $u' = \sum_{i=1}^{\ell_{k+1}} a_iu_i$ and $v' = \sum_{i=1}^{\ell_{k+1}} a_iv_i$. By the induction hypothesis, the width of $[u', v']$ is at most the distance between some two consecutive elements in the list of subset sums of $\mathbf{A}'$. The last block of $\mathbf{u}$ is $[u_{\ell_{k+1}+1}, ..., u_n]$. The entries of this block are either all zeros or all ones. Consider each case in-turn.

First suppose $u_{\ell_{k+1}+1} = \cdots = u_n = 0$. Since none of the a_i for $i = \ell_{k+1} + 1, ..., n$ were added to u, it must be the case that $u' + a_i > w$ for all such i. Thus the interval $[u, v] = [u', \min(v', u' + a_n)]$ has width at most a_n. Since 0 and a_n are consecutive in $\mathcal{S}(\mathbf{A})$, as $|a_n|$ is the entry with the smallest magnitude in $\mathbf{A}$, the output interval satisfies our requirements.

Next suppose $u_{\ell_{k+1}+1} = \cdots = u_n = 1$. It must be the case that $u = u' + a_{\ell_{k+1}+1} + \cdots + a_n \leq w$. Observe that $a_{\ell_{k+1}}$ is in the k^{th} block and so $u_{\ell_{k+1}} = 0$. Let $[u'', v'']$ be our interval after processing the $k-1^{\text{st}}$ block i.e. $u'' = \sum_{i=1}^{\ell_k} a_iu_i$ and $v'' = \sum_{i=1}^{\ell_k} a_iv_i$. Notice that since none of the entries in the k^{th} block were added to u'', we must have $u'' + a_i > w$ for all $i = \ell_k + 1, ..., \ell_{k+1}$. In such cases, we always update $v'' \leftarrow \min(v'', u'' + a_i)$ after each such i, thus the interval $[u', v']$ has width at most $a_{\ell_{k+1}}$. Thus it suffices to show that $a_{\ell_{k+1}+1} + \cdots + a_n$ and $a_{\ell_{k+1}}$ are consecutive in $\mathcal{S}(\mathbf{A})$. First note that $a_{\ell_{k+1}+1} + \cdots + a_n \leq a_{\ell_{k+1}}$ since $u' + a_{\ell_{k+1}+1} + \cdots + a_n \leq w \leq v' \leq u' + a_{\ell_{k+1}}$. The two subset sums then are also consecutive, since $a_i > a_{\ell_{k+1}}$ for all $i < \ell_{k+1}$.

Now consider the case where $\mathbf{A}$ can have both positive and negative entries. Without loss of generality we can assume that none of the entries are zero. Let $A_- = \{a_i \in \mathbf{A} : a_i < 0\}$ and $A_+ = \{a_i \in \mathbf{A} : a_i > 0\}$. It suffices to set $u_0 = \sum_{a \in A_-} a$ and $v_0 = \sum_{a \in A_+} a$ and let $\mathbf{u}$ and $\mathbf{v}$ be the indicator vectors of $\mathbf{A}_-$ and $\mathbf{A}_+$ respectively. The remainder of the algorithm is identical except that the matrix should be sorted in decreasing order of *magnitude* and every time an element $a_i \in \mathbf{A}_-$ is added to u, its entry in $\mathbf{u}$ should be set to zero. ◀

4.2 Constant Rows with Bounded Matrix Entries

Let $\mathbf{A} \in \mathbb{Z}^{d \times n}$ with $\max_{i,j} |A_{i,j}| \leq \delta$. Let $Z = \mathbf{A}[0, 1]^d$ be the zonotope of $\mathbf{A}$ and let $T = [-n\delta, n\delta]^d \cap \mathbb{Z}^d$ be the set of all integer lattice points of Z. The following algorithm computes lindisc($\mathbf{A}$) in polynomial time with respect to n for fixed d and δ. The algorithm makes use of Lemma 11, which is proved in the Appendix.

Algorithm 2 Finding a close subset sum to $\mathbf{Aw}$.

Input: A vector $\mathbf{w} \in [0,1]^n$ and a row matrix $\mathbf{A} = [a_1, ..., a_n]$ of positive integers sorted in increasing order.

Output: A vector $\mathbf{x} \in \{0,1\}^n$ such that $\|\mathbf{A}(\mathbf{w}-\mathbf{x})\|_\infty \leq \text{lindisc}(\mathbf{A})$.

```
A ← SORT-DECREASING(A)
u ← ZEROS(n)
v ← ONES(n)
w ← Aw, u ← Au, v ← Av
return v if w == v
for k = 1..n do
    if u + a_k > w then
        v ← min(v, u + a_k)
        if v == u + a_k then
            v ← COPY(u)
            v[k] ← 1
    else
        u ← u + a_k
        u[k] ← 1
return u if u is closer to w else v
```

Proof of Theorem 5. For every one of the $(2n\delta+1)^d$ integral points $\mathbf{b} \in T$, compute whether $\mathbf{Ax} = \mathbf{b}$ for some $\mathbf{x} \in \{0,1\}^n$ using dynamic programming. This procedure generalizes dynamic programming algorithms for knapsack and subset sum and will be outlined in the following. Let $\mathbf{a}_1, ..., \mathbf{a}_n$ be the columns of $\mathbf{A}$. Construct a matrix $\mathbf{M}$ with dimensions $[-n\delta, n\delta]^d \times n$. Cell $(\mathbf{v}, i)$ of $\mathbf{M}$ contains the indicator $[\mathbf{M}(\mathbf{v}-\mathbf{a}_i, i-1) \vee \mathbf{M}(\mathbf{v}, i-1)]$; this corresponds to a linear combination of the first $i-1$ columns of $\mathbf{A}$ which adds up to $\mathbf{v}-\mathbf{a}_i$ or a linear combination of the first $i-1$ columns which adds up to $\mathbf{v}$. The first column of $\mathbf{M}$ is the indicator vector for $\{\mathbf{a}_1\}$. Computing the entries of $\mathbf{M}$ takes time $O(2n\delta)^{d+1}$. $\mathbf{M}(\mathbf{b}, n)$ indicates the feasibility of $\mathbf{Ax} = \mathbf{b}$. Computing this for all $\mathbf{b}$ takes time $O(2n\delta)^{d+1}$. Let $S \subseteq T$ be the set of points $\mathbf{b}$ in Z such that $\mathbf{Ax} = \mathbf{b}$ for some $\mathbf{x} \in \{0,1\}^n$, and set $|S| = N$.

Apply Lemma 11 to the points of S in ℓ_∞-norm. The output is some radius r and point $\mathbf{x}^*$ such that the ℓ_∞-ball centered at $\mathbf{x}^*$ with radius r is the largest such ball with center inside the convex hull of S not containing any points of S. Note that r is in-fact the linear discrepancy of $\mathbf{A}$. Since r and $\mathbf{x}^*$ can be computed in time $O(N^d)$, $\text{lindisc}(\mathbf{A})$ can be computed in time $O(2n\delta)^{d^2+d}$. ◀

4.3 Poly-time Approximation Algorithm

Next, we prove Theorem 6 present a 2^n-approximation algorithm for linear discrepancy. Recall that $\text{rd}(\mathbf{w})$ is the function which rounds each coordinate of $\mathbf{w}$ to its nearest integer (with ties broken arbitrarily). Let the operator norms of a matrix $\mathbf{A}$ be:

$$\|\mathbf{A}\|_{p\to q} = \max_{\mathbf{x}\in\mathbb{R}^n\setminus\{0\}} \frac{\|\mathbf{Ax}\|_q}{\|\mathbf{x}\|_p}.$$

Note that

$$\text{lindisc}(\mathbf{A}) \leq \max_{\mathbf{w}\in[0,1]^n} \|\mathbf{A}(\mathbf{w} - \text{rd}(\mathbf{w}))\|_\infty \leq \frac{1}{2} \max_{\mathbf{z}\in[-1,1]^n} \|\mathbf{A}\mathbf{z}\|_\infty = \frac{1}{2}\|\mathbf{A}\|_{\infty\to\infty}.$$

To bound $\text{lindisc}(\mathbf{A})$ from below, we show that $\|\mathbf{A}\|_{\infty\to\infty} \leq 2^{n+1} \cdot \text{lindisc}(\mathbf{A})$. This completes the proof of the theorem, since $\|\mathbf{A}\|_{\infty\to\infty}$ equals the largest ℓ_1 norm of any row of $\mathbf{A}$, and can be computed in polynomial time.

Let us try to interpret the statement $\|\mathbf{A}\|_{\infty\to\infty} \leq 2^{n+1} \cdot \text{lindisc}(\mathbf{A})$. Note that $\|\mathbf{A}\mathbf{z}\|_\infty$ is equal to the Minkowski $\mathcal{P}$-norm $\|\mathbf{z}\|_\mathcal{P}$ for $\mathcal{P} = \{\mathbf{x} : \|\mathbf{A}\mathbf{x}\|_\infty \leq 1\}$ i.e. $\|\mathbf{z}\|_\mathcal{P} = \inf\{t \geq 0 : \mathbf{z} \in t\mathcal{P}\}$ so

$$\|\mathbf{A}\|_{\infty\to\infty} = \max_{\mathbf{z}\in[-1,1]^n} \|\mathbf{A}\mathbf{z}\|_\infty = \max_{\mathbf{z}\in[-1,1]^n} \|\mathbf{z}\|_\mathcal{P}.$$

By interpreting $\mathbf{z}$ as the difference of two vectors $\mathbf{x}, \mathbf{x}' \in [0,1]^n$ we have that

$$\|\mathbf{A}\|_{\infty\to\infty} = \max_{\mathbf{z}\in[-1,1]^n} \|\mathbf{z}\|_\mathcal{P} = \max_{\mathbf{x},\mathbf{x}'\in[0,1]^n} \|\mathbf{x} - \mathbf{x}'\|_\mathcal{P}.$$

It is an easy, and well-known fact that $\text{lindisc}(\mathbf{A})$ is the smallest t such that $[0,1]^n \subseteq \bigcup_{\mathbf{x}\in\{0,1\}^n} (\mathbf{x} + \mathcal{P})$; see [15]. We then just need to show that the diameter of the unit hypercube with respect to the Minkowski $\mathcal{P}$-norm is no more than this scale-factor t times $O(2^n)$. We prove the following more general statement.

▶ **Lemma 10.** *Let $\mathcal{K}$ be a convex symmetric polytope and $S \subset \mathbb{R}^n$ be convex. Suppose there exist N elements $x_1, ..., x_N \in S$ such that*

$$S \subseteq \bigcup_{x_i} x_i + t\mathcal{K}.$$

Then $\max_{x,x'\in S} \|x - x'\|_\mathcal{K} \leq 2tN$.

Proof. Fix any two points x and x' in S. Let $\mathcal{P}_i$ be the polytope $x_i + t\mathcal{K}$. Since S is convex, the line segment $\lambda x + (1-\lambda)x'$ for $\lambda \in [0,1]$ is in S. Therefore $\lambda x + (1-\lambda)x'$ intersects a sequence of polytopes $\mathcal{P}_{k_1}, ..., \mathcal{P}_{k_r}$ with centres $x_{k_1}, ..., x_{k_r}$, such that any two consequtive polytopes in the sequence intersect. Since the polytopes are convex, we can assume that they appear in the sequence at most once, so $r \leq N$. By the triangle inequality we have

$$\begin{aligned}\|x - x'\|_K &= \|(x - x_{k_1}) + (x_{k_1} - x_{k_2}) + \cdots + (x_{k_r} - x')\|_K \\ &\leq \|x - x_{k_1}\|_K + \|x_{k_1} - x_{k_2}\|_K + \cdots + \|x_{k_r} - x'\|_K \\ &\leq t + 2t(N-1) + t = 2tN\end{aligned}$$

where the last inequality follows as $x \in \mathcal{P}_{k_1}$, $x' \in \mathcal{P}_{k_r}$, and $\|x_{k_i} - x_{k_{i+1}}\|_K \leq 2t$. ◀

Proof of Theorem 6. In Lemma 10, set $\mathcal{K}$ to be the parallelepiped defined by $\mathbf{A}$, $S = [0,1]^n$, $t = \text{lindisc}(\mathbf{A})$, and $\{x_1, ..., x_N\} = \{0,1\}^n$. ◀

5 Open Problems

Because of the similarity between the closest vector problem and linear discrepancy, we suspect that linear discrepancy is also Π_2-complete, and the hardness result of Theorem 2 is, in this sense, not tight. Further, Haviv and Regev also showed that CRP is Π_2-hard to approximate to with-in a factor of $\frac{3}{2}$, and we conjecture that a similar hardness of approximation result should hold for linear discrepancy.

We suspect that the algorithm used to prove Theorem 3 can be generalized to matrices $\mathbf{A} \in \mathbb{Q}^{d \times n}$ with running time $\tilde{O}(n^d)$. This would be a substantial improvement on the $O\left(d(n\delta)^{d^2}\right)$ running time algorithm used to prove Theorem 5, and would be independent of the magnitude of the largest entry of $\mathbf{A}$.

It is also interesting to extend the largest empty ball algorithm from Lemma 11 to other ℓ_p norms, or even arbitrary norms, given appropriate access to the norm ball. Currently, this seems rather difficult as Voronoi diagrams with respect to the ℓ_p-norm for $p \in (2, \infty)$ are poorly behaved. For the standard ℓ_2-norm Voronoi diagram in $\mathbb{R}^d$, it is the case that $d+1$ affinely independent vertices are equidistant to exactly one point. This is no longer the case even in $\mathbb{R}^3$ for ℓ_4-norm [13]. In particular, there exists a set of four vertices such that the intersection of their pair-wise bisectors has size three. The situation is even worse for general strictly convex norms. There exists such norms where the pair-wise bisectors of a set of four points in $\mathbb{R}^3$ can have arbitrarily many intersections.

We currently also have no evidence that the approximation factor in Theorem 6 is tight. One possibility is that there exists an approximation preserving reduction from the closest vector problem in lattices to linear discrepancy. This would show that one cannot expect a significant improvement to Theorem 6 without also improving the best polynomial time approximation to the covering radius, which is currently also exponential in the dimension n. On the other hand, we also conjecture that the approximation factor in Theorem 6 can be taken to be a function of $\min\{m, n\}$, or even of the rank of the matrix $\mathbf{A}$.

References

1. Christoph Aistleitner, Dmitriy Bilyk, and Aleksandar Nikolov. Tusnády's problem, the transference principle, and non-uniform qmc sampling. In Art B. Owen and Peter W. Glynn, editors, *Monte Carlo and Quasi-Monte Carlo Methods*, pages 169–180. Springer International Publishing, 2016.
2. Per Austrin, Venkatesan Guruswami, and Johan Håstad. (2+ε)-sat is np-hard. *SIAM Journal on Computing*, 46(5):1554–1573, 2017.
3. Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. Better scalable algorithms for broadcast scheduling. *ACM Trans. Algorithms*, 11(1):3:1–3:24, 2014.
4. József Beck and Vera T Sós. Discrepancy theory. In *Handbook of combinatorics (vol. 2)*, pages 1405–1446. MIT Press, 1996.
5. Jean-Daniel Boissonnat, Micha Sharir, Boaz Tagansky, and Mariette Yvinec. Voronoi diagrams in higher dimensions under certain polyhedral distance functions. *Discrete & Computational Geometry*, 19(4):485–519, 1998. doi:10.1007/PL00009366.
6. Moses Charikar, Alantha Newman, and Aleksandar Nikolov. Tight hardness results for minimizing discrepancy. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1607–1614. Society for Industrial and Applied Mathematics, 2011.
7. Bernard Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(4):377–409, 1993.
8. Bernard Chazelle. *The discrepancy method: randomness and complexity*. Cambridge University Press, 2001.
9. E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.
10. Venkatesan Guruswami, Daniele Micciancio, and Oded Regev. The complexity of the covering radius problem. *Computational Complexity*, 14(2):90–121, 2005.

11 Ishay Haviv and Oded Regev. Hardness of the covering radius problem on lattices. In *Computational Complexity, 2006. CCC 2006. Twenty-First Annual IEEE Conference on*, pages 14–pp. IEEE, 2006.

12 Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2625. SIAM, 2017.

13 Christian Icking, Rolf Klein, Ngoc-Minh Lé, and Lihong Ma. Convex distance functions in 3-space are different. *Fundamenta Informaticae*, 22(4):331–352, 1995.

14 László Lovász, Joel Spencer, and Katalin Vesztergombi. Discrepancy of set-systems and matrices. *European Journal of Combinatorics*, 7(2):151–160, 1986.

15 Jiri Matousek. *Geometric discrepancy: An illustrated guide*. Springer, 1999.

16 Jiří Matoušek, Aleksandar Nikolov, and Kunal Talwar. Factorization Norms and Hereditary Discrepancy. *International Mathematics Research Notices*, 2020(3):751–780, March 2018. doi:10.1093/imrn/rny033.

17 A. Nikolov. Tighter bounds for the discrepancy of boxes and polytopes. *Mathematika*, 63(3):1091–1113, 2017. doi:10.1112/S0025579317000250.

18 Thomas Rothvoß. Approximating bin packing within o (log opt* log log opt) bins. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 20–29. IEEE, 2013.

19 Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

20 Godfried T Toussaint. Computing largest empty circles with location constraints. *International journal of computer & information sciences*, 12(5):347–358, 1983.

A Appendix

A.1 Bit Complexity of Linear Discrepancy

Proof of Lemma 7. Let $\mathbf{r}_i$ for $i \in [m]$ be the rows of $\mathbf{A}$, $\text{lindisc}(\mathbf{A}) = \lambda_A$, and $\mathbf{w}^*$ be a deep-hole of $\mathbf{A}$. For every $\mathbf{x} \in \{0,1\}^n$ there exists an $i \in [m]$ and $\sigma \in \{-1,1\}$ such that $\sigma\mathbf{r}_i(\mathbf{w}^* - \mathbf{x}) \geq \lambda_A$. Let $\mathbf{b}_x = \sigma\mathbf{r}_i$ and consider the following linear program over the variables $\mathbf{w} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$:

$$\begin{aligned} \text{Maximize: } & \lambda \\ \text{Subject to: } & \mathbf{b_x}(\mathbf{w} - \mathbf{x}) \geq \lambda && \text{for all } \mathbf{x} \in \{0,1\}^n \\ & \mathbf{0} \leq \mathbf{w} \leq \mathbf{1} \end{aligned}$$

Let λ^* be the optimum value of this linear program. First note that $\lambda_A \leq \lambda^*$ since $(\mathbf{w}^*, \lambda)$ satisfies the constraints. Next we show that $\lambda_A \geq \lambda^*$. Suppose, towards contradiction, that $\lambda_A < \lambda^*$. Then there exists $\mathbf{w}' \in [0,1]^n$ such that

$$\|\mathbf{A}(\mathbf{w}' - \mathbf{x})\|_\infty \geq \mathbf{b}_x(\mathbf{w}' - \mathbf{x}) \geq \lambda^* > \lambda_A$$

for every $\mathbf{x} \in \{0,1\}^n$. Since $\lambda_A = \text{lindisc}(\mathbf{A})$, we cannot have $\text{lindisc}(\mathbf{A}, \mathbf{w}') > \lambda_A$. Thus $\lambda^* = \text{lindisc}(\mathbf{A})$. Since this LP has n variables, the number of bits required to express the linear discrepancy and some deep-hole $\mathbf{w}^*$ of $\mathbf{A}$ are polynomial in n and the bit complexity of the largest entry of $\mathbf{A}$ [19]. ◀

A.2 Largest Empty Ball Problem

Let V be a set of n points in the plane and let $\text{ch}(V)$ denote the convex hull of V. The largest empty circle problem, denoted LEC, takes V and outputs both a radius r and point $\mathbf{x}^* \in \text{ch}(V)$ such that the circle centered at $\mathbf{x}^*$ with radius r is the largest empty circle not containing any point of V. We generalize this problem to other norms and to higher

dimensions as follows: V is a set of n points in $\mathbb{R}^d$, and the goal is to compute a point $\mathbf{x}^*$ in $\mathrm{ch}(V)$ such that $\mathbf{x}^* + rB$ does not contain any point of V, where B is the unit ball of either the ℓ_2^d or the ℓ_∞^d norm. In the following we present an algorithm which solves this largest empty ball (LEB) problem.

▶ **Lemma 11** (LEC in Higher Dimensions)**.** *Let V be a set of n points in $\mathbb{R}^d$ for some fixed constant d. The LEB of V, in both ℓ_2- and ℓ_∞-norms, can be computed in time $O(n^d)$.*

Proof. We use the following terminology. Define a face F of the Voronoi diagram $\mathrm{vd}(V)$ of V to be a subset of $\mathbb{R}^d$ such that, for some $S \subseteq V$, and every $\mathbf{x} \in F$, S are the points in V closest to $\mathbf{x}$. In particular, this means that any $\mathbf{x} \in F$ is equidistant from all points in S.

The algorithm of Toussaint [20] computes the LEB of n points V in the plane with respect to the ℓ_2-norm as follows,

1. Compute $\mathrm{vd}(V)$. Note that $\mathrm{vd}(V)$ is the union of Voronoi faces of dimension k, the set of which we denote $\mathrm{vd}_k(V)$, over all $k = 0, ..., d-1$.
2. Compute the convex hull of V, denoted $\mathrm{ch}(V)$. Let h be the number of facets of $\mathrm{ch}(V)$.
3. Preprocess the points of $\mathrm{ch}(V)$ so that queries of the form "Is a point x in $\mathrm{ch}(V)$?" can be answered in time $O(\log h)$. For every $v \in \mathrm{vd}_0(V)$, determine if $v \in \mathrm{ch}(V)$. Let $C_1 = \{v \in \mathrm{vd}_0(V) : v \in \mathrm{ch}(V)\}$.
4. Determine the intersection points of faces in $\mathrm{vd}_k(V)$ with faces of $\mathrm{ch}(V)$ of co-dimension k, for pairs of such faces that intersect at a unique point. Let C_2 be the set of all such intersection points.
5. For all points $v \in C_1 \cup C_2$, find the largest empty circle centered at v. Output a v which maximizes this radius.

We find the analogue of each step for points in $\mathbb{R}^d$ with respect to the ℓ_2-norm, and then adapt the algorithm to the ℓ_∞-norm.

In the following let $N = n^{\lceil d/2 \rceil}$. The complexity, i.e. total number of faces of every dimension, of the ℓ_2-Voronoi diagram in $\mathbb{R}^d$ for fixed d is $O(N)$ and can be computed in time $O(N + n \log n)$ by a classic result of Chazelle [7]. The complexity of $\mathrm{ch}(V)$ is $O(N)$ and can also be computed in time $O(N + n \log n)$.

To determine the set C_1 of Voronoi intersection points inside the convex hull, we let $\mathcal{H}$ be the set of bounding hyperplanes of $\mathrm{ch}(V)$. Assume, without loss of generality, that $\mathrm{ch}(V)$ contains the origin, and, for each $H \in \mathcal{H}$, let H^- be the half-space with H as its boundary containing the origin. Then $\mathrm{ch}(V) = \bigcap_{H \in \mathcal{H}} H^-$. We simply test, for each Voronoi intersection point $\mathbf{v}$, whether $\mathbf{v} \in H^-$ for each $H \in \mathcal{H}$, in total time $O(N)$. Since there are at most $O(N)$ Voronoi intersection points, we can find C_1 in time $O(N^2)$.

To determine the set C_2 of all unique intersection points of k-faces of $\mathrm{vd}_k(V)$ and faces of $\mathrm{ch}(V)$ of co-dimension k will require solving several linear systems. Note that the points in each face F in $\mathrm{vd}_k(V)$ satisfy $d-k$ equality constraints $\langle \mathbf{a}_1, \mathbf{x} \rangle = b_1, \langle \mathbf{a}_k, \mathbf{x} \rangle = b_k$ for linearly independent vectors $\mathbf{a}_1, ...\mathbf{a}_k \in \mathbb{R}^d$. Similarly, the points in each face of co-dimension k of $\mathrm{ch}(V)$ satisfy k linearly independent equality constraints. Since there are at most $O(2^d N) = O(N)$ faces of $\mathrm{ch}(V)$, there are at most that many faces of $\mathrm{ch}(V)$ of co-dimension k. We can then go over all Voronoi faces F of dimension k, and all faces G of $\mathrm{ch}(V)$ of co-dimension k, and solve the corresponding system of $(d-k)+k = d$ linear equations. If the system has a unique solution, we check if that solution is in $F \cap G$, and, if so, we add it to C_2. Thus, for constant d, the size of C_2 and the time to compute it are bounded bounded above by $O(N \cdot 2^d N) = O(N^2)$.

In total there are at most $O(N + N^2)$ points in $C_1 \cup C_2$ which can be computed in time $O(N^2)$. Thus solving the largest empty ball problem in dimension d for constant d takes time $O(n^d)$.

Next we consider the largest empty ball problem in ℓ_∞-norm. The convex hull remains the same, so we just have to consider the Voronoi diagram with respect to the ℓ_∞-norm. Again, constructing the Voronoi diagram can be done in expected time $O(n^{\lceil d/2 \rceil} \log^{d-1} n)$ using the randomized algorithm of Boissonnat et al. [5]. Next we consider the number of intersections between the Voronoi diagram and the convex hull. First note that Voronoi diagrams with respect to the ℓ_∞-norm need not consist of only hyperplanes and their intersections. Indeed, in $\mathbb{R}^d$, for two points with the same y-coordinate, there exists regions with affine dimension two which are equidistant to both points. To remedy this, we assume that no two points in V have the same i-th coordinate, for any $i \in [d]$. This is without loss of generality, by perturbing the points in V slightly. It remains to consider the complexity of each bisector in ℓ_∞-norm. By Claim 12, in constant dimension d, each such bisector can have at most $O(d^2)$ facets. Therefore, the complexity of any face of the Voronoi diagram, being the intersection of at most d bisectors, is bounded by a function of d. Thus the bounds of the ℓ_2-norm algorithm still hold, up to constant factors that depend on d. ◀

▷ Claim 12 (Bound on Number of Facets of ℓ_∞ Bisectors). Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ be such that assume that $u_i \neq v_i$ for all $i \in [d]$. Then the bisector $\{\mathbf{x} : \|\mathbf{x} - \mathbf{u}\|_\infty = \|\mathbf{x} - \mathbf{v}\|_\infty\}$ has at most $O(d^2)$ facets.

Proof. Let $\mathbf{x}$ be a point in the bisector at ℓ_∞ distance r from $\mathbf{u}$ and $\mathbf{v}$. Pick coordinates i and j and signs σ and τ in $\{-1, +1\}$ such that

$$\sigma_i(x_i - u_i) = \tau_j(x_j - v_j) = r. \tag{4}$$

Moreover, let us make this choice so that either $i \neq j$ or $\sigma_i \neq \tau_i$. This is always possible, since, otherwise, the assumption on $\mathbf{u}$ and $\mathbf{v}$ is violated. Then, (4) defines a hyperplane in $\mathbb{R}^d$, namely $H_{i,j,\sigma,\tau} = \{\mathbf{x} : \sigma_i x_i - \tau_j x_j = \sigma_i u_i - \tau_j v_j\}$. Note that there are at most $\binom{2d}{2} \in O(d^2)$ such hyperplanes, and each $\mathbf{x}$ in the bisector lies in at at least one of them. Moreover, a point $\mathbf{x}$ in $H_{i,j,\sigma,\tau}$ lies in the bisector if and only if it satisfies the inequalities

$$|x_k - u_k| \leq \sigma_i(x_i - u_i) \quad \forall k \in [d],$$
$$|x_k - v_k| \leq \tau_j(x_j - v_j) \quad \forall k \in [d].$$

Thus, the bisector is the union of $(d-1)$-dimensional convex polyhedra, one per each of the $O(d^2)$ hyperplanes $H_{i,j,\sigma,\tau}$. ◁

Augmenting the Algebraic Connectivity of Graphs

Bogdan-Adrian Manghiuc
School of Informatics, University of Edinburgh, UK
b.a.manghiuc@sms.ed.ac.uk

Pan Peng
Department of Computer Science, University of Sheffield, UK
p.peng@sheffield.ac.uk

He Sun
School of Informatics, University of Edinburgh, UK
h.sun@ed.ac.uk

Abstract

For any undirected graph $G = (V, E)$ and a set E_W of candidate edges with $E \cap E_W = \emptyset$, the (k, γ)-spectral augmentability problem is to find a set F of k edges from E_W with appropriate weighting, such that the algebraic connectivity of the resulting graph $H = (V, E \cup F)$ is least γ. Because of a tight connection between the algebraic connectivity and many other graph parameters, including the graph's conductance and the mixing time of random walks in a graph, maximising the resulting graph's algebraic connectivity by adding a small number of edges has been studied over the past 15 years, and has many practical applications in network optimisation.

In this work we present an approximate and efficient algorithm for the (k, γ)-spectral augmentability problem, and our algorithm runs in almost-linear time under a wide regime of parameters. Our main algorithm is based on the following two novel techniques developed in the paper, which might have applications beyond the (k, γ)-spectral augmentability problem:

- We present a fast algorithm for solving a feasibility version of an SDP for the algebraic connectivity maximisation problem from [16]. Our algorithm is based on the classic primal-dual framework for solving SDP, which in turn uses the multiplicative weight update algorithm. We present a novel approach of unifying SDP constraints of different matrix and vector variables and give a good separation oracle accordingly.
- We present an efficient algorithm for the subgraph sparsification problem, and for a wide range of parameters our algorithm runs in almost-linear time, in contrast to the previously best known algorithm running in at least $\Omega(n^2mk)$ time [22]. Our analysis shows how the randomised BSS framework can be generalised in the setting of subgraph sparsification, and how the potential functions can be applied to approximately keep track of different subspaces.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph sparsification, Algebraic connectivity, Semidefinite programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.70

Related Version The full version of the paper is available at `https://arxiv.org/abs/2006.14449`.

Funding *He Sun*: Support by an EPSRC Early Career Fellowship (EP/T00729X/1).

1 Introduction

Graph expansion is the metric quantifying how well vertices are connected in a graph, and has applications in many important problems of computer science: in complexity theory, graphs with good expansion are used to construct error-correcting codes [38, 42] and pseudorandom generators [18]; in network design, expander graphs have been applied in constructing super concentrators [40]; in probability theory, graph expansion is closely related to the behaviours of random walks in a graph [29, 37]. On the other side, as most graphs occurring in practice might not be expander graphs and a subset of vertices of low expansion is usually viewed as the

bottleneck of a graph, finding the set of vertices with minimum expansion has many practical applications including image segmentation [27], community detection [31, 36], ranking web pages, among many others. Because of these, both the approximation algorithms for the graph expansion problem and the computational complexity of the problem itself have been extensively studied over the past three decades.

In this paper we study the following graph expansion optimisation problem: given an undirected and weighted graph $G = (V, E, w)$, a set E_W of candidate edges, and a parameter $k \in \mathbb{N}$ as input, we are interested in (i) finding a set $F \subseteq E_W$ of k edges and their weights such that the resulting graph $H = (V, E \cup F, w')$ with weight function[1] $w' : E \cup F \to \mathbb{R}_{\geq 0}$ has good expansion, or (ii) showing that it's impossible to significantly improve the graph's expansion by adding k edges from E_W. Despite sharing many similarities with the sparest cut problem, our proposed problem has many of its own applications: for example, assume that the underlying graph G is a practical traffic or communication network and, due to physical constraints, only certain links can be used to improve the network's connectivity. For any given k and a set of feasible links, finding the best k links to optimise the network's connectivity is exactly the objective of our graph expansion optimisation problem.

To formalise the problem, we follow the work of [15, 16, 22] and define the *algebraic connectivity* of G by the second smallest eigenvalue $\lambda_2(L_G)$ of the Laplacian matrix L_G of G defined by $L_G \triangleq D_G - A_G$, where D_G is the diagonal matrix consisting of the degrees of the vertices and A_G is the adjacency matrix of G. Given an undirected and weighted graph $G = (V, E, w)$ with n vertices, $O(n)$ edges[2], a set E_W of candidate edges defined on V satisfying $E_W \cap E = \emptyset$ and a parameter k, we say that G is (k, γ)-*spectrally-augmentable with respect to* $W = (V, E_W)$, if there is $F \subseteq E_W$ with $|F| = k$ together with edge weights $\{w_e\}_{e \in F}$ such that $H = (V, E \cup F, w)$ satisfies $\lambda_2(L_H) \geq \gamma$. The main result of our work is an almost-linear time[3] algorithm that either (i) finds a set of $O(k)$ edges from E_W if G is (k, γ)-spectrally augmentable for some $\gamma \geq \Delta \cdot n^{-1/q}$, or (ii) returns "no" if G is not $(O(kq), O(\Delta \cdot n^{-2/q}))$-spectrally augmentable, where Δ is an upper bound of both the maximum degree of G and W, and $q \geq 10$ is an arbitrary integer. The formal description of our result is as follows:

▶ **Theorem 1.** *Let $G = (V, E, w)$ be a base graph with n vertices, $O(n)$ edges, and weight function $w : E \to \mathbb{R}_{\geq 0}$, and let $W = (V, E_W)$ be the candidate graph of m edges such that the maximum degrees of G and W is at most Δ. Then, there is an algorithm such that for any integer $k \geq 1$ and $q \geq 10$, the following statements hold:*

- *if G is $\left(k, \Delta \cdot n^{-1/q}\right)$-spectrally-augmentable with respect to W, then the algorithm finds a set $F \subseteq E_W$ of edges and a set of edge weights $\{w(e) : e \in F\}$ such that $|F| = O(qk)$, $\sum_{e \in F} w(e) \leq O(k)$, and the resulting graph $H = (V, E \cup F)$ satisfies that $\lambda_2(L_H) \geq c\lambda_\star^2 \Delta$, for some constant $c > 0$, where $\lambda_\star \cdot \Delta$ is the optimum solution[4].*

---

[1] We remark that the weight function w' needs to satisfy that $w'(e) = w(e)$ for any edge $e \in E$.

[2] Since a spectral sparsifier of G with $O(n)$ edges preserves the eigenvalues of the Laplacian matrix of G, we assume that G has $O(n)$ edges throughout the paper. Otherwise one can always run the algorithm in [25] to get a linear-sized spectral sparsifier of G and use this sparsifier as the input of our problem. Therefore, the number of edges in G will not be mentioned in our paper to simplify the notation.

[3] We say that a graph algorithm runs in almost-linear time if the algorithm's runtime is $O((m+n)^{1+c})$ for an arbitrary small constant c, where n and m are the number of vertices and edges of G, respectively. Similarly, we say that a graph algorithm runs in nearly-linear time if the algorithm's runtine is $O((m+n) \cdot \log^c(n))$ for some constant c.

[4] Note that since G is $(k, n^{-1/q} \cdot \Delta)$-spectrally-augmentable with respect to W, it always holds that $\lambda_\star \geq n^{-1/q}$.

- *if G is not $\left(O(kq), O(\Delta \cdot n^{-2/q})\right)$-spectrally-augmentable with respect to W, then the algorithm rejects the input G, W.*

Moreover, the algorithm runs in $\widetilde{O}\left(\min\left\{qn^{\omega+O(1/q)}, q(m+n)n^{O(1/q)}k^2\right\}\right)$ time. Here, the $\widetilde{O}(.)$ notation hides $\operatorname{poly}\log n$ factors, and ω is the constant for matrix multiplication.

We remark that the most typical application of our problem is the scenario in which only a low number of edges are needed such that the resulting graph enjoys good expansion, and these correspond to the regime of $k = n^{o(1)}$ and $\lambda_\star \in (n^{-1/q}, O(1))$ [34], under which our algorithm runs in almost-linear time and achieves an $\Omega(\lambda_\star)$-approximation. In particular, when it is possible to augment G to be an expander graph, i.e. $\lambda_\star = \Theta(1)$, our algorithm achieves a constant-factor approximation. Our algorithm runs much faster than the previously best-known algorithm for a similar problem that runs in *at least* $\Omega\left(n^2mk\right)$ time [22], though their algorithm solves the more general problem: for any instance G, W, k, if the optimum solution is $\lambda_\star\Delta$, i.e., G is $(k, \lambda_\star\Delta)$-spectrally-augmentable with respect to W, for *any* $\lambda_\star \in [0, 1)$, then their algorithm finds a graph $H = (V, E \cup F)$ with $\lambda_2(L_H) \geq c\lambda_\star^2\Delta$ such that $|F| = O(k)$ and the total sum of weights of edges in F is at most k. Our algorithm can only find a graph H when $\lambda_\star \in (n^{-1/q}, 1)$.

To give an overview of the proof technique for Theorem 1, notice that our problem is closely linked to the *algebraic connectivity maximisation problem* studied in [16], which looks for k edges from the candidate set to maximise $\lambda_2(L_H)$ of the resulting graph H. It is known that the algebraic connectivity maximisation problem is **NP**-hard [30], and Ghosh and Boyd [16] show that this problem can be formulated as an SDP, which we call the GB-SDP. Inspired by this, we study the following P-SDP, which is the feasibility version of the GB-SDP parameterised by some γ. Here, $P_\perp$ is the projection on the space orthogonal to $\mathbf{1} \triangleq (1, \ldots, 1)^\intercal$, i.e., $P_\perp = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\intercal$.

$$\text{P-SDP}(G, W, k, \gamma) \qquad \begin{aligned} \lambda &\geq \gamma \\ L_G + \sum_{e \in E_W} w_e L_e &\succeq \lambda\Delta P_\perp \\ k - \sum_{e \in E_W} w_e &\geq 0 \\ 1 - w_e &\geq 0, \qquad \forall e \in E_W \\ w_e &\geq 0, \qquad \forall e \in E_W \\ \gamma &\geq 0. \end{aligned}$$

Notice that, if G is $(k, \gamma\Delta)$-spectrally-augmentable with respect to W, then there is a set F of k edges such that, by setting $w_e = 1$ if $e \in F$ and $w_e = 0$ otherwise, it holds that $L_G + \sum_{e \in E_W} L_e \succeq \gamma\Delta P_\perp$. Therefore, there is a feasible solution of P-SDP(G, W, k, γ). Our algorithmic result for solving the P-SDP is summarised as follows:

▶ **Theorem 2.** *Let $\delta' > 0$ be any constant. There exists an algorithm running in $\widetilde{O}((m+n)/\gamma^2)$ time that either finds a solution to P-SDP$(G, W, k, (1-\delta')\gamma)$ or certifies that there is no feasible solutions for P-SDP(G, W, k, γ).*

Since the solution to the P-SDP only guarantees that the total weights of the selected edges are at most k if G is $(k, \gamma\Delta)$-spectrally augmentable, following [22] we use a subgraph sparsification algorithm to round our SDP solution, such that there are only $O(k)$ edges selected in the end. To give a high-level overview of this rounding step, we redefine the set E_W of candidate edges, and assume that E_W consists of the edges whose weight from the P-SDP solution is non-zero. Therefore, our objective is to find $O(k)$ edges from E_W and new weights, which form an edge set F, such that the Laplacian matrix L_H of the

resulting graph $H = (V, E \cup F)$ is close to L_{G+W}. That is, the subgraph sparsification problem asks for a sparse representation of $G + W$ while keeping the entire base graph G in the resulting representation. Our improved algorithm shows that, as long as $k = n^{o(1)}$, a subgraph sparsifier can be computed in almost-linear time[5]. Our result on subgraph sparsification will be formally described in Theorem 9.

1.1 Our techniques

In this section we will explain the techniques used to design the fast algorithm for the P-SDP, and an almost-linear time algorithm for subgraph sparsification.

Faster algorithm for solving the P-SDP. Our efficient SDP solver is based on the primal-dual framework developed in [5], which has been used in many other works [19, 33]. In this primal-dual framework, we will work on both the original SDP P-SDP(G, W, k, γ) and its dual D-SDP(G, W, k, γ) that is defined as follows:

$$\begin{aligned} Z \bullet L_G + kv + \sum_{e \in E_W} \beta_e &< \gamma \\ Z \bullet \Delta P_\perp &= 1 \\ \text{D-SDP}(G, W, k, \gamma) \qquad Z \bullet L_e &\le v + \beta_e, \qquad \forall e \in E_W \\ Z &\succeq 0 \\ \beta_e &\ge 0, \qquad \forall e \in E_W \\ v &\ge 0. \end{aligned}$$

We then apply the matrix multiplicative weight update (MWU) algorithm. Formally speaking, starting with some initial embedding $X^{(1)}$, for each $t \ge 1$ our algorithm iteratively uses a carefully constructed oracle Oracle for D-SDP(G, W, k, γ) to check whether the current embedding $X^{(t)}$ is good or not:

- If $X^{(t)}$ satisfies some condition, denoted by $\mathcal{C}\left(X^{(t)}\right)$, then the oracle fails and this implies that we can find a feasible solution from $X^{(t)}$ to D-SDP(G, W, k, γ). This implies that the primal SDP P-SDP(G, W, k, γ) has no feasible solution, which certifies that G is not (k, γ)-spectrally-augmentable with respect to W.
- If $X^{(t)}$ does not satisfy the condition $\mathcal{C}\left(X^{(t)}\right)$, then the oracle does not fail, which certifies that the current solution from $X^{(t)}$ is not feasible for D-SDP(G, W, k, γ), and will output a set of numbers $\left(\lambda^{(t)}, w^{(t)}\right)$ for updating the embedding.

The procedure above will be iterated for T times, for some T depending on the oracle and the approximate parameter $\delta' > 0$: if the oracle fails in any iteration, then P-SDP is infeasible; otherwise, the oracle does not fail for all T iterations and we find a feasible solution to P-SDP$(G, W, k, (1 - \delta')\gamma)$.

The main challenge for applying the above framework in our setting is to construct the Oracle and deal with the complicated constraints in our SDPs, which include both matrix inequality constraint and vector inequality constraints of different variables. To work with these constraints, our strategy is to unify them through a diagonal block matrix X, and through this we turn all individual constraints into a single matrix constraint. The embedding in each iteration is constructed in nearly-linear time in $n + m$ by the definition

[5] We remark that, when $k = \Theta(n)$, our problem can be solved directly by using a spectral sparsifier $\widetilde{W}$ of the graph W with $O(n)$ edges, which can be computed in nearly-linear time. This will imply that the two Laplacians $L_{G+\widetilde{W}} = L_G + L_{\widetilde{W}}$ and $L_{G+W} = L_G + L_W$ are close.

of the embedding. To construct the ORACLE, we carefully design the condition $\mathcal{C}(X)$ with the intuition that if the candidate solution corresponding to X has a relatively small dual objective value, then a re-scaling of X gives a feasible solution to D-SDP. Then we use a case analysis to show that if $\mathcal{C}(X)$ is not satisfied, we can very efficiently find updating numbers $(\lambda^{(t)}, w^{(t)})$ by distinguishing edges satisfying one constraint (in D-SDP) from those that do not satisfy it and assigning different weights $w^{(t)}$ to them accordingly.

Faster algorithm for subgraph sparsification. The second component behind proving our main result is an efficient algorithm for the subgraph sparsification problem. Our algorithm is inspired by the the original deterministic algorithm for subgraph sparsification [22] and the almost-linear time algorithm for constructing linear-sized spectral sparsifiers [26]. In particular, both algorithms follow the BSS framework, and proceed in iterations: it is shown that, with the careful choice of barrier values u_j and ℓ_j in each iteration j and the associated potential functions, one or more vectors can be selected in each iteration and the final barrier values can be used to bound the approximation ratio of the constructed sparsifier.

However, in contrast to most algorithms for linear-sized spectral sparsifiers [4, 25, 26], both the barrier values and the potential functions in [22] are employed for a slightly different purpose. In particular, instead of expecting the final constructed ellipsoid to be close to a sphere, the final constructed ellipsoid for subgraph sparsification could be still very far from being a sphere, since the total number of added edges is $O(k)$. Because of this, the two potential functions in [22] are used to quantify the contribution of the added vectors towards *two different subspaces*: one fixed k-dimensional subspace denoted by S, and one variable space defined with respect to the currently constructed matrix. Based on analysing two different subspaces for every added vector, which is computationally expensive, the algorithm in [22] ensures that the added vectors will significantly benefit the "worst subspace", the subspace in $\mathbb{R}^n$ that limits the approximation ratio of the final constructed sparsifier.

Because of these different roles of the potential functions in [22] and [6, 26], when applying the randomised BSS framework [26] for the subgraph sparsification problem, more technical issues need to take into account: (1) Since [22] crucially depends on some projection matrix denoted by P_S, of which the exact computation is expensive, to obtain an efficient algorithm for subgraph sparsification one needs to obtain some projection matrix close to P_S and such a projection matrix can be computed efficiently. (2) Since the upper and lower potential functions keep track of different subspaces whose dimensions are of different orders in most regimes, analysing the impact of multiple added vectors to the potential functions are significantly more challenging than [26].

To address the first issue, we show that the problem of computing an approximate projection close to P_S while preserving relevant proprieties can be reduced to the generalised eigenvalue problem, which in turn can be efficiently approximated by a recent algorithm [2]. For the second issue, we meticulously bound the *intrinsic dimension* of the matrix corresponding to the multiple added vectors, and by a more refined matrix analysis than [26] we show that the potential functions and the relative effective resistances decease in each iteration. We highlight that developing a fast procedure to computing all the quantities that involve a fixed projection matrix and analysing the impact of multiple added vectors with respect to two different subspaces constitute the most challenging part of the design of our algorithm.

Finally, we remark that, although the almost-linear time algorithm [26] has been improved by the subsequent paper [25], it looks more challenging to adapt the technique developed in [25] for the setting of subgraph sparsification. In particular, since the two potential functions

in [25] are used to analyse the same space $\mathbb{R}^n$, it is shown in [25] that it suffices to analyse the one-sided case through a one-sided oracle. However, the two potential functions defined in our paper are used to analyse two different subspaces, and it remains unclear whether we can reduce our problem to the one-sided case. We will leave this for future work.

1.2 Other related work

Spielman and Teng [39] present the first algorithm for constructing spectral sparsifiers: for any parameter $\varepsilon \in (0,1)$, and any undirected graph G of n vertices and m edges, they prove that a spectral sparsifier of G with $\widetilde{O}\left(n/\varepsilon^2\right)$ edges exists, and can be constructed in $\widetilde{O}\left(m/\varepsilon^2\right)$ time. Since then, there has been extensive studies on different variants of spectral sparsifiers and their efficient constructions in various settings. In addition to several results on several constructions of linear-sized spectral sparsifiers mentioned above, there are many studies on constructing spectral sparsifiers in streaming and dynamic settings [1, 20, 21]. The subgraph sparsification problem has many applications, including constructing precondtioners and nearly-optimal ultrasparsifiers [22, 35], optimal approximate matrix product [11], and some network optimisation problems [28]. Our work is also related to a sequence of research on network design, in which the goal is to find minimum cost subgraphs under some "connectivity constraints". Typical examples include constraints on vertex connectivity [8, 9, 10, 14, 23, 24], shortest path distances [12, 13], and spectral information [3, 7, 17, 32].

2 A fast SDP solver

We use the primal-dual framework introduced in [5] to solve the SDP $\mathsf{P\text{-}SDP}(G, W, k, \gamma)$ and prove Theorem 2. The framework is based on the matrix multiplicative weight update (MWU) algorithm on both the primal SDP $\mathsf{P\text{-}SDP}(G, W, k, \gamma)$ and its dual $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$.

Notation. For any given vector β, we use $\mathbf{Diag}(\beta)$ to denote the diagonal matrix such that each diagonal entry $[\mathbf{Diag}(\beta)]_{ii} = \beta_i$. Given matrix Z, scalar v and vector β, we use $\mathbf{Diag}(Z, v, \beta)$ to denote the diagonal 3-block matrix with blocks Z, v and $\mathbf{Diag}(\beta)$. We use I_V and I_{E_W} to denote the identity matrices on vertex set V and edge set E_W with $|E_W| = m$, respectively. We further define

$$\begin{aligned} &E \triangleq \mathbf{Diag}(\Delta \cdot I_V, m, I_{E_W}), \quad \Pi \triangleq \mathbf{Diag}(P_\perp, 1, I_{E_W}), \\ &N \triangleq \mathbf{Diag}(\Delta \cdot P_\perp, m, I_{E_W}) = E^{1/2}\Pi E^{1/2}. \end{aligned} \tag{1}$$

For any given parameter λ and vector w, we define $V(\lambda, w) \triangleq \lambda$, $A(\lambda, w) \triangleq L_G + \sum_{e\in E_W} w_e L_e - \lambda\Delta P_\perp$, and $B(\lambda, w) \triangleq k - \sum_{e\in E_W} w_e$. Let $c = c(\lambda, w) \in \mathbb{R}^m$ denote the vector with $c_e = 1 - w_e$ for each $e \in E_W$, and $C = C(\lambda, w) = \mathbf{Diag}(c(\lambda, w))$ be the diagonal $m \times m$ matrix with the diagonal entry $1 - w_e$ corresponding to edge e. Then we define

$$M(\lambda, w) \triangleq \mathbf{Diag}\left(A(\lambda, w), B(\lambda, w), C(\lambda, w)\right) = \begin{bmatrix} A(\lambda, w) & 0 & 0 \\ 0 & B(\lambda, w) & 0 \\ 0 & 0 & C(\lambda, w) \end{bmatrix}. \tag{2}$$

▶ **Definition 3.** *An (ℓ, ρ)-oracle for $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$ is an algorithm that on input $\langle Z, v, \beta\rangle$ with $\mathbf{Diag}(Z, v, \beta) \bullet N = 1$, either fails or outputs (λ, w) with $\lambda \geq 0$, $w \in \mathbb{R}^m_{\geq 0}$ that satisfies*

$$V(\lambda, w) \geq \gamma, \qquad A(\lambda, w) \bullet Z + B(\lambda, w) \cdot v + c(\lambda, w) \cdot \beta \geq 0, \qquad -\ell N \preceq M(\lambda, w) \preceq \rho N.$$

▶ **Fact 4.** *If an (ℓ, ρ)-oracle for D-SDP(G, W, k, γ) does not fail on input $\langle Z, v, \beta\rangle$ with $\boldsymbol{Diag}(Z, v, \beta) \bullet N = 1$, then $\langle Z, v, \beta\rangle$ is infeasible for D-SDP(G, W, k, γ).*

In order to apply the MWU algorithm, in the following we use $U_\varepsilon(A)$ to denote the matrix

$$U_\varepsilon(A) \triangleq \frac{E^{-1/2}(1-\varepsilon)^{E^{-1/2}AE^{-1/2}}E^{-1/2}}{\Pi \bullet (1-\varepsilon)^{E^{-1/2}AE^{-1/2}}},$$

where E and Π are matrices defined in (1).

2.1 The MWU algorithm

In the framework of MWU for solving our SDP, we sequentially produce candidate dual solutions $\langle Z^{(t)}, v^{(t)}, \beta^{(t)}\rangle$ such that $\mathbf{Diag}(Z^{(t)}, v^{(t)}, \beta^{(t)}) \bullet N = 1$ for any t. Specifically, for any given k, γ, we start with a solution $Z^{(1)} = \frac{1}{\Delta(n-1)}I$, $v^{(1)} = \frac{2}{n-1}$ and $\beta_e^{(1)} = 0$ for any $e \in E_W$. At each iteration t, we invoke a good separation oracle that takes $\mathbf{Diag}(Z^{(t)}, v^{(t)}, \beta^{(t)})$ as input, and either guarantees that $\mathbf{Diag}(Z^{(t)}, v^{(t)}, \beta^{(t)})$ is already good for dual SDP (and thus certifies infeasibility of primal SDP), or outputs $(\lambda^{(t)}, w^{(t)})$ certifying the infeasibility of $\mathbf{Diag}(Z^{(t)}, v^{(t)}, \beta^{(t)})$.

If $\left(\lambda^{(t)}, w^{(t)}\right)$ is returned by the oracle, then the algorithm updates the next candidate solution based on $X^{(t)} = U_\varepsilon\left(\frac{1}{2\rho}\sum_{s=1}^{t-1} M^{(s)}\right)$, where $M^{(s)} \triangleq M\left(\lambda^{(s)}, w^{(s)}\right)$ is as defined before and ε is a parameter of the algorithm. By definition, we have that $X^{(t)} \bullet N = 1$. Moreover, since $M^{(t)}$ can be viewed as a 3-block diagonal matrix with diagonal entries $A^{(t)}, B^{(t)}, C^{(t)}$, we have that $\exp(M^{(t)}) = \mathbf{Diag}\left(\exp\left(A^{(t)}\right), \exp\left(B^{(t)}\right), \exp\left(C^{(t)}\right)\right)$. Therefore, we can decompose $X^{(t)}$ as

$$X^{(t)} = \mathbf{Diag}\left(Z^{(t)}, v^{(t)}, \beta^{(t)}\right).$$

Note that $X^{(t)} \bullet N = 1$ is equivalent to

$$\Delta \cdot Z^{(t)} \bullet P_\perp + m \cdot v^{(t)} + \sum_{e \in E_W} \beta_e^{(t)} = 1.$$

The following theorem guarantees that, after a small number of iterations, the algorithm either finds a good enough dual solution, or a feasible solution to the primal SDP.

▶ **Theorem 5.** *Let Oracle be an (ℓ, ρ)-oracle for D-SDP(G, W, k, γ), and let $\delta > 0$. Let N, $X^{(t)}$, and $M^{(t)}$ be defined as above, for any $t \geq 1$. Let $\varepsilon = \min\{1/2, \delta/2\ell\}$. Suppose that Oracle does not fail for T rounds, where*

$$T = O\left(\frac{\rho \log n}{\delta\varepsilon}\right) \leq \max\left\{O\left(\frac{\rho \log n}{\delta}\right), O\left(\frac{\rho\ell \log n}{\delta^2}\right)\right\},$$

then $(\bar{\lambda} - 3\delta, \bar{w} - \delta)$ is a feasible solution to P-SDP$(G, W, k, \gamma - 3\delta)$, where $\bar{\lambda} \triangleq \frac{1}{T}\sum_{t=1}^{T}\lambda^{(t)}$ and $\bar{w} \triangleq \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$.

Approximate computation. By applying the Johnson-Linderstrauss (JL) dimensionality reduction to the embedding corresponding to U_ε, we can approximate $X^{(t+1)}$ by $\widetilde{X}^{(t+1)}$ while preserving the relevant properties. Specifically, let $\widetilde{U}_\varepsilon$ be a randomised approximation to U_ε from applying the JL Lemma, and we compute in nearly-linear time the matrix $\widetilde{X}^{(t+1)} = \widetilde{U}_\varepsilon\left(\frac{1}{2\rho}\sum_{s=1}^{t-1} M^{(s)}\right)$ and decompose it into 3 blocks:

$$\widetilde{X}^{(t+1)} = \mathbf{Diag}\left(\widetilde{Z}^{(t+1)}, \widetilde{v}^{(t+1)}, \mathbf{Diag}\left(\widetilde{\beta}^{(t+1)}\right)\right).$$

Moreover, $\widetilde{X}^{(t+1)} \bullet L_H$ well approximates $X^{(t+1)} \bullet L_H$ for any graph H, which suffices for our oracle. Hence, we assume that the oracle receives $\widetilde{X}^{(t+1)}$ as input instead of $X^{(t+1)}$. We defer the formal lemma we are using to the full version.

2.2 The oracle

In this subsection, we will present and analyse the oracle for our SDP $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$, which is presented in Algorithm 1. For the simplicity of presentation, we abuse notation and use $X = \mathbf{Diag}(Z, v, \beta)$ to denote the input to the oracle, although it should be clear that the input is the approximate embedding $\widetilde{X} = \mathbf{Diag}(\widetilde{Z}, \widetilde{v}, \mathbf{Diag}(\widetilde{\beta}))$ of X.

Algorithm 1 ORACLE for SDP $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$.

Require: Candidate solution $\langle Z, v, \beta \rangle$ with $\Delta \cdot Z \bullet P_\perp + m \cdot v + \sum_{e \in E_W} \beta_e = 1$, target value γ

Let $B := \{e : v + \beta_e < L_e \bullet Z\}$, $\Gamma := \sum_{e \in B}(L_e \bullet Z - v - \beta_e)$, and $T := Z \bullet \Delta P_\perp$.
Let $T_{\mathrm{tol}} := L_G \bullet Z + kv + \sum_{e \in E_W} \beta_e$.
if $\Gamma \leq T\gamma - T_{\mathrm{tol}}$ **then**
    Output "fail". ▷ In this case, $\langle Z, v, \beta \rangle$ is "good" enough
else if $T_{\mathrm{tol}} > \gamma m - \gamma \sum_{e \in E_W} Z \bullet L_e$ **then**
    return $w_e = \gamma$, and $\lambda = \gamma$.
else
    return $w_e = 1$ for $e \in B$, $w_e = 0$ for $e \in E_W \setminus B$ and $\lambda = \gamma$

To analyse the oracle, we prove the following technical lemma. First of all, we show that if the ORACLE fails, then we can find a dual feasible solution for $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$.

▶ **Lemma 6.** *Let $\langle Z, v, \beta \rangle$ be a candidate solution. Suppose that for $B \triangleq \{e : v + \beta_e - L_e \bullet Z < 0\}$, $T \triangleq Z \bullet \Delta P_\perp$ and $T_{tol} \triangleq L_G \bullet Z + kv + \sum_{e \in E_W} \beta_e$, then it holds that $\Gamma \triangleq \sum_{e \in B}(L_e \bullet Z - v - \beta_e) \leq T\gamma - T_{tol}$. Moreover, by setting $Z' = \frac{Z}{T}$, $v' = \frac{v}{T}$, and $\beta'_e = \frac{\beta_e}{T}$ if $e \in E_W \setminus B$ and $\beta'_e = \frac{L_e \bullet Z - v}{T}$ if $e \in B$, we have that $\langle Z', v', \beta' \rangle$ is a dual feasible for $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$.*

Secondly, we show that, if ORACLE does not fail, then it returns (λ, w) that satisfies the properties of (ℓ, ρ)-oracle for $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$ for appropriate ℓ, ρ.

▶ **Lemma 7.** *When ORACLE described in the algorithm does not fail, it returns a vector w and value λ such that $V(\lambda, w) \geq \gamma$, and for the matrix $M(\lambda, w) = \mathbf{Diag}(A(\lambda, w), B(\lambda, w), C(\lambda, w))$, $A(\lambda, w) \bullet Z + B(\lambda, w) \cdot v + C(\lambda, w) \cdot \beta \geq 0$. Moreover, it holds that $-N \preceq M(\lambda, w) \preceq 3N$.*

Combining these two lemmas together, we obtain the following theorem which summarise the performance of Algorithm 1.

▶ **Theorem 8.** *On input $\widetilde{X}^{(t)}$, there exists an algorithm ORACLE that runs in $\widetilde{O}(n + m)$ time and is an (ℓ, ρ)-oracle for SDP $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$, where $\ell = 1$ and $\rho = 3$.*

2.3 Proof of Theorem 2

Now we are ready to prove Theorem 2.

Proof of Theorem 2. Let $\delta' > 0$ be any constant. We specify $\delta = \frac{\delta'\gamma}{3}$ in our MWU algorithm, which is described in the previous subsections. We set $\rho = 3$ and $\ell = 1$, and let

$$T \triangleq O\left(\frac{\rho \ell \log n}{\delta^2}\right) = O\left(\frac{\log n}{(\delta')^2 \gamma^2}\right) = O\left(\frac{\log n}{\gamma^2}\right).$$

In the MWU algorithm, if the ORACLE fails in the t-th iteration for some $1 \le t \le T$, then the corresponding embedding $\widetilde{X}^{(t)} = \mathbf{Diag}(\widetilde{Z}^{(t)}, \widetilde{v}^{(t)}, \mathbf{Diag}(\beta^{(t)}))$ provides a good enough solution: the precondition of Lemma 6 is satisfied, which further implies that $\widetilde{X}^{(t)}$ can be turned into a dual feasible solution with objective at most γ, i.e., we find a solution to D-SDP(G, W, k, γ). Therefore, the primal SDP P-SDP(G, W, k, γ) is infeasible. Otherwise, we know that the ORACLE does not fail for T iterations; by Theorem 5 and Lemma 7, it holds that for $\bar{\lambda} \triangleq \frac{1}{T}\sum_{t=1}^{T} \lambda^{(t)}, \bar{w} \triangleq \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$, $(\bar{\lambda} - 3\delta, \bar{w} - \delta)$ is a feasible solution for P-SDP$(G, W, k, \gamma - 3\delta)$ = P-SDP$(G, W, k, \gamma - \delta'\gamma)$.

Now we analyse the algorithm's runtime. By Theorem 8 and the approximate computation of $X^{(t+1)}$, each iteration can be implemented in $\widetilde{O}(n+m)$ time. Thus, in $\widetilde{O}((n+m)/\gamma^2)$ time, we either find a solution to our SDP with objective value at least $(1-\delta')\gamma$ for any constant $\gamma' > 0$, or we certify that the P-SDP(G, W, k, γ) is infeasible (in case the ORACLE fails). ◀

3 Algorithm for subgraph sparsification

Now we give an overview of our efficient algorithm for constructing subgraph sparsifiers. Recall that, for any $k \in \mathbb{N}$, parameter $\kappa \ge 1$, and two weighted graphs $G = (V, E)$ and $W = (V, E_W)$, the subgraph sparsification problem is to find a set $F \subseteq E_W$ of $|F| = O(k)$ edges with weights $\{w_e\}_{e\in F}$, such that the resulting graph $H = (V, E + F)$ is a κ-approximation of $G + W$, i.e.,

$$L_{G+W} \preceq L_G + \sum_{e\in F} w_e b_e b_e^{\intercal} \preceq \kappa \cdot L_{G+W}. \tag{3}$$

To construct the required edge set F, we apply the standard reduction for constructing graph sparsifiers by setting $v_e \triangleq L_{G+W}^{\dagger/2} b_e$ for every $e \in E_W$, and (3) is equivalent to

$$I_{\mathsf{im}(L_{G+W})} \preceq L_{G+W}^{\dagger/2} L_G L_{G+W}^{\dagger/2} + \sum_{e\in F} w_e v_e v_e^{\intercal} \preceq \kappa \cdot I_{\mathsf{im}(L_{G+W})},$$

where $I_{\mathsf{im}(L_{G+W})}$ is the identity on $\mathsf{im}(L_{G+W})$. Our main result is summarised as follows:

▶ **Theorem 9.** *Let ε and q be arbitrary constants such that $\varepsilon \le 1/20$ and $q \ge 10$. Then, there is a randomised algorithm such that, for any two graphs $G = (V, E)$ and $W = (V, E_W)$ defined on the same vertex set as input, by defining $X = \left(L_{G+W}^{\dagger/2} L_G L_{G+W}^{\dagger/2}\right)\Big|_{\mathrm{Im}(L_{G+W})}$ and $\overline{M} \triangleq \left(\sum_{i=1}^{m} v_i v_i^{\intercal}\right)|_{\mathrm{Im}(L_{G+W})}$ where every v_i is of the form $L_{G+W}^{\dagger/2} b_e$ for some edge $e \in E_W$, the algorithm outputs a set of non-negative coefficients $\{c_i\}_{i=1}^{m}$ with $|\{c_i \mid c_i \ne 0\}| = K$ for some $K = O\left(qk/\varepsilon^2\right)$ such that it holds for some constant C that*

$$C \cdot (1 - O(\varepsilon)) \cdot \min\{1, K/T\} \cdot \lambda_{k+1}(X) \cdot I \preceq X + \sum_{i=1}^{m} c_i v_i v_i^{\intercal} \preceq (1 + O(\varepsilon)) \cdot I, \tag{4}$$

where $T \triangleq \lceil \mathrm{tr}\left(\overline{M}\right) \rceil$. Moreover, if we assume that every v_i is associated with some cost denoted by $cost_i$ such that $\sum_{i=1}^{m} cost_i = 1$, then with constant probability the coefficients $\{c_i\}_{i=1}^{m}$ returned by the algorithm satisfy $\sum_{i=1}^{m} c_i \cdot cost_i \le O(1/\varepsilon^2) \cdot \min\{1, k/T\}$. The algorithm runs in time

$$\widetilde{O}\left(\min\left\{n^{\omega}, \frac{mk + nk^2}{\sqrt{\lambda_{k+1}(X)}}\right\} + q \cdot n^{O(1/q)} \left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^{\omega}, mk + nk^2 + k^{\omega}\right\}\right) \Big/ \varepsilon^5\right).$$

Without loss of generality, we assume that $\overline{M}$ is a full-rank matrix, which can be achieved by adding n self-loops each of small weight $\gamma = \Theta(1/\mathrm{poly}(n))$, so that with constant probability these self-loops will not be sampled by the algorithm.

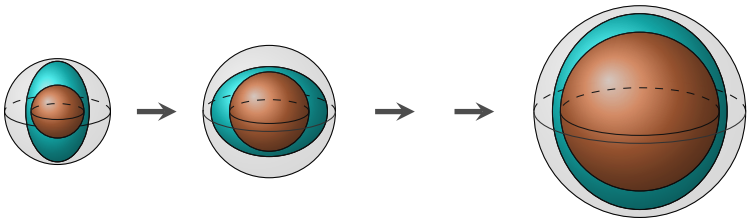


Figure 1 Illustration of the BSS framework: the light grey and orange balls in iteration j represent the spheres $u_j \cdot I$ and $\ell_j \cdot I$, and the cyan ellipsoid sandwiched between the two balls corresponds to the constructed ellipsoid in iteration j. After each iteration j, the algorithm increases the value of ℓ_j and u_j by some $\delta_{\ell,j}$ and $\delta_{u,j}$ so that the invariant (5) holds in iteration $j+1$. This process is repeated for τ iterations, so that the final constructed ellipsoid is close to be a sphere.

3.1 Overview of our algorithm

The BSS framework. At a high level, our algorithm follows the BSS framework for constructing spectral sparsifiers [6]. The BSS algorithm proceeds by iterations: in each iteration j the algorithm chooses one or more vectors, denoted by $v_{j_1}, \ldots, v_{j_k}$, and adds $\Delta_j = \sum_{i=1}^{k} c_{j_i} v_{j_i} v_{j_i}^\intercal$ to the currently constructed matrix by setting $A_j = A_{j-1} + \Delta_j$, where $c_{j_1}, \ldots, c_{j_k}$ are scaling factors, and $A_0 = \mathbf{0}$ initially. Moreover, two barrier values, the *upper barrier* u_j and the *lower barrier* ℓ_j, are maintained such that the constructed ellipsoid $\mathsf{Ellip}(A_j)$ is sandwiched between the outer sphere $u_j \cdot I$ and the inner sphere $\ell_j \cdot I$ for any iteration j. To ensure this, all the previous analysis uses a potential function $\Phi(A_j, u_j, \ell_j)$ defined by

$$\Phi\left(A_j, u_j, \ell_j\right) = \operatorname{tr}[f(u_j I - A_j)] + \operatorname{tr}[f(A_j - \ell_j I)]$$

for some function f, and a bounded value of $\Phi\left(A_j, u_j, \ell_j\right)$ implies that

$$\ell_j \cdot I \prec A_j \prec u_j \cdot I. \tag{5}$$

After each iteration, the two barrier values ℓ_j and u_j are increased properly by setting $u_{j+1} = u_j + \delta_{u,j}$ and $\ell_{j+1} = \ell_j + \delta_{\ell,j}$ for some positive values $\delta_{u,j}$ and $\delta_{\ell,j}$. The careful choice of $\delta_{u,j}$ and $\delta_{\ell,j}$ ensures that after τ iterations $\mathsf{Ellip}(A_\tau)$ is close to being a sphere, which implies that A_τ is a spectral sparsifier of $\ell_\tau \cdot I$, see Figure 1 for illustration.

The BSS framework for subgraph sparsification. The BSS framework ensures that, when starting with the zero matrix, after choosing $O(n)$ vectors, the final constructed matrix is close to I. However, applying the BSS framework to construct a subgraph sparsifier is significantly more challenging due to the following two reasons:

- Instead of starting with the zero matrix, we need to start with some *non-zero* matrix $A_0 = X$, and the number of added vectors is $K = O(k)$, which could be much smaller than n. This implies that the ellipsoid corresponding to the final constructed matrix could be still very far from being a sphere.
- Because of this and every rank-one update has different contribution towards each direction in $\mathbb{R}^n$, to "optimise" the contribution of $O(k)$ rank-one updates we have to ensure that the added vectors will significantly benefit the "worst subspace", the subspace in $\mathbb{R}^n$ that limits the approximation ratio of the final constructed sparsifier.

To address these two challenges, in the celebrated paper Kolla et al. [22] propose to keep track of the algorithm's progress with respect to two subspaces, each of which is measured by some potential function. Specifically, in each iteration j they define $A_j \triangleq X + \sum_i c_i v_i v_i^\intercal$,

where $\sum_i c_i v_i v_i^\intercal$ is the sum of currently picked rank-one matrices after reweighting during the first j iterations. For the upper barrier value u_j in iteration j, they define the upper potential function

$$\Phi^{u_j}(A_j) \triangleq \operatorname{tr}\left(P_{L(A_j)}\left(u_j I - A_j\right) P_{L(A_j)}\right)^\dagger,$$

where $L(A_j)$ is the T-dimensional subspace of A_j spanned by the T largest eigenvectors of A_j and $P_{L(A_j)}$ is the projection onto that subspace. Notice that $\Phi^{u_j}(A_j)$ is defined with respect to a variable space $L(A_j)$ that *changes after every rank-one update*, in order to upper bound the maximum eigenvalue of the final constructed matrix in the entire space. Similarly, for the same matrix A_j and lower barrier ℓ_j in iteration j, they define the lower potential function by

$$\Phi_{\ell_j}(B_j) \triangleq \operatorname{tr}\left(P_S(B_j - \ell_j I)P_S\right)^\dagger,$$

where P_S is the orthogonal projection onto S, the subspace generated by the bottom k eigenvectors of X, and the matrix B_j is defined by $B_j = Z(A_j - X)Z$, for $Z = \left(P_S(I - X)P_S\right)^{\dagger/2}$. Since the total number of chosen vectors is $K = O(k)$, instead of expecting the final constructed matrix A_τ approximating the identity matrix, the objective of the subgraph sparsification is to find coefficients $\{c_i\}$ with $K = O(k)$ non-zeros such that the following two conditions hold for some positive constants $\theta_{\min}, \theta_{\max}$:

- it holds that $X + \sum_{i=1}^m c_i v_i v_i^\intercal \preceq \theta_{\max} I$, and
- it holds that $\sum_{i=1}^m c_i Z v_i v_i^\intercal Z \succeq \theta_{\min} P_S$.

Informally, the first condition above states that the length of any axis of $\mathsf{Ellip}(A_j)$ is upper bounded, and the second condition ensures that the final matrix A_τ has significant contribution towards the bottom k eigenspace X. In other words, instead of ensuring $\ell_j \cdot I \prec A_j \prec u_j \cdot I$, $\Phi^{u_j}(A_j)$ and $\Phi_{\ell_j}(B_j)$ are used to "quantify" the shapes of the two ellipsoids with different dimensions:

- The function $\Phi^{u_j}(A_j)$ studies the ellipsoid A_j projected onto its own top eigenspaces, the subspace that *changes* after each iteration;
- The function $\Phi_{\ell_j}(B_j)$ studies $A_j - X$ projected onto the bottom k eigenspace of X, the subspace that remains *fixed* during the entire BSS process.

Proving the existence of some vector in each iteration so that the algorithm will make progress is much more involved, and constitutes one of the key lemmas used in [22] for constructing a subgraph sparsifier. However, the subgraph sparsification algorithm in [22] requires the computation of the projection matrices $P_{L(A_j)}$ in each iteration. Because of this, the algorithm presented in [22] runs in $\Omega\left(n^2 mk\right)$ time.

Our approach. At a very high level, our algorithm and its analysis can be viewed as a neat combination of the algorithm presented in [26] and the algorithm presented in [22]. Specifically, for any iteration j with the constructed matrix A_j, we set $B_j \triangleq Z(A_j - X)Z$, where $Z \triangleq \left(P_{\mathcal{V}}(I - X)P_{\mathcal{V}}\right)^{\dagger/2}$, and define the two potential functions by

$$\Phi^{u_j}(A_j) \triangleq \operatorname{tr}\left(P_{L(A_j)}\left(u_j I - A_j\right) P_{L(A_j)}\right)^{\dagger q},$$

and

$$\Phi_{\ell_j}(B_j) \triangleq \operatorname{tr}\left(P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}}\right)^{\dagger q}$$

for some fixed projection matrix $P_{\mathcal{V}}$, projecting on a k-dimensional subspace S'. Similar with [26], with the help of the q-th power in the definition of $\Phi^{u_j}(A_j)$ and $\Phi_{\ell_j}(B_j)$ we show that the eigenvalues of our constructed matrices A_j and B_j are never very close to the

two barrier values u_j and ℓ_j. Moreover, although the top T-eigenspace of the currently constructed matrix A_j changes after every rank-one update, multiple vectors can still be selected according to some probability distribution in each iteration.

However, when combining the randomised BSS framework [26] with the algorithm presented in [22], we have to take many challenging technical issues into account. In particular, we need to address the following issues: (1) Both the algorithm and its analysis in [22] crucially depend on the projection matrix P_S, of which the exact computation is expensive. Therefore, in order to obtain an efficient algorithm for subgraph sparsification, one needs to obtain some projection matrix close to P_S and such projection matrix can be computed efficiently. (2) As indicated by our definition of $\Phi_{\ell_j}(B_j)$ above, developing a fast subgraph sparsification algorithm would require efficient approximation of polynomials of the matrix $(P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}})^q$. In comparison with [26], the fixed projection matrix $P_{\mathcal{V}}$ sandwiched between two consecutive $(B_j - \ell_j I)$ makes computing the required quantities much more challenging.

To address these issues, we prove that there is a k-dimensional subspace S' close to S, and all of our required quantities that involve the projection onto S', denoted by $P_{\mathcal{V}}$, can be computed efficiently. Moreover, we prove that the quality of our constructed subgraph sparsifer based on the "approximate projection" $P_{\mathcal{V}}$ is the same as the one constructed by [22], in which the "optimal projection" P_S is needed. Our result regarding the approximate subspace S' is summarised as follows:

▶ **Lemma 10.** *There is an algorithm that computes a matrix $\mathbb{V} = L^{-1/2}V$ for matrix V in $t_{10} \triangleq \min\left\{O(n^\omega), \widetilde{O}\left(\frac{mk+nk^2}{\sqrt{\lambda_{k+1}(X)}}\right)\right\}$ time, such that with constant probability the following two properties hold: (1) $P_{\mathcal{V}} = VV^\intercal$ is a projection matrix on a k-dimensional subspace S' of $\mathbb{R}^n$; (2) For any $u \in \mathbb{R}^n$ satisfying $u^\intercal V = 0$, we have that*

$$\frac{u^\intercal X u}{u^\intercal u} \geq \frac{\lambda_{k+1}(X)}{2}.$$

We highlight that, in comparison to [26], in our setting the upper and lower potential functions *keep track of two different subspaces whose dimensions are of different orders in most regimes*, i.e., k versus T, and this makes our analysis much more involved than [26]. On the other side, we also show that the algorithm in [26] can be viewed as a special case of our algorithm, and from this aspect our algorithm presents a general framework for constructing spectral sparsifiers and subgraph sparsifiers.

3.2 Description of our algorithm

Our algorithm proceeds in iterations in which multiple vectors are sampled with different probabilities. In each iteration j, A_j is updated by setting $A_{j+1} = A_j + \Delta_j$, where Δ_j is the sum of the sampled rank-one matrices with reweighting. To compensate for this change, the two barriers u_j and ℓ_j are increased by $\delta_{u,j}$ and $\delta_{\ell,j}$. The algorithm terminates when the difference of the barriers is greater than α, defined by $\alpha \triangleq 4k/\Lambda$. Specifically, in the initialisation step, the algorithm sets $A_0 \triangleq X$,$u_0 \triangleq 2 + \lambda_{\max}(X)$, $\ell_0 \triangleq -2k/\Lambda$, where $\Lambda \triangleq \max\{k, T\}$. In each iteration j, the algorithm keeps track of the currently constructed matrix A_j and hence, also of the matrix $B_j \triangleq Z(A_j - X)Z$, where $Z \triangleq (P_{\mathcal{V}}(I - X)P_{\mathcal{V}})^{\dagger/2}$ for some fixed projection matrix $P_{\mathcal{V}}$. Intuitively, the projection matrix $P_{\mathcal{V}}$ used here is close to P_S, but can be approximated more efficiently than computing P_S precisely. In each iteration j, the algorithm starts by computing the *relative effective resistances*, which are defined as

$$R_i(A_j, B_j, u_j, \ell_j) \triangleq v_i^\intercal \left(u_j I - A_j\right)^{-1} v_i + v_i^\intercal Z \left(P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}}\right)^\dagger Z v_i,$$

for all vectors v_i. Then, the algorithm computes the number of vectors N_j that will be sampled, which can be written as

$$N_j \triangleq \left(\frac{\varepsilon}{4\rho_j} \cdot \lambda_{\min}\left[(u_j I - A_j)^{-1}\overline{M}\right] \cdot \frac{\lambda_{\max}\left((u_j I - A_j)^{-1}\overline{M}\right)}{\operatorname{tr}\left[(u_j I - A_j)^{-1}\overline{M}\right]}\right)^{2\varepsilon/q} \cdot \rho_j \cdot \min\left\{\frac{1}{\lambda_{\max}\left((u_j I - A_j)^{-1}\overline{M}\right)}, \frac{1}{\lambda_{\max}\left(P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}}\right)^\dagger}\right\},$$

where

$$\rho_j \triangleq \sum_{t=1}^{m} R_t(A_j, B_j, u_j, \ell_j) = \operatorname{tr}\left[(u_j I - A_j)^{-1}\overline{M}\right] + \operatorname{tr}\left[P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}}\right]^\dagger.$$

Next, the algorithm samples N_j vectors such that every v_i is sampled with probability proportional to $R_i(A_j, B_j, u_j, \ell_j)$, i.e., the sampling probability of every v_i is defined by

$$p(v_i) \triangleq \frac{R_i(A_j, B_j, u_j, \ell_j)}{\sum_{t=1}^{m} R_t(A_j, B_j, u_j, \ell_j)}.$$

For every sampled v_i, the algorithm scales it to

$$w_i \triangleq \sqrt{\frac{\varepsilon}{q \cdot R_i(A, B, u, \ell)}} \cdot v_i,$$

and gradually adds $w_i w_i^\intercal$ to A_j. After each rank-one update, the algorithm increases the barrier values by the average increases

$$\overline{\delta}_{u,j} \triangleq \frac{(1+3\varepsilon)\cdot\varepsilon}{q\cdot\rho_j} \qquad \text{and} \qquad \overline{\delta}_{\ell,j} \triangleq \frac{(1-3\varepsilon)\cdot\varepsilon}{q\cdot\rho_j},$$

and checks whether the terminating condition of the algorithm is satisfied. Note that between two consecutive iterations j and $j+1$ of the algorithm, the two barriers u_j and ℓ_j are increased by $\delta_{u,j} \triangleq N_j \cdot \overline{\delta}_{u,j}$ and $\delta_{\ell,j} \triangleq N_j \cdot \overline{\delta}_{\ell,j}$, respectively.

The formal description of our algorithm is presented in Algorithm 2. We remark that, in contrast to the algorithm for constructing a spectral sparsifier [26], the total number of vectors needed in the final iteration j could be much smaller than $O(N_j)$. This is why our algorithm performs a sanity check in Line 15 after every rank-1 update $w_i w_i^\intercal$.

3.3 Proof sketch of Theorem 9

In this subsection we give an overview of the main techniques used for proving Theorem 9. We refer the reader to the full version of our paper for a more detailed discussion.

Approximation Guarantee

Firstly, we will focus on showing that, at the end of Algorithm 2, (4) holds. The result is summarised in the following lemma, whose proof will be left for the end of this subsection.

Algorithm 2 Algorithm for constructing subgraph spectral sparsifiers.

Require: $\varepsilon \leq 1/20, q \geq 10$

$u_0 = 2 + \lambda_{\max}(X)$, $\ell_0 = -2k/\Lambda$ ▷ Here u_0 and ℓ_0 are the initial barrier values

$\widehat{u} = u_0$ and $\widehat{\ell} = \ell_0$ ▷ Here $\widehat{u}$ and $\widehat{\ell}$ are the current barrier values

$j = 0$ ▷ j will be the index of the current iteration

$A_0 = X$, $B_0 = \mathbf{0}$

while $\widehat{u} - \widehat{\ell} > \alpha + u_0 - \ell_0$ **do** ▷ Start of iteration j

  Compute $R_t(A_j, B_j, \ell_j, u_j)$ and hence $p(v_t)$ for all vectors v_t

  Compute N_j

  Sample N_j vectors $v_1, \dots v_{N_j}$ according to p

  Set $W_j \leftarrow 0$

  for every subphase $i = 1 \dots N_j$ **do** ▷ Start of subphase i

    $w_i \leftarrow \sqrt{\frac{\varepsilon}{q \cdot R_i(A_j, B_j, u_j, \ell_j)}} \cdot v_i$

    $W_j \leftarrow W_j + w_i w_i^\intercal$

    $\widehat{u} \leftarrow \widehat{u} + \overline{\delta}_{u,j}$

    $\widehat{\ell} \leftarrow \widehat{\ell} + \overline{\delta}_{\ell,j}$

    if $\widehat{u} - \widehat{\ell} > \alpha + u_0 - \ell_0$ **then**

      Stop at the current subphase ▷ End of subphase i

  $A_{j+1} \leftarrow A_j + W_j$

  $B_{j+1} \leftarrow Z(A_{j+1} - X)Z$

  $j = j + 1$ ▷ End of iteration j

Return $M = A_j$

▶ **Lemma 11.** *The condition number of the returned matrix A_τ after τ iterations is at most $1 + O(\varepsilon) \cdot \max\{1, T/k\}$. Moreover, it holds that*

$$\lambda_{\min}(A_\tau) \geq c \cdot (1 - O(\varepsilon)) \cdot \lambda_{k+1}(X) \min\{1, k/T\},$$

for some constant c.

The above result is based on the following technical lemma:

▶ **Lemma 12.** *For all iterations j, the following invariant is preserved by Algorithm 2*

$$A_j \prec (1 - \eta) u_j \cdot I \qquad \text{and} \qquad P_{\mathcal{V}} B_j P_{\mathcal{V}} \succeq (\ell_j + |\ell_j|\eta) P_{\mathcal{V}},$$

for some parameter $\eta = O\left(\frac{\varepsilon^{2+2/q}}{n^{2/q}}\right)$.

In order to prove Lemma 12, we need to develop a sequence of results. For the moment, we fix an iteration j. Recall that in this iteration the algorithm samples N_j vectors independently from $\{v_i\}_{i=1}^m$ such that each sampled vector v_i is further scaled to w_i. Moreover, the algorithm keeps track of the matrix

$$W_j \triangleq \sum_{i=1}^{N_j} w_i w_i^\intercal.$$

Our choice of N_j ensures that, with high probability, the matrix W_j has bounded eigenvalues with respect to the matrix A_j.

▶ **Lemma 13.** *Assume that the number of samples satisfies*

$$N_j \leq \left(\frac{\varepsilon}{4\rho_j} \cdot \lambda_{\min}\left[(u_j I - A_j)^{-1}\overline{M}\right] \cdot \frac{\lambda_{\max}\left((u_j I - A_j)^{-1}\overline{M}\right)}{\operatorname{tr}\left[(u_j I - A_j)^{-1}\overline{M}\right]} \right)^{2\varepsilon/q} \cdot \rho_j \cdot \frac{1}{\lambda_{\max}\left((u_j I - A_j)^{-1}\overline{M}\right)}.$$

Then it holds that

$$\mathbf{P}\left[\mathbf{0} \preceq W_j \preceq \frac{1}{2}(u_j I - A_j)\right] \geq 1 - \frac{\varepsilon}{2n}.$$

Notice that if $W_j \preceq \frac{1}{2}(u_j I - A_j)$, W_j's contribution towards the direction of A_j's eigenvector associating with its largest eigenvalue is upper bounded. Thus, conditioned on this event, we can control better the eigenvalues of the resulting matrix $A_{j+1} = A_j + W_j$. Formally, we show the following result:

▶ **Lemma 14.** *It holds that*

$$\mathbf{E}\left[\Phi^{u_{j+1}}(A_{j+1}) \middle| 0 \preceq W_j \preceq \frac{1}{2}(u_j I - A_j)\right] \leq \Phi^{u_j}(A_j) \quad \textit{and}$$
$$\mathbf{E}\left[\Phi_{\ell_{j+1}}(B_{j+1}) \middle| 0 \preceq W_j \preceq \frac{1}{2}(u_j I - A_j)\right] \leq \Phi_{\ell_j}(B_j).$$

Finally, we show that the careful choice of N_j ensures that a sufficiently large number of vectors are sampled in each iteration. This implies that the total number of iterations executed by the algorithm cannot be too large. The result is summarised below:

▶ **Lemma 15.** *With probability at least* $4/5$*, Algorithm 2 finishes in at most*

$$\tau \leq \frac{80q}{3\varepsilon^2} \cdot \frac{1}{c_N} \cdot \Lambda^{(1+2\varepsilon)/q}$$

iterations, where $c_N = \Omega\left((1/(\operatorname{poly}(n))^{2\varepsilon/q}\right)$.

We are now ready to prove the main technical lemma.

Proof of Lemma 12. By Lemma 13, Lemma 15 and the union bound, with probability at least $3/4$, all matrices picked in

$$\tau \leq \frac{80q}{3\varepsilon^2} \cdot \frac{1}{c_N} \cdot \Lambda^{(1+2\varepsilon)/q} \leq \frac{80q}{3\varepsilon^2} \cdot n^{c/q}$$

iterations, for some small constant $c < q$, satisfy $W_j \preceq \frac{1}{2}(u_j I - A_j)$ for all iterations j. Therefore, by Lemma 14 and conditioning on the event that $\forall i : W_i \preceq 1/2 \cdot (u_i I - A_i)$ we have that

$$\mathbf{E}\left[\Phi^{u_j}(A_j) \middle| \forall i : W_i \preceq (1/2) \cdot (u_i I - A_i)\right] \leq \Phi^{u_0}(A_0) \leq \frac{T}{2^q}$$

and

$$\mathbf{E}\left[\Phi_{\ell_j}(B_j) \middle| \forall i : W_i \preceq (1/2) \cdot (u_i I - A_i)\right] \leq \Phi_{\ell_0}(B_0) \leq k \cdot \left(\frac{\Lambda}{2k}\right)^q.$$

By Markov's inequality, it holds with high probability that

$$(\Phi^{u_j}(A_j))^{1/q} = O\left(T^{1/q} \cdot \tau^{1/q}\right) \quad \text{and} \quad \left(\Phi_{\ell_j}(B_j)\right)^{1/q} = O\left(k^{1/q} \cdot \frac{\Lambda}{k} \cdot \tau^{1/q}\right).$$

For any eigenvalue of A_j, say λ_i, we have

$$(u_j - \lambda_i)^{-q} \leq (u_j - \lambda_{\max}(A_j))^{-q} < \sum_{t=n-T+1}^{n} (u_j - \lambda_t(A_j))^{-q} = \Phi^{u_j}(A_j).$$

Therefore, it holds that

$$\lambda_i < u_j - (\Phi^{u_j}(A_j))^{-1/q} \leq u_j - O\left(\frac{1}{T^{1/q}} \cdot \frac{1}{\tau^{1/q}}\right) \leq u_j - O\left(\frac{2}{T^{1/q}} \cdot \left(\frac{\varepsilon^2}{qn^{c/q}}\right)^{1/q}\right).$$

Since u_j is $O(1/\varepsilon^2)$ and $T \leq n$, we can choose $\eta = O\left(\frac{\varepsilon^{2+2/q}}{n^{2/q}}\right)$ such that $A_j \prec (1-\eta)u_j I$.

The second statement can be shown in a similar way, i.e., we show that for any nonzero eigenvalue λ_i of B_j, it holds that $\lambda_i \geq \ell_j + \left(\Phi_{\ell_j}(B_j)\right)^{-1/q}$. Hence

$$\lambda_i \geq \ell_j + \Omega\left(\frac{1}{k^{1/q}} \cdot \frac{k}{\Lambda} \cdot \left(\frac{\varepsilon^2}{qn^{c/q}}\right)^{1/q}\right).$$

Since $|\ell_j| = O\left(\frac{k}{\Lambda} \cdot 1/\varepsilon\right)$ and $k \leq n$, we can choose $\eta = O\left(\frac{\varepsilon^{2+2/q}}{n^{2/q}}\right)$ such that $P_{\mathcal{V}} B_j P_{\mathcal{V}} \succeq (\ell_j + |\ell_j|\eta) P_{\mathcal{V}}$. ◀

Proof sketch of Lemma 11. Notice that it holds for any iteration j that

$$\frac{\overline{\delta}_{u,j} - \overline{\delta}_{\ell,j}}{\overline{\delta}_{u,j}} = \frac{6\varepsilon}{1+3\varepsilon},$$

which implies that

$$\overline{\delta}_{u,j} = \frac{1+3\varepsilon}{6\varepsilon}\left(\overline{\delta}_{u,j} - \overline{\delta}_{\ell,j}\right) \geq \frac{1}{6\varepsilon}\left(\overline{\delta}_{u,j} - \overline{\delta}_{\ell,j}\right).$$

Let u_τ and ℓ_τ be the barrier values when the algorithm terminates, and our goal is to show that

$$\frac{u_\tau}{\ell_\tau} = \left(1 - \frac{u_\tau - \ell_\tau}{u_\tau}\right)^{-1} = 1 + O(\varepsilon) \cdot \max\{1, T/k\},$$

which suffices to prove that

$$\frac{u_\tau - \ell_\tau}{u_\tau} = O(\varepsilon) \cdot \max\{1, T/k\}.$$

By definition, we know that

$$\frac{u_\tau - \ell_\tau}{u_\tau} \leq \frac{u_0 - \ell_0 + \alpha}{u_0 + (6\varepsilon)^{-1}\alpha} \leq \frac{3 + 6k/\Lambda}{2 + (6\varepsilon)^{-1}4k/\Lambda} \leq O(\varepsilon) \cdot \max\{1, T/k\},$$

where the last inequality holds by the definition of Λ. Similar with [22], in the full version of the paper we prove that

$$\lambda_{\min}(A_K) \geq \frac{\theta_{\min}\lambda_{k+1}(X)/2}{\left((\lambda_{k+1}(X)/2)^{1/2} + \theta_{\min}^{1/2} + \theta_{\max}^{1/2}\right)^2}.$$

assuming that $\lambda_{\max}(A_\tau) \leq \theta_{\max}$ and $\lambda_{\min}\left(B_K\big|_{S'}\right) \geq \theta_{\min}$. By setting $\theta_{\min} = \ell_\tau$ and $\theta_{\max} = u_\tau$, the inequality above implies the second statement of the lemma. ◀

The results for the total number of edges (vectors) sampled as well as the assigned costs are summarised below. Due to space constraints, we defer the proofs to the full version of the paper.

▶ **Lemma 16.** *With probability at least* $3/4$*, Algorithm 2 terminates after choosing at most*

$$K = \frac{20 \cdot q\ k}{3 \cdot \varepsilon^2}$$

vectors.

▶ **Lemma 17.** *It holds with constant probability that* $\sum_{i=1}^{m} c_i \cdot cost_i = O(1/\varepsilon^2) \cdot \min\{1, k/T\}$.

Runtime analysis

Now we discuss a fast approximation of the quantities needed for our subgraph sparsification algorithm. We fix an arbitrary iteration j, and drop this subscript for simplicity. A careful inspection tells us that the efficiency of our algorithm is based on the fast approximation of the following quantities:

1. $v_i^\intercal Z \left(P_{\mathcal{V}} \left(B - \ell I\right) P_{\mathcal{V}}\right)^\dagger Z v_i$
2. $\lambda_{\max} \left(P_{\mathcal{V}}(B - \ell I)P_{\mathcal{V}}\right)^\dagger$
3. $\lambda_{\min} \left[(uI - A)^{-1}\overline{M}\right]$
4. $\lambda_{\max} \left[(uI - A)^{-1}\overline{M}\right]$
5. $\operatorname{tr} \left[(uI - A)^{-1}\overline{M}\right]$
6. $v_i^\intercal \left(uI - A\right)^{-1} v_i$

These are precisely the nontrivial quantities required to compute the values N and $R_i(A, B, u, \ell)$ for all vectors v_i. As previously mentioned, for the first two items we use the approximate projection $P_{\mathcal{V}}$ instead of the actual projection P_S on the bottom k eigenspace of X. This is done in order to overcome the expensive exact computation of P_S. We also remark that, while $P_{\mathcal{V}} = VV^\intercal$ for some unitary matrix V is used in our previous analysis, we do not need to compute the matrices V or $P_{\mathcal{V}}$ explicitly. Instead, it suffices to compute the matrix $\mathbb{V} \triangleq L_{G+W}^{-1/2} V$ in order to approximate our required quantities (1), (2). The fast computation of $\mathbb{V}$ builds upon the work of [2] and our result is summarised in Lemma 10.

Once we have access to the matrix $\mathbb{V}$, we can efficiently approximate the above quantities (1)–(6). The techniques we used are inspired from the previous work [4, 26]. However, the presence of the matrix $\overline{M}$ as well as the projection $P_{\mathcal{V}}$ make the computations nontrivial and require extra work. We summarise our results bellow and refer the reader to the full version of the paper for the details of each individual approximation.

▶ **Lemma 18.** *Let j be an arbitrary iteration of Algorithm 2. We can approximately compute the quantities (1)–(6), for all vectors v_i in time*

$$t_{\text{iteration}} = \widetilde{O}\left(\left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^\omega, mk + nk^2 + k^\omega\right\}\right)/\varepsilon^3\right).$$

The running time of Algorithm 2 is analysed in the next lemma.

▶ **Lemma 19.** *Assuming Algorithm 2 finishes in $\tau = O\left(q \cdot n^{O(1/q)}/\varepsilon^2\right)$ iterations, then the total running time is*

$$t_{\text{alg}} = \widetilde{O}\left(\min\left\{n^\omega, \frac{mk + nk^2}{\sqrt{\lambda_{k+1}(X)}}\right\} + q \cdot n^{O(1/q)}\left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^\omega, mk + nk^2 + k^\omega\right\}\right)/\varepsilon^5\right).$$

Proof. By Lemma 10, we can compute the matrix $\mathbb{V}$ in time $t_{10} = \min\left\{O(n^\omega), \widetilde{O}\left(\frac{mk+nk^2}{\sqrt{\lambda_{k+1}(X)}}\right)\right\}$. This is computed only once and will be used throughout every iteration.

By Lemma 18, the running time in each iteration is

$$t_{\text{iteration}} = \widetilde{O}\left(\left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^\omega, mk + nk^2 + k^\omega\right\}\right)/\varepsilon^3\right).$$

Thus, the algorithm's overall running time is

$$\begin{aligned} t_{\text{alg}} &\triangleq t_{10} + \tau \cdot t_{\text{iteration}} \\ &= \widetilde{O}\left(\min\left\{n^\omega, \frac{mk+nk^2}{\sqrt{\lambda_{k+1}(X)}}\right\} + q \cdot n^{O(1/q)}\left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^\omega, mk + nk^2 + k^\omega\right\}\right)/\varepsilon^5\right). \end{aligned}$$

◀

4 Proof of the main theorem

Finally we apply our fast SDP solver and the subgraph sparsification algorithm to design an algorithm for the spectral-augmentability problem, and prove Theorem 1. We first give an overview of the main algorithm: for any input $G = (V, E)$, the set E_W of candidate edges, and parameter k, our algorithm applies the doubling technique to enumerate all the possible γ under which the input instance is (k, γ)-spectrally augmentable: starting with the initial γ, which is set to be $1/n^{1/q}$ and increases by a factor of 2 each time, the algorithm runs the SDP solver, a subgraph sparsification algorithm, and a Laplacian solver to verify the algebraic connectivity of the output of our subgraph sparsification algorithm. The algorithm terminates if the algebraic connectivity is greater than some threshold at some iteration, or it is below the initial threshold. See Algorithm 3 for formal description.

The following lemma will be used in our analysis.

▶ **Lemma 20.** *Let $\gamma > 0$. If $G = (V, E)$ is $(k, \gamma\Delta)$-spectrally-augmentable with respect to $W = (V, E_W)$, then the SDP solver finds a feasible solution $(\widehat{\lambda}, w)$ to P-SDP$(G, W, k, (1-\delta')\gamma)$, and the subgraph sparsification algorithm with input $G, E_W, k, \varepsilon, q$ and weights $\{w_e : e \in E \cup E_W\}$ will find a graph $H = (V, E \cup F)$ with $F \subseteq E_W$, $\lambda_2(L_H) \geq c_1\gamma^2 \cdot \Delta$, $|F| \leq O(qk/\varepsilon^2)$ and total new weights of edges in F at most $O(k/\varepsilon^2)$.*

Proof. If G is $(k, \gamma\Delta)$-spectrally-augmentable with respect to W, then there exists a feasible solution to P-SDP(G, W, k, γ) and our SDP solver will find a solution $(\widehat{\lambda}, w)$ to P-SDP$(G, W, k, (1-\delta')\gamma)$, for any constant $\delta' > 0$. Note that $\widehat{\lambda} \geq (1-\delta')\gamma$. Now we use the subgraph sparsification algorithm to sparsify the SDP solution.

We apply Theorem 9 to graphs G, W, by setting $V = \text{im}(L_{G+W}) = \ker(L_{G+W})^\perp$, $X = \left(L_{G+W}^{\dagger/2} L_G L_{G+W}^{\dagger/2}\right)_{|V}$ and $Y_e = w_e\left(L_{G+W}^{\dagger/2} L_e L_{G+W}^{\dagger/2}\right)_{|V}$, and $\overline{M} = \sum_{e\in E_W} Y_e$, $K = O(qk/\varepsilon^2)$, $\lambda^* = \lambda_{k+1}(X)$ and $cost_e = \frac{w_e}{\sum_{f\in E_W} w_f}$. Note that $T = \lceil \text{tr}(\overline{M}) \rceil \leq k$. This is true since $\sum_{e\in E_W} w_e \leq k$, $L_{G+W}^{\dagger/2} L_e L_{G+W}^{\dagger/2} \preceq I$, $L_{G+W}^{\dagger/2} L_e L_{G+W}^{\dagger/2}$ is a rank one matrix and has trace at most 1. We get a set of coefficients $\{c_e\}$ supported on at most K edges, such that

$$\begin{aligned} C(1 - O(\varepsilon)) \cdot \min\{1, K/T\} \cdot \lambda_{k+1}(X) &\leq \lambda_{\min}\left(X + \sum_{e\in E_W} c_e Y_e\right) \\ &\leq \lambda_{\max}\left(X + \sum_{e\in E_W} c_e Y_e\right) \leq 1 + O(\varepsilon). \end{aligned}$$

Algorithm 3 Algorithm for augmenting the algebraic connectivity.

Require: the base graph $G = (V, E)$, and the set E_W of m edges defined on V, and $k \in \mathbb{Z}^+$.

$\gamma_0 \leftarrow 1/n^{\frac{1}{q}}$;
$\gamma \leftarrow \gamma_0$;
$\alpha \leftarrow 0$;
$F \leftarrow \emptyset$; ▷ the set of edges added to G
while $\gamma < 1$ **do**
  $\gamma \leftarrow 2 \cdot \gamma$, and run the SDP solver from Theorem 2 for P-SDP(G, W, k, γ)
  if the solver certifies that P-SDP(G, W, k, γ) is infeasible **then**
    if $\alpha = 0$ **then**
      Abort and output **Reject**.
    else
      return graph $H = (V, E(G) \cup F)$. ▷ $\lambda_2(L_H) \geq c_1\alpha^2\Delta$
  else the solver finds a feasible solution for P-SDP$(G, W, k, 0.9\gamma)$ with weights $\{w_e\}_{e\in E_W}$
    $\alpha \leftarrow \gamma$
    Let $H = (V, E(G) \cup F)$ be the output of our subgraph sparsification algorithm with edge weights $\{w_e\}_{e\in E_W}$, q, k and a sufficiently small constant ε.
    $\eta_2 \leftarrow$ a 1.1-approximation of $\lambda_2(L_H)$ ▷ apply the Laplacian solver to compute η_2
    if $\eta_2 \leq O\left(\Delta \cdot n^{-2/q}\right)$ **then**
      Abort and output **Reject**.

From the above and the fact that $T \leq k$, $K = O(qk/\varepsilon^2)$, we have that

$$\begin{aligned}\lambda_2\left(L_G + \sum_e c_e w_e L_e\right) &\geq C(1 - O(\varepsilon)) \cdot \min\{1, K/T\} \cdot \lambda_{k+1}(X) \cdot \lambda_2\left(L_G + \sum_e w_e L_e\right)\\ &\geq C' \cdot \frac{\lambda_{k+2}(L_G)}{4\Delta} \cdot \widehat{\lambda} \cdot \Delta\\ &= \frac{C'}{4} \cdot \widehat{\lambda} \cdot \lambda_{k+2}(L_G)\end{aligned}$$

for some constant $C' > 0$, where the last inequality follows from the fact that

$$\lambda_i(X) = \lambda_i\left(\left(L_{G+W}^{\dagger/2} L_G L_{G+W}^{\dagger/2}\right)\Big|_V\right) \geq \frac{\lambda_{i+1}(L_G)}{4D},$$

for any $i \geq 1$.

▷ Claim 21. It holds that $\lambda_{k+2}(L_G) \geq \lambda_{\mathrm{OPT}}$, the maximum algebraic connectivity of adding a subset set of k edges from E_W to G.

Proof. Let L_R be the Laplacian matrix of the graph which is formed by the optimum solution R. Then $\dim\ker(L_R) \geq n-k$ as $\mathrm{rank}(L_R) \leq |E| \leq k$. Consider the space S spanned by all the eigenvectors of L_G corresponding to $\lambda_2(L_G), \ldots, \lambda_{k+2}(L_G)$. Since $\dim(S) + \dim\ker(L_R) > n$, there exists a unit vector $v \in \ker(L_R) \cap \dim(S)$ such that $v \perp \mathbf{1}$, and

$$v(L_G + L_R)v^{\intercal} \leq \lambda_{k+2}(L_G) + 0 = \lambda_{k+2}(L_G).$$

This further implies that $\lambda_{\mathrm{OPT}} = \lambda_2(L_G + L_R) \leq \lambda_{k+2}(L_G)$. ◁

Therefore, if we let $F = \{e : e \in E_W, c_e > 0\}$ and set the edge weights to be $\{c_e \cdot w_e : e \in F\}$, then the resulting graph $H = (V, E + F)$ with the corresponding weights satisfies that

$$\lambda_2(L_H) = \lambda_2\left(L_G + \sum_e c_e w_e L_e\right) \geq c \cdot \gamma \cdot \lambda_{\mathrm{OPT}} \geq c \cdot \gamma^2 \Delta$$

for some constant $c > 0$, where the last inequality follows from the assumption G is $(k, \gamma\Delta)$-spectrally-augmentable with respect to W and thus $\lambda_{\mathrm{OPT}} \geq \gamma\Delta$. Since

$$\sum_{e \in E_W} cost_e \cdot c_e \leq O(1/\varepsilon^2) \min\{1, K/T\} = O(1/\varepsilon^2),$$

the total weights of added edges become

$$\sum_{e \in E_W} c_e w_e = \left(\sum_{e \in E_W} w_e\right) \cdot \left(\sum_{e \in E_W} cost_e \cdot c_e\right) O(1/\varepsilon^2) \cdot k = O(k/\varepsilon^2). \qquad \blacktriangleleft$$

Finally, we are ready to prove the main theorem of the paper.

Proof of Theorem 1. Let G and W be the input to Algorithm 3. Note that the algorithm only returns a subgraph H with $\lambda_2(L_H) \geq c_1\gamma_0^2\Delta$, and H contains at most $K = O(kq)$ edges from E_W. Hence, if G is not $(O(kq), c_1\gamma_0^2\Delta)$-spectrally-augmentable with respect to W, then the algorithm will reject the input instance.

Without loss of generality, in the following analysis we assume that G is $(k, \lambda_\star\Delta)$-augmentable for some $\lambda_\star > \gamma_0$, where $\lambda_\star\Delta$ is the optimum solution. In this case, by the geometric search over γ in the algorithm, when $\gamma \in (\frac{\lambda_\star}{2}, \lambda_\star)$, the SDP solver will find a feasible solution for P-SDP$(G, W, k, 0.9\gamma)$ and the graph H returned by the subgraph sparsification algorithm with input G, W, q, k and constant ε satisfies that $\lambda_2(L_H) \geq c_1\gamma^2\Delta \geq c_1'\lambda_\star^2\Delta$. If $\gamma \geq \lambda_\star$, then the algorithm will either return the graph H that we constructed corresponding to the value $\gamma \in (\frac{\lambda_\star}{2}, \lambda_\star)$, or finds a graph H with $\lambda_2(L_H) \geq c_1\gamma^2\Delta \geq c_1'\lambda_\star^2\Delta$. By Lemma 20, the number of added edges and the total sum of their weights are $O(qk)$ and $O(k)$, respectively.

Furthermore, since $\lambda_\star \geq \gamma_0$, it only takes $O(\log n)$ iterations to reach γ with $\gamma \in (\frac{\lambda_\star}{2}, \lambda_\star)$. In each iteration, by Theorem 2, the running time for solving P-SDP$(G, W, k, 0.9\gamma)$ for $\gamma \geq \gamma_0$ is $\widetilde{O}(m+n)/\gamma^2) = \widetilde{O}((m+n)n^{O(1/q)})$; by Theorem 9, the time for applying subgraph sparsification with input G, W and constant ε is $\widetilde{O}(\min\left\{qn^{\omega+O(1/q)}, q(m+n)n^{O(1/q)}k^2\right\})$. For the latter, we note that whenever we apply the subgraph sparsification from Theorem 9, the corresponding matrix X satisfies that

$$\lambda_{k+1}(X) \geq \frac{\lambda_{k+2}(L_G)}{4\Delta} \geq \frac{\lambda_{\mathrm{OPT}}}{4\Delta} = \frac{\lambda_\star\Delta}{4\Delta} \geq \frac{\gamma_0\Delta}{4\Delta} = \Omega\left(n^{-1/q}\right)$$

and thus we obtain the claimed runtime. Furthermore, we can compute an estimate η_2 of $\lambda_2(L_H)$ by the algorithm given in [41], which takes $\widetilde{O}(|E(H)|+n) = \widetilde{O}(n+k)$ time. Thus, the total running time is $\widetilde{O}\left(\min\left\{qn^{\omega+O(1/q)}, q(m+n)n^{O(1/q)}k^2\right\}\right)$. This completes the proof of the theorem. $\blacktriangleleft$

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *57th Annual IEEE Symposium on Foundations of Computer Science (FOCS'16)*, pages 335–344, 2016.

2 Zeyuan Allen-Zhu and Yuanzhi Li. Doubly accelerated methods for faster CCA and generalized eigendecomposition. In *34th International Conference on Machine Learning (ICML'17)*, pages 98–106, 2017.

3 Zeyuan Allen-Zhu, Yuanzhi Li, Aarti Singh, and Yining Wang. Near-optimal design of experiments via regret minimization. In *34th International Conference on Machine Learning (ICML'17)*, pages 126–135, 2017.

4 Zeyuan Allen-Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond multiplicative updates. In *47th Annual ACM Symposium on Theory of Computing (STOC'15)*, pages 237–245, 2015.

5 Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. *Journal of the ACM*, 63(2):12, 2016.

6 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

7 Stephen Boyd, Persi Diaconis, and Lin Xiao. Fastest mixing Markov chain on a graph. *SIAM review*, 46(4):667–689, 2004.

8 Tanmoy Chakraborty, Julia Chuzhoy, and Sanjeev Khanna. Network design for vertex connectivity. In *40th Annual ACM Symposium on Theory of Computing (STOC'08)*, pages 167–176, 2008.

9 Joseph Cheriyan and László A Végh. Approximating minimum-cost k-node connected subgraphs via independence-free graphs. *SIAM Journal on Computing*, 43(4):1342–1362, 2014.

10 Julia Chuzhoy and Sanjeev Khanna. An $O(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. In *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS'09)*, pages 437–441, 2009.

11 Michael B. Cohen, Jelani Nelson, and David P. Woodruff. Optimal approximate matrix product in terms of stable rank. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP'16)*, pages 1–14, 2016.

12 Michael Dinitz and Zeyu Zhang. Approximating low-stretch spanners. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*, pages 821–840, 2016.

13 Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In *31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, pages 750–759, 1999.

14 Jittat Fakcharoenphol and Bundit Laekhanukit. An $O(\log^2 k)$-approximation algorithm for the k-vertex connected spanning subgraph problem. *SIAM Journal on Computing*, 41(5):1095–1109, 2012.

15 Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

16 Arpita Ghosh and Stephen Boyd. Growing well-connected graphs. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 6605–6611, 2006.

17 Arpita Ghosh, Stephen Boyd, and Amin Saberi. Minimizing effective resistance of a graph. *SIAM review*, 50(1):37–66, 2008.

18 Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *26th Annual ACM Symposium on Theory of Computing (STOC'94)*, pages 356–364, 1994.

19 Rahul Jain, Zhengfeng Ji, Sarvagya Upadhyay, and John Watrous. QIP= PSPACE. *Journal of the ACM (JACM)*, 58(6):30, 2011.

20 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477, 2017.

21 Jonathan A Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. *Theory of Computing Systems*, 53(2):243–262, 2013.

22 Alexandra Kolla, Yury Makarychev, Amin Saberi, and Shang-Hua Teng. Subgraph sparsification and nearly optimal ultrasparsifiers. In *42nd Annual ACM Symposium on Theory of Computing (STOC'10)*, pages 57–66, 2010.

23 Guy Kortsarz, Robert Krauthgamer, and James R Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM Journal on Computing*, 33(3):704–720, 2004.

24 Bundit Laekhanukit. Parameters of two-prover-one-round game and the hardness of connectivity problems. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 1626–1643, 2014.

25 Yin Tat Lee and He Sun. An SDP-based algorithm for linear-sized spectral sparsification. In *49th Annual ACM Symposium on Theory of Computing (STOC'17)*, pages 678–687, 2017.

26 Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. *SIAM Journal on Computing*, 47(6):2315–2336, 2018.

27 Marina Meila and Jianbo Shi. Learning segmentation by random walks. In *Advances in Neural Information Processing Systems*, pages 873–879, 2001.

28 William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. *Proceedings of the 17th ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

29 Milena Mihail. Conductance and convergence of Markov chains-a combinatorial treatment of expanders. In *30th Annual IEEE Symposium on Foundations of Computer Science (FOCS'89)*, pages 526–531, 1989.

30 Damon Mosk-Aoyama. Maximum algebraic connectivity augmentation is NP-hard. *Operations Research Letters*, 36(6):677–679, 2008.

31 Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

32 Aleksandar Nikolov, Mohit Singh, and Uthaipon Tao Tantipongpipat. Proportional volume sampling and approximation algorithms for a-optimal design. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, pages 1369–1386, 2019.

33 Lorenzo Orecchia and Nisheeth K Vishnoi. Towards an sdp-based approach to spectral methods: A nearly-linear-time algorithm for graph partitioning and decomposition. In *22th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'11)*, pages 532–545, 2011.

34 Shayan Oveis Gharan and Luca Trevisan. Partitioning into expanders. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 1256–1266, 2014. doi:10.1137/1.9781611973402.93.

35 Richard Peng. Algorithm Design Using Spectral Graph Theory. *PhD Thesis*, 2013. doi:10.1184/R1/6714635.v1.

36 Richard Peng, He Sun, and Luca Zanetti. Partitioning well-clustered graphs: Spectral clustering works! *SIAM Journal on Computing*, 46(2):710–743, 2017.

37 Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

38 Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Trans. Information Theory*, 42(6):1710–1722, 1996.

39 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

40 Leslie G. Valiant. Graph-theoretic properties in computational complexity. *J. Comput. Syst. Sci.*, 13(3):278–285, 1976.

41 Nisheeth K. Vishnoi. $Lx = b$. *Foundations and Trends in Theoretical Computer Science*, 8(1-2):1–141, 2013.

42 Gilles Zémor. On expander codes. *IEEE Trans. Information Theory*, 47(2):835–837, 2001.

Chordless Cycle Packing Is Fixed-Parameter Tractable

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
marx@cispa.saarland

Abstract

A *chordless cycle* or *hole* in a graph G is an induced cycle of length at least 4. In the Hole Packing problem, a graph G and an integer k is given, and the task is to find (if exists) a set of k pairwise vertex-disjoint chordless cycles. Our main result is showing that Hole Packing is fixed-parameter tractable (FPT), that is, can be solved in time $f(k)n^{O(1)}$ for some function f depending only on k.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Graph algorithms analysis

Keywords and phrases chordal graphs, packing, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.71

Funding Supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.

Acknowledgements The author is very grateful to O-joung Kwon and Eun Jung Kim for helpful comments on the manuscript.

1 Introduction

The area of graph modification problems contains algorithmic tasks of the following form: given a graph G, find the minimum number of allowed editing operations to make the graph belong to a certain target graph class $\mathcal{G}$. For example, if we allow only vertex deletions and the target graph class is the set of edgeless graphs, forests, directed acyclic graphs, bipartite graphs, then we get well-known Vertex Cover, Feedback Vertex Set, Directed Feedback Vertex Set, Bipartite Deletion problems, respectively. Most of the natural graph modification problems are NP-hard [21, 30, 31]. However, there is a large literature on the fixed-parameter tractability of graph modification problems (see, e.g., [4, 8, 12, 13, 15]). Several problems of this form are known to be solvable in time $f(k)n^{O(1)}$, where k is the number of editing operations allowed (e.g., maximum number of vertices to be deleted) and f is a computable function depending only on k [9].

Let us consider the $\mathcal{G}$ Vertex Deletion problem where, given a graph G and an integer k, the task is to delete k vertices to make the graph belong to class $\mathcal{G}$. If $\mathcal{G}$ is closed under taking induced subgraphs, then there is a (finite or infinite) set $\mathcal{F}$ of obstructions such that a graph is in $\mathcal{G}$ if and only if it does not have an induced subgraph isomorphic to a member of $\mathcal{F}$. Then $\mathcal{G}$ Vertex Deletion can be equivalently expressed as finding k vertices that cover every induced copy of a member of $\mathcal{F}$. For many natural graph properties, the obstruction set $\mathcal{F}$ contains graphs that are simple to describe. For example, the problems Vertex Cover, Feedback Vertex Set, Directed Feedback Vertex Set, and Bipartite Deletion correspond to covering every edge, undirected cycle, directed cycle, and odd cycle, respectively.

Given the interpretation of $\mathcal{G}$ Vertex Deletion as covering objects from the obstruction set $\mathcal{F}$, there is natural dual problem: in the $\mathcal{F}$ Packing problem a graph G and an integer k are given, the task is to find k vertex-disjoint induced subgraphs isomorphic to members of $\mathcal{F}$.

In many cases, packing problems seem to be harder than the corresponding covering problems. First of all, if the graph class $\mathcal{G}$ is recognizable in polynomial-time, then the covering problem can be solved in $n^{O(k)}$ time by brute force, while there is no such immediate argument for the packing problem, even when the class $\mathcal{F}$ consists of very simple objects, such as cycles. For example, Directed Feedback Vertex Set is FPT [6], while the dual problem Directed Cycle Packing is W[1]-hard [28] and it requires a highly nontrivial result to show that it is polynomial-time solvable for fixed k [25]. Even when both problems are FPT, the techniques behind the algorithms could be significantly different. Bipartite Deletion has a very elegant elementary FPT algorithm using iterative compression [26], while the fixed-parameter tractability of the dual problem Odd Cycle Packing required the use of sophisticated techniques, including the introduction of odd minors [18, 19]. Another aspect from which the packing problem proved to be more difficult is the existence of polynomial kernels. For example, Feedback Vertex set (that is, covering cycles) admits a polynomial kernel [29], while the dual problem of finding k vertex-disjoint cycles does not have a polynomial kernel, under the standard complexity assumption $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$ [3].

There is a natural combinatorial question connecting the covering and packing problems. A classic result of Erdős and Pósa [11] shows that if the maximum number of vertex-disjoint cycles in graph G is k, then every cycle of G can be covered by $O(k \log k)$ vertices. A similar question can be asked about other obstructions, connecting the packing and covering problems: if the maximum number of disjoint obstructions from the set $\mathcal{F}$ is at most k, then is it true that every obstruction can be covered by $f(k)$ vertices for some function f? A positive answer is known for example when $\mathcal{F}$ is the set of edges (easy), undirected cycles going through a set S [17, 23], or directed cycles [25], but it is known that no such Erdős-Pósa property holds for odd cycles [24]. Let us observe that, as the cases of odd cycles and directed cycles show, even when the covering problem is FPT, the existence of the Erdős-Pósa property does not give a good prediction on the fixed-parameter tractability of the packing problem.

Chordal graphs. After this general introduction, let us turn our attention to chordal graphs, the main topic of the current paper. A *chordless cycle* or *hole* in a graph G is an induced cycle of length at least 4 (for brevity, we will use the term "hole" throughout the paper). A graph is *chordal* if it does not contain any hole. Chordal graphs form a well-known class of perfect graphs and it is known that a graph is chordal if and only if it can be represented as the intersection graph of a set of subtrees of a tree [14]. Chordal graphs can be recognized in linear time [27]. In the Chordal Vertex Deletion problem, a graph G and an integer k are given, and the task is to find a set S of at most k vertices such that $G - S$ is chordal. While the problem can be solved in time $n^{O(k)}$ by trying every subset of size at most k and then testing for chordality, it is also known to be FPT.

▶ **Theorem 1** ([1, 5, 16, 22]). Chordal Vertex Deletion *is FPT.*

Kim and Kwon gave a constructive proof showing that holes have the Erdős-Pósa property.

▶ **Theorem 2** (Kim and Kwon [20]). *There is a polynomial-time algorithm that, given a graph G and integer k, produces either*

1. *a set of $k+1$ disjoint holes, or*
2. *a set of $O(k^2 \log k)$ vertices covering every hole.*

Our main result concerns the Hole Packing problem, where given a graph G and an integer k, the task is to find a set of k pairwise vertex-disjoint holes.

▶ **Theorem 3** (Main Result). Hole Packing *is FPT.*

Let us remark that it is known that Chordal Vertex Deletion admits a polynomial kernel [1,16], while an easy reduction gives negative evidence for Hole Packing. Bodlaender et al. [3] showed that the problem of finding k pairwise vertex-disjoint cycles does not admit a polynomial kernel under the complexity assumption $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. If we subdivide every edge of a simple graph, then every cycle has length at least 6, which means that the holes of the new graph are in one-to-one correspondence with the cycles of the original graph. Therefore, the result of Bodlaender et al. [3] immediately implies that Hole Packing has no polynomial kernel, assuming $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. This can be seen as an indication that the dual problem Hole Packing is more challenging than Chordal Vertex Deletion, and it can be expected that more involved algorithmic ideas are needed for this problem.

Our techniques. To explain the main ideas and challenges behind the algorithm of Theorem 3, let us briefly overview the Chordal Vertex Deletion algorithm of Marx [22]; our algorithm mirrors the technical ideas from that result up to a certain point, but then it needs to deviate from it significantly. When solving Chordal Vertex Deletion, the standard technique of iterative compression [26] allows us to assume that we know a set W of $k+1$ vertices such that $G-W$ is chordal and furthermore we can assume that the solution S of size k we are looking for is disjoint from W. If the size of the largest clique in $G-W$ can be bounded by a function of k, then the treewidth of the chordal graph $G-W$ and hence also the treewidth of the slightly larger (not necessarily chordal) G can be bounded by a function of k. In this case, the problem can be solved on the graph G using standard algorithmic techniques on graphs of bounded treewidth, for example, using Courcelle's Theorem [7].

If $G-W$ has a large clique K, then, intuitively, we want to argue that a large part of the clique is not really important for the problem. More formally, we want to identify a vertex $v \in K$ such that removing k from G does not make the problem any easier. We say that v is *irrelevant* if for every set of S of size at most k disjoint from W, if there is a hole in $G-S$, then there is a hole in $G-(S \cup \{v\})$ as well. The algorithm of Marx [22] marks a certain number of vertices in K as important and then it is argued that every other vertex $v \in K$ is irrelevant in this sense. The proof is mostly a rerouting argument: if there is a hole going through v in $G-S$, then it has to be shown that the hole can be modified to avoid v.

To solve the Hole Packing problem, let us observe that Theorem 2 allows us to assume that we have a set W of $O(k^2 \log k)$ vertices such that $G-W$ is chordal: if the algorithm of Theorem 2 terminates with Outcome 1, then we are done. If $G-W$ has maximum clique size bounded by function of k, then G has bounded treewidth and we can use standard techniques to find a set of k vertex-disjoint holes. Thus our goal again is to argue that we can find an irrelevant vertex v in a large clique K. But now our notion of irrelevant vertex is different: for Chordal Vertex Deletion, a vertex needed to be irrelevant with respect to a deletion set S of at most k vertices, while for Hole Packing, vertex v needs to be irrelevant with respect to a set of $k-1$ holes. Formally, now we can say that v is *irrelevant* if whenever G has a set $\mathcal{H}$ of k disjoint holes, then there is such a set avoiding v. The set W splits $\mathcal{H}$ into at most k induced paths. Therefore, we again need a rerouting argument: the induced path going through v needs to be rerouted to avoid the other at most $|W|-1$ induced paths.

Rerouting a path to avoid a bounded number of induced paths seems to be a significantly more challenging task compared to avoiding a bounded number of vertices: the paths can be arbitrarily long and we cannot bound the number of vertices they contain. However, it is useful to observe that an induced path can contain at most two vertices from a clique. Therefore, looking locally at a clique, avoiding a bounded number of induced paths is not all that different from avoiding a bounded number of vertices. Indeed, it seems that we can

reuse many of the technical ideas from [22] to Hole Packing (we found it convenient to use somewhat different notation and streamlined some of the proofs, but we face essentially the same difficulties and similar arguments are needed). However, we did not manage to fully translate the approach to Hole Packing and to reduce the maximum clique size of $G - W$ (and hence the treewidth of G) to be bounded by a function of k. There is a particular situation where no vertex of a large clique can be declared irrelevant under our definition; Section 5 describes an example.

The first part of our algorithm uses these irrelevant vertex arguments to find a bounded-treewidth subgraph of G that contains the solution, except a few vertices of the solution that appear in a very specific situation (Section 3). This part of the proof uses ideas similar to the Chordal Vertex Deletion algorithm of [22], with appropriate modifications to account for induced paths. The main technical novelty of the paper appears in the way the problem is treated after this step. We find a way of encoding the problem in a bounded-treewidth labeled graph; however, for this to work, we need to leave the setting of the Hole Packing problem and introduce a technical variant of the problem which we call Special Hole Packing (Section 4). This problem involves finding k pairwise disjoint holes subject to certain technical conditions on the labels of vertices used by the holes. We show that when we move to this problem, then the large cliques can be reduced. Essentially, if there is a vertex v in a large clique K, then we look at the subtree intersection representation of the chordal graph $G - W$, and replace the subtree T_v representing v with a set of vertices representing the leaves of T_v. Applying this operation to every vertex of every large clique results in a graph with bounded treewidth. An appropriate choice of labeling, provided by the Color Coding [2] method, ensures that the reduction results in an instance of Special Hole Packing whose solution gives a solution to the original problem.

2 Preliminaries

We use standard graph-theoretic notation, see e.g. [10]. For background on parameterized algorithms, see [9]. In this section, we only discuss notation and basic results related to treewidth and chordal graphs.

Treewidth. A *tree decomposition* of a graph G is a pair $(T, \mathcal{B})$ in which T is a tree and $\mathcal{B} = \{B_t \mid t \in V(T)\}$ is a family of subsets of $V(G)$ such that

1. $\bigcup_{t \in V(T)} B_i = V(G)$;
2. for each edge $e = uv \in E(G)$, there exists an $t \in V(T)$ such that both u and v belong to B_t; and
3. the set of nodes $\{t \in V(T) \mid v \in B_t\}$ forms a connected subtree of T for every $v \in V(G)$.

To distinguish between vertices of the original graph G and vertices of T in the tree decomposition, we call vertices of T *nodes* and their corresponding B_i's *bags*. The *width* of the tree decomposition is the maximum size of a bag in $\mathcal{B}$ minus 1. The *treewidth* of a graph G, denoted by $\mathrm{tw}(G)$, is the minimum width over all possible tree decompositions of G.

Sentences in *Monadic Second Order Logic of Graphs* (MSO) contain quantifiers, logical connectives ($\neg$, $\vee$, and $\wedge$), vertex variables, vertex set variables, binary relations $\in$ and $=$, and the atomic formula $E(u, v)$ expressing that u and v are adjacent. Courcelle's Theorem states that if a graph property can be described in this language, then this description can be turned into an algorithm:

▶ **Theorem 4** (Courcelle [7]). *If a graph property can be described as a formula ϕ in the Monadic Second Order Logic of Graphs, then it can be recognized in time $f(|\phi|, \mathrm{tw}(G)) \cdot (|E(G)| + |V(G)|)$ if a given graph G has this property.*

Courcelle's Theorem works also in the more general setting of relational structures. For our purposes, it will be sufficient to know that the result can be extended in such a way that the input graph G comes with a labeling $\lambda : V(G) \to [c]$ of the vertices and the formula ϕ may contain unary predicates $C_1(v), \ldots, C_c(v)$, stating that vertex v has a label.

Chordal graphs. A *chordless cycle* or *hole* in a graph G is an induced cycle of length at least 4. A graph is *chordal* if it does not contain any hole. It is well known that a graph is chordal if and only if it can be represented as the intersection graph of subtrees of a tree. That is, every chordal graph G can be represented by a tree T and a subtree T_v of T corresponding to every $v \in V(G)$ such that $u, v \in V(G)$ are adjacent if and only if T_u and T_v share at least one node in T. Equivalently, a graph G is chordal if and only if it has a tree decomposition $(T, \mathcal{B})$ where every bag B_t induces a clique $G[B_t]$ for every $t \in V(T)$. Such a tree decomposition is also called a *clique tree decomposition*. We will use the well-known fact that if K is a clique in the chordal graph, then the clique tree decomposition contains a node t with $K \subseteq B_t$.

The following lemma is straightforward:

▶ **Lemma 5.** *Let x and y be two nonadjacent neighbors of v and let P be an $x - y$ path whose internal vertices are not in the closed neighborhood of v. Then the graph induced by $V(P) \cup \{v\}$ contains a hole.*

Induced paths. Suppose that $\mathcal{H}$ is a collection of holes in a graph G and W is a set of vertices that intersects each hole in $\mathcal{H}$. Then W splits each hole in $\mathcal{H}$ into some number of induced paths, that is, what remains from $\mathcal{H}$ is a collection of at most $|W|$ induced paths. This motivates the following definition.

▶ **Definition 6.** *A set X of vertices of a graph G is a* k-IP set *if it has partition $(X_1, \ldots, X_{k'})$ into $k' \leq k$ classes such that each $G[X_i]$ is an induced path (possibly of length 0, i.e., consisting only of a single vertex).*

Note that this definition allows the existence of arbitrary edges between X_i and X_j for $i \neq j$. The first basic observation is that such a set has small intersection with a clique.

▶ **Lemma 7.** *Let X be a k-IP set in a graph G and let K be a clique in G. Then X contains at most $2k$ vertices of K.*

Proof. Let $(X_1, \ldots, X_{k'})$ be a partition of X into induced paths. It is clear that an induced path can contain at most two vertices of the clique K, hence $|X \cap K| \leq 2k' \leq 2k$. ◀

The second observation is that a k-IP set can enter only a bounded number of components after the removal of a clique (or, more generally, if the neighborhood of each resulting component is a clique).

▶ **Lemma 8.** *Let X be a k-IP set in graph G and let Y be a set of vertices such that it is true for every component C of $G - Y$ that the neighborhood of C in Y is a clique. Then X intersects at most $2k$ components of $G - Y$.*

Proof. Let $(X_1, \ldots, X_{k'})$ be a partition of X into induced paths. We claim that each X_i can intersect at most two components of $G - Y$. If X_i intersects three components, then it is true for some component C that the induced path $G[P_i]$ enters C from Y and then later leaves C to Y. But the neighborhood of C in Y is a clique, contradicting the assumption that $G[X_i]$ is an induced path. Thus in total X can intersect at most $2k' \leq 2k$ components of $G - Y$. ◀

3 Part 1: Treewidth reduction (almost)

Given an instance (G, k) of Hole Packing, an application of Theorem 2 gives us a set W of $O(k^2 \log k)$ vertices such that $G - W$ is chordal. The main result of this section is a marking procedure (Lemma 9) that identifies a bounded-treewidth part of the graph that *almost* contains the solution. As $G - W$ is chordal, every hole in G contains at least one vertex of W. A *special hole* is a hole that contains exactly one vertex of W. The *special vertices* of a special hole H going through $w \in W$ are the two neighbors of w in H. Note that, as every hole has length at least 4, the two special vertices of a special hole are not adjacent.

▶ **Lemma 9.** *Let G be graph and W be a set of vertices such that $G - W$ is chordal, and let k be an integer. In polynomial time, we can find a set $S \subseteq V(G) \setminus W$ of vertices such that the following holds:*

1. *If G has a set of k pairwise disjoint holes, then there is such a set $\mathcal{H}$ where every vertex of every hole is in $S \cup W$, except possibly some of the special vertices of the special holes.*
2. *The maximum clique size in $G[S]$ is $12(|W| + 2)^4$.*

The proof of Lemma 9 starts with $S = V(G) \setminus W$ and if $G[S]$ contains a large clique K, then it tries to identify a vertex $v \in K$ that can be excluded from S without violating Requirement 1. Towards this goal, the following lemma either finds a collection of paths that are useful for creating holes going through K (Outcome 1), or marks a bounded-sized set M of vertices that are somehow important in the clique and a path reaching the clique at a vertex of $K \setminus M$ can be rerouted to reach the clique at some other vertex (Outcome 2).

▶ **Lemma 10.** *Let K be a clique in a chordal graph G, A be a set of vertices, and k be an integer. There is a polynomial-time algorithm that produces one of the following two outcomes:*

1. *A collections $\mathcal{P}$ of paths such that*
   - *every path in $\mathcal{P}$ is a path of length at least one from A to K with exactly one vertex in A,*
   - *the first endpoints of the paths in $\mathcal{P}$ form an independent set of A, and*
   - *if X is a k-IP set, then at least* two *of the paths in $\mathcal{P}$ are disjoint from X.*
2. *A subset M of K having size at most $(2k + 1)(4k + 2)$ such that following holds: if P is a path of length at least one from $a \in A$ to $v \in K \setminus M$ having exactly one vertex in A, and X is a k-IP set disjoint from $V(P)$, then there is a path P' from vertex $a \in A$ to a vertex of K such that P' is disjoint from $X \cup \{v\}$ and moreover $V(P') \setminus V(P) \subseteq K$. (Note that path P' can have length 0 and may contain more than one vertex from A.)*

Proof. Let us consider a subtree representation of the chordal graph G over the tree T. For every node $x \in V(T)$, let us denote by bag B_x the set of those vertices whose subtrees contain x. Let U be the set of nodes y for which $|B_y \cap K| > 2k + 1$. It is easy to see that U induces a connected subtree of T. Let subtrees $T_1, \ldots, T_c$ be the components of $T - U$ and let C_i contain those vertices of G whose subtrees are completely contained in the subtree T_i. Every T_i has a unique node w_i that is adjacent to U. Let us observe that the neighborhood

of every C_i is a clique: if a vertex v has a neighbor in C_i, but it is not itself in C_i, then the subtree T_v has to contain a node of T_i and a node not in T_i. This is only possible if the subtree T_v contains node w_i and such vertices v form a clique.

For every $1 \leq i \leq c$, let $M_i \subseteq K$ be defined the following way. If for a vertex $v \in K$, there is a path P_v^i of length at least one from a vertex of A to v such that P_v^i has exactly one vertex in A and every vertex of P_v^i except v is in C_i, then we put v into M_i. For the rest of the proof, let us fix such a path P_v^i for every vertex $v \in M_i$. Let us observe that if $v \in M_i$, then the subtree of v contains a node of T_i and a node of U, hence it contains the node w_i. As $w_i \notin U$ by definition, there are at most $2k+1$ vertices of K whose subtree contains w_i and $|M_i| \leq 2k+1$ follows.

Let $M = \bigcup_{i=1}^{c} M_i$. We consider two cases. If $|M| > (2k+1)(4k+1)$, then we claim the set of paths required by Outcome 1 exist. In this case, a simple greedy selection argument shows the existence of a subset of $t = 4k+2$ paths $P_{v_1}^{i_1}, \ldots, P_{v_t}^{i_t}$ of the paths defined above such that the integers $i_1, \ldots, i_t$ and the vertices $v_1, \ldots, v_t$ are all distinct. That is, if we have already selected $j \leq 4k+1$ of these paths, then they together can block at most $(2k+1)j \leq (2k+1)(4k+1)$ vertices of M, hence we can add one more path $P_{v_{j+1}}^{i_{j+1}}$ to our collection. We claim that these paths satisfy the requirements. Path $P_{v_j}^{i_j}$ starts in a vertex of $A \cap C_{i_j}$, whose subtree is fully contained in T_{i_j}. As $i_1, \ldots, i_t$ are distinct integers, the start vertices of these paths are independent vertices of A, as required. Let now X be a k-IP. By Lemma 7, X contains at most $2k$ vertices of K, thus it can intersect at most $2k$ of the vertices $v_1, \ldots, v_h$. By Lemma 8, X can intersect at most $2k$ of the sets $C_{i_1}, \ldots, C_{i_h}$: recall that the neighborhood of each C_j is a clique. In summary, X contain at most $2k$ of $v_{i_1}, \ldots, v_{i_t}$ and intersects at most $2k$ of $C_{i_t}, \ldots, C_{i_t}$, hence there are at least *two* values of j for which X is disjoint from $P_{v_j}^{i_j}$, as required.

The second case is when $|M| \leq (2k+1)(4k+2)$. In this case, we show that M satisfies the requirements of Outcome 2. Let $P = p_1, \ldots, p_\ell$ be a path as in the statement of the lemma with $\ell \geq 2$, $p_1 = a \in A$, and $p_\ell = v \in K \setminus M$ and suppose that $V(P)$ is disjoint from a k-IP set X. If $p_{\ell'} \in K$ for some $1 \leq \ell' < \ell$, then the subpath P' of P from a to $p_{\ell'}$ satisfies the requirements (this includes the case when $a \in K$; Outcome 2 allows that P' has length 0). If $p_{\ell'}$ has more than $2k+1$ neighbors in K for some $1 \leq \ell' < \ell$, then $p_{\ell'}$ has a neighbor $v' \in K \setminus (X \cup \{v\})$ (as $|X \cap K| \leq 2k$ by Lemma 5). Then the path $P' = p_1, \ldots, p_{\ell'}, v'$ satisfies the requirements. Suppose therefore that each vertex $p_{\ell'}$ with $\ell' \in [\ell - 1]$ has at most $2k+1$ neighbors in K. This implies that the subtrees corresponding to these vertices do not intersect U and it follows that all these vertices are in the same set C_i. Now the path P shows that $v \in M_i \subseteq M$, a contradiction. ◀

There is a particularly problematic special case that we handle in a separate lemma. It concerns the case when a hole contains a single vertex v from a clique K and the two neighbors of v are in W. It may seem like an easy, degenerate case (after all, rerouting to avoid v means finding another common neighbor of the two neighbors of w in W), but actually it is relatively complicated to find a replacement of v without introducing unwanted adjacencies.

▶ **Lemma 11.** *Let G be a graph with two vertices w_x and w_y such that $G - \{w_x, w_y\}$ is chordal, let K be a clique in $G - \{w_x, w_y\}$, and let k be an integer. In polynomial time, we can find a set $M \subseteq K$ of at most $(2k+4)(2k+1)$ vertices such that the following holds: If X is a k-IP set, and H is a hole disjoint from X and with $V(H) \cap W = \{w_x, w_y\}$ such that H has a vertex $v \in K \setminus M$ adjacent to both w_x and w_y, then there is a hole H' disjoint from $X \cup \{v\}$.*

Proof. Let us consider a subtree representation of the chordal graph G over the tree T. For every node $x \in V(T)$, let us denote by bag B_x the set of those vertices whose subtrees contain x. Let us assume that T is a rooted at a node r and that subtree T_u for every $u \in K$ contains r. For a node x of T, let us define $C_x \subseteq V(G) \setminus \{w_1, w_2\}$ the following way: a vertex u is in C_x if the subtree T_u of u is fully contained in the subtree of T rooted at x. Observe that the neighborhood of C_x (in $G - \{w_1, w_2\}$) is a clique: if u has a neighbor in C_x, but is not itself in C_x, then T_u has to contain the parent node of x.

We say that path P is a *good path* if it is either a path of length 0 consisting a single vertex adjacent to both w_x and w_y, or a path of length at least 1 where the unique vertex adjacent to w_x is one of the endpoints, and the unique vertex adjacent to w_y is the other endpoint. Observe that if P_1 and P_2 are two good paths that have no adjacent vertices, then P_1, P_2, w_x, w_y together form a hole of length at least 4.

Let Z be the set of nodes of T with the following property: a node x is in Z if $G[C_x]$ contains a good path. By definition, Z induces a subtree of T rooted at r: if a node is in Z, then all its ancestors are also in Z. Let $\ell_1, \ldots, \ell_t$ be the leaves of Z.

Let us consider first the case when $T[Z]$ has at least $2k + 5$ leaves. We claim that in this case returning $M = \emptyset$ is a valid answer. As the sets $C_{\ell_1}, \ldots, C_{\ell_t}$ are disjoint, nonadjacent, and the neighborhood of each of them is a clique, Lemma 8 implies that the $(k + 1)$-IP set $X \cup \{v\}$ can intersect at most $2k + 3$ of the sets $C_{\ell_1}, \ldots, C_{\ell_t}$. This means that there are at least two such sets that are disjoint from X; assume, without loss of generality that $X \cup \{v\}$ is disjoint from C_{ℓ_1} and C_{ℓ_2}. By assumption, there are two good paths P_i for $i = 1, 2$ such that P_i is in $G[C_{\ell_i}]$. As there are no edges between P_1 and P_2, we have that P_1, P_2, w_x, w_y together form a hole H' that is disjoint from $X \cup \{v\}$.

Assume therefore that $t \leq 2k + 4$. We construct M using the following procedure. For every $x \in Z$, let M_x contain the vertices $u \in K$ with the property that u is adjacent to both w_1 and w_2, and moreover T_u does not contain x. Observe that $M_x \subseteq M_y$ if x is an ancestor of y. Set $M = \emptyset$ initially. Let us consider the nodes of Z in a top down order, i.e., in a nondecreasing ordering by the distance from the root r. When considering $x \in Z$, extend M using vertices of M_x until either $|M \cap M_x| \geq 2k + 1$ or $M_x \subseteq M$. This completes the definition of M.

We claim that $|M| \leq (2k + 4)(2k + 1)$. Let the weight h_x be the number of vertices added to M when considering node x. We claim that the total weight of a node x and all its ancestors is at most $(2k + 1)$. To prove this, consider a node x such that the total weight of its proper ancestors is $h < 2k + 1$, but $h + h_x > 2k + 1$. Observe that the h vertices added to M by the proper ancestors all appear in M_x and hence we should have added only $2k + 1 - h < h_x$ new vertices of M_x when considering node x, a contradiction. In particular, the statement holds for the leaves of $G[Z]$, hence it follows that the total weight (i.e., the size of M) is at most $t(2k + 1) \leq (2k + 4)(2k + 1)$.

It remains to show that M satisfies the statement of the lemma. Observe that if we remove v, w_x, w_y from the hole H, then what remains is a good path P. Let us choose x to be node of T such that C_x contains P and x has maximum distance to the root r. This means that the bag B_x contains a vertex of P, which also implies that T_v cannot contain x, that is, $v \in M_x$. If $|M_x| \leq 2k + 1$, then the construction of M ensures $M_x \subseteq M$, contradicting $v \in K \setminus M$. If $|M_x| \geq 2k + 2$, then by Lemma 7, M_x contains a vertex v' disjoint from the k-IP set X and different from v. As P is in C_x and $v' \in M_x$, vertex v' is not adjacent to any vertex of P. Thus replacing v with v' in the hole H results in a hole H' that is disjoint from $X \cup \{v\}$. ◀

We are now ready to prove the main result of the section, Lemma 9.

Proof (of Lemma 9). The set $S = V(G) \setminus W$ trivially satisfies Requirement 1. We show that if S satisfies Requirement 1 and has a large clique violating Requirement 2, then we can remove a vertex from S in a way that Requirement 1 remains satisfied. After repeated applications of this argument, we eventually arrive to a set S that satisfies both Requirements 1 and 2.

Suppose that S satisfies Requirement 1, but $G[S]$ has a clique K of size greater than $12(|W|+2)^4$ (as $G[S]$ is chordal, such a clique can be found in polynomial time). We define a set $M \subseteq K$ by invoking the procedure of Lemma 10 with different values for the parameters (G, K, A, k) and then argue that removing from S any vertex $v \in K \setminus M$ does not violate Requirement 1. We need some definitions first. For $w \in W$, let A_w be the neighborhood of w in S. For $w_1, w_2 \in W$, let $A_{w_1,w_2} = A_{w_1} \cap A_{w_2}$ and $A_{w_1,\overline{w_2}} = A_{w_1} \setminus A_{w_2}$. Let $K_{w_1} = K \cap A_{w_1}$, $K_{\overline{w_1}} = K \setminus A_{w_1}$, $K_{w_1,w_2} = K \cap A_{w_1,w_2}$, and $K_{w_1,\overline{w_2}} = K \cap A_{w_1,\overline{w_2}}$.

The set M is the defined the following way. We invoke the algorithm of Lemma 10 with various graphs and sets as listed below.

1. For every $w \in W$, we invoke the procedure with $(G[S], K_{\overline{w}}, A_w, |W|+1)$ and we let $\mathcal{P}_w$ be the set of paths in case of Outcome 1 and M_w be the resulting set in case of Outcome 2.
2. For every $w_1, w_2 \in W$ with $w_1 \neq w_2$, we invoke the procedure with $(G[S \setminus A_{w_2}], K_{\overline{w_2}}, A_{w_1,\overline{w_2}}, |W|+1)$ and we let $\mathcal{P}_{w_1,\overline{w_2}}$ be the set of paths in case of Outcome 1 and $M_{w_1,\overline{w_2}}$ be the resulting set in case of Outcome 2.

The set M is defined to be the union of all these sets M_w and $M_{w_1,\overline{w_2}}$ (for all the values w and $(w_1, \overline{w_2})$ for which the algorithm of Lemma 10 terminated with Outcome 2). Additionally, for every distinct $w_1, w_2 \in W$, we extend M the following way:

1. We put arbitrarily $2|W|+1$ vertices of $K_{w_1,\overline{w_2}}$ into M or all such vertices, if fewer than $2|W|+1$ such vertices exist.
2. We put arbitrarily $2|W|+1$ vertices of K_{w_1,w_2} into M or all such v
3. We extend M with the set returned by the algorithm of Lemma 11 for $G[S \cup \{w_1, w_2\}]$, the clique K, and $k = |W|$.

To bound the size of M, let us observe that we invoke the algorithm of Lemma 10 exactly $|W|+|W|(|W|-1)$ times, each time with parameter $k = |W|+1$. As each call returns a set of size at most $(2k+1)(4k+2)$, the total number of vertices in M obtained this way is at most $8(|W|+2)^4$. Additionally, we include at most $|W|^2(2(2|W|+1)+(2|W|+4)(2|W|+1)) \leq 4(|W|+2)^4$ vertices into M, resulting in at most $12(|W|+2)^4$ vertices in total. We assumed that K has size greater than $12(|W|+2)^4$, hence there exists at least one vertex in $K \setminus M$.

The rest of the proof is devoted to showing that Requirement 1 remains satisfied after removing a vertex $v \in K \setminus M$ from S. If G has no k pairwise disjoint holes, then there is nothing to show; otherwise, as let us fix a collection $\mathcal{H}$ of k pairwise disjoint holes such that every vertex of these holes is in $S \cup W$, except possibly some of the special vertices. Let us choose $\mathcal{H}$ such that the total number of vertices used from W is minimized.

If no hole goes through vertex v, then we are done. Otherwise, let $H \in \mathcal{H}$ be the hole containing v. We may assume that v is not a special endpoint of H, otherwise $S \setminus \{v\}$ still satisfies Requirement 1. Let X be the union of the vertices in $V(G) \setminus W$ used by the holes in $\mathcal{H} \setminus \{H\}$. It is clear that X is a $|W|$-IP set: each hole uses at least one vertex of W and the vertices of W split these holes into a collection of at most $|W|$ induced paths. It also follows that $X \cup \{v\}$ is a $(|W|+1)$-IP set.

As $G - W$ is chordal, H contains at least one vertex of W. The following claim shows that in the definition of M above, the set M_w was defined for every vertex $w \in V(H) \cap W$ (and the same is true for $M_{w,\overline{w'}}$ for any $w' \in W$).

▷ Claim 12. For every $w \in V(H) \cap W$,

1. the set M_w is defined,
2. the set $M_{w,\overline{w'}}$ is defined for every $w' \in W$ with $w' \neq w$.

Proof. If M_w was not defined, then the algorithm of Lemma 10 terminated with Outcome 1 on input $(G[S], K_{\overline{w}}, A_w, |W|+1)$, resulting in a set $\mathcal{P}_w$ of paths satisfying the requirements of Outcome 1. This means that there are two paths $P_1, P_2 \in \mathcal{P}_w$ of length at least one that are disjoint from the $(|W|+1)$-IP set $X \cup \{v\}$. For $i = 1, 2$, let $a_i \in A_w$ and $z_i \in K$ be the endpoints of P_i. Now z_1 and z_2 either coincide or are adjacent in the clique K, thus concatenating them gives a walk that contains an $a_1 - a_2$ simple path P. From the conditions on P_1 and P_2, we also know that a_1 and a_2 are independent and the internal vertices of P are not in A_w, that is, not adjacent to w. Thus Lemma 5 shows that w and P form a hole H' of length at least 4 disjoint from $X \cup \{v\}$ and fully contained in $S \cup W$. Replacing H with H' in $\mathcal{H}$ would give a collection of k holes satisfying Requirement 1, even if v is removed from S, what we wanted to show.

In an analogous way, we can show that if the algorithm of Lemma 10 terminated with Outcome 1 on input $(G[S \setminus A_{w'}], K_{\overline{w'}}, A_{w,\overline{w'}}, |W|+1)$, then there is a path of length at least 2 connecting two independent vertices of $A_{w,\overline{w'}}$ in $G[S \setminus A_w]$ that is disjoint from $X \cup \{v\}$, forming a hole with w. This proves the second statement. ◁

The set $V(H) \setminus W$ induces a set of at least one and at most $|W|$ induced paths in $G - W$ (a path can consists of only a single vertex). Let P be the subpath of H that contains v. Fixing an orientation of H, let $w_x, w_y \in W$ be the previous and next vertices of H (note that $|V(H) \cap W| = 1$ if and only if $w_x = w_y$). We try to reroute P to obtain a path P' that avoids v. Then we replace P with P' in the hole H and try to obtain a hole H' that avoids v, showing that there is a collection of k disjoint holes that satisfies Requirement 1 even if v is removed from S. For this, we need to ensure that the new path P' is independent from the rest of H in a certain way. Thus there are two challenges here: finding the rerouted path P' that avoids v and ensuring that replacing P with P' results in hole.

The rest of the proof depends on the size $V(H) \cap W$. The most generic case is when $V(H) \cap W$ has at least 3 vertices. In this case, we can use that H visits a third vertex $w_z \in H$ different w_x, w_y and P is not adjacent to P; then it is sufficient to ensure that P' is also not adjacent to w_y. The case $|V(H) \cap W| = 2$ is similar, but then we need different arguments to ensure that H' is a hole. In the case $|V(H) \cap W| = 1$, an additional complication is that $w_x = w_y$, hence the endpoints of P' need to be nonadjacent.

Case A: $|V(H) \cap W| \geq 3$. Then $w_x \neq w_y$ and there is at least one other $w_z \in V(H) \cap W$ different from w_x and w_y. Our goal is to show that there is an $A_{w_x,\overline{w_z}} - A_{w_y,\overline{w_z}}$ path P' in $G[S \setminus A_{w_z}]$ that is disjoint from $X \cup \{v\}$. Assuming there is such a path P', let u_1 and u_2 be the two neighbors of w_z on the hole H (note that $u_1, u_2 \notin P$). Replacing P with P' in the hole H shows that that there is a $u_1 - u_2$ walk whose internal vertices are not adjacent to w_z (here we use that the endpoints of P' are adjacent to w_x and w_y, respectively, and the vertices of P' are not adjacent to w_z). Thus by Lemma 5, there is a hole disjoint from $X \cup \{v\}$.

The path P' is constructed as follows. We will construct two paths P'_x and P'_y in $G[S \setminus A_{w_z}]$ such that P'_x and P'_y are $A_{w_x,\overline{w_z}} - K_{\overline{w_z}}$ and $A_{w_y,\overline{w_z}} - K_{\overline{w_z}}$ paths, respectively, and they are both disjoint from $X \cup \{v\}$. As any two vertices of $K_{\overline{w_z}}$ are adjacent, the concatenation of the two paths gives a walk that can be simplified to the required path P'.

The path P can be split into a path P_x going from a vertex $v_x \in A_{w_x,\overline{w_z}}$ to v, and into a path P_y going from a vertex $v_y \in A_{w_y,\overline{w_z}}$ to v (now one or both of these paths can be of length 0). We show how to construct P'_x; the construction of P'_y is analogous. If P_x has length at least one, then we use that, by Claim 12, Outcome 2 was the result of applying Lemma 10 when defining $M_{w_x,\overline{w_z}}$. As P_x is a path from v_x to $v \in K_{\overline{w_z}} \setminus M \subseteq K_{\overline{w_z}} \setminus M_{w_x,\overline{w_z}}$ and it is disjoint from X, Outcome 2 guarantees the existence of a path P'_x from v_x to $K_{\overline{w_z}}$ that is disjoint from $X \cup \{v\}$. If P_x has length 0, then $v_x = v$ is in $K_{w_x,\overline{w_z}}$. When defining the set M, we tried to put $2|W|+1$ vertices of $K_{w_x,\overline{w_z}}$ into M. As v was not put into this set, we have that M contains $2|W|+1$ other vertices of $K_{w_x,\overline{w_z}}$. As the $|W|$-IP set X can contain at most $2|W|$ vertices of $K_{\overline{w_z}}$ (Lemma 7), there is a vertex $v' \in K_{w_x,\overline{w_z}} \setminus X$ different from v. Now the path P'_x of length 0 consisting of only v' satisfies the requirements. Path P'_y can be obtained in a similar way, completing our proof for the existence of the path P'.

Case B: $|V(H) \cap W| = 2$. In this case, $H - W$ consists of either only the path P, or two paths P and P^*. Note that if $H - W$ consists of only P, then the two vertices in $w_x, w_y \in V(H) \cap W$ are adjacent and P has length at least two. Let us first handle the case when P has length 0, i.e., it consist of a single vertex v adjacent to both w_x and w_y. Then the fact that M includes the set returned by Lemma 11 for graph $G[S \cup \{w_x, w_y\}]$, clique K, $k = |W|$ implies that there is a hole H' disjoint from the $|W|$-IP set X and from v.

In the following, we assume that P has length at least 1 (which in particular implies that P has no vertex adjacent to both w_x and w_y). If P^* exists, then we claim that no vertex of P^* is adjacent to a vertex $u \in K \setminus (X \cup K_{w_x,w_y})$. Otherwise, without loss of generality, assume that u is not adjacent to w_x (the other case being symmetric). Let v_x be an endpoint of P adjacent to w_x, and consider the subpath Q of P from v to v_x. Let v^*_x be an endpoint of P^* adjacent to w_x and consider the subpath Q^* of P^* from v^*_x to a vertex u^* that is adjacent to u. Now v_x and v^*_x are nonadjacent neighbors of w_x (because the paths P and P^* are not adjacent) and the walk Q^*u^*uvQ goes from v^*_x to v_x. Note that no internal vertex of this walk is adjacent to w_x (as we assumed that this is true for u). Therefore, there is a hole H' that is disjoint from the holes in $\mathcal{H} \setminus \{H\}$ (the only vertex possibly used by H' that is not used by H is $u \in K \setminus X$) and uses only one vertex of W, contradicting the minimal choice of $\mathcal{H}$.

If P has length at least 1, then it has an endpoint $v_x \in A_{w_x,\overline{w_y}}$ and an endpoint $v_y \in A_{w_y,\overline{w_x}}$. Then we proceed very similarly to Case A above. The path P can be split into a path P_x going from vertex $v_x \in A_{w_x,\overline{w_y}}$ to v, and into a path P_y going from vertex $v_y \in A_{w_y,\overline{w_y}}$ to v (one, but not both, of these paths can be of length 0). We construct two paths P'_x and P'_y in G such that P'_x and P'_y are $A_{w_x,\overline{w_y}} - K$ and $A_{w_y,\overline{w_x}} - K$ paths, respectively, and they are both disjoint from $X \cup \{v\}$. If P_x has length at least 1, then the definition of M_{w_x} obtained by Outcome 2 of Lemma 10 (which exists by Claim 12) shows that a path P'_x disjoint from $X \cup \{v\}$ exists, and moreover any vertex of P'_x not in $V(P_x)$ is in $K_{\overline{w_x}}$. If P_x is only a single vertex, $v_x = v$ and the definition of M includes at least $2|W|+1$ vertices from $K_{w_x,\overline{w_y}}$ and there exists a vertex $v' \in K_{w_x,\overline{w_y}} \setminus (X \cup \{v\})$. Path P'_y can be constructed similarly. Putting together these two paths gives an $A_{w_x,\overline{w_y}} - A_{w_y,\overline{w_x}}$ walk Q. Observe that no vertex of Q is in A_{w_x,w_y}: there is no such vertex in $V(P_x) \cup V(P_y)$ and the new vertices we may introduce when defining P'_x or P'_y came from $K_{\overline{w_y}}$ or from $K_{\overline{w_x}}$, hence they cannot be in A_{w_x,w_y} either. Therefore, walk Q has a simple $A_{w_x,\overline{w_y}} - A_{w_y,\overline{w_x}}$ subpath P' of length at least 1 whose internal vertices are disjoint from A_{w_x} and A_{w_y}. As P' was constructed in such a way that any vertex of it not in $V(P)$ is in the clique $K \setminus (X \cup K_{w_x,w_y})$, our claim in the previous paragraph shows that P^* (if exists) is not adjacent to P'. Thus replacing w_xPw_y with $w_xP'w_y$ in the hole H shows the existence of a hole H' disjoint from $X \cup \{v\}$.

Case C: $|V(H) \cap W| = 1$. Let $V(H) \cap W = \{w\}$. We again proceed similarly as in Case A, but we use the fact v is not a special endpoint of H, that is, v is not an endpoint of P. The path P can be split into a path P_x going from a vertex $v_x \in A_w$ to v, and into a path P_y going from a vertex $v_y \in A_w$ to v; both paths have length at least 1 (as v is not a special endpoint) and v_x and v_y are not adjacent. We will construct a $v_x - K$ path P'_x and a $v_y - K$ path P'_y in G that are disjoint from $X \cup \{v\}$ and moreover v_x and v_y are the only vertices of A_w on these paths. To construct P'_x, consider the set M_w, which was defined by Outcome 2 of Lemma 10 applied on $(G[S], K_{\overline{w}}, A_w, |W| + 1)$ (by Claim 12, the set M_w is defined). As P_x has length at least 1 and $v \notin A_w$, the assumption $v \notin K \setminus M_w$ implies that there is a path P'_x from v_x to a vertex of K that is disjoint from $X \cup \{v\}$ and has exactly one vertex in A_w (namely, v_x). Moreover, any vertex of P'_x not in $V(P_x)$ is in $K_{\overline{w}}$, it follows that v_x is the only vertex of P'_x in A_w, as required. Path P'_y can be constructed in a similar way. Then P'_x and P'_y show that existence of a path P' from v_x to v_y with no internal vertices in A_w. Finally, Lemma 5 applied on w and P' shows the existence of a hole H' disjoint from $X \cup \{v\}$. ◀

4 Part 2: Special Hole Packing

Lemma 9 did not manage to fully reduce the treewidth of the Hole Packing instance. To proceed, we introduce a generalization Hole Packing and show that in this generalization it is possible to encode the original instance of Hole Packing in a way that the treewidth is reduced to a function of size of the chordal deletion set W given in the input.

We define the Special Hole Packing problem the following way. The input is a vertex-labeled graph G and two integers $\ell \leq k$. The possible labels of the vertices are 0, i, or i^* for $i \in [\ell]$ (i.e., there are $2\ell + 1$ different labels). The task is to find k pairwise vertex-disjoint holes $H_1, \ldots, H_k$ with the following additional conditions. For $\ell < i \leq k$, every vertex of H_i should have label 0. For $i \in [\ell]$, hole H_i should have exactly one vertex v_i with label i and every other vertex of H_i has label 0. Moreover, if v'_i, v''_i are the neighbors of v_i in H_i, then there should exist a $v'_i - v''_i$ path P_i whose internal vertices have label i^*.

Using Courcelle's Theorem (Theorem 4), it is not difficult to show that Special Hole Packing is FPT with combined parameters k and the treewidth of G: it is routine to describe the problem in Monadic Second Order Logic. A tedious but standard dynamic programming algorithm can also show that running time $2^{\mathrm{poly}(k+\mathrm{tw}(G))} \cdot n$ is also possible.

▶ **Lemma 13.** Special Hole Packing *is FPT parameterized by* $k + \mathrm{tw}(G)$.

The final part of the proof is to reduce Hole Packing to Special Hole Packing on a bounded treewidth graph. The main idea is the following. We first use Lemma 9 to mark a set S of vertices in the chordal graph $G - W$. We can assume that if a vertex v is not in $S \cup W$, then it may only be used as a special vertex of a special hole. We treat such vertices v as follows. Suppose that v is adjacent to w_i and represented by a subtree T_v in the clique tree decomposition of $G - W$. Let $a_1, \ldots, a_t$ be the leaves of T_v. Then we "blow up" v: we remove it and for each a_j, we introduce a new vertex v_j that is adjacent to w_i and whose representation in the clique tree decomposition is just the single node a_j. We perform this step for every $v \notin S \cup W$. Using easy transformations of the clique tree, we can ensure that every node of the clique tree is the leaf of at most one subtree, hence we add at most one new vertex at each node. This ensures that the resulting modified graph has bounded treewidth.

There are at least two obvious problems with this transformation. First, vertex v is replaced by multiple vertices $v_1, \ldots, v_t$ and the solution may use more than one of them, effectively using v in more than one hole. However, we can use the Color Coding [2] technique

in a straightforward way to enforce that each vertex is used only in one hole as special vertex. The second problem is that the transformation loses information about adjacency. Suppose that we find a solution that contains a special hole H_i that consists of vertex w_i and a path P, where path P connects a newly introduced vertex v_j with a vertex u (that is not adjacent to v_j). Now the orginial vertex v could be adjacent to u in the original graph, hence replacing v_j with v may not give a hole in the original graph. This is the point where the required path P_i on vertices with label i^* comes into play: we introduce these vertices in a way that forces the $v_j - u$ path to "go away" from the tree T_v, making sure that it ends at a vertex u that is not a neighbor of v. More precisely, we introduce a tree-like "scaffolding" with label i^* and then we break this scaffolding in a way that potential paths of label i^* can touch only the leaves of every such tree T_v.

We state the main result of the section as the fixed-parameter tractability of Hole Packing parameterized by the size of a chordal deletion set given in the input.

▶ **Lemma 14.** *Given a graph G, integer k, and vertex set $W \subseteq V(G)$ such that $G - W$ is chordal, the* Hole Packing *instance (G, k) can be solved in time $f(|W|)n^{O(1)}$.*

Proof. Let S be the set given by Lemma 9 and let us fix a hypothetical solution $\mathcal{H}$ of k holes such that every vertex of every hole is in $S \cup W$, except perhaps some of the special vertices. If $w_1, \ldots, w_\ell$ are distinct vertices from W, then we say that $\mathcal{H}$ is *consistent* with the tuple $(w_1, \ldots, w_\ell)$ if each w_i for $i \in [\ell]$ is in a special hole and no other vertex of W is in a special hole; in particular, this implies that there are exactly ℓ special holes. The algorithm first guesses a tuple $(w_1, \ldots, w_\ell)$ with which $\mathcal{H}$ is consistent (as the order of the w_i's do not matter, we have $2^{|W|}$ different possibilities). In the following, let $H_i \in \mathcal{H}$ be the special hole going through w_i (note that each special hole uses exactly one vertex of W hence the H_i's are distinct).

Let $\lambda : V(G) \setminus (W \cup S) \to [\ell]$ be an arbitrary labeling and let X_i be the set of vertices with label i. We say that $\mathcal{H}$ is *consistent* with $(\lambda; w_1, \ldots, w_\ell)$ if it is consistent with $(w_1, \ldots, w_\ell)$ and moreover the special endpoints of H_i are in $X_i \cup S$. Observe that this definition puts a requirement on the labeling of at most 2ℓ vertices. Thus we can use Color Coding [2]: by going through a 2ℓ-perfect family of hash functions of size $2^{O(\ell)} \cdot n$, we can assume that we have a fixed $(\lambda; w_1, \ldots, w_\ell)$ with which the hypothetical solution $\mathcal{H}$ is consistent.

Given G and $(\lambda; w_1, \ldots, w_\ell)$, our goal is to obtain a labeled bounded-treewidth graph G' and invoke the algorithm of Lemma 13 for Special Hole Packing on (G', k). To define G', let us fix a clique tree decomposition of the chordal graph $G - W$. It will be convenient to assume the following extra properties of the clique tree decomposition:

(P1) Every tree T_x has at least two vertices.
(P2) The maximum degree of T is at most 3.
(P3) If u and v are adjacent in T and one of them has degree 3, then $B_u = B_v$.
(P4) Every leaf of every subtree T_x is in a degree-2 node of T.
(P5) Every node of u is the leaf of at most one subtree T_x.

Property (P1) can be achieved by attaching a new leaf to each node u, having the same bag B_u. Properties (P2) and (P3) can be achieved by replacing each node u of degree $d \geq 3$ with a binary tree having d leaves and each node having the same bag B_u. Then the neighbors of u can be connected to the leaves of this tree. This replacement also ensures that every leaf of every subtree T_x is in a node with degree at most 2. Therefore, Property (P4) can be achieved simply by attaching a new node with empty bag to each leaf node of T (to avoid that some T_x has a leaf in a degree-1 node). Property (P5) can be achieved by an appropriate sequence of subdivisions at each edge of T.

To construct G', let us start with $G[S \cup W]$, let us assign label i to w_i and label 0 to every other vertex. For every $i \in [\ell]$, we proceed the following way. We will add new vertices to G' and the way we are describing these new vertices is by adding new subtrees to the clique tree decomposition of $G - W$ (which defines how these new vertices are adjacent to the vertices not in W) and explicitly specifying how the new vertices are adjacent to W. Note that $G' - W$ will be a chordal graph defined by this clique tree decomposition. First, for every edge uv of T, we introduce a new vertex with label i^* whose subtree consists of nodes u and v. Next, for every vertex $x \in X_i$ that is adjacent to w_i, and for every leaf u of T_x, we do the following. By (P1), subtree T_u has at least two vertices, thus u has a unique neighbor v that is in T_x. We remove the vertex with label i^* whose subtree consists of $\{u, v\}$ and introduce a new vertex with label 0 that is adjacent to w_i and whose subtree consists of only $\{u\}$. This completes the description of G'.

We claim that G' has treewidth at most $12(|W|+2)^2+4|W|$. The graph $G[S]$ is a chordal graph with maximum clique size $12(|W|+2)^4$, hence each bag contains at most that many vertices. For every i, we introduce at most 3 new vertices with label i^* in each bag (as (P2) requires that every node of T have degree at most 3). Furthermore, as each node of T contains the leaf of at most one subtree T_x by (P5), we may introduce at most one new vertex with label 0 in each bag. Therefore, $G' - W$ has a clique tree decomposition where every bag has size at most $12(|W|+2)^2+3|W|+1$. This means that $G' - W$ has treewidth at most $12(|W|+2)^2+3|W|$ and hence G' has treewidth at most $12(|W|+2)^2+4|W|$. Therefore, by Lemma 13, we can solve Special Hole Packing for (G', k) in time $f(|W|) \cdot n^{O(1)}$. The following claim shows that in case we find a solution to this Special Hole Packing instance, then it allows us to find k disjoint holes in G.

▷ Claim 15. Given a solution for the Special Hole Packing instance (G', k), we can construct in polynomial time a set of k pairwise vertex-disjoint holes in G.

Proof. Let $H'_1, \ldots, H'_k$ be the solution of the Special Hole Packing instance (G', k). We show first that for $i > \ell$, hole H'_i in G' is a hole in G as well. Recall that for $i > \ell$, hole H'_i contains only vertices of label 0. The only potential problem is that H'_i contains a vertex x^* that do not appear in G, but was added during the construction of G'. But recall that every such vertex x^* was defined by introducing a subtree T_{x^*} containing only a single node u of T and x^* was also made adjacent to a single w_j for some $j \in [\ell]$. Thus the neighbors of x^* is w_j plus a clique, which means that any hole going through x^* has to go through w_j as well. As w_j is the only vertex with label j, we have that H'_j has to contain it and hence hole H'_i for $i > \ell$ does not contain w_j. This shows that H'_i is a hole in G as well.

Consider now the hole H'_i for some $i \in [\ell]$ and the path P_i that connect neighbors x, y of w_i in H'_i and whose internal vertices are labeled i^*. This hole may contain vertices not present in G, but the argument in the previous paragraph shows that there are at most two such vertices: x and y. We show that if these vertices are not present in G, then they can be replaced by vertices in X_i, resulting in a hole H_i of G.

As x and y are not adjacent, there is a path $p_1 p_2 \ldots p_t$ in T with $t \geq 1$ such that p_1 is the only vertex of the path in T_x and p_t is the only vertex of the path in T_y. For every $j \in [t-1]$, the construction of G' involved introducing an i^*-labeled vertex z_j whose subtree is exactly $\{p_j, p_{j+1}\}$. It is clear that the path P_i consists of the vertices x, z_1, $\ldots$, z_t, y: there is no other way of connecting x and y with a simple path whose internal vertices have label i^*. If x is not a vertex of G, then x was introduced in the construction of G' because there is a vertex $x_0 \in X_i$ that is adjacent to w_i and p_1 is the leaf of the subtree T_{x_0}; let us replace x with x_0 in the hole H'_i. Similarly, if y is not part of G, then y can be replaced by a vertex $y_0 \in X_i$ that is adjacent to w_i and whose subtree T_{y_0} contains p_t. We have to verify

that the hole H_i obtained by replacing x with x_0 and/or y with y_0 is indeed a hole. First, $V(T_x) \subseteq V(T_{x_0})$ (as T_x contains only node p_1), hence x_0 is adjacent to every neighbor of x. Therefore, we only need to verify that no unwanted new edge appears in H_i after the replacement. The crucial point here is that p_2 is not in T_{x_0}: in that case, we would have removed vertex z_1 during the construction of G'. Thus p_1 is the only node of T_{x_0} on the path $p_1, \ldots, p_t$. Similarly, if y is replaced by y_0, then p_t is the only node of T_{y_0} on this path. It is easy to see that the subtree of every vertex of $V(H'_i) \setminus \{w_i\}$ contains a node from this path and therefore if, e.g., x_0 is adjacent to such a vertex, then already x was adjacent to that. Thus after the replacements, we indeed obtain a hole H_i in G. Performing this step for every $i \in [\ell]$ gives a set $H_1, \ldots, H_\ell, H'_{\ell+1}, \ldots, H'_k$ of holes. The disjointness of these holes follow from the disjointness of $H'_1, \ldots, H'_k$ and from the fact that $V(H_i) \setminus V(H'_i)$ is in X_i, and these vertices cannot be used by any hole other than H_i. ◁

Conversely, the (contraposition of the) following claim shows that if the answer to the Special Hole Packing instance is no, then we know that there is no set of disjoint holes consistent with our current choice of $(\lambda; w_1, \ldots, w_\ell)$.

▷ Claim 16. If G has a set $\mathcal{H}$ of pairwise disjoint holes consistent with $(\lambda; w_1, \ldots, w_\ell)$, then Special Hole Packing for (G', k) has a solution.

Proof. Let $H_1, \ldots, H_k$ be disjoint holes in G such that H_i for $i \in [\ell]$ is a special hole going through vertex w_i. For special hole H_i where vertices x and y are the neighbors of w_i, we define the *gap size* of H_i to be the distance of T_x and T_y in T. As x and y are not adjacent, the gap size is positive. Let us choose $H_1, \ldots, H_k$ such that the sum of gap sizes is minimum possible.

For $i > \ell$, the vertices of H_i are contained in $S \cup W$, hence they are also holes in G' as well with every vertex having label 0. Consider now the special hole H_i for $i \in [\ell]$ and suppose it has the form $w_i x x' P y' y$ where P is an $x' - y'$ path (with $x' = y'$ if H_i has length 4).

As x and y are not adjacent, there is a path $p_1 p_2 \ldots p_t$ in T with $t \geq 1$ such that p_1 is the only vertex of the path in T_x and p_t is the only vertex of the path in T_y. Observe that this means that the gap size of H_i is exactly $t - 1$. It easy to see that every bag B_{p_i} for $i \in [t]$ contains at least one vertex of the path P. By (P4), the leaves are in degree-2 nodes, hence there is a unique vertex p_0 before p_1 and a unique vertex p_{t+1} after p_t. For every $j \in [t-1]$, the construction of G' involved introducing an i^*-labeled vertex z_j whose subtree is exactly $\{p_j, p_{j+1}\}$. We argue that none of these vertices z_j were removed during the construction of G'. Recall that z_j was removed if there was a vertex $q \in X_i$ that is adjacent to w_i and either T_q has a leaf in p_{j+1} and T_q contains p_j, or T_q has a leaf in p_j and T_q contains p_{j+1}. We show that if a z_j was removed during the construction of G', then H_i can be replaced with another hole that has strictly smaller gap size and is still disjoint from the rest of the holes. This would contradict the minimality of the choice of $\mathcal{H}$.

Let α be the largest integer $\leq t$ such that there is a vertex in $x^* \in X_i \cup \{x\}$ whose subtree has a leaf in p_α and contains $p_{\alpha-1}$. Vertex x shows that α is well-defined and at least 1. Furthermore, vertex y shows that α cannot be t: by (P5), node p_t is the leaf of only T_y and this subtree does not contain p_{t-1}. Let $\beta \geq \alpha$ be the smallest integer such that there is a vertex in $y^* \in X_i \cup \{y\}$ whose subtree has a leaf in p_β and contains $p_{\beta+1}$. Vertex y shows that β is well-defined and $\beta \leq t$. Furthermore, we have that $\beta \neq \alpha$, as otherwise both x^* and y^* would have a leaf at the same node (and they are distinct vertices). Now $\beta > \alpha$ implies that x^* and y^* are not adjacent. It can be also observed that vertex z_j with $\alpha \leq j \leq \beta - 1$ was not removed: removing it because of a subtree with leaf in p_{j+1} and containing p_j would contradict the maximality of α; removing it because of a subtree with

leaf in p_j and containing p_{j+1} would violate the minimality of β. Therefore, if $\alpha = 1$ and $\beta = t$, then none of $z_1, \ldots, z_{t-1}$ was removed, what we wanted to show. Otherwise, there is an $x^* - y^*$ path whose internal vertices are in $V(P)$ (as every bag $B_{p_\alpha}, \ldots, B_{p_\beta}$ contains a vertex of P). Then Lemma 5 implies that there is a hole H_i^* going through x^*, w_i, and y^*. This hole is disjoint from the other holes in $\mathcal{H} \setminus \{H_i\}$: vertices in $V(H_i^*) \setminus V(H_i)$ are from X_i, and consistency of $\mathcal{H}$ with $(\lambda; w_1, \ldots, w_\ell)$ means that only H_i could use vertices from X_i. Now the gap size of H_i^* is $\beta - \alpha < t - 1$, strictly smaller than the gap size of H_i, contradicting the minimal choice of $\mathcal{H}$. This completes the proof that all of the vertices z_1, $\ldots$, z_{t-1} are in G'.

Let us show now the existence of a hole H_i' required by the Special Hole Packing problem. If $x, y \in S$, then they are present in G' with label 0, hence H_i is a hole in G' with every vertex except w_i having label 0. Then $P_i = xz_1 \ldots z_{t-1}y$ is a path with internal vertices labeled i^*, as required. Otherwise, suppose that $x \notin S$. As $\mathcal{H}$ is consistent with $(\lambda; w_1, \ldots, w_\ell)$, this is only possible if $x \in X_i$. As x is adjacent to w_i and p_1 is a leaf of T_x, we have introduced a vertex x^* that is adjacent to w_i and whose subtree consists of only p_1. Let us observe that x^* is adjacent to x': this follows from the fact that the induced path $xx'Py'y$ connects x and y, hence the subtree $T_{x'}$ should contain a node of the component of $T - V(T_x)$ that contains T_y. As $V(T_{x^*}) \subseteq V(T_x)$, vertex x^* cannot be adjacent to any vertex that is not a neighbor of x. Let us replace x with x^*; in a similar way, if $y \notin S$, then we can replace it with an appropriate vertex y^* that is adjacent to w_i and y'. Then the resulting hole H_i' is a hole in G' where every vertex except w_i has label 0. Furthermore, the two neighbors of w_i in H_i' can be connected by a path P_i whose internal vertices are $z_1, \ldots, z_{t-1}$, as required in the definition of Special Hole Packing. ◁

In summary, we can solve Hole Packing the following way. We enumerate every possibility for $(w_1, \ldots, w_\ell)$ and every mapping λ in a 2ℓ-perfect family of hash functions. For each choice of $(\lambda; w_1, \ldots, w_\ell)$, we construct the graph G' and solve the Special Hole Packing instance (G', k) using the algorithm of Lemma 13. If it returns a solution, then Claim 15 allows us to turn this into a solution of Hole Packing. The correctness of the algorithm follows from the fact if there is a set $\mathcal{H}$ of k pairwise disjoint holes, then at some point we reach a tuple $(\lambda; w_1, \ldots, w_\ell)$ with which $\mathcal{H}$ is consistent. At this point, Claim 16 shows that the Special Hole Packing instance (G', k) has a solution, and hence our algorithm indeed returns a solution for Hole Packing. ◀

To prove our main result Theorem 3, let us invoke the algorithm of Theorem 2 on the Hole Packing instance (G, k). If it returns a set of $k + 1$ disjoint holes, then we are done. Otherwise, we can assume that we have a set W of $O(k^2 \log k)$ vertices such that $G - W$ is chordal. Then we can use Lemma 14 to solve the problem in time $f(|W|)n^{O(1)} = f'(k)n^{O(1)}$.

5 A difficult situation

The set S computed by Lemma 9 does not necessarily cover the special vertices of the special holes. If we could ensure that set S covers every vertex of the solution, then we could immediately apply Courcelle's Theorem and we would not need the arguments in Section 4. This raises the question whether we could improve the proof of Lemma 9 in such a way.

Of course, a solution can use at most $2k$ vertices from a clique, so a large clique certainly has a vertex v not needed for a solution. Therefore, technically speaking, we can prove a variant of Lemma 9 without the extra condition for the special vertices: using our algorithm for Hole Packing, we can find a set k disjoint holes and we can set S to be the vertices of these holes. A better question is whether we can prove the inductive rerouting argument in the proof of Lemma 9 without the extra provision for the special vertices. Notice that the

proof of Lemma 9 shows the following: if a hole of the solution goes through a vertex $v \notin S$ of a large clique, then we can reroute that hole *without modifying any of the other holes.* Therefore, the question is whether we can find a vertex in a large clique that is irrelevant for the existence of a hole, even after the deletion of a set of other holes.

▶ **Question 17.** *Let G be a graph and W a set of vertices such that $G - W$ is chordal and let K be a clique in $G - W$. We say that $v \in K$ is* irrelevant *if whenever $\mathcal{H}$ is a collection of disjoint holes*[1] *in G such that $G-(\bigcup_{H\in\mathcal{H}} V(H))$ contains a hole, then $G-(\bigcup_{H\in\mathcal{H}} V(H)\cup\{v\})$ contains a hole as well. Is there a function f such that every clique K of $G - W$ having size at least $f(|W|)$ contains an irrelevant vertex?*

We construct a simple example that gives a negative answer to this question. Therefore, when trying to declare a vertex v as irrelevant, we cannot argue just by rerouting the hole going through v in the solution: we may need to reroute some other holes of the solution as well. This shows that a version of Lemma 9 without the provision for the special vertices would require a proof of very different flavor.

We define first a chordal graph G_0 the following way. Let the tree T contain nodes x_0, $\ldots$, x_{n+1} forming a path and let y_i be a degree-1 neighbor of x_i for $i \in [n]$. We define chordal graph G_0 as the intersection graph of subtrees of T:

- For $i \in [n]$, vertex $a_i \in V(G_0)$ corresponds to a subtree with nodes $\{x_{i-1}, x_i, x_{i+1}, y_i\}$.
- For $3 \leq i \leq n-2$, vertex $b_i \in V(G_0)$ corresponds to a subtree with nodes $\bigcup_{j\in[n]}\{x_j, y_j\} \setminus \{y_i\}$.
- For $i \in [n]$, vertex $c_i \in V(G_0)$ corresponds to a subtree with node $\{y_i\}$.

Observe that the a_i's form a path P, the b_i's form a clique K and the c_i's form an independent set I. Graph G is defined by adding three new vertices w_1, w_2, w_3 to G_0, making w_1 and w_2 adjacent to I, and making w_3 adjacent to $K \cup I$.

Let us choose an arbitrary $3 \leq i \leq n-2$. Consider the holes $H_1 = w_1c_1a_1a_2 \ldots a_{i-1}c_{i-1}w_1$ and $H_2 = w_2c_{i+1}a_{i+1}a_{i+2} \ldots a_nc_nw_2$. We claim that $H_3 = w_3c_ia_ib_iw_3$ is the only hole in $G - (V(H_1) \cup V(H_2))$, showing that b_i is not an irrelevant vertex. As $G - \{w_1, w_2, w_3\}$ is chordal, such a hole has to go through a w_3 and then contain two nonadjacent neighbors of w_3. As K is a clique, this means that hole H contains c_j for some $j \in [n]$. Consider the neighbor of c_j in H different from w_3. This vertex cannot be from K (as it cannot be a neighbor of w_3), hence a_j is the only possible neighbor of c_j. Hole H_1 uses the vertices a_1, $\ldots$, a_{i-1}, while hole H_2 uses the vertices a_{i+1}, $\ldots$, a_n, which leaves only a_i, and $i = j$ follows. Therefore, hole H contains vertices $w_3c_ia_i$, which can be completed to a hole only by vertex b_i (the only vertex of K not adjacent to c_i), as claimed. As this argument holds for every $3 \leq i \leq n-2$, none of the $n-4$ vertices of the clique K are irrelevant. The construction is valid for arbitrary large n and we have $|W| = 3$, which rules out the possibility that the function f of Question 17 depending only on $|W|$ exists.

References

1 Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Feedback vertex set inspired kernel for chordal vertex deletion. *ACM Trans. Algorithms*, 15(1):11:1–11:28, 2019. doi:10.1145/3284356.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

[1] Note that $\mathcal{H}$ can contain at most $|W|$ holes.

3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernel bounds for path and cycle problems. *Theor. Comput. Sci.*, 511:117–136, 2013. doi:10.1016/j.tcs.2012.09.006.

4 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

5 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016. doi:10.1007/s00453-015-0014-x.

6 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.

7 Bruno Courcelle. Graph rewriting: an algebraic and logic approach. In *Handbook of theoretical computer science, Vol. B*, pages 193–242. Elsevier, Amsterdam, 1990.

8 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *CoRR*, abs/2001.06867, 2020. arXiv:2001.06867.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 P. Erdős and L. Pósa. On independent circuits contained in a graph. *Canad. J. Math.*, 17:347–352, 1965.

12 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. On the parameterized complexity of graph modification to first-order logic properties. *Theory Comput. Syst.*, 64(2):251–271, 2020. doi:10.1007/s00224-019-09938-8.

13 Fedor V. Fomin, Saket Saurabh, and Neeldhara Misra. Graph modification problems: A modern perspective. In Jianxin Wang and Chee-Keng Yap, editors, *Frontiers in Algorithmics - 9th International Workshop, FAW 2015, Guilin, China, July 3-5, 2015, Proceedings*, volume 9130 of *Lecture Notes in Computer Science*, pages 3–6. Springer, 2015. doi:10.1007/978-3-319-19647-3_1.

14 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, New York, 1980.

15 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004. doi:10.1007/s00453-004-1090-5.

16 Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. *SIAM J. Discrete Math.*, 32(3):2258–2301, 2018. doi:10.1137/17M112035X.

17 Naonori Kakimura, Ken-ichi Kawarabayashi, and Dániel Marx. Packing cycles through prescribed vertices. *J. Comb. Theory, Ser. B*, 101(5):378–381, 2011. doi:10.1016/j.jctb.2011.03.004.

18 Ken-ichi Kawarabayashi and Bruce A. Reed. Odd cycle packing. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 695–704. ACM, 2010. doi:10.1145/1806689.1806785.

19 Ken-ichi Kawarabayashi, Bruce A. Reed, and Paul Wollan. The graph minor algorithm with parity conditions. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 27–36. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.52.

20 Eun Jung Kim and O-joung Kwon. Erdős-pósa property of chordless cycles and its applications. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1665–1684. SIAM, 2018. doi:10.1137/1.9781611975031.109.

21 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

22 Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010. doi:10.1007/s00453-008-9233-8.

23 M. Pontecorvi and Paul Wollan. Disjoint cycles intersecting a set of vertices. *J. Comb. Theory, Ser. B*, 102(5):1134–1141, 2012. doi:10.1016/j.jctb.2012.05.004.

24 Bruce A. Reed. Mangoes and blueberries. *Combinatorica*, 19(2):267–296, 1999. doi:10.1007/s004930050056.

25 Bruce A. Reed, Neil Robertson, Paul D. Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996. doi:10.1007/BF01271272.

26 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

27 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. doi:10.1137/0205021.

28 Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM J. Discrete Math.*, 24(1):146–157, 2010. doi:10.1137/070697781.

29 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.

30 Mihalis Yannakakis. Edge-deletion problems. *SIAM J. Comput.*, 10(2):297–309, 1981. doi:10.1137/0210021.

31 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981. doi:10.1137/0210022.

Incompressibility of H-Free Edge Modification Problems: Towards a Dichotomy

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
marx@cispa.saarland

R. B. Sandeep
Department of Computer Science and Engineering, Indian Institute of Technology Dharwad, India
sandeeprb@iitdh.ac.in

Abstract

Given a graph G and an integer k, the H-free Edge Editing problem is to find whether there exist at most k pairs of vertices in G such that changing the adjacency of the pairs in G results in a graph without any induced copy of H. The existence of polynomial kernels for H-free Edge Editing (that is, whether it is possible to reduce the size of the instance to $k^{O(1)}$ in polynomial time) received significant attention in the parameterized complexity literature. Nontrivial polynomial kernels are known to exist for some graphs H with at most 4 vertices (e.g., path on 3 or 4 vertices, diamond, paw), but starting from 5 vertices, polynomial kernels are known only if H is either complete or empty. This suggests the conjecture that there is no other H with at least 5 vertices were H-free Edge Editing admits a polynomial kernel. Towards this goal, we obtain a set $\mathcal{H}$ of nine 5-vertex graphs such that if for every $H \in \mathcal{H}$, H-free Edge Editing is incompressible and the complexity assumption $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$ holds, then H-free Edge Editing is incompressible for every graph H with at least five vertices that is neither complete nor empty. That is, proving incompressibility for these nine graphs would give a complete classification of the kernelization complexity of H-free Edge Editing for every H with at least 5 vertices.

We obtain similar result also for H-free Edge Deletion. Here the picture is more complicated due to the existence of another infinite family of graphs H where the problem is trivial (graphs with exactly one edge). We obtain a larger set $\mathcal{H}$ of nineteen graphs whose incompressibility would give a complete classification of the kernelization complexity of H-free Edge Deletion for every graph H with at least 5 vertices. Analogous results follow also for the H-free Edge Completion problem by simple complementation.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases incompressibility, edge modification problems, H-free graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.72

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.11761`.

Funding Research supported by the European Research Council (ERC) grant SYSTEMATIC-GRAPH: "Systematic mapping of the complexity landscape of hard algorithmic graph problems", reference 725978.
R. B. Sandeep: Partially supported by SERB Grant SRG/2019/002276: "Complexity Dichotomies for Graph Modification Problems".

1 Introduction

In a typical graph modification problem, the input is a graph G and an integer k, and the task is to make at most k allowed editing operations on G to make it belong to a certain graph class or satisfy a certain property. For example, Vertex Cover (remove k vertices to make the graph edgeless), Feedback Vertex Set (remove k vertices to make the graph acyclic), Odd Cycle Transversal (remove k edges/vertices to make the graph bipartite),

and Minimum Fill-in (add k edges to make the graph chordal) are particularly well-studied members of this problem family. Most natural graph modification problems are known to be NP-hard, in fact, there are general hardness results proving hardness for many problems [19, 21, 22]. On the other hand, most of these problems are fixed-parameter tractable (FPT) parameterized by k: it can be solved in time $f(k)n^{O(1)}$, where f is a computable function depending only on k [3, 8, 17, 18]. Looking at the parameterized complexity literature, one can observe that, even though there are certain recurring approaches and techniques, these FPT results are highly problem specific, and often rely on a very detailed understanding of the graph classes at hand.

A class of problems that can be treated somewhat more uniformly is H-free Edge Editing. This is a separate problem for every fixed graph H: given a graph G and an integer k, the task is to find whether there exist at most k pairs of vertices in G such that changing the adjacency of the pairs in G results in a graph without any induced copy of H. Aravind et al. [2] proved that H-free Edge Editing is NP-hard for every graph H with at least 3 vertices. However, a simple application of the technique of bounded-depth search trees shows that H-free Edge Editing is FPT parameterized by k for every fixed H [3].

Graph modification problems were explored also from the viewpoint of polynomial kernelization: is there a polynomial-time preprocessing algorithm that does not necessarily solve the problem, but at least reduces the size of the an instance to be bounded by a polynomial of k? The existence of a polynomial kernelization immediately implies that the problem is FPT (after the preprocessing, one can solve the reduced instance by brute force or any exact method). Therefore, one can view polynomial kernelization as a special type of FPT result that tries to formalize the question whether the problem can be efficiently preprocessed in a way that helps exhaustive search methods. There is a wide literature on algorithms for kernelization (see, e.g., [14]). Conversely, incompressibility results can show, typically under the complexity assumption $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, that a parameterized problem has no polynomial kernelization.

Most of the highly nontrivial FPT algorithms for graph modification problems do not give kernelization results and, in many cases, it required significant amount of additional work to obtain kernelization algorithms. In particular, the FPT algorithm for H-free Edge Editing based on the technique of bounded-depth search trees does not give polynomial kernels. For the specific case when $H = K_r$ is a complete graph, it is easy to see that there is a solution using only deletions. Now the problem essentially becomes a Hitting Set problem with sets of bounded size: we have to select at least one edge from the edge set of each copy of K_r. Therefore, known kernelization results for Hitting Set can be used to show that K_r-free Edge Editing has a polynomial kernel for every fixed r. A similar argument works if H is an empty graph on r vertices.

Besides cliques and empty graphs, it is known for certain graphs H of at most 4 vertices (diamond [5, 9], path [6, 15, 16], paw [7, 13], and their complements) that H-free Edge Editing has a polynomial kernel, but these algorithms use very specific arguments exploiting the structure of H-free graphs. As there is a very deep known structure theory of claw-free (i.e, $K_{1,3}$-free) graphs, it might be possible to obtain a polynomial kernel for Claw-free Edge Editing, but this is currently a major open question [4, 10, 12]. However, besides cliques and empty graphs, no H with at least 5 vertices is known where H-free Edge Editing has a polynomial kernel and there is no obvious candidate H for which one would expect a kernel. This suggests the following conjecture:

▶ **Conjecture 1.** *If H is a graph with at least 5 vertices, then H-free Edge Editing has a polynomial kernel if and only if H is a complete or empty graph.*

We are not able to resolve this conjecture, but make substantial progress towards it by showing that only a finite number of key cases needs to be understood. Our main result for H-free Edge Editing is the following.

▶ **Theorem 1.** *There exists a set $\mathcal{H}^E$ of nine graphs, each with five vertices such that if H-free Edge Editing is incompressible for every $H \in \mathcal{H}^E$, then for a graph H with at least five vertices H-free Edge Editing is incompressible if and only if H is neither complete nor empty, where the incompressibility assumes* NP $\not\subseteq$ coNP/poly.

The set $\mathcal{H}^E$ of nine graphs are shown in Figure 1a. Note that a simple reduction by complementation shows that H-free Edge Editing and $\overline{H}$-free Edge Editing have the same complexity. Therefore, for each of these nine graphs, we could put either it or its complement into the set $\mathcal{H}^E$. As it will be apparent later, we made significant efforts to reduce the size of $\mathcal{H}^E$ as much as possible. However, the known techniques for proving incompressibility do not seem to work for these graphs. Let us observe that most of these graphs are very close to the known cases that admit a polynomial kernel: for example, they can be seen as a path, paw, or diamond with an extra isolated vertex or with an extra degree-1 vertex attached. Thus resolving the kernelization complexity of H-free Edge Editing for any of these remaining graphs seems to be a particularly good research question: either one needs to extend in a nontrivial way the known kernelization results, or a significant new ideas are needed for proving hardness.

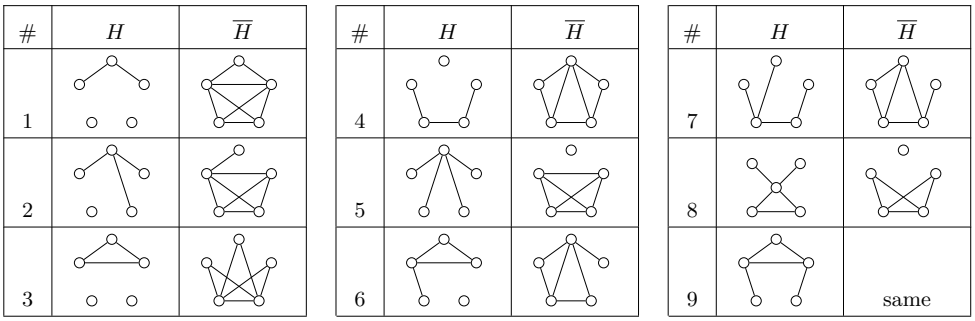
The reader might not be convinced of the validity of Conjecture 1 and may wonder about the value of Theorem 1 when the conjecture is false. However, we can argue that Theorem 1 is meaningful even in this case. It shows that if there is any H violating Conjecture 1, then one of the 9 graphs in $\mathcal{H}^E$ also violates it. That is, if we believe that there are kernelization results violating the conjecture, then we should focus on the 9 graphs in $\mathcal{H}^E$, as these are the easiest cases where we may have a kernelization result. In other words, Theorem 1 precisely shows the frontier where new algorithmic results are most likely to exist.

H-free Edge Deletion is the variant of H-free Edge Editing where only edge removal is allowed. For the same fixed graph H, it seems that H-free Edge Deletion should be a simpler problem than H-free Edge Editing, but we want to emphasize that H-free Edge Deletion is *not* a special case of H-free Edge Editing. There is no known general reduction from the former to the latter, although the technique of completion enforcers (see Section 5 and [4]) can be used for many specific graphs H. There is a known case where H-free Edge Deletion seems to be strictly easier: if H has at most one edge, then there is only one way of destroying a copy of an induced H by edge removal, making the problem polynomial-time solvable. Aravind et al. [1] showed that having at most one edge is the only condition that makes H-free Edge Deletion polynomial-time solvable: if H has at least two edges, then the problem is NP-hard. Therefore, the counterpart of Conjecture 1 for H-free Edge Deletion should take this case also into account.

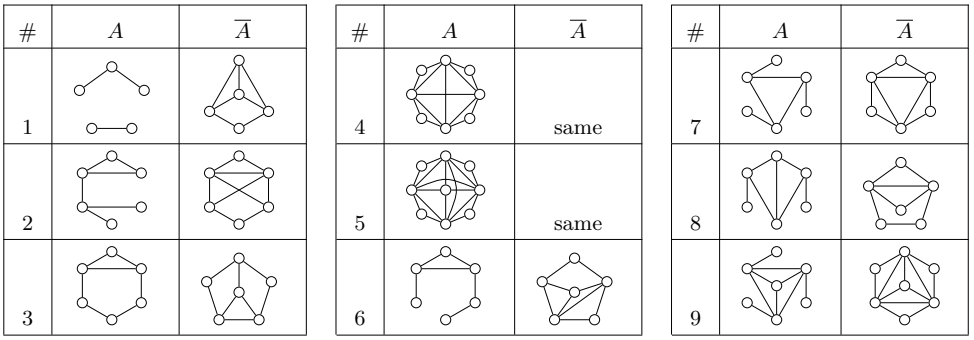
▶ **Conjecture 2.** *If H is a graph with at least 5 vertices, then H-free Edge Deletion has a polynomial kernel if and only if H is a complete graph or has at most one edge.*

Working toward this conjecture, we show that only a finite number of cases needs to be shown incompressible.

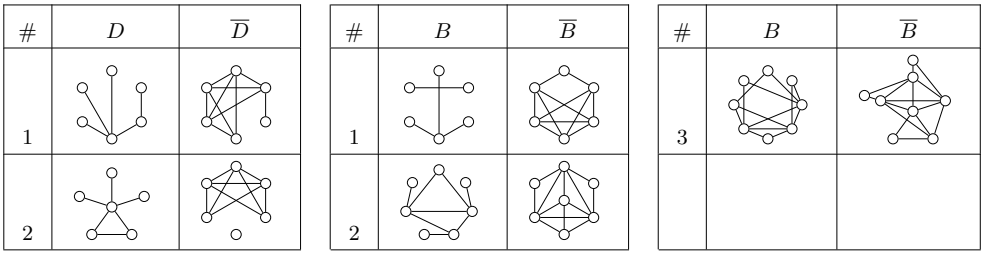
▶ **Theorem 2.** *There exists a set $\mathcal{H}^D$ of nineteen graphs, each with either five or six vertices such that if H-free Edge Deletion is incompressible for every $H \in \mathcal{H}^D$ then for a graph H with at least five vertices, H-free Edge Deletion is incompressible if and only if H is a graph with at least two edges but not complete, where the incompressibility assumes* NP $\not\subseteq$ coNP/poly.



(a) The set $\mathcal{H}$ of graphs.

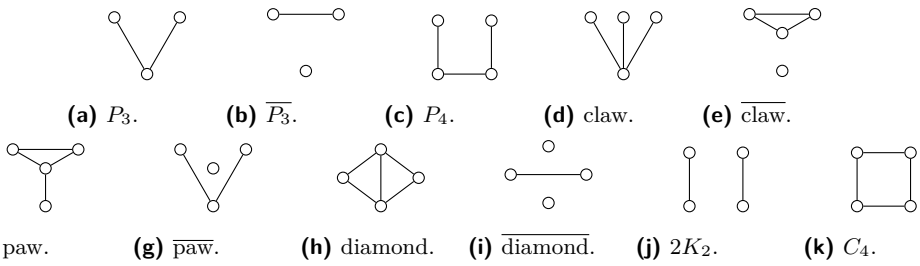


(b) The set $\mathcal{A}$ of graphs.



(c) The sets $\mathcal{D}$ and $\mathcal{B}$ of graphs.

Figure 1



(a) P_3. **(b)** $\overline{P_3}$. **(c)** P_4. **(d)** claw. **(e)** $\overline{\text{claw}}$.

(f) paw. **(g)** $\overline{\text{paw}}$. **(h)** diamond. **(i)** $\overline{\text{diamond}}$. **(j)** $2K_2$. **(k)** C_4.

Figure 2 All non-empty and non-complete graphs with at most four vertices.

The set $\mathcal{H}^D$ contains the graphs in set $\mathcal{H}^E$, as well their complements. This seems reasonable and hard to avoid: if we do not have an incompressibility result for H-free Edge Editing for some $H \in \mathcal{H}^E$, then it is unlikely that we can find such a result for H-free Edge Deletion (even though, as discussed above, there is no formal justification for this). Together with these 17 graphs (note that H_9 is the same as its complement), we need to include into

$\mathcal{H}^D$ the two graphs D_1 and D_2 shown in Figure 1c. In the case of editing, we can prove incompressibility for these two graphs by a reduction from H-free Edge Editing where H is the graph with 5 vertices and one edge. However, H-free Edge Deletion for this H is polynomial-time solvable.

Finally, let us consider the H-free Edge Completion problem, where we have to make G induced H-free by adding at most k edges. As H-free Edge Completion is essentially the same problem as $\overline{H}$-free Edge Deletion, we can obtain a counterpart of Theorem 2 by simple complementation:

▶ **Theorem 3.** *There exists a set $\mathcal{H}^C$ of nineteen graphs, each with either five or six vertices such that if H-free Edge Completion is incompressible for every $H \in \mathcal{H}^C$ then for a graph H with at least five vertices, H-free Edge Completion is incompressible if and only if H is a graph with at least two nonedges but not empty, where the incompressibility assumes* $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$.

Our techniques. We crucially use two earlier results. First, Cai and Cai [4] proved that H-free Edge Editing is incompressible (assuming $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$) when H or $\overline{H}$ is a cycle or a path of length at least 4, or 3-connected but not complete. While these result handle many graphs and prove to be very useful for our proofs, they do not come close to a complete classification. Second, we use a key tool in the polynomial-time dichotomy result of Aravind et al. [1]: if V_ℓ is the set of lowest degree vertices of H, then $(H - V_\ell)$-free Edge Editing can be reduced to H-free Edge Editing. The same statement holds for the set V_h of highest degree vertices.

Our proofs of Theorems 1–3 introduce new incompressibility results and new reductions, which we put together to obtain an almost complete classification by a graph-theoretic analysis. Additionally, to make the arguments simpler, we handle small graphs by an exhaustive computer search. In the following, we highlight some of the main ideas that appear in the paper.

- **Analysis of graphs.** Our goal is to prove Theorem 1 by induction on the size of H. First we handle the case when H is regular: we show that this typically implies that either H or $\overline{H}$ is 3-connected, and the result of Cai and Cai [4] can be used. If H is not regular, then the graphs $H - V_\ell$ and $H - V_h$ are nonempty and have stricly fewer vertices than H. If one of them, say $H - V_\ell$, has at least 5 vertices and is neither complete nor empty, then the induction hypothesis gives an incompressibility result for $(H - V_\ell)$-free Edge Editing, which gives an incompressibility result for H-free Edge Editing by the reduction of Aravind et al. [1]. Therefore, we only need to handle those graphs H where it is true for both $H - V_\ell$ and $H - V_h$ that they are either small, complete, or empty. But we can obtain a good structural understanding of H in each of these cases, which allows us to show that either H or $\overline{H}$ is 3-connected, or H has some very well defined structure. With these arguments, we can reduce the problem to the incompressibility of H-free Edge Editing for a few dozen specific graphs H and for a few well-structured infinite families (such as $K_{2,t}$).
  For H-free Edge Deletion, we have the additional complication that one or both of $H - V_\ell$ and $H - V_h$ can be near-empty (i.e., has exactly one edge), which is not an incompressible case for this problem. We need additional case analysis to cover such graphs, but the spirit of the proof remains the same.
- **Computer search.** Our analysis of graphs becomes considerably simpler if we assume that H is not too small. In this case, we can assume that at least one of $H - V_\ell$ and $H - V_h$ is a complete or empty graph of certain minimum size, which is a very helpful

starting point for proving the 3-connectivity of H or $\bar{H}$, respectively. Therefore, we handle every graph with at most 9 vertices using an exhaustive computer search and assume in the proof that H has at least 10 vertices. The list provided by McKay [20] shows that there are 288266 different graphs with at most 9 vertices, which is feasible for a computer search. In principle, it would be possible to extend our case analysis to avoid this computer search, but it would significantly complicate the proof and is not clear what additional insight it would give.

- **Reductions.** We investigate different reductions that allow us to reduce H'-free Edge Editing to H-free Edge Editing when H' is an induced subgraph of H satisfying certain conditions. With extensive use of such reductions, we can reduce the remaining cases of H-free Edge Editing that needs to be handled to a smaller finite set.
- **Incompressibility results.** We carefully revisit the proof of Cai and Cai [4] showing the incompressibility of H-free Edge Editing when H is 3-connected, and observe that, with additional ideas, it can be made to work also for certain 2-connected graphs that are not 3-connected (the set $\mathcal{A}$ of graphs shown in Figure 1b and the set $\mathcal{B}$ of graphs shown in Figure 1c). This allows us to handle every graph, except those finite sets that are mentioned in Theorems 1–3. A key step in many of these incompressibility results is to establish first incompressibility for the Restricted H-free Edge Deletion problem, which is the generalization of H-free Edge Deletion where some of the edges of G are marked as forbidden in the input, and the solution is not allowed to delete forbidden edges. Then we use deletion and completion enforcer gadgets specific to H to reduce Restricted H-free Edge Deletion to H-free Edge Editing.

The paper is organized as follows. Preliminaries are in Section 2. Section 3 presents the churning procedure, our main technical tool in the analysis of graphs, and shows that it reduces the problem to a finite number of graphs, plus a few well-defined infinite families. Section 4 presents reductions (old and new) that allow us to further reduce the number of graphs we need to handle. Finally, in Setion 5, we give new incompressibility results, showing that only the cases stated in Theorems 1–3 need to be proved incompressible to complete the exploration of the complexity landscape of the problems. All proofs have been moved to a full version of the paper due to space constraints.

2 Preliminaries

Graph-theoretic notation and terminology. For a graph G, $V(G)$ and $E(G)$ denote the set of vertices and the set of edges of G respectively. For a set $V' \subseteq V(G)$, $G - V'$ denotes the graph obtained by removing all vertices in V' and their incident edges from G. For a set F of pairs of vertices and a graph G, $G \triangle F$ denotes the graph G' such that $V(G') = V(G)$ and $E(G') = \{(u,v) \mid ((u,v) \in E(G) \text{ and } (u,v) \notin F) \text{ or } (u,v \in V(G),\ (u,v) \notin E(G), \text{ and } (u,v) \in F)\}$. Whenever we say that a set of (non)edges F is a solution of an instance (G,k) of a problem, we refers to a subset of F containing all (non)edges where both the end vertices are in $V(G)$. A graph is *empty* if it does not have any edges. A graph is *near-empty* if it has exactly one edge. A graph is *complete* if it has no nonedges. A component of a graph is a *largest component* if it has maximum number of vertices among all components of the graph. Similarly, a component of a graph is a *smallest component* if it has minimum number of vertices among all components of the graph. For a graph H which is not complete, the vertex connectivity of H is the minimum integer c such that there exists a set $S \subseteq V(H)$ such that $|S| = c$ and $H - S$ is disconnected. For a graph H with vertex connectivity 1, a vertex v in H is known as a cut vertex if $H - v$ is disconnected. A graph is k-connected, if

the vertex connectivity of it is at least k. An induced subgraph H' of H is known as a 2-connected component if H' is a maximal 2-connected induced subgraph of H. The adjectives "largest" and "smallest" can be applied to 2-connected components as done for components. A *twin-star* graph T_{ℓ_1,ℓ_2} for $\ell_1, \ell_2 \geq 0$ is defined as the tree with two adjacent vertices u and v such that $|N(u) \setminus \{v\}| = \ell_1, |N(v) \setminus \{u\}| = \ell_2$, and every vertex in $N(u) \cup N(v) \setminus \{u, v\}$ has degree 1. A graph G is H-free if G does not contain any induced copy of H. For two graphs G_1 and G_2, the disjoint union of G_1 and G_2 denoted by $G_1 \cup G_2$ (or $G_2 \cup G_1$) is the graph G such that $V(G) = V(G_1) \cup V(G_2)$ and $E(G) = E(G_1) \cup E(G_2)$. For two graphs G_1 and G_2, the join of G_1 and G_2 denoted by $G_1 \boxtimes G_2$ (or $G_2 \boxtimes G_1$), is the graph G such that $V(G) = V(G_1) \cup V(G_2)$ and $E(G) = E(G_1) \cup E(G_2) \cup \{(x, y) \mid x \in V(G_1), y \in V(G_2)\}$. A complete graph, a cycle, and a path with t vertices are denoted by K_t, C_t, and P_t respectively. By $K_t - e$, we denote the graph obtained by deleting an edge from a complete graph on t vertices. We call a graph *non-regular* if it is not regular. A *modular decomposition* $\mathcal{M}$ of a graph G is a partitioning of its vertices into maximal sets, known as *modules*, such that for every set $M \in \mathcal{M}$, every vertex in M has the same neighborhood outside M. Let $\mathcal{M}' \subseteq \mathcal{M}$. Let $V' = \bigcup_{M \in \mathcal{M}'} M$. Then we say that $\mathcal{M}'$ corresponds to V'. For a set $\mathcal{S}$ of graphs, by $\overline{\mathcal{S}}$ we denote the set of complements of graphs in $\mathcal{S}$. Figure 2 shows all graphs with at most four vertices which are neither empty nor complete.

For $t \geq 3$, let J_t be the graph obtained from $K_2 \boxtimes tK_1$ and C_4 by identifying an edge of C_4 with the edge between the highest degree vertices in $K_2 \boxtimes tK_1$. Let Q_t be the graph graph obtained from $K_{2,t}$, for some $t \geq 3$, by adding a path of length three between the highest degree vertices in $K_{2,t}$. Let $\mathcal{H}$, $\mathcal{A}$, $\mathcal{D}$, $\mathcal{B}$, $\mathcal{S}$ denote the graphs (H, A, D, B, S respectively) shown in Figures 1a, 1b, 1c, and 3a. Let $\mathcal{F}$ be the union of graphs in the classes of graphs shown in column $\mathcal{F}$ of Figure 3b. The graphs in $\mathcal{S}$ and $\mathcal{F}$ are handled in Section 4 and the graphs in $\mathcal{A}$ and $\mathcal{B}$ are handled in Section 5. For all these classes of graphs, we use subscripts to identify each graph/graph class. For example H_1 is $P_3 \cup 2K_1$ and $\mathcal{F}_1$ is the class of graphs $K_{1,t}$. Let $\mathcal{W}$ be the set $\mathcal{H} \cup \overline{\mathcal{H}} \cup \mathcal{A} \cup \overline{\mathcal{A}} \cup \mathcal{D} \cup \overline{\mathcal{D}} \cup \mathcal{B} \cup \overline{\mathcal{B}} \cup \mathcal{S} \cup \overline{\mathcal{S}} \cup \mathcal{F} \cup \overline{\mathcal{F}}$. We observe that $\overline{\mathcal{W}} = \mathcal{W}$.

Parameterized problems and transformations. A parameterized problem is a classical problem with an additional integer input known as the parameter. A parameterized problem admits a polynomial kernel if there is a polynomial-time algorithm which takes as input an instance (I, k) of the problem and outputs an instance (I', k') of the same problem, where $|I'|, k' \leq p(k)$, where $p(k)$ is a polynomial in k, such that (I, k) is a yes-instance if and only if (I', k') is a yes-instance. A parameterized problem is incompressible if it does not admit a polynomial kernel. A *Polynomial Parameter Transformation* (PPT) from one parameterized problem Q to another parameterized problem Q' is a polynomial-time algorithm which takes as input an instance (I, k) of Q and produces an instance (I', k') of Q' such that (I, k) is a yes-instance of Q if and only if (I', k') is a yes-instance of Q', and $k' \leq p(k)$, for some polynomial $p(.)$. It is known that if there is a PPT from Q to Q', then if Q is incompressible, then so is Q'. We refer to the book [11] for various concepts in parameterized algorithms and complexity.

The parameterized problems we deal with in this paper are listed below.

H-free Edge Editing: Given a graph G and an integer k, do there exist at most k edges such that editing (adding or deleting) them in G results in an H-free graph?
Parameter: k

H-free Edge Deletion: Given a graph G and an integer k, do there exist at most k edges such that, deleting them from G results in an H-free graph? **Parameter**: k

H-free Edge Completion: Given a graph G and an integer k, do there exist at most k edges such that, adding them in G results in an H-free graph? **Parameter**: k

Basic results. Proposition 4 follows from the observations that (G, k) is a yes-instance of H-free Edge Editing(Deletion) if and only if $(\overline{G}, k)$ is a yes-instance of $\overline{H}$-free Edge Editing(Completion). It enables us to focus only on H-free Edge Editing and H-free Edge Deletion.

▶ **Proposition 4** (folklore). *Let H be any graph. Then H-free Edge Deletion is incompressible if and only if $\overline{H}$-free Edge Completion is incompressible. Similarly, H-free Edge Editing is incompressible if and only if $\overline{H}$-free Edge Editing is incompressible.*

For graphs H and H', by "*H simulates H'*" and by "*H' is simulated by H*", we mean that, there is a PPT from H'-free Edge Editing to H-free Edge Editing, there is a PPT from H'-free Edge Deletion to H-free Edge Deletion, and there is a PPT from H'-free Edge Completion to H-free Edge Completion. We observe that this is transitive, i.e., if H simulates H' and H' simulates H'', then H simulates H''. A set of graphs $\mathcal{H}$ is called a *base* for a set $\mathcal{G}$ of graphs if for every graph $H \in \mathcal{G}$ there is a graph $H' \in \mathcal{H}$ such that H simulates H'. The objective of the rest of the paper is to find, for each of the problems, a base $\mathcal{H} \cup \mathcal{X}$ for all graphs with at least five vertices, except the trivial cases, such that the following conditions are satisfied: (i) $\mathcal{H}$ is finite and the incompressibility is not known for any graph in it; (ii) for every graph in $\mathcal{X}$, the problem is known to be incompressible.

Proposition 4 implies Corollary 5 and Proposition 6 can be deduced directly from the definitions.

▶ **Corollary 5.** *Let H and H' be graphs such that H simulates H'. Then $\overline{H}$ simulates $\overline{H'}$.*

▶ **Proposition 6.** *Let $\mathcal{H}$ be a base for a set $\mathcal{G}$ of graphs. Assume that for every graph $H' \in \mathcal{H}$, H'-free Edge Editing (Deletion) is incompressible. Then for every graph $H \in \mathcal{G}$, H-free Edge Editing (Deletion) is incompressible.*

Intuitively, if H' is an induced subgraph of H, then H-free Edge Editing (Deletion) seems harder than H'-free Edge Editing (Deletion). However, there is no general argument why this should be true: there does not seem to be a completely general reduction that would reduce H'-free Edge Editing (Deletion) to H-free Edge Editing (Deletion). There is, however, a fairly natural idea for trying to do such a reduction: we extend the graph by attaching copies of $H - H'$ at every place where a copy of H' can potentially appear. The following construction is essentially the same as the main construction used in [2].

▶ **Construction 1** (see [2]). *Let (G', k, H, V') be an input to the construction, where G' and H are graphs, k is a positive integer and V' is a subset of vertices of H. We construct a graph G from G' as follows. For every injective function $f : V' \longrightarrow V(G')$, do the following:*

- *Introduce $k+1$ sets of vertices $V_1, V_2, \ldots, V_{k+1}$, each of size $|V(H) \setminus V'|$, and $k+1$ bijective functions $g_i : V(H) \longrightarrow (f(V') \cup V_i)$, for $1 \leq i \leq k+1$, such that $g_i(v') = f(v')$ for every $v' \in V'$;*

- *For each set V_i, introduce an edge set $E_i = \{(u,v) \mid u \in (f(V') \cup V_i), v \in V_i, (g_i^{-1}(u), g_i^{-1}(v)) \in E(H)\}$.*

This completes the construction. Let the constructed graph be G.

For convenience, we call every set V_i of vertices introduced in the construction as a *satellite* and the vertices in it as *satellite vertices*. This reduction works correctly in one direction: it ensures that the operations that make the new graph G H-free should ensure that the copy of G' inside G is H'-free.

▶ **Proposition 7** (see Lemma 2.6 in [2]). *Let G be obtained by Construction 1 on the input (G', k, H, V'), where G' and H are graphs, k is a positive integer and $V' \subseteq V(H)$. Then, if (G, k) is a yes-instance of H-free Edge Editing (Deletion), then (G', k) is a yes-instance of H'-free Edge Editing (Deletion), where H' is $H[V']$.*

However, the other direction of the correctness of the reduction does not hold in general (this is easy to see for example for $H = K_{1,2}$ and $H' = K_2$). As we shall see, there are particular cases where we can prove the converse of Proposition 7, for example, when $H - H'$ consists of exactly the highest- or lowest-degree vertices. Application of such arguments will be our main tool in reducing the complexity of H-free Edge Editing (Deletion) to simpler cases. Propositions 8 to 11 summarize the major results on the incompressibility of H-free edge modification problems known so far.

▶ **Proposition 8** ([4]). *Assuming* NP $\not\subseteq$ coNP/poly, *H-free Edge Editing, H-free Edge Deletion, and H-free Edge Completion are incompressible if H is either of the following graphs.*

- **(i)** *C_ℓ for any $\ell \geq 4$;*
- **(ii)** *P_ℓ for any $\ell \geq 5$;*
- **(iii)** *$2K_2$.*

We observe that Proposition 8(iii) follows from Proposition 8(i) and Proposition 4.

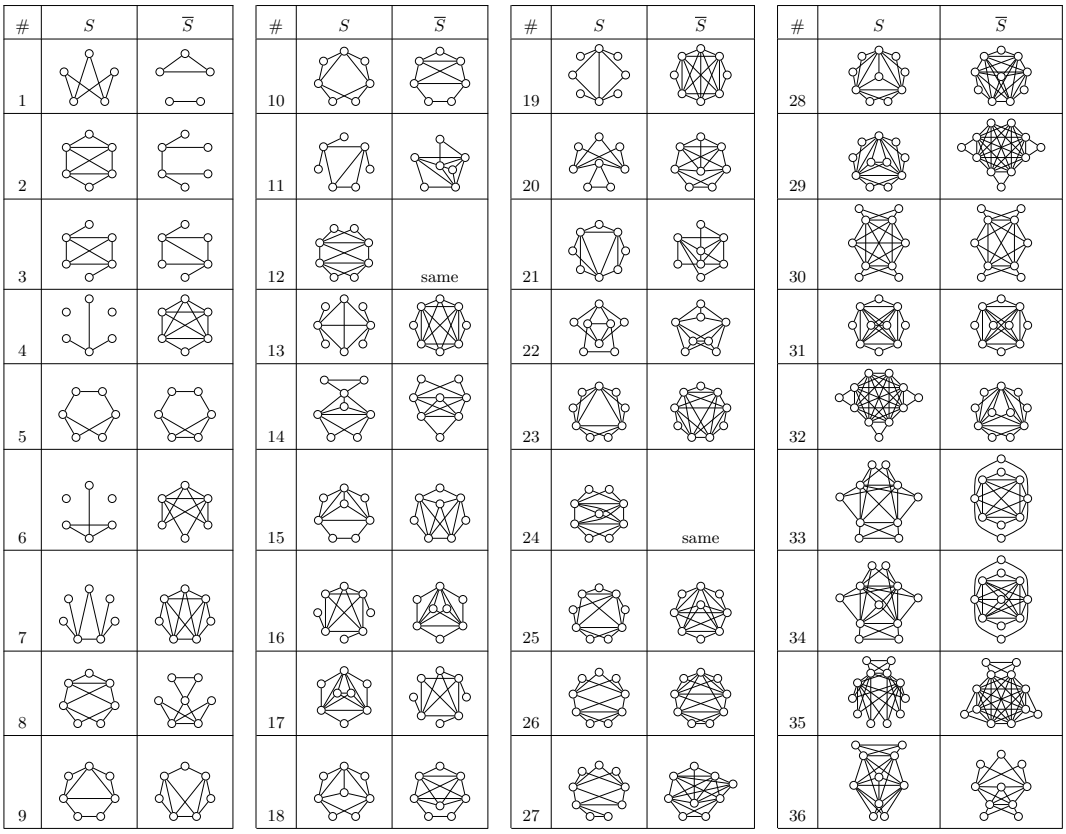
▶ **Proposition 9** ([4]). *Assuming* NP $\not\subseteq$ coNP/poly, *for 3-connected graphs H, H-free Edge Editing and H-free Edge Deletion are incompressible if H is not complete and H-free Edge Completion is incompressible if H has at least two nonedges.*

▶ **Proposition 10** ([4], folklore). *If H is a complete or empty graph, then H-free Edge Editing admits polynomial kernelization. If H is complete or has at most one edge then H-free Edge Deletion admits polynomial kernelization. If H is an empty graph or has at most one nonedge then H-free Edge Completion admits polynomial kernelization.*

▶ **Proposition 11.** *H-free Edge Editing, H-free Edge Deletion, and H-free Edge Completion admit polynomial kernels when H is a P_3 [6, 15], P_4 [16], paw [7, 13], or a diamond [5, 9].*

3 Churning

In this section, we introduce and analyze the churning procedure. The main result of the section is that assuming incompressibility for the class $\mathcal{W}$ of graphs defined in the previous section explains incompressibility for every graph with at least five vertices, except the trivial cases. Recall that $\mathcal{W}$ is not finite, as it contains the infinite families shown in Figure 3b. In Sections 4 and 5, we will further reduce $\mathcal{W}$ to a finite set.



(a) The set $\mathcal{S}$ of graphs.

#	$\mathcal{F}$	$\overline{\mathcal{F}}$	Comment
1	$K_{2,t}$	$K_t \cup K_2$	$4 \leq t$
2	$K_{1,t}$	$K_t \cup K_1$	$5 \leq t$
3	$K_2 \boxtimes tK_1$	$K_t \cup 2K_1$	$4 \leq t$
4	$T_{t,1}$	$\overline{T_{t,1}}$	$4 \leq t$
5	$\overline{(K_t - e) \cup 2K_1}$	$(K_t - e) \cup 2K_1$	$4 \leq t$
6	$\overline{(K_t - e) \cup K_2}$	$(K_t - e) \cup K_2$	$4 \leq t$
7	$K_{1,t} \cup K_2$	$\overline{K_{1,t} \cup K_2}$	$4 \leq t$
8	$\overline{(K_t - e) \cup K_1}$	$(K_t - e) \cup K_1$	$6 \leq t$
9	J_t	$\overline{J_t}$	$3 \leq t$
10	Q_t	$\overline{Q_t}$	$3 \leq t$

(b) The set $\mathcal{F}$ of infinite sets of graphs.

Figure 3

▶ **Lemma 12.** *If H-free Edge Editing is incompressible for every $H \in \mathcal{W}$, then H-free Edge Editing is incompressible for every H having at least five vertices but is neither complete nor empty, where the incompressibility assumes* NP $\not\subseteq$ coNP/poly.

▶ **Lemma 13.** *If H-free Edge Deletion is incompressible for every $H \in \mathcal{W}$, then H-free Edge Deletion is incompressible for every H having at least five vertices and at least two edges but not complete, where the incompressibility assumes* NP $\not\subseteq$ coNP/poly.

Corollary 14 follows from Lemma 13, Proposition 4 and from the fact that $\overline{\mathcal{W}} = \mathcal{W}$.

▶ **Corollary 14.** *If H-free Edge Completion is incompressible for every $H \in \mathcal{W}$, then H-free Edge Completion is incompressible for every H having at least five vertices and at least two nonedges but not empty, where the incompressibility assumes* NP $\not\subseteq$ coNP/poly.

By $\mathcal{X}_E$ we denote the set of all graphs (and their complements) listed in Proposition 8, Proposition 9, and Theorem 18 for which the incompressibility is known (assuming $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$) for H-free Edge Editing. By $\mathcal{Y}_E$, we denote the set of all graphs (and their complements) listed in Proposition 10 and 11 for which there exist polynomial kernels for H-free Edge Editing; additionally, we include into $\mathcal{Y}_E$ the claw and its complement (as we do not want to conjecture the incompressibility for these cases). Similarly, we define the set $\mathcal{X}_D$ of "hard" and the set $\mathcal{Y}_D$ of "nonhard" cases for H-free Edge Deletion. More formally,

$\mathcal{X}_D = \{C_\ell, \overline{C_\ell}$ for all $\ell \geq 4$,
$P_\ell, \overline{P_\ell}$ for all $\ell \geq 5$,
H such that H is regular but is neither complete nor empty,
H such that either H is 3-connected but not complete
or $\overline{H}$ is 3-connected with at least two nonedges$\}$

$\mathcal{X}_E = \mathcal{X}_D \cup \{H$ such that H has at most one edge and at least five vertices$\}$

$\mathcal{Y}_E = \{K_t, \overline{K_t}$ for all $t \geq 1$,
$P_3, \overline{P_3}, P_4$,
diamond, $\overline{\text{diamond}}$, paw, $\overline{\text{paw}}$, claw, $\overline{\text{claw}}\}$

$\mathcal{Y}_D = \mathcal{Y}_E \cup \{H$ such that H has at most one edge and at least five vertices$\}$

Additionally we define $\mathcal{Y}' = \{P_3, \overline{P_3}, P_4, \text{claw}, \overline{\text{claw}}, \text{paw}, \overline{\text{paw}}, \text{diamond}, \overline{\text{diamond}}\}$. We observe that $\mathcal{Y}' \subseteq \mathcal{Y}_E \cap \mathcal{Y}_D$ and the set of graphs with at most four vertices is a subset of $\mathcal{X}_E \cup \mathcal{Y}_E$ and $\mathcal{X}_D \cup \mathcal{Y}_D$. Further, we observe that near-empty graphs with at least five vertices are in $\mathcal{Y}_D$ but their complements are 3-connected and are in $\mathcal{X}_D$. We also note that both these graphs and their complements are in $\mathcal{X}_E$.

The main technical result of the section is the following lemma. It states that if a graph is not in the set $\mathcal{Y}_D$ of "easy" graphs, then it simulates a "hard" graph in $\mathcal{X}_\mathcal{D}$ or $\mathcal{W}$ (and there is a similar result for $\mathcal{X}_E$ and $\mathcal{Y}_E$).

▶ **Lemma 15.** *If $H \notin \mathcal{Y}_D$, then H simulates a graph in $\mathcal{X}_D \cup \mathcal{W}$. If $H \notin \mathcal{Y}_E$, then H simulates a graph in $\mathcal{X}_E \cup \mathcal{W}$.*

In the rest of the paper, integer ℓ and set V_ℓ denote the lowest degree and the set of lowest degree vertices in H respectively; integer h and set V_h denote the highest degree and the set of highest degree vertices in H respectively; and set V_m denotes the set $V(H) \setminus (V_\ell \cup V_h)$. By h^* we denote the degree of vertices of V_h in $\overline{H}$, i.e., $h^* = |V(H)| - h - 1$.

Now we introduce a procedure (Churn) which is similar to the one used to obtain dichotomy results on the polynomial-time solvable and NP-hard cases of these problems (see Section 5 in [2]). The basic observation is that H can simulate the graphs $H - V_\ell$ and $H - V_h$. This follows from proving that Construction 1 gives a PPT in these cases.

▶ **Proposition 16** (Corollary 2.9 in [2]). *Let H' be $H - V_\ell$ or $H - V_h$. Then H simulates H'.*

To deal with both H-free Edge Editing and H-free Edge Deletion in a uniform way, we define $\mathcal{X} = \mathcal{X}_E$ and $\mathcal{Y} = \mathcal{Y}_D$. We observe that $\overline{\mathcal{X} \cup \mathcal{Y}} = \mathcal{X} \cup \mathcal{Y}$ and $\mathcal{X} \cup \mathcal{Y} = \mathcal{X}_E \cup \mathcal{Y}_E = \mathcal{X}_D \cup \mathcal{Y}_D$.

Churn(H):
Step 1: If H is regular, then return H.
Step 2: If $H - V_\ell \notin \mathcal{Y}$, then return Churn($H - V_\ell$).
Step 3: If $H - V_h \notin \mathcal{Y}$, then return Churn($H - V_h$).
Step 4: Return H.

Proposition 16 implies Corollary 17.

▶ **Corollary 17.** *Let H' be the output of Churn(H). Then H simulates H'.*

We prove Lemma 15 by analyzing Churn() and showing that the graph returned by it always satisfies the requirements of the lemma. The procedure first handles the case when H is regular. In Section 3.1, we show that if H is regular, then it is safe to return H, as it is already in $\mathcal{X}_D \subseteq \mathcal{X}_E$. If H is not regular, then $H - V_\ell$ and $H - V_h$ are both defined. If one of these two graphs is not in $\mathcal{Y}$, then Proposition 16 allows us to proceed by recursion on that graph. Step 4 is reached when both $H - V_\ell$ and $H - V_h$ are in $\mathcal{Y}$. However, at this point the conditions on $H - V_\ell$ and $H - V_h$ give us important structural information about the graph H, which can be exploited to show that it is in $\mathcal{X}_D \cup \mathcal{W}$. Recall that $\mathcal{Y}$ is the union of complete, empty, near-empty, and the finite graphs in $\mathcal{Y}'$. This means we can split the problem into $4 \cdot 4$ different cases, with very strict structural restrictions on H in each case. These cases are analysed in a sequence of lemmas/corollaries (Lemma 19 to Lemma 34 in Sections 3.2–3.5).

3.1 Regular graphs

In this section, we handle the case when H is regular.

▶ **Theorem 18.** *Let H be a regular graph. Then H-free Edge Deletion, H-free Edge Completion, and H-free Edge Editing are incompressible if and only if H is neither complete nor empty, where the incompressibility assumes* NP $\not\subseteq$ coNP/poly.

3.2 Small graphs

If both $H - V_\ell$ and $H - V_h$ are in the in the finite set $\mathcal{Y}'$ of graphs, then H has bounded size. An exhaustive computer search showed the correctness of the procedure in this case.

▶ **Lemma 19.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that both $H - V_\ell$ and $H - V_h$ are in $\mathcal{Y}'$. Then $H \in \mathcal{W}$.*

3.3 Cliques and empty graphs

In this section, we consider the cases when both $H - V_\ell$ and $H - V_h$ are cliques or empty graphs. In this case, the structure of H is very limited. In principle, we need to consider four cases separately depending on the type of $H - V_\ell$ and $H - V_h$. However, a simple complementation argument shows that the case when both of them are cliques is equivalent to the case when both of them are empty.

▶ **Lemma 20.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that both $H - V_\ell$ and $H - V_h$ are complete graphs. Then $H \in \mathcal{W}$.*

Corollaries in this section and in Sections 3.4 and 3.5 use the facts that various sets we consider are self-complementary, i.e., $\mathcal{X} \cup \mathcal{Y} = \overline{\mathcal{X} \cup \mathcal{Y}}, \mathcal{W} = \overline{\mathcal{W}}, \mathcal{Y}' = \overline{\mathcal{Y}'}$.

▶ **Corollary 21.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that both $H - V_\ell$ and $H - V_h$ are empty graphs. Then $H \in \mathcal{W}$.*

▶ **Lemma 22.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that $H - V_\ell$ is a complete graph and $H - V_h$ is an empty graph. Then $H \in \mathcal{W}$.*

Our last case is when $H - V_\ell$ is empty and $H - V_h$ is complete. Let us observe that this case *does not* follow from Lemma 22 by complementation. If $\overline{V_\ell}$ and $\overline{V_h}$ are the lowest- and highest-degree vertices in $\overline{H}$, then $\overline{V_\ell} = V_h$, $\overline{V_h} = V_\ell$ and hence $\overline{H} - \overline{V_\ell}$ is empty and $\overline{H} - \overline{V_h}$ is a clique, that is, we have the same condition as for H. Fortunately, this last case is very simple to handle.

▶ **Lemma 23.** *There exists no graph $H \notin \mathcal{X} \cup \mathcal{Y}$ such that $H - V_\ell$ is an empty graph and $H - V_h$ is a complete graph.*

3.4 Cliques/empty graphs plus small graphs

Next we consider the cases when one of $H - V_\ell$ or $H - V_h$ is a clique or an empty graph, while the other is a graph from the finite set $\mathcal{Y}'$. Assuming that H is not too small, this means that H is essentially a clique or an empty graph, and intuitively it should follow that H or $\overline{H}$ is 3-connected, respectively. However, this requires a detailed proof considering several cases.

▶ **Lemma 24.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that $H - V_\ell \in \mathcal{Y}'$ and $H - V_h$ is a complete graph. Then $H \in \mathcal{W}$.*

▶ **Corollary 25.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that $H - V_\ell$ is an empty graph and $H - V_h \in \mathcal{Y}'$. Then $H \in \mathcal{W}$.*

▶ **Lemma 26.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that $H - V_\ell \in \mathcal{Y}'$ and $H - V_h$ is an empty graph. Then $H \in \mathcal{W}$.*

▶ **Corollary 27.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that $H - V_\ell$ is a complete graph and $H - V_h \in \mathcal{Y}'$. Then $H \in \mathcal{W}$.*

3.5 Near-empty graphs

Finally, we consider the cases when one of $H - V_\ell$ or $H - H_h$ is near empty. These cases are similar to the corresponding ones for empty graphs, but more technical and a higher number of corner cases need to be handled. Let us remark that this part of the proof is needed only for the H-free Edge Deletion problem: near-empty graphs are not in $\mathcal{Y}_E$, hence if our goal is to prove Theorem 1 for H-free Edge Editing, then the churning procedure can recurse on such graphs.

▶ **Lemma 28.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that $H - V_\ell$ is a complete graph and $H - V_h$ is a near-empty graph. Then $H \in \mathcal{W}$.*

▶ **Lemma 29.** *There exists no $H \notin \mathcal{X} \cup \mathcal{Y}$ such that $H - V_\ell$ is a near-empty graph and $H - V_h$ is a complete graph.*

▶ **Lemma 30.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ such that $H - V_\ell$ is an empty graph and $H - V_h$ is a near-empty graph. Then $H \in \mathcal{W}$.*

▶ **Lemma 31.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ such that $H - V_\ell$ is a near-empty graph and $H - V_h$ is an empty graph. Then $H \in \mathcal{W}$.*

▶ **Lemma 32.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that both $H - V_\ell$ and $H - V_h$ are near-empty graphs. Then $H \in \mathcal{W}$.*

▶ **Lemma 33.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that $H - V_\ell$ is a near-empty graph and $H - V_h \in \mathcal{Y}'$. Then $H \in \mathcal{W}$.*

▶ **Lemma 34.** *Let $H \notin \mathcal{X} \cup \mathcal{Y}$ be such that $H - V_\ell \in \mathcal{Y}'$ and $H - V_h$ is a near-empty graph. Then $H \in \mathcal{W}$.*

4 Reductions

Recall that we defined $\mathcal{W} = \mathcal{H} \cup \overline{\mathcal{H}} \cup \mathcal{A} \cup \overline{\mathcal{A}} \cup \mathcal{D} \cup \overline{\mathcal{D}} \cup \mathcal{B} \cup \overline{\mathcal{B}} \cup \mathcal{S} \cup \overline{\mathcal{S}} \cup \mathcal{F} \cup \overline{\mathcal{F}}$ and Section 3 reduced our main questions to assuming incompressibility for the set $\mathcal{W}$. In this section, we further refine the result and show that incompressibility needs to be assumed only for the finite set $\mathcal{W}' = \mathcal{H} \cup \overline{\mathcal{H}} \cup \mathcal{A} \cup \overline{\mathcal{A}} \cup \mathcal{B} \cup \mathcal{D}$. That is, we recall and introduce some further simple reductions and use them to prove that every graph in $\mathcal{W} \setminus \mathcal{W}'$ simulates a graph in $\mathcal{W}' \cup \mathcal{X}_D$. To begin with, we observe that deleting the lowest degree vertices in the graphs in $\overline{\mathcal{B}} \cup \overline{\mathcal{D}}$ results in 3-connected graphs which are not complete. Then by Proposition 16, we have:

▶ **Proposition 35.** *If $H \in \overline{\mathcal{B}} \cup \overline{\mathcal{D}}$, then H-free Edge Editing and H-free Edge Deletion are incompressible, assuming* NP $\not\subseteq$ coNP/poly.

The following reductions are based on Construction 1 and a few other similar constructions.

4.1 Reductions based on Construction 1

The following lemma can be proved using a straight-forward application of Construction 1.

▶ **Lemma 36.** *Let H be $J \cup K_t$, for some graph J and integer $t \geq 1$, where the K_t is induced by V_ℓ. Let V' be $V(H) \setminus \{v\}$, where v is any vertex in the K_t. Let H' be $H[V']$. Then H simulates H'. In particular, H simulates $J \cup K_1$.*

▶ **Corollary 37.**
- **(i)** *Let H be $K_t \cup K_2$, for $t \geq 4$ ($\in \overline{\mathcal{F}_1}$). Then H simulates $K_t \cup K_1$ ($\in \{\overline{H_5}\} \cup \overline{\mathcal{F}_2}$).*
- **(ii)** *Let H be $(K_t - e) \cup K_2$, for $t \geq 4$ ($\in \overline{\mathcal{F}_6}$). Then H simulates $(K_t - e) \cup K_1$ ($\in \{\overline{H_8}, \overline{D_2}\} \cup \overline{\mathcal{F}_8}$).*

Next we consider the removal of a path of degree-2 vertices. We can prove the correctness of the reduction only under a certain uniqueness condition on the path.

▶ **Lemma 38.** *Let H be a graph with minimum degree two and let $p \geq 2$ be an integer such that there is a unique induced path P of length p with the property that all the internal vertices of the path are having degree exactly two in H. Let H' be obtained from H by removing all internal vertices of P. Then H simulates H'.*

▶ **Corollary 39.** *Let H be J_t, for some $t \geq 3$ ($\in \mathcal{F}_9$). Then H simulates $K_2 \boxtimes tK_1$ ($\in \{\overline{H_3}\} \cup \mathcal{F}_3$).*

▶ **Corollary 40.** *Let H be Q_t, for some $t \geq 3$ ($\in \mathcal{F}_{10}$). Then H simulates $K_{2,t}$ ($\in \{S_1\} \cup \mathcal{F}_1$).*

▶ **Corollary 41.**
- **(i)** *S_5 simulates C_4.*
- **(ii)** *S_9 simulates $\overline{H_4}$.*
- **(iii)** *Let H be S_{15}. Then H simulates $\overline{H_9 \cup K_1}$. Further H simulates H_9 (Proposition 16).*
- **(iv)** *S_{22} simulates $\overline{diamond \cup K_2}$ ($\in \overline{\mathcal{F}_6}$).*

Lemma 42 essentially says the following: If H has vertex connectivity 1 and has a unique smallest 2-connected component which is a "leaf" in the tree formed by the 2-connected components, then H simulates a graph obtained by removing all vertices in the 2-connected component except the cut vertex.

▶ **Lemma 42.** *Let H be a graph with vertex connectivity 1 and be not a complete graph. Let $\mathcal{C}$ be the set of all 2-connected components of H having exactly one cut vertex of H. Assume that there exists a unique smallest (among $\mathcal{C}$) 2-connected component J in $\mathcal{C}$. Let v be the cut vertex of H in J. Let H' be $H - \{J \setminus \{v\}\}$. Then H' is simulated by H.*

▶ **Corollary 43.**

(i) *S_{20} simulates $K_{2,4}$ ($\in \mathcal{F}_1$).*

(ii) *S_{27} simulates $\overline{(K_5 - e) \cup K_2}$ ($\in \mathcal{F}_6$).*

▶ **Lemma 44.** *Let H be S_{35}. Then H simulates a 3-connected graph, which is not complete ($\in \mathcal{X}_D$).*

4.2 Reductions based on Construction 2

The following is a simplified version of Construction 1.

▶ **Construction 2.** *Let (G', k, ℓ) be an input to the construction, where G' is a graph and k and ℓ are positive integers. For every set S of ℓ vertices in G' introduce a clique C of $k+1$ vertices and make all the vertices in C adjacent to all the vertices in S.*

As before, we call every clique C introduced during the construction as a *satellite* and the vertices in it as *satellite vertices*. Lemma 45 can be proved using a straight-forward application of Construction 2. It says that if H satisfies some properties, then H simulates H' where H' is obtained by removing one vertex from each module of H contained within V_ℓ.

▶ **Lemma 45.** *Let H be a non-regular graph such that the following conditions hold true:*

(i) *$1 \leq \ell \leq 2, |V(H)| \geq 5$;*

(ii) *V_ℓ is an independent set, $V_h \cup V_m$ induces a connected graph, and every vertex in V_h is adjacent to at least one vertex in V_ℓ;*

(iii) *Every vertex in V_m has at least $\ell + 1$ neighbors outside V_m or there exists no pair u, v of adjacent vertices in V_m such that $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.*

Consider a modular decomposition $\mathcal{M}$ of H. Let $\mathcal{M}' \subseteq \mathcal{M}$ corresponds to V_ℓ. Let H' be the graph obtained from H by removing one vertex from each module in $\mathcal{M}'$. Then H simulates H'.

The following corollary lists many graphs that can be handled by Lemma 45.

▶ **Corollary 46.**

(i) *S_7 simulates H_9.*

(ii) *S_8 simulates $\overline{A_1}$.*

(iii) *S_{10} simulates C_4.*

(iv) *S_{11} simulates $\overline{H_7}$.*

(v) *S_{12} simulates S_2.*

(vi) *S_{13} simulates S_3.*

(vii) *$\overline{S_{14}}$ simulates $\overline{B_1}$.*

(viii) *S_{18} simulates $\overline{A_1}$.*

(ix) *S_{19} simulates $\overline{S_3}$.*

(x) *S_{21} simulates $\overline{H_4}$.*

(xi) *S_{23} simulates $\overline{A_7}$.*

(xii) *S_{24} simulates $\overline{S_7}$.*

(xiii) *S_{25} simulates $\overline{H_1}$.*

(xiv) *S_{26} simulates S_8.*

(xv) *S_{28} simulates $\overline{A_9}$.*

(xvi) *S_{29} simulates S_{17}.*

(xvii) *S_{30} simulates $\overline{S_{19}}$.*

(xviii) *$\overline{S_{32}}$ simulates $\overline{S_{16}}$.*

(xix) *$\overline{S_{33}}$ simulates $\overline{K_{1,4} \cup K_2} \in \overline{\mathcal{F}_7}$.*

(xx) *$\overline{S_{34}}$ simulates $\overline{K_{1,5} \cup K_2} \in \overline{\mathcal{F}_7}$.*

(xxi) *$\overline{S_{36}}$ simulates $\overline{S_{14}}$.*

The following three corollaries are obtained by application of Lemma 45: they show that in certain families of graphs, every member simulates the simplest member. Corollary 47 deals with star graphs ($K_{1,t}$). For every graph H in this class, V_ℓ is a single module of the graph and H simulates a graph H', where H' is obtained by removing one vertex from V_ℓ. Corollary 48 handles $K_2 \boxtimes sK_1$, where V_ℓ forms a single module of the graph. As in the previous case, H' is obtained by removing one vertex from V_ℓ. Corollary 49 deals with the set of twin-star graphs (T_{t_1,t_2}). For every graph H in this class, there are two modules of H in V_ℓ : t_1 vertices adjacent to one vertex in $H - V_\ell$ and t_2 vertices adjacent to the other vertex in $H - V_\ell$. Then H simulates a graph H', where H' is obtained by removing one vertex each from the two modules.

▶ **Corollary 47** (see Lemma 6.4 in [2] for a partial result)**.** *Let H be $K_{1,t}$, for any $t \geq 5$ ($\in \mathcal{F}_2$). Let H' be $K_{1,t-1}$. Then H simulates H'. Furthermore, H simulates H_5 ($K_{1,4}$).*

▶ **Corollary 48** (see Lemma 4.5 in [1] for a partial result)**.** *Let H be $K_2 \boxtimes sK_1$, for any $s \geq 4$ ($\in \mathcal{F}_3$) and let H' be $K_2 \boxtimes (s-1)K_1$. Then H simulates H'. Furthermore, H simulates $\overline{H_3}$ ($K_2 \boxtimes 3K_1$).*

▶ **Corollary 49** (see Lemma 6.6 in [2] for a partial result)**.** *Let H be a twin-star graph T_{t_1,t_2}, such that $t_1, t_2 \geq 1$. Let H' be T_{t_1-1,t_2-1}. Then H simulates H'. In particular, if H is $T_{t,1}$, for some $t \geq 4$ ($\in \mathcal{F}_4$), then H simulates $K_{1,t}$ ($\in \{H_5\} \cup \mathcal{F}_2$).*

▶ **Lemma 50.** *$K_{2,3}$ ($= S_1$) simulates C_4.*

4.3 Reductions based on Construction 3

Now we give another construction that will be used in a few reductions.

▶ **Construction 3.** *Let (G', k, t) be an input to the construction, where G' is a graph and k and t are positive integers. For every set S of t vertices in G' introduce an independent set I_S of $k+2$ vertices such that every vertex in I_S is adjacent to every vertex in G' except those in S. Let $\bigcup_{S \subseteq V(G'), |S|=t} I_S = I$. Let the resultant graph be G.*

▶ **Lemma 51.** *Let H be a graph such that V_h forms a clique and for every pair of vertices $u, v \in V_h$, $H - u$ is isomorphic to $H - v$. Further assume that there exists no independent set S of size $s \geq 2$ where each vertex in S has degree at least $h - s + 1$ in H. Then H simulates $H - u$, where u is any vertex in V_h.*

Additional results used by Corollary 52 are shown in parenthesis.

▶ **Corollary 52.**

(i) *S_2 simulates $\overline{H_7}$.*
(ii) *S_3 simulates H_6.*
(iii) *S_{16} simulates S_3 (Proposition 16).*
(iv) *S_{17} simulates $\overline{H_1}$ (Proposition 16).*
(v) *S_{31} simulates S_3 (Proposition 16)*

▶ **Lemma 53.** *Let H be $(K_t - e) \cup K_1$ for $t \geq 6$ ($\overline{\mathcal{F}_8}$). Let H' be $K_{t-2} \cup K_1$ ($\in \{\overline{H_5}\} \cup \overline{\mathcal{F}_2}$). Then H simulates H'.*

4.4 Other reductions

To resolve graphs in $\mathcal{F}_7$ ($= K_{1,t} \cup K_2$), we resort to a known reduction. There is a PPT in [2] from H'-free Edge Editing to H-free Edge Editing, where H' is a largest component in H. It is a composition of two reductions: one from H'-free Edge Editing to H''-free Edge Editing and another from H''-free Edge Editing to H-free Edge Editing, where H'' is the union of all components in H isomorphic to H'. The first reduction uses a simple construction (take a disjoint union of the input graph and join of $k+1$ copies of H') and the second reduction uses Construction 1.

▶ **Proposition 54** (see Lemma 3.5 in [2]). *Let H' be a largest component of H. Then H simulates H'.*

▶ **Corollary 55.** *Let H be $K_{1,t} \cup K_2$, for $t \geq 4$ ($\in \mathcal{F}_7$). Then H simulates $K_{1,t}$ ($\in \{H_5\} \cup \mathcal{F}_2$).*

The following statement consider reduction that involve the removal of independent vertices.

▶ **Lemma 56.** *Let H be $J \cup tK_1$, for any $t \geq 2$ such that J has no component which is a clique. Let H' be $J \cup (t-1)K_1$. Then H simulates H'. In particular, H simulates $J \cup K_1$.*

▶ **Corollary 57.**

(i) *S_4 simulates H_2.*
(ii) *S_6 simulates H_6.*
(iii) *Let H be $(K_t - e) \cup 2K_1$, for $t \geq 4$ ($\in \overline{\mathcal{F}_5}$). Then H simulates $(K_t - e) \cup K_1$ ($\in \{\overline{H_8}, \overline{D_2}\} \cup \overline{\mathcal{F}_8}$).*

Summary of results in this section handling graphs in $\mathcal{S}$ and $\mathcal{F}$ are given in Figure 4a and 4b respectively. Lemma 58 follows from Corollary 5, Proposition 35, the transitivity of PPTs, and other results in this section (see Figures 4a and 4b) for details.

▶ **Lemma 58.** *Let $H \in \mathcal{W} \setminus \mathcal{W}'$. Then H simulates a graph in $\mathcal{W}' \cup \mathcal{X}_D$.*

5 Incompressibility results for the graphs in $\mathcal{A}$ and $\mathcal{B}$

In this section, we prove that for every graph $H \in \mathcal{A} \cup \overline{\mathcal{A}}$, all three problems H-free Edge Editing, H-free Edge Deletion, and H-free Edge Completion are incompressible, assuming NP $\not\subseteq$ coNP/poly. With the same assumption, we prove that H-free Edge Deletion is incompressible for every graph $H \in \mathcal{B}$; Proposition 4 then implies incompressibility of H-free Edge Completion for every $H \in \overline{\mathcal{B}}$.

▶ **Theorem 59.** *Assuming* NP $\not\subseteq$ coNP/poly*:*

(i) *Let $H \in \mathcal{A}$. Then H-free Edge Editing is incompressible.*
(ii) *Let $H \in \mathcal{A} \cup \overline{\mathcal{A}} \cup \mathcal{B}$. Then H-free Edge Deletion is incompressible.*
(iii) *Let $H \in \mathcal{A} \cup \overline{\mathcal{A}} \cup \overline{\mathcal{B}}$. Then H-free Edge Completion is incompressible.*

We apply the technique used by Cai and Cai [4] by which they obtained a complete dichotomy on the incompressibility of H-free edge modification problems on 3-connected graphs H. We will give a self-contained summary of their proof technique, with only a few references to proofs of formal statements. The reader is referred to [4] for a more detailed exposition of terminology and concepts discussed in this section.

The first step in the proof is to establish incompressibility for the *restricted* versions of H-free Edge Deletion and H-free Edge Completion, where only allowed edges can be deleted/added. Then deletion and completion *enforcer gadgets* can be used to reduce

H	Simulates	By
S_1	C_4	Lemma 50
S_2	$\overline{H_7}$	Corollary 52
S_3	H_6	Corollary 52
S_4	H_2	Corollary 57
S_5	C_4	Corollary 41
S_6	H_6	Corollary 57
S_7	H_9	Corollary 46
S_8	$\overline{A_1}$	Corollary 46
S_9	$\overline{H_4}$	Corollary 41
S_{10}	C_4	Corollary 46
S_{11}	$\overline{H_7}$	Corollary 46
S_{12}	S_2	Corollary 46

H	Simulates	By
S_{13}	S_3	Corollary 46
S_{14}	B_1	Corollary 46
S_{15}	H_9	Corollary 41
S_{16}	S_3	Corollary 52
S_{17}	$\overline{H_1}$	Corollary 52
S_{18}	$\overline{A_1}$	Corollary 46
S_{19}	$\overline{S_3}$	Corollary 46
S_{20}	a graph in $\mathcal{F}_1$	Corollary 43
S_{21}	$\overline{H_4}$	Corollary 46
S_{22}	a graph in $\overline{\mathcal{F}_6}$	Corollary 41
S_{23}	$\overline{A_7}$	Corollary 46
S_{24}	$\overline{S_7}$	Corollary 46

H	Simulates	By
S_{25}	$\overline{H_1}$	Corollary 46
S_{26}	S_8	Corollary 46
S_{27}	a graph in $\mathcal{F}_6$	Corollary 43
S_{28}	$\overline{A_9}$	Corollary 46
S_{29}	S_{17}	Corollary 46
S_{30}	$\overline{S_{19}}$	Corollary 46
S_{31}	S_3	Corollary 52
S_{32}	S_{16}	Corollary 46
S_{33}	a graph in $\mathcal{F}_7$	Corollary 46
S_{34}	a graph in $\mathcal{F}_7$	Corollary 46
S_{35}	a graph in $\mathcal{X}_D$	Lemma 44
S_{36}	S_{14}	Corollary 46

(a) Summary of results in Section 4 handling graphs in $\mathcal{S}$.

$H \in$	Simulates a graph in	By
$\mathcal{F}_1$	$\{H_5\} \cup \mathcal{F}_2$	Corollary 37
$\mathcal{F}_2$	$\{H_5\}$	Corollary 47
$\mathcal{F}_3$	$\{\overline{H_3}\}$	Corollary 48
$\mathcal{F}_4$	$\{H_5\} \cup \mathcal{F}_2$	Corollary 49
$\mathcal{F}_5$	$\{H_8, D_2\} \cup \mathcal{F}_8$	Corollary 57

$H \in$	Simulates a graph in	By
$\mathcal{F}_6$	$\{H_8, D_2\} \cup \mathcal{F}_8$	Corollary 37
$\mathcal{F}_7$	$\{H_5\} \cup \mathcal{F}_2$	Corollary 55
$\mathcal{F}_8$	$\{H_5\} \cup \mathcal{F}_2$	Lemma 53
$\mathcal{F}_9$	$\{\overline{H_3}\} \cup \mathcal{F}_3$	Corollary 39
$\mathcal{F}_{10}$	$\{S_1\} \cup \mathcal{F}_1$	Corollary 40

(b) Summary of results in Section 4 handling graphs in $\mathcal{F}$.

Figure 4

the restricted problems to the original versions. Cai and Cai [4] presented constructions that were proved to work correctly when H is 3-connected. We show, by careful inspection, that the same technique works for certain graphs H that are not 3-connected. For certain graphs H, we can prove incompressibility of the restricted problem, but enforcer gadgets of the required form provably do not exist. In these cases, we use ad hoc ideas to reduce the restricted version to the original one. In yet further cases, we need even trickier reductions, where we reduce H'-free Edge Deletion to H-free Edge Deletion for some $H' \neq H$.

5.1 Incompressibility results for the restricted problems

A graph is called *edge-restricted* if a subset of its edges are marked as *forbidden*. All edges other than forbidden are *allowed*. A graph is called *nonedge-restricted* if a subset of its nonedges are marked as *forbidden*. All nonedges other than forbidden are *allowed*.

Restricted H-free Edge Deletion: Given a graph G, an integer k, and a set R of edges of G, do there exist at most k edges disjoint from R such that deleting them from G results in an H-free graph? **Parameter**: k

Restricted H-free Edge Completion: Given a graph G, an integer k, and a set R of nonedges of G, do there exist at most k nonedges disjoint from R such that adding them in G results in an H-free graph? **Parameter**: k

Propagational formula satisfiability. A ternary Boolean function $f(x, y, z)$ (where x, y, and z are either Boolean variables or constants 0 or 1) is *propagational* if $f(1, 0, 0) = 0, f(0, 0, 0) = f(1, 0, 1) = f(1, 1, 0) = f(1, 1, 1) = 1$. This has the meaning: if x is true then either y is true or z is true.

Propagational-f Satisfiability: Given a conjunctive formula φ of a propagational ternary function f with distinct variables in each clause of φ, find whether there exists a satisfying truth assignment with weight at most k. The parameter we consider is k.

▶ **Proposition 60** (Theorem 3.4 in [4]). *For any propagational ternary Boolean function f,* Propagational-f Satisfiability *on 3-regular conjunctive formulas (every variable appears exactly three times) admits no polynomial kernel, assuming* NP $\not\subseteq$ coNP/poly.

Satisfaction-testing components. For H-free Edge Deletion, a *satisfaction-testing component* $S_D(x, y, z)$ is a constant-size edge-restricted H-free graph with exactly three allowed edges $\{x, y, z\}$ such that there is a propagational Boolean function $f(x, y, z)$ such that $f(x, y, z) = 1$ if and only if the graph obtained from $S_D(x, y, z)$ by deleting edges in $\{x, y, z\}$ with value 1 is H-free. For H-free Edge Completion, a *satisfaction-testing component* $S_C(x, y, z)$ is a constant-size nonedge-restricted H-free graph with exactly three allowed nonedges $\{x, y, z\}$ such that there is a propagational Boolean function $f(x, y, z)$ such that $f(x, y, z) = 1$ if and only if the graph obtained from $S_C(x, y, z)$ by adding edges in $\{x, y, z\}$ with value 1 is H-free.

There is an easy construction (Lemma 4.3 in [4]) showing that $S_D(x, y, z)$ exists for every connected graph H with at least four vertices but not complete and $S_C\{x, y, z\}$ exists for every connected graph with at least four vertices and at least two nonedges. The construction for this is as follows. $S_D\{x, y, z\}$: Let x be a nonedge, and y and z be two edges in H. Then $H + x$ is a $S_D(x, y, z)$ where x, y, z are the only allowed edges. $S_C\{x, y, z\}$: Let x be an edge, and y and z be two nonedges in H. Then $H - x$ is a $S_C(x, y, z)$ where x, y, z are the only allowed nonedges.

Truth-setting components. For H-free Edge Deletion, a *truth-setting component* $(T_D(u))$ is a constant-sized, edge-restricted H-free graph such that it contains at least three allowed edges x, y, z without a common vertex and admits exactly two deletion sets $\emptyset$ and the set of all allowed edges. For H-free Edge Completion, a *truth-setting component* $(T_C(u))$ is a constant-sized, nonedge-restricted H-free graph such that it contains at least three allowed nonedges x, y, z without a common vertex and admits exactly two completion sets $\emptyset$ and the set of all allowed nonedges.

There is a construction given in [4] for $T_D(u)$ and $T_C(u)$ when H is 3-connected but not complete. The constructions are given below.

Construction of $T_D(u)$: Let e', e be a nonedge and an edge sharing no common vertex in H. Let the *basic unit* $U = H + e'$ and set all edges except e and e' in U as forbidden. Let p be the number of vertices in H. Take p copies $U_1, U_2, \ldots, U_p$ of U. Identify the edge e of U_i with the edge e' of U_{i+1} to form a chain of U's. This is a *basic chain* $B(u)$. Let us call the unidentified edge e' of U_1 as the *left-most allowed edge* of $B(u)$ and unidentified edge of U_p as the *right-most allowed edge* of $B(u)$. Take three basic chains B_0, B_1, and B_2. Attach them in a cyclic fashion: Identify the right-most allowed edge of B_i with the left-most allowed edge of B_{i+1}, where indices are taken mod 3. This is the claimed truth-setting component $T_D(u)$. Let us call the allowed edges thus identified as *variable edges*. We note that there are exactly three variable edges in $T_D(u)$.

It is easy to see that, for every H, there are only two possible deletion sets in $T_D(u)$: the empty set and the set of all allowed edges. To see this, observe that if we remove any of the allowed edges, then it creates a copy of H in one of the units, forcing us to remove the next allowed edge as well. However, it is not clear if these two deletion sets really make the graph H free. As Cai and Cai [4] show, this construction for $T_D(u)$ works correctly for 3-connected graphs H: Since the "cycle" of basic units is long enough, every subgraph having vertices from different basic units and having at most $|V(H)|$ vertices has vertex connectivity at most 2. In general, the construction may not give correct truth-setting components for 2-connected graphs H. But, as we shall see later, by carefully choosing e and e' in these constructions, we can obtain truth-setting components for many 2-connected graphs H.

Construction of $T_C(u)$: Let e', e be a nonedge and an edge sharing no common vertex in H. Let the *basic unit* $U = H - e$ and set all nonedges except e and e' in U as forbidden. Let p be the number of vertices in H. Take p copies $U_1, U_2, \ldots, U_p$ of U. Identify the nonedge e of U_i with the nonedge e' of U_{i+1} to form a chain of U's. This is a *basic chain* $B(u)$. Let us call the unidentified nonedge e' of U_1 as the *left-most allowed nonedge* of $B(u)$ and unidentified nonedge of U_p as the *right-most allowed nonedge* of $B(u)$. Take three basic chains B_0, B_1, and B_2. Attach them in a cyclic fashion: Identify the right-most allowed nonedge of B_i with the left-most allowed nonedge of B_{i+1}, where indices are taken mod 3. This is the claimed truth-setting component $T_C(u)$. Let us call the allowed nonedges thus identified as *variable nonedges*. We note that there are exactly three variable nonedges in $T_C(u)$. Similarly to $T_D(u)$, we can argue that for any H, there are only two potential completion sets (the empty set and the set of all allowed nonedges), and for 3-connected H, these two sets are indeed completion sets.

The following is the construction used in the reduction from Propagational-f Satisfiability to Restricted H-free Edge Deletion (Completion).

▶ **Construction 4.** *Let (φ, k, H) be an input to the construction, where φ is a 3-regular conjunctive formula on a propagational ternary Boolean function f, and k is a positive integer. The construction gives a graph G_φ, an integer k', and a set of restricted (non)edges in G_φ.*

- *For every clause in φ, introduce a satisfaction-testing component $S_D(x, y, z)$ ($S_C(x, y, z)$) for H-free Edge Deletion (Completion).*
- *If $c \in \{x, y, z\}$ is 1, then the corresponding allowed (non)edge is deleted(added) and if $c = 0$ then the corresponding allowed (non)edge is set as forbidden.*
- *For every variable u in f, introduce a truth-setting component $T_D(u)$ ($T_C(u)$) for H-free Edge Deletion (Completion)*
- *For every variable u, identify each of the variable (non)edges in $T_D(u)$ ($T_C(u)$) with an allowed (non)edge in a satisfaction-testing component corresponds to a different clause in which u appears – since φ is 3-regular, u appears in exactly three clauses.*

Let the graph obtained be G_φ and let $k' = 3|V(H)|k$. For the deletion problem the set R of forbidden edges is all the edges in G_φ except the allowed edges in the units. For the completion problem, the set R of forbidden nonedges contains every nonedge of G_φ except the allowed nonedges in the units.

Let H be a graph and (φ, k) be an instance of a Propagational-f Satisfiability problem. Let (G_φ, k', R) be the output of the Construction 4 applied on (φ, k, H). The construction works correctly in one direction: If (G_φ, k', R) is a yes-instance of Restricted H-free Edge Deletion (Completion), then (φ, k) is a yes-instance of Propagational-f Satisfiability. To see this, let F be a solution of (G_φ, k', R). By the definition of $T_D(u)$ ($T_C(u)$), if an allowed (non)edge is in F then so is every allowed (non)edge in it. Therefore,

since $|F| \leq k' = 3k|V(H)|$ and every truth-setting component has exactly $3|V(H)|$ many allowed (non)edges, only at most k (non)edges of satisfaction-testing components are in F. By the definition of $S_D(x,y,z)$ ($S_C(x,y,z)$), if $x \in F$ then either y or z is in F, otherwise there is an induced H in $G_\varphi + F$. Therefore, setting the variables to 1 corresponding to the (non)edges, which are part of F, in satisfaction-testing components, we obtain that (φ, k) is a yes-instance of Propagational-f Satisfiability. Thus we have the following Proposition.

▶ **Proposition 61** (see Lemma 5.1 in [4]). *Let (φ, k) be an instance of* Propagational-f Satisfiability. *For a graph H, let (G_φ, k', R) be obtained by applying Construction 4 on (φ, k, H). Then, if (G_φ, k', R) is a yes-instance of* Restricted H-free Edge Deletion *(Completion) then (φ, k) is a yes-instance of* Propagational-f Satisfiability.

We remark that the proof of Proposition 61 works even if we use a gadget for the truth-setting component which satisfies only a weak property: it has at most two deletion (completion) sets, the $\emptyset$ and the set of all allowed (non)edges. As we have seen, the construction of $T_D(u)$ and $T_C(u)$ discussed above satisfies this weak property. To prove the other direction, one needs to show that there is no induced H in the "vicinity" of a satisfaction-testing component after deleting (adding) the (non)edges corresponding to the variables being set to 1 in φ. This can be done very easily for 3-connected graphs H. Proving this direction for 2-connected graphs H (if provable) requires careful structural analysis of the constructed graph G_φ. In Figure 5, we give various gadgets required for the proofs of this section. We use *unit* as a general term to refer to a satisfaction-testing component or a basic unit.

▶ **Lemma 62.** *Let $H \in \{\overline{A_1}, \overline{A_2}, A_3, \overline{A_3}, A_4, A_5, \overline{A_7}, \overline{A_9}\}$. Then* Restricted H-free Edge Deletion *is incompressible, assuming* NP $\not\subseteq$ coNP/poly.

The following corollary follows from the fact that there is no subgraph isomorphic to a C_4 where all edges are allowed in the graph G_φ constructed in the proof of Lemma 62 for Restricted H-free Edge Deletion, when H is a $\overline{A_1}$. We will be using this result later to handle $\overline{A_7}$.

▶ **Corollary 63.** *Let H be $\overline{A_1}$. Then, assuming* NP $\not\subseteq$ coNP/poly, Restricted H-free Edge Deletion *is incompressible even if input graphs does not contain a subgraph (not necessarily induced) isomorphic to H where all the side-edges of the diamond (a side-edge of a diamond is an edge incident to a degree-2 vertex in the diamond) in the subgraph are allowed.*

▶ **Lemma 64.** *Let $H \in \{\overline{A_2}, \overline{A_7}, \overline{A_8}, \overline{A_9}, \overline{B_1}, \overline{B_2}, \overline{B_3}\}$. Then* Restricted H-free Edge Completion *is incompressible, assuming* NP $\not\subseteq$ coNP/poly.

5.2 Using enforcers to reduce to the unrestricted problems

If we want to reduce Restricted H-free Edge Deletion to H-free Edge Deletion, then there is a fairly natural idea to try: for each restricted edge $e' = x'y'$, we introduce a copy of H on set U of new vertices and identify $x'y'$ with xy, where $x, y \in U$ are nonadjacent vertices. Now U induces a copy of H plus an extra edge, but as soon as e' is deleted, it becomes a copy of H, effectively preventing the deletion of e'.

There are two problems with this approach. First, the solution could delete other edges from the new copy of H, and then it is not necessarily true that the removal of e' automatically creates an induced copy of H. However, this problem is easy to avoid by repeating this

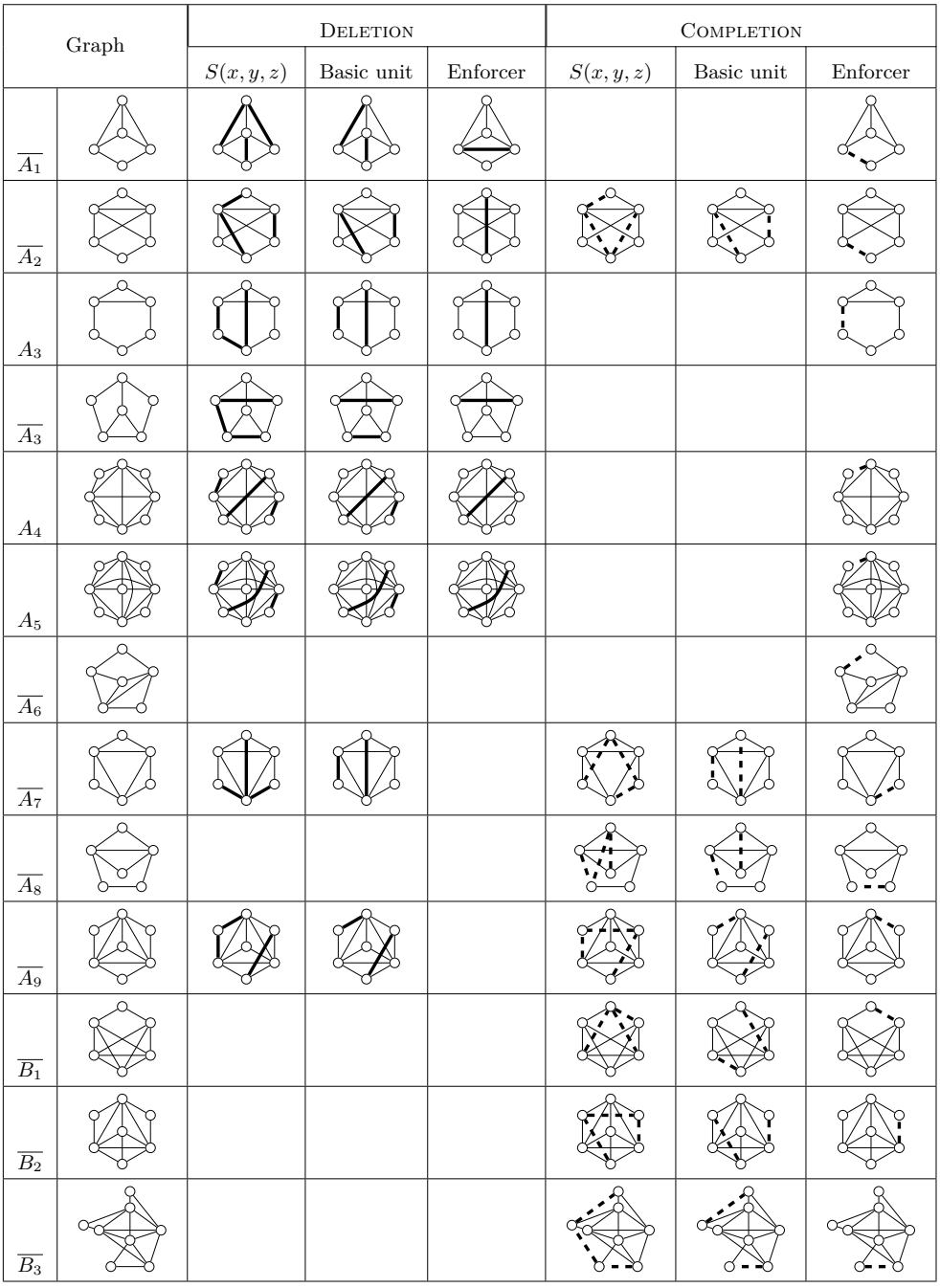


Figure 5 Various gadgets used in the proofs of this section. In a satisfaction-testing component $S_D(x,y,z)$ ($S_C(x,y,z)$), x is the darkened (non)edge added(deleted) in H to obtain the gadget, and y and z are the other two darkened (non)edges. Both the allowed (non)edges in basic units are darkened. The distinguished edge in a deletion enforcer and the distinguished nonedge in a completion enforcer are darkened.

gadget construction $k+1$ times: a solution of size at most k cannot interfere with all $k+1$ gadgets. The second problem is more serious: it is possible that attaching the new vertices creates a copy of H, even when e is not deleted. For certain graphs H, with a careful choice of x and y we can ensure that this does not happen: no induced copy of H can go through the separator x, y.

An *H-free deletion enforcer* (X, e) consists of an H-free graph X and a distinguished edge e in X such that (a) $X - e$ contains an induced H, and (b) for any graph G vertex disjoint with X, and any edge e' of G, all induced copies of H in the graph obtained by attaching X to G through identifying e with e' reside entirely inside G. Similarly, an *H-free completion enforcer* (X, e) consists of an H-free graph X and a distinguished nonedge e such that (a) $X + e$ contains an induced H, and (b) for any graph G vertex disjoint with X, and any nonedge e' in G, all induced copies of H in the graph obtained by attaching X to G through identifying e with e' reside entirely inside G. It can be shown that if we can come up with enforcer gadgets satisfying these conditions, then the ideas sketched above can be made to work, and we obtain a reduction from the restricted problem to the unrestricted version.

▶ **Proposition 65** (See Lemma 6.5 in [4]). *For a graph H:*

(i) *If* Restricted H-free Edge Deletion *is incompressible and there exists an H-free deletion enforcer, then H-*free Edge Deletion *is incompressible.*

(ii) *If* Restricted H-free Edge Completion *is incompressible and there exists an H-free completion enforcer, then H-*free Edge Completion *is incompressible.*

(iii) *If H-*free Edge Deletion *is incompressible and there exists an H-free completion enforcer, then H-*free Edge Editing *is incompressible.*

In the rest of the section, we establish the existence of enforcer gadgets for certain graphs H.

▶ **Lemma 66.** *Let $H \in \{\overline{A_1}, \overline{A_2}, A_3, \overline{A_3}, A_4, A_5\}$. Then the gadget X with a distinguished edge e shown in the corresponding cell in the column "Enforcer" (under* Deletion*) in Figure 5 is an H-free deletion enforcer.*

▶ **Lemma 67.** *Let $H \in \{\overline{A_1}, \overline{A_2}, A_3, A_4, A_5, \overline{A_6}, \overline{A_7}, \overline{A_8}, \overline{A_9}, \overline{B_1}, \overline{B_2}, \overline{B_3}\}$. Then the gadget X with a distinguished nonedge e shown in the corresponding cell in the column "Enforcer" (under* Completion*) in Figure 5 is an H-free completion enforcer.*

▶ **Lemma 68.** *Let $H \in \{\overline{A_1}, \overline{A_2}, A_3, \overline{A_3}, A_4, A_5\}$. Then H-*free Edge Deletion *and H-*free Edge Editing *are incompressible, assuming* NP $\not\subseteq$ coNP/poly.

Similarly, we can prove Lemma 69. The cases of H being A_4 or A_5 follows from the fact that H and $\overline{H}$ are isomorphic (see Proposition 4).

▶ **Lemma 69.** *Let $H \in \{\overline{A_2}, A_4, A_5, \overline{A_7}, \overline{A_8}, \overline{A_9}, \overline{B_1}, \overline{B_2}, \overline{B_3}\}$. Then H-*free Edge Completion *is incompressible, assuming* NP $\not\subseteq$ coNP/poly.

5.3 Further tricky reductions

There are graphs for which we can show that no completion/deletion enforcers, as defined in the previous section, exist (this can be checked by going through every pair x, y of (non)adjacent vertices). For some of these graphs, we can find a different way of enforcing that certain edges are forbidden; typically, we introduce some vertices that are used globally by every enforcer gadget. Furthermore, there are graphs H, where we were unable to obtain a reduction from Restricted H-free Edge Deletion (Completion), but could choose an induced subgraphs $H' \subseteq H$ and obtain a reduction from Restricted H'-free Edge Deletion (Completion), whose incompressibility was established earlier.

▶ **Lemma 70.** *Assuming* NP $\not\subseteq$ coNP/poly, H-free Edge Editing *and* H-free Edge Deletion *are incompressible, when* $H \in \{\overline{A_6}, \overline{A_7}, \overline{A_8}, \overline{A_9}\}$ *and* H-free Edge Completion *is incompressible when* $H \in \{\overline{A_1}, \overline{A_6}\}$.

Now, Theorem 59(i) follows from Lemma 68, Lemma 70, and Proposition 4. Similarly, Theorem 59(ii) follows from Lemma 68, 70, 69, and Proposition 4. Theorem 59(iii) follows from Theorem 59(ii) and Proposition 4. Theorem 1 follows from Lemma 12, 58, Theorem 59(i), and Proposition 6. Similarly, Theorem 2 follows from Lemma 13, 58, Theorem 59(ii), and Proposition 6.

References

1. N. R. Aravind, R. B. Sandeep, and Naveen Sivadasan. Parameterized lower bounds and dichotomy results for the NP-completeness of H-free edge modification problems. In *Proc. LATIN 2016*, pages 82–95, 2016. doi:10.1007/978-3-662-49529-2_7.
2. N. R. Aravind, R. B. Sandeep, and Naveen Sivadasan. Dichotomy results on the hardness of H-free edge modification problems. *SIAM J. Discrete Math.*, 31(1):542–561, 2017. doi:10.1137/16M1055797.
3. Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.
4. Leizhen Cai and Yufei Cai. Incompressibility of H-free edge modification problems. *Algorithmica*, 71(3):731–757, 2015. doi:10.1007/s00453-014-9937-x.
5. Yufei Cai. Polynomial kernelisation of H-free edge modification problems. Mphil thesis, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR, China, 2012.
6. Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012. doi:10.1007/s00453-011-9595-1.
7. Yixin Cao, Yuping Ke, and Hanchun Yuan. Polynomial kernels for paw-free edge modification problems. In *Proc. TAMC 2020*, pages –, 2020.
8. Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016. doi:10.1007/s00453-015-0014-x.
9. Yixin Cao, Ashutosh Rai, R. B. Sandeep, and Junjie Ye. A polynomial kernel for diamond-free editing. In *Proc. ESA 2018*, pages 10:1–10:13, 2018. doi:10.4230/LIPIcs.ESA.2018.10.
10. Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *arXiv*, 2020. arXiv:2001.06867.
11. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
12. Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, Erik Jan van Leeuwen, and Marcin Wrochna. Polynomial kernelization for removing induced claws and diamonds. *Theory Comput. Syst.*, 60(4):615–636, 2017. doi:10.1007/s00224-016-9689-x.
13. Eduard Eiben, William Lochet, and Saket Saurabh. A polynomial kernel for paw-free editing. *arXiv*, 2019. arXiv:1911.03683.
14. Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.
15. Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Fixed-parameter algorithms for clique generation. In *Proc CIAC 2003*, pages 108–119, 2003. doi:10.1007/3-540-44849-7_17.
16. Sylvain Guillemot, Frédéric Havet, Christophe Paul, and Anthony Perez. On the (non-)existence of polynomial kernels for P_ℓ-free edge modification problems. *Algorithmica*, 65(4):900–926, 2013. doi:10.1007/s00453-012-9619-5.

17 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006. doi:10.1016/j.jcss.2006.02.001.

18 Ken-ichi Kawarabayashi and Bruce A. Reed. Computing crossing number in linear time. In *Proc. STOC 2007*, pages 382–390. ACM, 2007. doi:10.1145/1250790.1250848.

19 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-Complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

20 Brendan McKay. *Graphs*, (accessed June 11, 2017). https://users.cecs.anu.edu.au/ bdm/data/-graphs.html.

21 Mihalis Yannakakis. Edge-deletion problems. *SIAM J. Comput.*, 10(2):297–309, 1981. doi:10.1137/0210021.

22 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981. doi:10.1137/0210022.

Approximating k-Connected m-Dominating Sets

Zeev Nutov
The Open University of Israel, Raanana, Israel
nutov@openu.ac.il

Abstract

A subset S of nodes in a graph G is a **k-connected m-dominating set** (**(k,m)-cds**) if the subgraph $G[S]$ induced by S is k-connected and every $v \in V \setminus S$ has at least m neighbors in S. In the k-Connected m-Dominating Set ((k,m)-CDS) problem the goal is to find a minimum weight (k,m)-cds in a node-weighted graph. For $m \geq k$ we obtain the following approximation ratios. For general graphs our ratio $O(k \ln n)$ improves the previous best ratio $O(k^2 \ln n)$ of [26] and matches the best known ratio for unit weights of [34]. For unit disk graphs we improve the ratio $O(k \ln k)$ of [26] to $\min\left\{\frac{m}{m-k}, k^{2/3}\right\} \cdot O(\ln^2 k)$ – this is the first sublinear ratio for the problem, and the first polylogarithmic ratio $O(\ln^2 k)/\epsilon$ when $m \geq (1+\epsilon)k$; furthermore, we obtain ratio $\min\left\{\frac{m}{m-k}, \sqrt{k}\right\} \cdot O(\ln^2 k)$ for uniform weights. These results are obtained by showing the same ratios for the Subset k-Connectivity problem when the set of terminals is an m-dominating set.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases k-connected graph, m-dominating set, approximation algorithm, rooted subset k-connectivity, subset k-connectivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.73

Acknowledgements I thank an anonymous referee for many useful comments.

1 Introduction

All graphs in this paper are assumed to be simple, unless stated otherwise. A (simple) graph is **k-connected** if it has k pairwise internally node disjoint paths between every pair of its nodes; in this case the graph has at least $k+1$ nodes. A subset S of nodes in a graph G is a **k-connected set** if the subgraph $G[S]$ induced by S is k-connected; S is an **m-dominating set** if every $v \in V \setminus S$ has at least m neighbors in S. If S is both k-connected and m-dominating set then S is a **k-connected m-dominating set**, or **(k,m)-cds** for short. A graph is a **unit disk graph** if its nodes can be located in the Euclidean plane such that there is an edge between u and v iff the Euclidean distance between u and v is at most 1. We consider the following problem for $m \geq k$ both in general graphs and in unit disk graphs.

k-Connected m-Dominating Set ((k,m)-CDS)
Input: A graph $G=(V,E)$ with node weights $\{w_v : v \in V\}$ and integers k, m.
Output: A minimum weight (k,m)-cds $S \subseteq V$.

The problem generalizes several classic problems including Set-Cover ($k=0, m=1$), Set-Multicover ($k=0$), and Connected Dominating Set ($k=m=1$). The Connected Dominating Set problem is closely related to the Node Weighted Steiner Tree problem, and both problems admit a tight ratio $O(\log n)$ [16, 12, 13]. In unit disk graphs, the problem is NP-hard [5], admits a PTAS for unit weights [3], and ratio $3+2.5\rho+\epsilon$ for arbitrary weights [33, 35], where ρ is the ratio for the edge-weighted Steiner Tree problem in general graphs. The (k,m)-CDS problem models (fault tolerant) virtual backbones in networks [7, 6], and it was studied extensively, c.f. [1, 3, 10, 12, 13, 21, 32, 26, 33, 31, 34, 35, 36] for the case $m \geq k$ and [2, 29] for the case $k=2, m=1$. For further motivation and history

survey we refer the reader to recent papers of Zhang, Zhou, Mo, and Du [31] and of Fukunaga [10], where they obtained in unit disk graphs ratios $O(k^3 \ln k)$ and $O(k^2 \ln k)$, respectively. This was improved to $O(k \ln k)$ in [26], where is also given ratio $O(k^2 \ln n)$ in general graphs.

Our main results is:

▶ **Theorem 1.** (k, m)-CDS *with* $m \geq k$ *admits the following approximation ratios:*

- $O(k \ln n)$ *in general graphs.*
- $\min\left\{\frac{m}{m-k}, k^{2/3}\right\} \cdot O(\ln^2 k)$ *in unit disk graphs.*
- $\min\left\{\frac{m}{m-k}, \sqrt{k}\right\} \cdot O(\ln^2 k)$ *in unit disk graphs with unit weights.*

For general graphs our ratio $O(k \ln n)$ improves the previous ratio $O(k^2 \ln n)$ of [26] and matches (while using totally different techniques) the best known ratio for unit weights of Zhang et. al. [34]. For unit disk graphs our ratio $\min\left\{\frac{k}{m-k}, k^{2/3}\right\} \cdot O(\ln^2 k)$ improves the previous best ratio $O(k \ln k)$ of [26]; this is the first sublinear ratio for the problem, and for any constant $\epsilon > 0$ and $m = k(1+\epsilon)$ the first polylogarithmic ratio $O(\ln^2 k)/\epsilon$.

Let us say that a graph with a set T of terminals and a root $r \in T$ is **k-(T, r)-connected** if it has k internally node disjoint rt-paths for every $t \in T \setminus \{r\}$. Similarly, a graph is **k-T-connected** if it has k internally node disjoint st-paths for every $s, t \in T$. A reason why the case $m \geq k$ is easier than the case $m < k$ is given in the following statement (a proof can be found in many papers, c.f. [31, 10, 26]).

▶ **Lemma 2.** *Let T be a k-dominating set in a graph $H = (U, F)$. If H is k-(T, r)-connected then H is k-(U, r)-connected; if H is k-T-connected then H is k-connected.*

The above lemma implies that in the case $m \geq k$ (k, m)-CDS has the property that the union $T \cup S$ of a partial solution T that is just m-dominating and a node set S such that $G[T \cup S]$ is T-k-connected, is a feasible solution – this enables to construct the solution iteratively. Specifically, most algorithms for the case $m \geq k$ start by computing just an m-dominating set T; the best ratios for m-Dominating Set are $\ln(\Delta + m)$ in general graphs [8] and $O(1)$ in unit disk graphs [10], where Δ is the maximum degree in G. By invoking just these ratios, Lemma 2 enables to reduce (k, m)-CDS with $m \geq k$ to following (node weighted) problem:

Subset k-Connectivity
Input: A graph $G = (V, E)$ with node-weights $\{w_v : v \in V\}$, a set $T \subseteq V$ of terminals, and an integer k.
Output: A minimum weight k-T-connected subgraph of G.

The ratios for this problem are usually expressed in terms of the best known ratio β for the following problem (in both problems we assume w.l.o.g. that $w_v = 0$ for all $v \in T$):

Rooted Subset k-Connectivity
Input: A graph $G = (V, E)$ with node-weights $\{w_v : v \in V\}$, a set $T \subseteq V$ of terminals, a root node $r \in T$, and an integer k.
Output: A minimum weight k-(T, r)-connected subgraph of G.

We refer the reader to recent surveys [28, 27] on approximation algorithm for node-connectivity problems and to [16, 22, 25, 9, 24] on approximation algorithm for various node-weighted connectivity problems and their generalizations. Currently, $\beta = O(k^2 \ln |T|)$ [22]. From previous work it can be deduced that Subset k-Connectivity with $|T| \geq k$

admits ratio $\beta + k^2$. Add a new root node r connected to a set $R \subseteq T$ of k nodes by edges of cost zero. Then compute a β-approximate solution to the obtained Rooted Subset k-Connectivity instance. Finally, augment this solution by computing for every $u, v \in R$ a min-weight set of k internally disjoint uv-paths. For the (k, m)-CDS problem with $m \geq k$ this already gives ratio $\beta + k^2 = O(k^2 \ln |T|)$ in general graphs. For the special case when T is a k-dominating set the ratio $\beta + k^2$ was improved in [26] to $\beta + k - 1$, since then in the final step it is sufficient to compute a min-weight set of k internally disjoint uv-paths only for pairs that form a forest on R (by the Critical Cycle Theorem of Mader [20]).

We now consider unit disk graphs. Zhang et al. [31] showed that any k-connected unit disk graph has a k-connected spanning subgraph of maximum degree $\leq 5k$. This implies that the node weighted case is reduced with a loss of factor $O(k)$ to the case of node induced edge costs – when $c_{uv} = w_u + w_v$ for every edge $e = uv \in E$. The edge costs version of Subset k-Connectivity admits ratio $O(k^2 \ln k)$, which gives ratio $O(k^3 \ln k)$ for (k, m)-CDS with $m \geq k$ in unit disk graphs. Independently, Fukunaga [10] obtained ratio $O(k^2 \ln k)$ using a different approach – he considered the Rooted Subset Connectivity Augmentation problem, when $G[T]$ is ℓ-(T, r)-connected and we seek a minimum weight $S \subseteq V \setminus T$ such that $G[T \cup S]$ is $(\ell + 1)$-(T, r)-connected. In [22] it is shown that the augmentation problem decomposes into $O(k)$ "uncrossable" subproblems (for precise definitions, see Definition 20 in Section 3), and Fukunaga [10] designed a primal-dual $O(1)$-approximation algorithm for such an uncrossable subproblem in unit disk graphs. This gives ratio $O(\ell)$ for Rooted Subset Connectivity Augmentation in unit disk graphs. Furthermore, using the so called "backward augmentation analysis" [11] Fukunaga showed that since his approximation is w.r.t. an LP, then sequentially increasing the T-connectivity by 1 invokes only a factor of $O(\ln k)$, thus obtaining ratio $O(k \ln k)$ for Rooted Subset Connectivity Augmentation. He then combined this result with a decomposition of the Subset k-Connectivity problem into k Rooted Subset k-Connectivity problems, and obtained ratio $O(k^2 \ln k)$. As was mentioned, in [26] it is proved that ratio β for Rooted Subset k-Connectivity implies ratio $\beta + k - 1$ for (k, m)-CDS with $m \geq k$, which improves the ratio to $O(k \ln k)$.

However, it seems that previous reductions and methods alone do not enable to obtain ratio better than $O(k^2 \ln |T|)$ in general graphs, or a sublinear ratio in unit disk graphs. These algorithm rely on the ratios and decompositions for the Rooted/Subset k-Connectivity problems from [22, 23, 19], but these do not consider the specific feature relevant to (k, m)-CDS with $m \geq k$ – that the set T of terminals is a k-dominating set; note that then Subset k-Connectivity is equivalent to the problem of finding the lightest k-connected subgraph containing T, by Lemma 2. Here we change this situation by asking the following question:

If the set T of terminals is an m-dominating set with $m \geq k$, what approximation ratio can be achieved for (node weighted) Subset k-Connectivity?

Our answer to this question is given in the following theorem, which is of independent interest, and note that it implies Theorem 1.

▶ **Theorem 3.** *The (node weighted)* Subset k-Connectivity *problem such that T is an m-dominating set with $m \geq k$ admits the following approximation ratios: $O(k \ln n)$ in general graphs, $O(\ln^2 k) \cdot \min\left\{\frac{m}{m-k+1}, k^{2/3}\right\}$ in unit disk graphs, and $O(\ln^2 k) \cdot \min\left\{\frac{m}{m-k+1}, \sqrt{k}\right\}$ in unit disk graphs with unit weights.*

In the proof of Theorem 3 we use several results and ideas from previous works [22, 23, 31, 10, 26]. As was mentioned, the best ratios for the Subset k-Connectivity are derived via reductions of [23, 26] from the ratios for the Rooted Subset k-Connectivity problem,

so we will consider the latter problem; the currently best known ratio for this problem is $O(k^2 \ln |T|)$ [22]. The algorithm of [22] has k iterations, where at iteration $\ell = 0, \ldots, k-1$ it considers the augmentation problem of increasing the connectivity from ℓ to $\ell + 1$. This is equivalent to "covering" a certain family $\mathcal{F}$ of "deficient sets" (see Section 2 for precise definitions), and the algorithm of [22] decomposes this problem into $O(\ell)$ uncrossable family covering problems; the ratio for covering each uncrossable family is $O(\ln n)$ in general graphs [22] and $O(1)$ in unit disk graphs [10].

However, a more careful analysis of the [22] algorithm reveals that in fact the number of uncrossable families is $O(\ell/q) + 1$, where q is the minimum number of terminals in a deficient set. Instances with $q \geq \ell + 1$ are often called "T-independence-free" (see Lemmas 6 and 7). In T-independence-free instances the entire family of deficient sets is uncrossable, hence such instances admit ratio $O(\ln n)$ in general graphs and $O(1)$ in unit disk graphs. The algorithm of [22] has an "inflation phase" that works towards reaching $q \geq \ell + 1$ – to make the instance T-independence-free, by repeatedly covering $O(\ell/q)$ uncrossable families to double q. Hence if q_0 is the initial value of q, the total number of uncrossable families that the algorithm covers is 1 plus order of $\frac{\ell}{q_0}\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right) = O(\ell/q_0)$. Note that a large part of the uncrossable families are covered when q is small. One of our contributions is designing different "lighter" inflation algorithms for increasing the parameter q. These algorithms just aim to cover the inclusion minimal deficient sets (a.k.a. "cores" – for precise definition see Definition 14), by adding a light set S of nodes, and then add S to the set T of terminals; if T is a k-dominating set then adding any set S to T does not make the problem harder, by Lemma 2.

Our algorithms for covering inclusion minimal deficient sets reduce the problem to a set covering type problem. In the case of general graphs the reduction is to a special case considered in [18] of the Submodular Covering problem; the ratio invoked by this procedure is only $O(\ln n)$ and if we apply it $p = \max\{2k - m - 1, 1\}$ times then we get $q \geq m - \ell + p(k - \ell) \geq k$ for all $\ell = 0, \ldots, k-1$. In fact, we apply this procedure before considering the augmentation problems, but it guarantees that $q \geq k$ through all augmentation iterations. The same procedure applies in the case of unit disk graphs, but to avoid the dependence on n in the ratio we use a different procedure. Specifically, we use the result of Zhang et. al. [31] that a minimally k-connected unit disk graph has maximum degree $\leq 5k$, to reduce the problem of covering the family of deficient sets to the Set Cover problem with soft capacities. This approach gives ratio $\min\left\{\frac{m}{m-k}, k^{2/3}\right\} \cdot O(\ln^2 k)$.

Our algorithms are simple and combinatorial. We omit the running time analysis, but it is polynomial and dominated by that of finding k times an approximate solution to Rooted Subset k-Connectivity in T-independence-free instances [22, 10]. Apart from significantly improving approximation ratios for (k, m)-CDS and special instances of Subset/Rooted k-Connectivity, which are extensively studied important fundamental problems, we also have the following contribution. The framework of approximating k-connectivity problems via independence-free graphs was very successful in [22] and [4, 30] (see also [14] for the first paper that used this framework, for an exact algorithm), but these are the only papers that succeeded to apply it. In general, it is not clear how to find a cheap partial solution to make the residual instance independence-free. In [22] this was achieved by "merging" deficient sets, so that at each iteration the minimum number of terminals in a deficient set is doubled. The method used in [4, 30] constructs an independence free instance in just two iterations, but it is tailor made for the k-Connected Subgraph problem considered there. We use a different method that reaches an almost independence-free instance by just repeatedly solving

a Set Cover (or a Submodular Cover) problem. We believe that the method used here can be also applied for other problems, maybe for activation network design problems [24, 9], that generalize node weighted network design problems.

In the rest of the paper we prove Theorem 3; Section 2 considers general graphs and Section 3 considers unit disk graphs.

2 General graphs

In order to prove our results we need to characterize k-connectivity in terms of "cuts" rather than in terms of paths. While edge-cuts of a graph correspond to node subsets, a natural way to represent a node-cut of a graph is by a pair of sets called a "biset".

▶ **Definition 4.** *An ordered pair $\mathbb{A} = (A, A^+)$ of subsets of V with $A \subseteq A^+$ is called a* **biset***; the set $\partial\mathbb{A} = A^+ \setminus A$ is called the* **cut** *of $\mathbb{A}$. We say that $\mathbb{A}$ is a* **(T, r)-biset** *if $A \cap T \neq \emptyset$ and $r \in V \setminus A^+$. For an edge set/graph J let $d_J(\mathbb{A})$ denote the number of edges in J that have one end in A and the other in $V \setminus A^+$.*

Let $\kappa_G(t, r)$ denote the maximum number of pairwise internally disjoint tr-paths in G. In biset terms, the node connectivity version of Menger's Theorem (that applies also for non-simple graphs) says that $\kappa_G(t, r)$ equals $\min\{|\partial\mathbb{A}| : t \in A, r \in V \setminus A^+\}$ plus the number of tr-edges. Here $\partial\mathbb{A}$ is a node cut that separates t from r in the graph obtained from G by removing the tr-edges. It is not hard to verify that if J is an edge set that contains the tr-edges then $\kappa_G(t, r) = \min\{|\partial\mathbb{A}| : t \in A, r \in V \setminus A^+\} + d_J(\mathbb{A})$. In particular, we have:

▶ **Lemma 5.** *Let $G = (V, E)$ be a graph and let $\{t, r\} \subseteq T \subseteq V$. Then*

$$\kappa_G(t, r) = \min_{\mathbb{A}}\{|\partial\mathbb{A}| + d_{G[T]}(\mathbb{A}) : t \in A, r \in V \setminus A^+\} .$$

Here the nodes in $\partial\mathbb{A}$ and the edges in $G[T]$ that go from A to $V \setminus A^+$ form a "mixed" st-cut of G that contains both nodes and edges. The original Menger's Theorem is the case $T = \{r, t\}$, while the case $T = V$ is also widely used in the literature, c.f. [28].

From Lemma 5 we get that G is k-(T, r)-connected iff $|\partial\mathbb{A}| + d_{G[T]}(\mathbb{A}) \geq k$ holds for every (T, r)-biset $\mathbb{A}$. Given a Rooted Subset k-Connectivity instance, we say that a (T, r)-biset $\mathbb{A}$ is a **deficient biset** if $|\partial\mathbb{A}| + d_{G[T]}(\mathbb{A}) \leq k - 1$. We use the algorithm from [22] for Rooted Subset k-Connectivity. Two deficient bisets $\mathbb{A}, \mathbb{B}$ are ***T*-independent** if $A \cap T \subseteq \partial\mathbb{B}$ or $B \cap T \subseteq \partial\mathbb{A}$. A Rooted Subset k-Connectivity instance is ***T*-independence-free** if no pair of deficient bisets are T-independent. We have the following from previous work [22].

▶ **Lemma 6** ([22])**.** *T-independence-free Rooted Subset k-Connectivity instances admit ratio $O(k \ln |T|)$.*

Clearly, a sufficient condition for an instance to be T-independence-free is:

▶ **Lemma 7.** *If for a Rooted Subset k-Connectivity instance $|A \cap T| \geq k$ holds for every deficient biset $\mathbb{A}$, then the instance is T-independence-free.*

In the next two lemmas we show how to find an $O(k \ln n)$-approximate set $S \subseteq V \setminus T$ such that adding S to T result in a T-independence-free instance.

▶ **Lemma 8** (Inflation Lemma for general graphs)**.** *There exists a polynomial time algorithm that given an instance of Rooted Subset k-Connectivity finds $S \subseteq V \setminus T$ such that $|A \cap S| \geq k - (|\partial\mathbb{A}| + d_{G[T \cup S]}(\mathbb{A}))$ holds for any (T, r)-biset $\mathbb{A}$, and $w(S) = O(\ln \Delta) \cdot \mathsf{opt}$.*

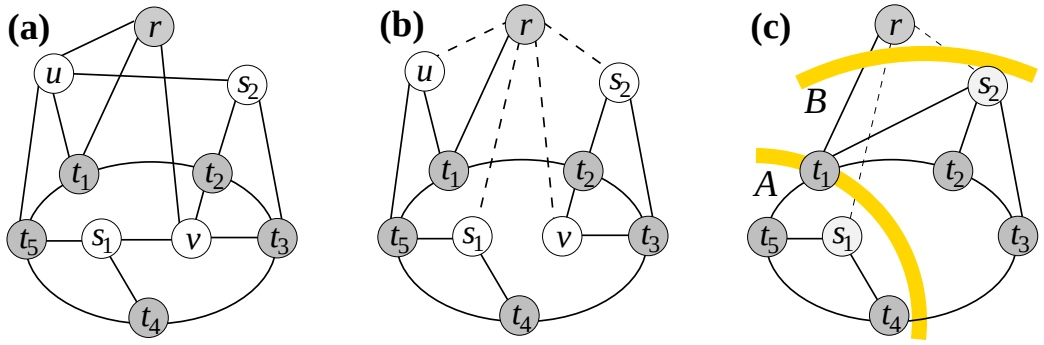


Figure 1 Illustration to Lemma 8 Algorithm 1 for $k = 3$. (a) Rooted Subset k-Connectivity instance; nodes in T are shown by gray circles. (b) Centered Rooted Subset k-Connectivity instance constructed in Algorithm 1; added edges are shown by dashed lines. (c) The set $S = \{s_1, s_2\}$ returned by the algorithm and two bisets $\mathbb{A}, \mathbb{B}$ in $G[T \cup S]$ (the dashed edges do not belong to $G[T \cup S]$), where $A = \{t_5, t_4, s_1\}, \partial\mathbb{A} = \{t_1\}$ and $B = (T \cup S) \setminus \{r\}, \partial\mathbb{B} = \emptyset$.

Proof. The Centered Rooted Subset k-Connectivity problem is a particular case of the Rooted Subset k-Connectivity problem when all nodes of positive weight are neighbors of the root. This problem admits ratio $O(\ln \Delta)$ [18], where here Δ is the maximum degree of a neighbor of the root. We use this in our algorithm as follows (see Fig. 1):

Algorithm 1 $(G = (V, E), w, r, T, k)$.

1 construct a Centered Rooted Subset k-Connectivity instance $(G' = (V, E'), w, T, r, k)$, where G' is obtained from G by removing edges in $G[(V \setminus T) \cup \{r\}]$ and adding an rv-edge for each $v \in V \setminus T$ (see Fig. 1(a,b))
2 compute an $O(\ln \Delta)$-approximate solution $S \subseteq V \setminus T$ for the obtained Centered Rooted Subset k-Connectivity instance (see Fig. 1(c))
3 **return** S

Let S^* and S^*_c be optimal solutions to Rooted Subset k-Connectivity and the constructed Centered Rooted Subset k-Connectivity instances, respectively. For every $t \in T$ fix some set of k internally disjoint rt-paths in the graph $G[T \cup S^*]$, and obtain a set P_t by picking for each path the node in S^* that is closest to t on this path, if such a node exists. Let $P = \cup_{t \in T} P_t$. Then P is a feasible solution to the constructed Centered Rooted Subset k-Connectivity instance, since for each $t \in T$, G' has $|P_t|$ internally disjoint rt-paths of length 2 each that go through P_t, and $k - |P_t|$ paths that have all nodes in T. Furthermore, since $P \subseteq S^*$, $w(P) \leq w(S^*)$. Thus $w(S^*_c) \leq w(P) \leq w(S^*)$, implying that $w(S) = O(\ln \Delta) \cdot w(S^*)$.

Now let $\mathbb{A}$ be a (T, r)-biset on $T \cup S$. Then:

- $d_{G'[T \cup S]}(\mathbb{A}) = |A \cap S| + d_{G[T \cup S]}(\mathbb{A})$ by the construction.
- $|\partial\mathbb{A}| + d_{G'[T \cup S]}(\mathbb{A}) \geq k$ since $G'[T \cup S]$ is k-(T, r)-connected.

Combining we get that $|\partial\mathbb{A}| + d_{G[T \cup S]}(\mathbb{A}) + |A \cap S| \geq k$, as claimed. ◀

Note that Lemma 8 does *not* assume that T has any domination properties, and it does *not* imply that $G[T \cup S]$ has higher (T, r)-connectivity than $G[T]$ – see the example in Fig. 1. The lemma just states that for every biset $\mathbb{A}$ in $G[T \cup S]$, $|A \cap S|$ is at least the "deficiency" $k - (|\partial\mathbb{A}| + d_{G[T \cup S]}(\mathbb{A}))$ of $\mathbb{A}$. E.g., in the example in Fig. 1(c) we have:

- $A \cap S = \{s_1\}$, $\quad k - (|\partial\mathbb{A}| + d_{G[T \cup S]}(\mathbb{A})) = 3 - (1 + 1) = 1$.
- $B \cap S = \{s_1, s_2\}$, $k - (|\partial\mathbb{B}| + d_{G[T \cup S]}(\mathbb{B})) = 3 - (0 + 1) = 2$.

Hence if we add S to T and $T \leftarrow T \cup S$ will become the new set of terminals, then the new Rooted Subset k-Connectivity instance will be "closer" to being T-independence-free than the original instance. And if also T is a k-dominating set (this is not the case in Fig. 1), then adding S to T does not increase the optimal solution value, by Lemma 2.

Our algorithms use the following simple procedure – Algorithm 2, that sequentially adds p sets $S_1, \ldots, S_p$ to an m-dominating set $T = T_0$ with $m \geq k$; in the case of general graphs considered in this section, each S_i is as in Lemma 8.

Algorithm 2 $(G = (V, E), c, r, T = T_0, k, 1 \leq p \leq k-1)$.

```
1 for i = 1 to p do
2 |  T ← T ∪ S_i
3 return T
```

▶ **Lemma 9.** *Suppose that we are given a Rooted Subset k-Connectivity instance such that T is an m-dominating set in G with $m \geq k$. If at each iteration i at step 2 of Algorithm 2 we add to T a set $S = S_i$ as in Lemma 8, then at the end of the algorithm $w(T \setminus T_0) = O(p \ln \Delta) \cdot \mathsf{opt}$, and $|A \cap T| \geq m - \ell + p(k - \ell)$ holds for any biset $\mathbb{A}$ on T with $|\partial\mathbb{A}| + d_{G[T]}(\mathbb{A}) = \ell \leq k - 1$. In particular, if $p \geq \max\{2k - m - 1, 1\}$ then the resulting instance is T-independence-free.*

Proof. The bound $w(T \setminus T_0) = O(p \ln \Delta) \cdot \mathsf{opt}$ follows from Lemma 2 and the bound $w(S) = O(\ln \Delta) \cdot \mathsf{opt}$ in Lemma 8.

Let $\mathbb{A}$ be a biset as in the lemma. Let $T_i = T_0 \cup S_1 \cup \cdots \cup S_i$ be the set stored in T at the end of iteration i, where T_0 is the initial set. Applying Lemma 8 on T_{i-1} and S_i we get

$$|A \cap S_i| \geq k - (|\partial\mathbb{A} \cap T_{i-1}| + d_{G[T_i]}(\mathbb{A})) \geq k - (|\partial\mathbb{A}| + d_{G[T]}(\mathbb{A})) = k - \ell \ .$$

In particular $A \cap S_1 \neq \emptyset$. Any $v \in A \cap S_1$ has in $G[T]$ at least m neighbors in T_0, and at most ℓ of them are not in A; thus v has at least $m - \ell$ neighbors in $A \cap T_0$, so $|A \cap T_0| \geq m - \ell$. Since $T_0, S_1, \ldots, S_p$ are pairwise disjoint we get $|A \cap T| \geq |A \cap T_0| + \sum_{i=1}^{p} |A \cap S_i| \geq m - \ell + p(k - \ell)$. If $p \geq \max\{2k - m - 1, 1\}$ then $m - \ell + p(k - l) \geq k$; thus, by Lemma 7, the resulting instance is T-independence-free. ◀

The proof of the following known statement can be found in [17], and the second part follows from Mader's Undirected Critical Cycle Theorem [20].

▶ **Lemma 10.** *Let $H_r = (U, F)$ be a k-(U, r)-connected graph and R the set of neighbors of r in H_r. The graph $H = H_r \setminus \{r\}$ can be made k-connected by adding a set J of new edges on R, and if J is inclusion minimal then J is a forest.*

Note that an inclusion minimal edge set J as in Lemma 10 can be computed in polynomial time, by starting with J being a clique on R and repeatedly removing from J an edge e if $H \cup (J \setminus e)$ remains k-connected.

Our algorithm for general graphs is as follows.

Algorithm 3 $(G=(V,E),w,T)$ **general graphs.**

1 construct a graph G_r by adding to G and to T a new node r connected to a set $R \subseteq T$ of k nodes by a set $F_r = \{rv : v \in R\}$ of new edges
2 apply the Lemma 9 algorithm with $p = \max\{2k-m-1, 0\}$
3 use the algorithm from Lemma 6 to compute an $O(k \ln n)$-approximate set $S \subseteq V \setminus T$ such that $H_r = G_r[T \cup S]$ is k-(T,r)-connected
4 let $H = H \setminus \{r\} = G[T \cup S]$ and let J be a forest of new edges on R as in Lemma 10 such that the graph $H \cup J$ is k-connected
5 for every $uv \in J$ find a minimum weight node set P_{uv} such that $G[T \cup S \cup P_{uv}]$ has k internally disjoint uv-paths; let $P = \bigcup_{uv \in J} P_{uv}$
6 return $T \cup S \cup P$

Except step 2, the algorithm is identical to the algorithm of [26] – the only difference is that step 2 improves the factor invoked by step 3. In [26] it is also proved that at the end of the algorithm $T \cup S \cup P$ is a k-connected set. The dominating terms in the ratio are invoked by steps 2 and 3, and they are both $O(k \ln n)$, while step 5 invokes just ratio $k-1$; thus the overall ratio is $O(k \ln n)$.

This concludes the proof of Theorem 3 for general graphs.

3 Unit disk graphs

Our goal in this section is to prove the following:

▶ **Lemma 11.** *Consider a* Subset k-Connectivity *instance on a unit disk graph $G=(V,E)$ where T is an (ℓ,m)-cds in G (so $G[T]$ is ℓ-connected and every $v \in V \setminus T$ has at least m neighbors in T), $m \geq k \geq \ell+1$. Then for any $1 \leq p \leq \ell+1$ there exists a polynomial time algorithm that computes $S \subseteq V \setminus T$ such that $G[T \cup S]$ is $(\ell+1)$-connected and*

$$\frac{w(S)}{\mathsf{opt}} = \frac{O(\ln k)}{k-\ell}\left(p + \frac{(m+p)^2}{(m+p-\ell)^2}\right) .$$

Furthermore, in the case of unit weights $\frac{w(S)}{\mathsf{opt}} = \frac{O(\ln k)}{k-\ell}\left(p + \frac{m+p}{m+p-\ell}\right)$.

Let us show that Lemma 11 implies the unit disk part of Theorem 3. We can apply the Lemma 11 algorithm sequentially, starting with an $O(1)$-approximate m-dominating set $T = T_0$, and at iteration $\ell = 0, \ldots, k-1$ add to T a set $S = S_\ell$ as in the lemma. In the case of arbitrary weights choosing $p=1$ if $m - \ell \geq \ell^{2/3}$ and $p = \ell^{2/3}$ otherwise gives $\frac{w(S_\ell)}{\mathsf{opt}} = \frac{O(\ln \ell)}{k-\ell} \min\left\{\frac{m}{m-\ell}, \ell^{2/3}\right\}$. Then denoting $S = S_0 \cup \cdots \cup S_{k-1}$ we get:

$$\frac{w(S)}{\mathsf{opt}} = \sum_{\ell=0}^{k-1} \frac{O(\ln \ell)}{k-\ell} \min\left\{\frac{m}{m-\ell}, \ell^{2/3}\right\} = O(\ln^2 k) \cdot \min\left\{\frac{m}{m-k+1}, k^{2/3}\right\}$$

In the case of unit weights, choosing $p=1$ if $m - \ell \geq \sqrt{\ell}$ and $p = \sqrt{\ell}$ otherwise gives $\frac{w(S_\ell)}{\mathsf{opt}} = \frac{O(\ln \ell)}{k-\ell} \min\left\{\frac{m}{m-\ell}, \sqrt{\ell}\right\}$, and then by a similar analysis we get that

$$\frac{w(S)}{\mathsf{opt}} = O(\ln^2 k) \cdot \min\left\{\frac{m}{m-k+1}, \sqrt{k}\right\} .$$

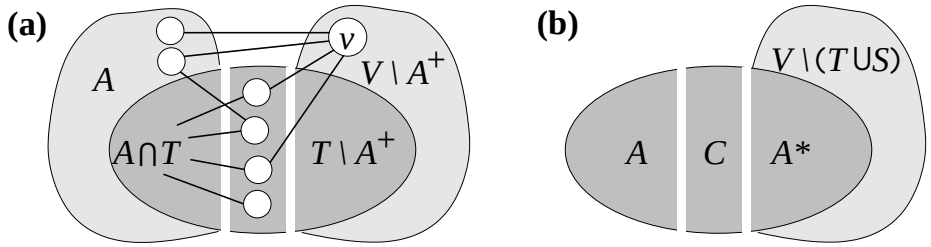


Figure 2 (a) Illustration to the definition of a biset $\mathbb{A} \in \mathcal{D}_T$ and a node v that covers $\mathbb{A}$. (b) Illustration to the proof that if $G[T \cup S]$ is not $(\ell+1)$-connected then S does not cover $\mathcal{D}_T$.

In the rest of this section we prove Lemma 11, so let G, T, and ℓ be as in the lemma; in particular, $G[T]$ is ℓ-connected. Define the following family of biset on V (see Fig. 2)

$$\mathcal{D}_T = \{\mathbb{A} : A \cap T \neq \emptyset \neq T \setminus A^+, d_{G[T]}(\mathbb{A}) = 0, |\partial\mathbb{A}| = \ell\} .$$

Let $\mathbb{A} \in \mathcal{D}_T$. Then $G[T] \setminus \partial\mathbb{A}$ is disconnected (since $d_{G[T]}(\mathbb{A}) = 0$), hence $\partial\mathbb{A} \cap T$ is a node cut of $G[T]$ that separates between the non-empty sets $A \cap T$ and $T \setminus A^+$. Since $|\partial\mathbb{A}| = \ell$ and since G is ℓ-connected, $\partial\mathbb{A}$ must be a minimum node cut of $G[T]$. Thus we have:

▶ **Corollary 12.** *A biset $\mathbb{A}$ on V belongs to $\mathcal{D}_T$ if and only if the following holds:*
- *$\partial\mathbb{A} \subseteq T$ and $\partial\mathbb{A}$ is a minimum node cut (of size ℓ) of the graph $G[T]$.*
- *$A \cap T$ is a union of some, but not all, connected components of $G[T] \setminus \partial\mathbb{A}$.*

We say that a **node v covers $\mathbb{A} \in \mathcal{D}_T$** (see Fig. 2(a)) if $v \in \Gamma(A) \setminus T$, where $\Gamma(A) \subseteq V \setminus A$ is the set of neighbors of A in G; $S \subseteq V \setminus T$ covers $\mathcal{F} \subseteq \mathcal{D}_T$ if every $\mathbb{A} \in \mathcal{F}$ is covered by some $v \in S$. Using Menger's Theorem and Lemma 2, one can see the following (see also [10, Section 5.2]).

▶ **Lemma 13.** *Let T be an $(\ell, \ell+1)$-cds in a graph $G = (V, E)$. Let $S \subseteq V \setminus T$. Then $G[T \cup S]$ is $(\ell+1)$-connected if and only if S covers $\mathcal{D}_T$.*

Proof. Let $\mathbb{A} \in \mathcal{D}_T$. Then $\partial\mathbb{A} \subseteq T$ is a node cut of size ℓ of the graph $G[T]$ that separates between the nonempty sets $A \cap T$ and $T \setminus A^+$. If S does not cover $\mathbb{A}$, then $S \cap \Gamma(A) = \emptyset$, and then $\partial\mathbb{A}$ is also a node cut of size ℓ of the graph $G[T \cup S]$ that separates between $A \cap T$ and $(T \setminus A^+) \cup S$; thus $G[T \cup S]$ is not ℓ-connected. Consequently, if $G[T \cup S]$ is $(\ell+1)$-connected then S must cover $\mathcal{D}_T$.

Suppose that $G[T \cup S]$ is not $(\ell+1)$-connected (see Fig. 2(b)). Then there is a node cut C of size ℓ of $G[T \cup S]$ that separates some $A \subseteq T \cup S$ from $A^* = (T \cup S) \setminus (A \cup C)$. Since T is an $(\ell+1)$-dominating set in $G[T \cup S]$ and since $|C| = \ell$, we must have $A \cap T \neq \emptyset$; otherwise, if $A \cap T = \emptyset$, then for any $u \in A$, there must exist $u' \in A^*$ with $uu' \in E$ since T is an $(\ell+1)$-dominating set, which contradicts that C is a node cut separating A from A^*. By a similar argument, $A^* \cap T \neq \emptyset$. Let $\mathbb{A} = (A, A \cup C)$. Then $A \cap T \neq \emptyset \neq T \setminus A^+$, $d_{G[T]}(\mathbb{A}) = 0$ (since $d_{G[T \cup S]}(\mathbb{A}) = 0$), and $|\partial\mathbb{A}| = |C| = \ell$. Thus $\mathbb{A} \in \mathcal{D}_T$. Furthermore, in $G[T]$, $C \cap T$ separates between the nonempty node sets $A \cap T$ and $T \setminus A^+$, hence $C = \partial\mathbb{A} = \Gamma(A) \subseteq T$, since $G[T]$ is ℓ-connected. Consequently, S does not cover $\mathbb{A}$, and the proof is complete. ◀

Thus we have the following LP-relaxation for the problem of finding a min-weight cover of $\mathcal{D}_T$ (a similar LP was used by Fukunaga in [10, Section 5.2]):

$$\begin{array}{lll} \tau(\mathcal{D}_T) = & \min & \sum_{v \in V \setminus T} w_v x_v \\ & \text{s.t.} & \sum_{v \in \Gamma(A) \setminus T} x_v \geq 1 \qquad \forall A \in \mathcal{D}_T \\ & & x_v \geq 0 \qquad\qquad\quad \forall v \in V \setminus T \end{array}$$

Note that if $\mathbb{A} \in \mathcal{D}_T$ then $\Gamma(A) \setminus T = \Gamma(A) \setminus \partial\mathbb{A}$, and thus the constraint $\sum_{v \in \Gamma(A)\setminus T} x_v \geq 1$ is equivalent to $\sum_{v \in \Gamma(A)\setminus \partial\mathbb{A}} x_v \geq 1$.

▶ **Definition 14.** *We say that* $\mathbb{A}$ ***contains*** $\mathbb{B}$ *and write* $\mathbb{A} \subseteq \mathbb{B}$ *if* $A \subseteq B$ *and* $A^+ \subseteq B^+$. *Inclusion minimal members of a biset family* $\mathcal{F}$ *are called* $\mathcal{F}$***-cores***. *The* ***intersection*** *and the* ***union*** *of two bisets* $\mathbb{A}, \mathbb{B}$ *are defined by*

$$\mathbb{A} \cap \mathbb{B} = (A \cap B, A^+ \cap B^+) \qquad \mathbb{A} \cup \mathbb{B} = (A \cup B, A^+ \cup B^+) \ .$$

▶ **Lemma 15.** *Let* $\mathcal{C}$ *be the family of* $\mathcal{D}_T$*-cores. Then* $C \cap C' = \emptyset$ *for any distinct* $\mathbb{C}, \mathbb{C}' \in \mathcal{C}$ *or* $|\mathcal{C}| \leq \ell(\ell+1)$.

Proof. let $\mathcal{F}$ be the family of "deficient bisets" of the ℓ-connected graph $G[T]$; namely, $\mathcal{F}$ is the family of those bisets $\mathbb{A}$ on T such that $\partial\mathbb{A}$ is a minimum node cut (of size ℓ) of the graph $G[T]$, and A is a union of some, but not all, connected components of $G[T] \setminus \partial\mathbb{A}$. Note that from Corollary 12 it follows that:

- $\mathcal{F} = \{\mathbb{A} \in \mathcal{D}_T : A \subseteq T\}$.
- Every $\mathcal{D}_T$-core belongs to $\mathcal{F}$, hence $\mathcal{C}$ coincides with the family of $\mathcal{F}$-cores.

Thus it is sufficient to prove the lemma with $\mathcal{D}_T$ replaced by $\mathcal{F}$. The family $\mathcal{F}$ has the following well known "symmetry" and "crossing" properties, c.f. [30].

(i) If $\mathbb{A} \in \mathcal{F}$ then $(V \setminus A^+, V \setminus A) \in \mathcal{F}$.
(ii) If $\mathbb{A}, \mathbb{B} \in \mathcal{F}$ and $A \cap B, T \setminus (A^+ \cup B^+)$ are non-empty then $\mathbb{A} \cap \mathbb{B}, \mathbb{A} \cup \mathbb{B} \in \mathcal{F}$.

In [[15], Lemma 3.5, Case II] it is proved that if $C \cap C' \neq \emptyset$ for some distinct $\mathbb{C}, \mathbb{C}' \in \mathcal{C}$ then there is $P \subseteq V$ with $|P| \leq \ell + 1$ such that $P \cap C \neq \emptyset$ for every $\mathbb{C} \in \mathcal{C}$. In this case $\mathcal{F}$ has at most $\ell(\ell+1)$ distinct cores, since:

- For every $\mathbb{C} \in \mathcal{C}$, there is $s \in P \cap C$, and there is $t \in P \cap (V \setminus C^+)$, by (i).
- For each $(s,t) \in P \times P$ there is at most one such $\mathbb{C}$, by (ii).

Hence if $C \cap C' \neq \emptyset$ for some distinct $\mathbb{C}, \mathbb{C}' \in \mathcal{C}$ then $|\mathcal{C}| \leq |P|(|P|-1) \leq \ell(\ell+1)$. ◀

To obtain an approximation ratio that depends on k rather than on n, we will need the following result.

▶ **Theorem 16** (Zhang, Zhou, Mo, & Du [31])**.** *Any* k*-connected unit disk graph has a* k*-connected spanning subgraph of maximum degree* $\leq 5k$.

▶ **Lemma 17** (Inflation Lemma for unit disk graphs)**.** *There exists a polynomial time algorithm that computes* $S \subseteq V \setminus T$ *that covers the family* $\mathcal{C}$ *of cores of* $\mathcal{D}_T$ *and* $w(S) = O(\ln k) \cdot \frac{\mathsf{opt}}{k-\ell}$.

Proof. The problem of covering $\mathcal{C}$ is essentially a (weighted) Set Cover problem where for each $v \in V \setminus T$ the corresponding set has weight w_v and consists of the cores covered by v. Then the greedy algorithm for Set Cover computes a solution of weight $O(\ln |\mathcal{C}|)$ times the value of the standard Set Cover LP

$$\begin{array}{rlll} \tau(\mathcal{C}) = & \min & \sum_{v \in V\setminus T} w_v x_v & \\ & \text{s.t.} & \sum_{v \in \Gamma(A)\setminus T} x_v \geq 1 & \forall A \in \mathcal{C} \\ & & x_v \geq 0 & \forall v \in V \setminus T \end{array}$$

For any $S' \subseteq V \setminus T$ such that $G[T \cup S']$ is k-connected, any set A has at least $k - \ell$ neighbors in $G[T \cup S']$, hence if x' is a characteristic vector of S' then $\frac{x'}{k-\ell}$ is a feasible solution to the LP. Consequently, $\tau(\mathcal{C}) \leq \frac{\mathsf{opt}}{k-\ell}$.

In the case $|\mathcal{C}| \leq \ell(\ell+1)$ we get a solution of weight $O(\ln \ell) \cdot \tau(\mathcal{C}) = O(\ln \ell) \cdot \frac{\mathsf{opt}}{k-\ell}$.

In the case $|\mathcal{C}| > \ell(\ell+1)$, $C \cap C' = \emptyset$ for any $\mathbb{C}, \mathbb{C}' \in \mathcal{C}$, by Lemma 15. Then relying on Theorem 16 we modify this reduction such that every $v \in V \setminus T$ can cover at most $5k$ cores; this is essentially the Set Cover with (soft) capacities problem. Specifically, for each pair (v, J) where $v \in V \setminus S$ and J is a set of at most $5k$ edges incident to v, we add a new node v_J of weight w_v with corresponding copies of the edges in J. In the obtained Set Cover instance the maximum size of a set is at most $5k$, since the sets in $\{C : \mathbb{C} \in \mathcal{C}\}$ are pairwise disjoint. Note that we do not need to construct this Set Cover instance explicitly to run the greedy algorithm – we just need to determine for each $v \in V$ the maximum number of at most $5k$ not yet covered cores that can be covered by v. Since the sets in $\{C : \mathbb{C} \in \mathcal{C}\}$ are pairwise disjoint, this can be done in polynomial time. Note that during the greedy algorithm we may pick pairs (v, J) and (v, J') with distinct J, J' but with the same node v, but this only makes the solution lighter. Since in the Set Cover instance the maximum set size is $5k$, the computed solution has weight $O(\ln k) \cdot \tau$, where here τ is an optimal LP-value of the modified instance. Now we argue in the same way as before that $\tau \leq \frac{\mathsf{opt}}{k-\ell}$. Consider a feasible solution $S' \subseteq V \setminus T$ and an edge J' such that $G[T] \cup S' \cup J'$ is a spanning k-connected subgraph of $G[T \cup S']$ and $\deg_{J'}(v) \leq 5k$ for all $v \in S'$; such J' exists by Theorem 16. Let x' be the characteristic vector of the pairs (v, J'_v) where $v \in S'$ and J'_v is the set of edges in J' incident to v. Any set A has at least $k - \ell$ neighbors in $G[T] \cup S' \cup J'$, hence $\frac{x'}{k-\ell}$ is a feasible solution to the LP. Consequently, $\tau \leq \frac{\mathsf{opt}}{k-\ell}$. ◀

▶ **Corollary 18.** *If at step 2 of Algorithm 2 we add $S = S_i$ is as in Lemma 17, then at the end of the algorithm $w(T \setminus T_0) = O(p \ln k) \cdot \tau^*$ and $|A \cap T| \geq m - \ell + p$ holds for any $\mathbb{A} \in \mathcal{D}_T$.*

Proof. We have $|A \cap S_i| \geq 1$ for all i. In particular $A \cap S_1 \neq \emptyset$. Any $v \in A \cap S_1$ has in $G[T]$ at least m neighbors in T_0, and at most ℓ of them are not in A; thus v has at least $m - \ell$ neighbors in $A \cap T_0$, so $|A \cap T_0| \geq m - \ell$. Since $T_0, S_1, \ldots, S_p$ are pairwise disjoint we get $|A \cap T| \geq |A \cap T_0| + \sum_{i=1}^{p} |A \cap S_i| \geq m - \ell + p$. ◀

Now we decompose the problem of covering $\mathcal{D}_T$ into several subproblems. For $r \in T$ let $\mathcal{D}_{(T,r)} = \{\mathbb{A} \in \mathcal{D}_T : r \in T \setminus A^+\}$; we note the sets in $\{A : \mathbb{A} \in \mathcal{D}_{(T,r)}\}$ are called "demand cuts" by Fukunaga [10, Section 5.2].

▶ **Theorem 19** ([23]). *Given an ℓ-T-connected graph with $|T| \geq \ell + 1$, one can find in polynomial time $R \subseteq T$ of size $|R| = O\left(\frac{|T|}{|T|-\ell} \ln \ell\right)$ such that $\mathcal{D}_T = \cup_{r \in R} \mathcal{D}_{(T,r)}$.*

We now describe how to cover the family $\mathcal{D}_{(T,r)}$ for given $r \in T$.

▶ **Definition 20.** *The biset $\mathbb{A} \setminus \mathbb{B}$ is defined by $\mathbb{A} \setminus \mathbb{B} = (A \setminus B^+, A^+ \setminus B)$. We say that a biset family $\mathcal{F}$ is:*

- ***uncrossable*** *if $\mathbb{A} \cap \mathbb{B}, \mathbb{A} \cup \mathbb{B} \in \mathcal{F}$ or if $\mathbb{A} \setminus \mathbb{B}, \mathbb{B} \setminus \mathbb{A} \in \mathcal{F}$ for all $\mathbb{A}, \mathbb{B} \in \mathcal{F}$.*
- ***T-intersecting*** *if $\mathbb{A} \cap \mathbb{B}, \mathbb{A} \cup \mathbb{B} \in \mathcal{F}$ for any $\mathbb{A}, \mathbb{B} \in \mathcal{F}$ with $A \cap B \cap T \neq \emptyset$.*
- ***T-co-crossing*** *if $\mathbb{A} \setminus \mathbb{B}, \mathbb{B} \setminus \mathbb{A} \in \mathcal{F}$ for any $\mathbb{A}, \mathbb{B} \in \mathcal{F}$ with $A \cap B^* \cap T \neq \emptyset$ and $B \cap A^* \cap T \neq \emptyset$.*

▶ **Lemma 21** ([22, 10]). *$\mathcal{D}_{(T,r)}$ is T-intersecting and T-co-crossing for any $r \in T$.*

▶ **Theorem 22** ([22]). *There exists a polynomial time algorithm that given a T-intersecting T-co-crossing biset family $\mathcal{F}$ sequentially finds $O\left(\frac{q+\ell}{q}\right)$ T-intersecting uncrossable subfamilies of $\mathcal{F}$, such that the union of covers of these subfamilies covers $\mathcal{F}$, where $q = \min\{|A \cap T| : \mathbb{A} \in \mathcal{F}\}$ and $\ell = \max_{\mathbb{A} \in \mathcal{F}} |\partial \mathbb{A} \cap T|$.*

In [10, Theorem 3 and Proof of Corollary 3] Fukunaga proved the following.

▶ **Theorem 23** (Fukunaga [10]). *If $\mathcal{F}$ is a T-intersecting uncrossable subfamily of $\mathcal{D}_{(T,r)}$ then there exists a polynomial time algorithm that computes a cover S of $\mathcal{F}$ of weight $w(S) \leq \frac{15\mathsf{opt}}{k-\ell}$.*

Combining Lemma 21 with Theorems 22 and 23 we get:

▶ **Corollary 24.** *For any $r \in T$, there exists a polynomial time algorithm that computes a cover S_r of $\mathcal{D}_{(T,r)}$ such that if $q = \min\{|A \cap T| : \mathbb{A} \in \mathcal{D}_{(T,r)}\}$ then*

$$\frac{w(S_r)}{\mathsf{opt}} = O\left(\frac{q+\ell}{q(k-\ell)}\right) .$$

The algorithm for unit disk graphs is as follows.

Algorithm 4 $(G=(V,E), w, T, p)$ **unit disk graphs**.

1 apply Algorithm 2 where at step 2 each S_i is as in Lemma 17
2 **if** $G[T]$ is $(\ell+1)$ connected then $S \leftarrow \emptyset$
3 **else** {comment: now $|T| \geq m+p$ and $|A \cap T| \geq m+p-\ell$ for all $\mathbb{A} \in \mathcal{D}_T$}
4 find a set R of $O\left(\frac{|T|}{|T|-\ell} \ln \ell\right)$ roots as in Theorem 19
5 for each $r \in R$ compute a cover S_r of $\mathcal{D}_{(T,r)}$ as in Corollary 24
6 $S \leftarrow \cup_{r \in R} S_r$
7 **return** $T \cup S$

We bound the weight of each of the sets computed. Let T_0 denote the initial set stored in T. By Lemma 17, at the end of step 1 we have

$$\frac{w(T \setminus T_0)}{\mathsf{opt}} = \frac{O(\ln k)}{k-\ell} \cdot p$$

Now we bound the weight of the set S computed in steps 3 to 6:

$$\frac{w(S)}{\mathsf{opt}} = |R| \cdot O\left(\frac{q+\ell}{q(k-\ell)}\right) = \frac{O(\ln \ell)}{k-\ell} \frac{|T|}{|T|-\ell} \frac{q+\ell}{q} = \frac{O(\ln k)}{k-\ell} \frac{(m+p)^2}{(m+p-\ell)^2}$$

The last equation is since $q = \min\{|A \cap T| : \mathbb{A} \in \mathcal{F}\} \geq m-\ell+p$ by Corollary 18 and since $|T| \geq m+p$. The overall weight of the augmenting set computed is as claimed in Lemma 11.

In the case of unit weights, we add arbitrary ℓ nodes to T; this step invokes an additive term of $O(1)$ to the ratio, and $|T| \geq 2\ell+1$ holds after this step. Hence by Theorem 19 we will have $|R| = O\left(\frac{|T|}{|T|-\ell} \ln \ell\right) = O\left(\frac{2\ell+1}{\ell+1} \ln \ell\right) = O(\ln \ell)$ and thus

$$\frac{w(S)}{\mathsf{opt}} = |R| \cdot O\left(\frac{q+\ell}{q(k-\ell)}\right) = \frac{O(\ln \ell)}{k-\ell} \frac{q+\ell}{q} = \frac{O(\ln k)}{k-\ell} \frac{m+p}{m+p-\ell} .$$

The overall weight of the augmenting set computed is as claimed in Lemma 11.

This concludes the proof of Lemma 11, and thus also the proof of Theorem 3.

References

1. C. Ambühl, T. Erlebach, M. Mihalák, and M. Nunkesser. Constant-factor approximation for minimum weight (connected) dominating sets in unit disk graphs. In *APPROX-RANDOM*, pages 3–14, 2006.
2. A. Belgi and Z. Nutov. An $\tilde{O}(\log^2 n)$-approximation algorithm for 2-edge-connected dominating set. CoRR abs/1912.09662, 2019. `arXiv:1912.09662`.
3. X. Cheng, X. Huang, D. Li, W. Weili, and D.-Z. Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks*, 42(4):202–208, 2003.
4. J. Cheriyan and L Végh. Approximating minimum-cost k-node connected subgraphs via independence-free graphs. *SIAM J. Comput.*, 43(4):1342–1362, 2014. Preliminary version in FOCS 2013.
5. B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.
6. B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *IEEE International Conference on Comunications, Montreal*, page 376–380, 1997.
7. A. Ephremides, J. E. Wieselthier, and D. J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE*, 75(1):56–73, 1987.
8. K.-T. Förster. Approximating fault-tolerant domination in general graphs. In *ANALCO*, pages 25–32, 2013.
9. T. Fukunaga. Spider covers for prize-collecting network activation problem. *ACM Trans. Algorithms*, 13(4):49:1–49:31, 2017. preliminary version in SODA 2015.
10. T. Fukunaga. Approximation algorithms for highly connected multi-dominating sets in unit disk graphs. *Algorithmica*, 80(11):3270–3292, 2018.
11. M. Goemans, A. Goldberg, S. Plotkin, D. Shmoys, E. Tardos, and D. Williamson. Improved approximation algorithms for network design problems. In *SODA*, pages 223–232, 1994.
12. S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998. preliminary version in ESA 1996.
13. S. Guha and S. Khuller. Improved methods for approximating node weighted steiner trees and connected dominating sets. *Inf. Comput.*, 150(1):57–74, 1999.
14. B. Jackson and T. Jordán. Independence free graphs and vertex connectivity augmentation. *J. Comb. Theory, Ser. B*, 94(1):31–77, 2005. Preliminary version in IPCO 2001.
15. T. Jordán. On the optimal vertex-connectivity augmentation. *J. Combin. Theory Ser. B*, 63:8–20, 1995.
16. P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *J. Algorithms*, 19(1):104–115, 1995. Preliminary version in IPCO 1993.
17. G. Kortsarz and Z. Nutov. Approximating node connectivity problems via set covers. *Algorithmica*, 37:75–92, 2003. Preliminary version in APPROX 2000.
18. G. Kortsarz and Z. Nutov. Approximating source location and star survivable network problems. *Theoretical Computer Science*, 674:32–42, 2017.
19. B. Laekhanukit. An improved approximation algorithm for the minimum cost subset k-connected subgraph problem. *Algorithmica*, 72(3):714–733, 2015. Preliminary version in ICALP 2011.
20. W. Mader. Ecken vom grad n in minimalen n-fach zusammenhängenden graphen. *Archive der Mathematik*, 23:219–224, 1972.
21. M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995.
22. Z. Nutov. Approximating minimum cost connectivity problems via uncrossable bifamilies. *ACM Transactions on Algorithms*, 9(1):1:1–1:16, 2012. Preliminary version in FOCS 2009.
23. Z. Nutov. Approximating subset k-connectivity problems. *J. Discrete Algorithms*, 17:51–59, 2012.

24 Z. Nutov. Activation network design problems. In T. F. Gonzalez, editor, *Handbook on Approximation Algorithms and Metaheuristics, Second Edition*, volume 2, chapter 12. Chapman & Hall/CRC, 2018.

25 Z. Nutov. Erratum: Approximating minimum cost connectivity problems via uncrossable bifamilies. *ACM Transactions on Algorithms*, 14(3):37:1–37:8, 2018.

26 Z. Nutov. Improved approximation algorithms for k-connected m-dominating set problems. *Information Processing Letters*, 140:30–33, 2018.

27 Z. Nutov. The k-connected subgraph problem. In T. F. Gonzalez, editor, *Handbook on Approximation Algorithms and Metaheuristics, Second Edition*, volume 2, chapter 15. Chapman & Hall/CRC, 2018.

28 Z. Nutov. Node-connectivity survivable network problems. In T. F. Gonzalez, editor, *Handbook on Approximation Algorithms and Metaheuristics, Second Edition*, volume 2, chapter 13. Chapman & Hall/CRC, 2018.

29 Z. Nutov. 2-node-connectivity network design. CoRR abs/2002.04048, 2020. `arXiv:2002.04048`.

30 Z. Nutov. A $4+\epsilon$ approximation for k-connected subgraphs. In *SODA*, pages 1000–1009, 2020.

31 Y. Shi, Z. Zhang, Y. Mo, and D.-Z. Du. Approximation algorithm for minimum weight fault-tolerant virtual backbone in unit disk graphs. *IEEE/ACM Transactions on networking*, 25(2):925–933, 2017.

32 M. Thai, N. Zhang, R. Tiwari, and X. Xu. On approximation algorithms of k-connected m-dominating sets in disk graphs. *Theoretical Computer Science*, 385:49–59, 2007.

33 J. Willson, Z. Zhang, W. Wu, and D.-Z. Du. Fault-tolerant coverage with maximum lifetime in wireless sensor networks. In *INFOCOM*, pages 1364–1372, 2015.

34 Z. Zhang, J. Zhou, S. Tang, X. Huang, and D.-Z. Du. Computing minimum k-connected m-fold dominating set in general graphs. *INFORMS Journal on Computing*, 30(2):217–224, 2018.

35 F. Zou, X. Li, S. Gao, and W. Weili. Node-weighted steiner tree approximation in unit disk graphs. *J. Combinatorial Optimization*, 18(4):342–349, 2009.

36 F. Zou, Y. Wang, X. XiaoHua, X. Li, D. Hongwei, P.-J. Wan, and W. Weili. New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs. *Theoretical Computer Science*, 412(3):198–208, 2011.

Full Complexity Classification of the List Homomorphism Problem for Bounded-Treewidth Graphs

Karolina Okrasa
Warsaw University of Technology, Faculty of Mathematics and Information Science, Poland
University of Warsaw, Institute of Informatics, Poland
k.okrasa@mini.pw.edu.pl

Marta Piecyk
Warsaw University of Technology, Faculty of Mathematics and Information Science, Poland
m.piecyk@mini.pw.edu.pl

Paweł Rzążewski
Warsaw University of Technology, Faculty of Mathematics and Information Science, Poland
University of Warsaw, Institute of Informatics, Poland
p.rzazewski@mini.pw.edu.pl

Abstract

A homomorphism from a graph G to a graph H is an edge-preserving mapping from $V(G)$ to $V(H)$. Let H be a fixed graph with possible loops. In the list homomorphism problem, denoted by LHOM(H), we are given a graph G, whose every vertex v is assigned with a list $L(v)$ of vertices of H. We ask whether there exists a homomorphism h from G to H, which respects lists L, i.e., for every $v \in V(G)$ it holds that $h(v) \in L(v)$.

The complexity dichotomy for LHOM(H) was proven by Feder, Hell, and Huang [JGT 2003]. The authors showed that the problem is polynomial-time solvable if H belongs to the class called *bi-arc graphs*, and for all other graphs H it is NP-complete.

We are interested in the complexity of the LHOM(H) problem, parameterized by the treewidth of the input graph. This problem was investigated by Egri, Marx, and Rzążewski [STACS 2018], who obtained tight complexity bounds for the special case of reflexive graphs H, i.e., if every vertex has a loop.

In this paper we extend and generalize their results for *all* relevant graphs H, i.e., those, for which the LHOM(H) problem is NP-hard. For every such H we find a constant $k = k(H)$, such that the LHOM(H) problem on instances G with n vertices and treewidth t

- can be solved in time $k^t \cdot n^{\mathcal{O}(1)}$, provided that G is given along with a tree decomposition of width t,
- cannot be solved in time $(k-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$, for any $\varepsilon > 0$, unless the SETH fails.

For some graphs H the value of $k(H)$ is much smaller than the trivial upper bound, i.e., $|V(H)|$.

Obtaining matching upper and lower bounds shows that the set of algorithmic tools that we have discovered cannot be extended in order to obtain faster algorithms for LHOM(H) in bounded-treewidth graphs. Furthermore, neither the algorithm, nor the proof of the lower bound, is very specific to treewidth. We believe that they can be used for other variants of the LHOM(H) problem, e.g. with different parameterizations.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Problems, reductions and completeness; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases list homomorphisms, fine-grained complexity, SETH, treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.74

Related Version A full version of the paper is available at `http://arxiv.org/abs/2006.11155`.

Funding *Karolina Okrasa*: This research is a part of a project that has received funding from

the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement no. 714704.
Marta Piecyk: Supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062.
Paweł Rzążewski: Supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062.

Acknowledgements We are grateful to Daniél Marx and László Egri for much advice regarding the topic and to Kamil Szpojankowski for an inspiring discussion about almost all graphs.

1 Introduction

A popular line of research in studying computationally hard problems is to consider restricted instances, in order to understand the boundary between easy and hard cases. For example, most of natural problems can be efficiently solved on trees, using a bottom-up dynamic programming. This observation led to the definition of *treewidth*, which, informally speaking, measures how much a given graph resembles a tree. The notion of treewidth appears to be very natural and it was independently discovered by several authors in different contexts [2, 8, 37, 13].

For many problems, polynomial-time algorithms for graphs with bounded treewidth can be obtained by adapting the dynamic programming algorithms for trees. Most of these straightforward algorithms follow the same pattern, which was captured by the seminal meta-theorem by Courcelle [11]: he proved that each problem expressible in *monadic second order logic* (MSO_2) can be solved in time $f(\mathrm{tw}(G)) \cdot n^{\mathcal{O}(1)}$ on graphs G with n vertices and treewidth $\mathrm{tw}(G)$, where f is some function, depending on the MSO_2 formula describing the particular problem. As a consequence of this meta-theorem, in order to show that some problem Π is *fixed-parameter tractable* (FPT), parameterized by the treewidth, it is sufficient to show that Π can be described in a certain way.

The main problem with using the meta-theorem as a black-box is that the function f it produces is huge (non-elementary). We also know that this bound cannot be improved, if we want to keep the full generality of the statement [23]. Because of this, as the area of the so-called *fine-grained complexity* gained popularity [32], researchers turned back to studying particular problems, and asking about the best possible dependence on treewidth, i.e., the function f. This led to many exciting algorithmic results which significantly improved the naive dynamic programming approach [40, 6, 31].

There is a little caveat that applies to most of algorithms mentioned above: Usually we assume that the input graph is given with its tree decomposition, and the running time is expressed in terms of the width of this decomposition. This might be a serious drawback, since finding an optimal tree decomposition is NP-hard [1]. However, finding a tree decomposition of given width (if one exists) can be done in FPT time [4, 7], which is often sufficient. Since we are interested in the complexity of certain problems, parameterized by the treewidth, we will not discuss the time needed to find a decomposition. Thus we will assume that the input graph is given with its tree decomposition.

In parallel to improving the algorithms, many lower bounds were also developed [33, 37, 13]. Let us point out that the main assumption from the classical complexity theory, i.e., $\mathrm{P} \neq \mathrm{NP}$, is too weak to provide any meaningful lower bounds in our setting. The most commonly used assumptions in the fine-grained complexity world, are the *Exponential-Time Hypothesis* (ETH) and the *Strong Exponential-Time Hypothesis* (SETH), both introduced by Impagliazzo and Paturi [27, 28]. Informally speaking, the ETH implies that 3-Sat with n variables cannot be solved in subexponential time, i.e., in time $2^{o(n)}$, and the SETH implies that CNF-Sat with n variables and m clauses cannot be solved in time $(2-\varepsilon)^n \cdot m^{\mathcal{O}(1)}$, for any $\varepsilon > 0$.

For example the straightforward dynamic programming algorithm for k-Coloring works in time $k^{\mathrm{tw}(G)} \cdot n^{\mathcal{O}(1)}$. As one of the first SETH-based lower bounds for problems parameterized by the treewidth, Lokshtanov, Marx, and Saurabh [33] proved that this bound is tight.

▶ **Theorem 1** (Lokshtanov, Marx, Saurabh [33]). *For any $k \geqslant 3$, there is no algorithm solving k-Coloring on a graph with n vertices and treewidth t in time $(k-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$, unless the SETH fails.*

Graph homomorphisms. In this paper we are interested in extending Theorem 1 for one of possible generalizations of the k-Coloring problem. For graph G and H (both with possible loops on vertices), a *homomorphism* from G to H is a mapping $h\colon V(G) \to V(H)$, which preserves edges, i.e., for every edge xy of G it holds that $h(x)h(y) \in E(H)$. The graph H is called a *target*. If h is a homomorphism from G to H, we denote it by writing $h : G \to H$. We also write $G \to H$ to indicate that some homomorphism from G to H exists.

By $\textsc{Hom}(H)$ we denote the problem of deciding if an instance graph G admits a homomorphism to H (usually we consider H a fixed graph, but we might also treat it as a part of the input). Observe that if $H = K_k$, then $\textsc{Hom}(H)$ is equivalent to the k-Coloring problem. Because of that, homomorphisms to H are often called *H-colorings*. We will also refer to vertices of H as *colors*.

Let us briefly survey some results concerning the complexity of variants of the $\textsc{Hom}(H)$ problem. For more, we refer the reader to the monograph by Hell and Nešetřil [25]. The complexity dichotomy for $\textsc{Hom}(H)$ was shown by Hell and Nešetřil [26]: the problem is polynomial-time-solvable if H contains a vertex with a loop or is bipartite, and NP-complete for all other graphs H. Since then, many interesting results concerning the complexity of graph homomorphisms have appeared [22, 41, 38, 12, 14, 10]. The fine-grained complexity of the $\textsc{Hom}(H)$ problem, parameterized by the treewidth of the input graph, was recently studied by Okrasa and Rzążewski [35]. They found tight SETH-bounds, conditioned on two conjectures from algebraic graph theory from early 2000s. As these conjectures remain wide open, we know no graph, for which the bounds from [35] do not apply.

A natural and interesting extension of the $\textsc{Hom}(H)$ problem is its *list* version. In the *list homomorphism problem*, denoted by $\textsc{LHom}(H)$, the input consists of a graph G and a *H-lists* L, which means that L is a function which assigns to each vertex of G a subset of vertices of H. We ask whether there is a homomorphism h from G to H, which respects lists L, i.e., for each $x \in V(G)$ it holds that $h(x) \in L(x)$. If h is such a list homomorphism, we denote it by $h : (G, L) \to H$. We also write $(G, L) \to H$ to indicate that some homomorphism $h : (G, L) \to H$ exists.

The complexity of the $\textsc{LHom}(H)$ problem was shown in three steps. First, Feder and Hell [16] provided a classification for the case that H is *reflexive*, i.e., every vertex has a loop. They proved that if H is an interval graph, then the problem is polynomial-time solvable, and otherwise it is NP-complete. The next step was showing the complexity dichotomy for *irreflexive graphs* (i.e., with no loops). Feder, Hell, and Huang [17] proved that if H is bipartite and its complement is a circular-arc graph, then the problem is polynomial-time solvable, and otherwise it is NP-complete. Interestingly, bipartite graphs whose complement is circular-arc were studied independently by Trotter and Moore [39] in the context of some poset problems. Finally, Feder, Hell, and Huang [18] provided the full classification for general graph H: the polynomial cases appear to be *bi-arc graphs*, which are also defined in terms of some geometric representation. Let us now skip the exact definition of bi-arc graphs, and we will get back to it in Section 4.1.

Let us point out that in all three papers mentioned above, the polynomial-time algorithms for LHOM(H) exploited the geometric representation of H. On the other hand, all hardness proofs followed the same pattern. First, for each "easy" class $\mathcal{C}$ (i.e., interval graphs, bipartite co-circular-arc graphs, and bi-arc graphs), the authors provided an alternative characterization in terms of forbidden subgraphs. In other words, they defined a (non-necessarily finite) family $\mathcal{F}$ of graphs, such that $H \in \mathcal{C}$ if and only if H does not contain any $F \in \mathcal{F}$ as an induced subgraph. Then, for each $F \in \mathcal{F}$, the authors showed that LHOM(F) is NP-complete. Note that this is sufficient, as every "hard" graph H contains some $F \in \mathcal{F}$, and every instance of LHOM(F) is also an instance of LHOM(H), where no vertex from $V(H) - V(F)$ appears in any list.

If the input graph G is given with a tree decomposition of width $\mathrm{tw}(G)$, then the straightforward dynamic programming solves the LHOM(H) problem in time $|V(H)|^{\mathrm{tw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$. The study of the fine-grained complexity of LHOM(H), parameterized by the treewidth of the input graph, was initiated by Egri, Marx, and Rzążewski [15]. They provided the full complexity classification for reflexive graphs H, i.e., corresponding to the first step of the above-mentioned complexity dichotomy. The authors defined a new graph invariant, denoted by i^*, which is based on *incomparable sets* and the existence of a certain decomposition in H, and proved the following tight bounds.

▶ **Theorem 2** (Egri, Marx, Rzążewski [15]). *Let H be a fixed connected reflexive non-interval graph, and let $k = i^*(H)$. The LHOM(H) problem on instances (G, L) with n vertices,*

(a) *can be solved in time $k^{\mathrm{tw}(G)} \cdot n^{\mathcal{O}(1)}$, provided that an optimal tree decomposition of G is given,*

(b) *cannot be solved in time $(k-\varepsilon)^{\mathrm{tw}(G)} \cdot n^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

Our results

In this paper we provide a full complexity classification of LHOM(H), parameterized by the treewidth of an instance graph. Our results heavily extend the ones of Egri, Marx, Rzążewski [15] and generalize Theorem 2 to all relevant graphs H. Let us point out that instead of designing ad-hoc algorithms and reductions that are fine-tailored for a particular problem, we rather build a general framework that allows us to provide tight bounds for a natural and important family of problems.

Bipartite graphs H. We first deal with the case that H is bipartite (in particular, irreflexive), with bipartition classes X and Y. Recall that we are interested in graphs H, for which the LHOM(H) problem is NP-hard, i.e., graphs that are not co-circular-arc graphs. Moreover, we consider only connected graphs H (as otherwise we can reduce to this case in polynomial time).

Let us present the high-level idea behind our algorithm for LHOM(H). Consider an instance (G, L), such that G is connected, and let $n = |V(G)|$. We may assume that G is bipartite, as otherwise (G, L) is clearly a no-instance. Furthermore, in any homomorphism from G to H, each bipartition class of G is mapped to a different bipartition class of H. We can assume that this is already reflected in the lists (we might have to solve two independent instances).

The algorithm is based on two main ideas. First, observe that if H contains two vertices u, v, which are in the same bipartition class, and each neighbor of u is a neighbor of v, then we can always use v instead of u, unless this is forbidden by lists. Thus we might always assume that each list is an *incomparable set*, i.e., it does not contain two vertices u, v as above. By $i(H)$ we denote the size of a largest incomparable set contained in one bipartition class.

The second idea is related to a certain decomposition of H. By a *bipartite decomposition* we mean a partition of the vertex set of H into three subsets D, N, R, such that:

- at least one of sets $(D \cap X)$ and $(D \cap Y)$ has at least 2 elements,
- N is non-empty, induces a biclique in H, and separates D and R,
- the sets $(D \cap X) \cup (N \cap Y)$ and $(D \cap Y) \cup (N \cap X)$ induce bicliques in H.

We show that if H has a bipartite decomposition, then we can reduce solving an instance (G, L) of LHom(H) to solving several instances of LHom(H_1) and LHom(H_2), where H_1 is the subgraph of H induced by D, and H_2 is obtained from H by collapsing $D \cap X$ and $D \cap Y$ to single vertices.

This leads to the definition of $i^*(H)$ as the maximum value of $i(H')$ over all connected undecomposable induced subgraphs H' of H, which are not complements of a circular-arc graph (a graph is undecomposable if it has no bipartite decomposition). As our first result, we show that the algorithm exploiting decompositions recursively runs in time $i^*(H)^{\mathrm{tw}(G)} \cdot n^{\mathcal{O}(1)}$.

One might wonder whether some additional observations could be used to improve the algorithm. As our second result, we show that this is not possible, assuming the SETH. This means that unless something unexpected happens in complexity theory, our algorithmic toolbox allows to solve LHom(H), parameterized by the treewidth, as fast as possible. More formally, we show the following theorem, which fully classifies the complexity of LHom(H) for bipartite graphs H.

▶ **Theorem 3.** *Let H be a connected bipartite graph, whose complement is not a circular-arc graph, and let $k = i^*(H)$. Let G be a bipartite graph with n vertices and treewidth* $\mathrm{tw}(G)$.

(a) *Even if H is given as an input, the LHom(H) problem with instance (G, L) can be solved in time $k^{\mathrm{tw}(G)} \cdot (n \cdot |H|)^{\mathcal{O}(1)}$ for any lists L, provided that G is given with an optimal tree decomposition.*

(b) *Even if H is fixed, there is no algorithm that solves LHom(H) for every G and L in time $(k - \varepsilon)^{\mathrm{tw}(G)} \cdot n^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

Note that for Theorem 3 a), if H is not considered to be a constant, $n \cdot |H|$ is a natural measure of the size of an instance, as it is an upper estimate on the sum of sizes of all lists.

The main tool used in the proof of Theorem 3 b) is the following technical lemma.

▶ **Lemma 4** (Constructing a NEQ(S)-gadget)**.** *Let H be a connected, bipartite, undecomposable graph, whose complement is not a circular-arc graph. Let S be an incomparable set of $k \geqslant 2$ vertices of H, contained in one bipartition class. Then there exists a* NEQ(S)*-gadget, i.e., a graph F with H-lists L and two special vertices $x, x' \in V(F)$, such that $L(x) = L(x') = S$ and*

- *for any list homomorphism $h : (F, L) \to H$, it holds that $h(x) \neq h(x')$,*
- *for any distinct $s, s' \in S$ there is $h : (F, L) \to H$, such that $h(x) = s$ and $h(x') = s'$.*

Let us point out that the graph constructed in Lemma 4 can be seen as a *primitive-positive definition* of the inequality relation on S (see e.g. Bulatov [9, Section 2.1]). However, we prefer to present our results using purely combinatorial terms.

The proof of Lemma 4 is technically involved, but as soon as we have it, the proof of Theorem 3 b) is straightforward. Consider an instance G of k-Coloring, where $k = i^*(H)$. Let H' be a connected, undecomposable, induced subgraph of H, whose complement is not a circular-arc graph, and contains an incomparable set S of size k. We construct a graph G^* by replacing each edge uv of G with a copy of the NEQ(S)-gadget, given by Lemma 4 (invoked for H' and S), so that u is identified with x and v is identified with x'. By the properties of the gadget, we observe that G^* has a list homomorphism to H if and only if

G is a yes-instance of k-Coloring. Furthermore, the construction of the NEQ(S)-gadget depends on H only, and H is assumed fixed, so we conclude that $\mathrm{tw}(G^*) = \mathrm{tw}(G) + \mathcal{O}(1)$. Therefore the statement of Theorem 3 b) follows from Theorem 1.

General graphs H. Next, we move to the general case. We aim to reduce the problem to the bipartite case. The main idea comes from Feder, Hell, and Huang [18] who showed a close connection between the LHom(H) problem and the LHom(H^*) problem, where H^* is the *associated bipartite graph* of H, i.e., the bipartite graph with bipartition classes $\{v' \colon v \in V(H)\}$ and $\{v'' \colon v \in V(H)\}$, where $u'v'' \in E(H^*)$ if and only if $uv \in E(H)$. Let us point out that we can equivalently define H^* as a categorical (direct) product of H and K_2 [24].

We extend the definition of i^* to non-bipartite graphs by setting $i^*(H) := i^*(H^*)$. Let us point out that this is consistent with the definition for bipartite graphs, and with the definition of i^* for reflexive graphs, introduced by Egri, Marx, and Rzążewski [15]. We show the following theorem, fully classifying the complexity of LHom(H), parameterized by the treewidth of the instance graph[1].

▶ **Theorem 5.** *Let H be a connected non-bi-arc graph (with possible loops), and let $k = i^*(H)$. Let G be a graph with n vertices and treewidth* $\mathrm{tw}(G)$.

(a) *Even if H is given as an input, the LHom(H) problem with instance (G, L) can be solved in time $k^{\mathrm{tw}(G)} \cdot (n \cdot |H|)^{\mathcal{O}(1)}$ for any lists L, provided that G is given with an optimal tree decomposition.*

(b) (♠) *Even if H is fixed, there is no algorithm that solves LHom(H) for every G and L in time $(k-\varepsilon)^{\mathrm{tw}(G)} \cdot n^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails.*

As we mentioned before, both statements of Theorem 5 follow from the corresponding statements in Theorem 3. For the algorithmic part, we define certain decompositions of general graphs H and show that they coincide with bipartite decompositions H^*. This lets us reduce solving an instance (G, L) of LHom(H) to solving some instances of LHom(H^*). On the complexity side, the reduction is even more direct: we show that an algorithm solving the LHom(H) problem on instances with treewidth t in time $(i^*(H) - \varepsilon)^t \cdot n^{\mathcal{O}(1)}$ could be used to solve the LHom(H^*) problem on instances with treewidth t in time $(i^*(H^*) - \varepsilon)^t \cdot n^{\mathcal{O}(1)}$, thus contradicting Theorem 3 b).

We also analyze the complexity of LHom(H) for *typical* graphs H, and prove the following.

▶ **Corollary 6 (♠).** *For almost all graphs H with possible loops the following holds. Even if H is fixed, there is no algorithm that solves LHom(H) for every instance (G, L) in time $\mathcal{O}\left((|V(H)| - \varepsilon)^{\mathrm{tw}(G)} \cdot n^{\mathcal{O}(1)}\right)$ for any $\varepsilon > 0$, unless the SETH fails.*

Finally, we show how to generalize our approach of reducing instances of LHom(H) to instances of LHom(H'), where H' is undecomposable. We believe that this idea could be exploited to study the complexity of LHom(H) in various regimes, e.g., for different parameterizations of input instances.

Comparison to the previous work. Let us briefly discuss similarities and differences between our work and previous, closely related results by Egri, Marx, and Rzążewski [15] (about the complexity of the LHom(H) problem for reflexive H), and by Okrasa and Rzążewski [35] (about the complexity of the Hom(H) problem).

[1] The proofs of results marked with ♠ can be found in the full version of the paper [34].

At the high level, we follow the direction used by Egri *et al.* [15], but since we generalize their result to *all* relevant graphs H, the techniques become much more involved. The crucial idea was to reduce the problem to the bipartite case, and to define decompositions of general graphs that correspond to the decompositions of H^*. On the contrary, the case of reflexive graphs H is much more straightforward. In particular, there is just one type of decomposition that could be exploited algorithmically. Also, the structure of "hard" subgraphs is much simpler in this case, so the necessary gadgets are significantly easier to construct.

On the other hand, in order to prove hardness for the HOM(H) problem, Okrasa and Rzążewski [35] used mostly *algebraic tools* that are able to capture the global structure of a graph. In contrast, our proofs are purely combinatorial. Furthermore, we are able to provide the full complexity classification for all graphs H, while the results of [35] are conditioned on two twenty-year-old conjectures.

Notation

Let H be a graph. By $\mathrm{comp}(H)$ we denote the set of connected components of H. For a vertex v, by $N(v)$ we denote the *open neighborhood* of v, i.e., the set of vertices adjacent to v (note that $v \in N(v)$ if and only if v has a loop). For a set $U \subseteq V(H)$, we define $N(U) := \bigcup_{u \in U} N(u) - U$ and $N[U] := \bigcup_{u \in U} N(u) \cup U$. If $U = \{u_1, \ldots, u_k\}$, we omit one pair of brackets and write $N(u_1, \ldots, u_k)$ (respectively $N[u_1, \ldots, u_k]$) instead of $N(\{u_1, \ldots, u_k\})$ (respectively $N[\{u_1, \ldots, u_k\}]$).

We say that two vertices x, y are *comparable* if $N(y) \subseteq N(x)$ or $N(x) \subseteq N(y)$. If two vertices are not comparable, we say that they are *incomparable*. A set of vertices is *incomparable* if all vertices are pairwise incomparable.

We say that a set $A \subseteq V(H)$ is *complete* to a set $B \subseteq V(H)$ if for every $a \in A$ and $b \in B$ the edge ab exists. On the other hand, A is *non-adjacent* to B if there are no edges with one endvertex in A and the other in B.

Let H be a bipartite graph, whose bipartition classes are denoted by X and Y. For a set $S \subseteq V(H)$ and $Z \in \{X, Y\}$, by S_Z we denote $S \cap Z$. For $A, B \subseteq V(H)$, we say that A is *bipartite-complete* to B if A_X is complete to B_Y and A_Y is complete to B_X.

A *walk* $\mathcal{P}$ is a sequence $\mathcal{P} = p_1, \ldots, p_\ell$ of vertices of H, such that $p_i p_{i+1} \in E(H)$, for every $i \in [\ell - 1]$.

2 Algorithm for bipartite target graphs

Observe that we might always assume that H is connected, as otherwise we can solve the problem for each connected component of H separately. Furthermore, without losing the generality we may assume certain properties of instances of LHOM(H) that we need to solve.

▶ **Observation 7 (♠).** *Let (G, L) be an instance of LHOM(H), where H is connected and bipartite with bipartition classes X, Y. Without loss of generality, we might assume the following.*

1. *The graph G is connected and bipartite, with bipartition classes X_G and Y_G,*
2. *$\bigcup_{x \in X_G} L(x) \subseteq X$ and $\bigcup_{y \in Y_G} L(y) \subseteq Y$,*
3. *for each $x \in V(G)$, the set $L(x)$ is incomparable.*

An instance of LHOM(H) that respects conditions in Observation 7 is called *consistent*. From now on we will restrict ourselves to consistent instances. Let us introduce a graph parameter, which will play a crucial role in our investigations.

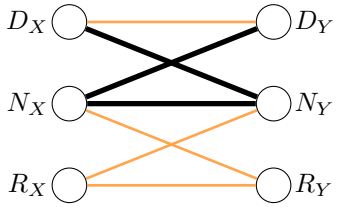


Figure 1 A schematic view of a bipartite decomposition. Disks correspond to independent sets of vertices. Thick black lines indicate that all possible edges between two sets exist, and thin orange lines depict edges that might exist, but do not have to. The lack of a line means that there are no edges between two sets.

▶ **Definition 8** ($i(H)$). *For a bipartite graph H, by $i(H)$ we denote the maximum size of an incomparable set in H, which is fully contained in one bipartition class.*

Clearly for every H we have $i(H) \leqslant |H|$. Note that by Observation 7 we obtain the following.

▶ **Corollary 9.** *Let (G, L) be a consistent instance of LHOM(H), where H is bipartite. Then $\max_{v \in V(G)} |L(v)| \leqslant i(H)$.*

2.1 Decomposition of bipartite graphs

Throughout this section we assume that the target graph H is bipartite with bipartition classes X and Y. In particular, it has no loops. Our algorithm for LHOM(H) is based on the existence of a certain decomposition of H.

▶ **Definition 10** (Bipartite decomposition). *A partition of $V(H)$ into an ordered triple of sets (D, N, R) is a* bipartite decomposition *if the following conditions are satisfied (see Figure 1).*

1. *N is non-empty and separates D and R,*
2. *$|D_X| \geqslant 2$ or $|D_Y| \geqslant 2$,*
3. *N induces a biclique in H,*
4. *D is bipartite-complete to N.*

Since so far we only consider bipartite decompositions, we will just call them decompositions. Later on we will introduce other types of decompositions and then the distinction will be important. If H admits a decomposition, then it is *decomposable*, otherwise it is *undecomposable*.

For a graph H with a decomposition (D, N, R), the *factors of the decomposition* are two graphs H_1, H_2 defined as follows. The graph H_1 is the subgraph of H induced by the set D. The graph H_2 is obtained in the following way. For $Z \in \{X, Y\}$, if D_Z is non-empty, then we contract it to a vertex d_Z. If there is at least one edge between the sets D_X and D_Y, we add the edge $d_X d_Y$.

Note that both H_1 and H_2 are proper induced subgraphs of H. For H_1 is follows directly from the definition and H_2 can be equivalently defined as a graph obtained from H by removing all but one vertex from D_X (if $D_X \neq \emptyset$) and all but one vertex from D_Y (if $D_Y \neq \emptyset$). We leave the vertices that are joined by an edge, provided that such a pair exists.

Now let us show how the bipartite decomposition can be used algorithmically. Let $T(H, n, t)$ denote an upper bound for the complexity of LHOM(H) on instances with n vertices, given along a tree decomposition of width t. In the following lemma we do not assume that $|H|$ is a constant.

▶ **Lemma 11** (Bipartite decomposition lemma). *Let H be a bipartite graph with bipartition classes X and Y, whose complement is not a circular-arc graph, and suppose H has a bipartite decomposition with factors H_1, H_2. Assume that there are constants $\alpha \geqslant 1$, $c \geqslant 1$, and $d > 2$, such that $T(H_1, n, t) \leqslant \alpha \cdot c^t \cdot (n \cdot |H_1|)^d$ and $T(H_2, n, t) \leqslant \alpha \cdot c^t \cdot (n \cdot |H_2|)^d$. Then $T(H, n, t) \leqslant \alpha \cdot c^t \cdot (n \cdot |H|)^d$, if n is sufficiently large.*

Proof. Consider an instance (G, L) of LHOM(H), recall that without loss of generality we may assume that it is consistent. Let the bipartition classes of G be X_G and Y_G and assume that $\bigcup_{x \in X_G} L(x) \subseteq X$ and $\bigcup_{y \in Y_G} L(y) \subseteq Y$.

Let (D, N, R) be a bipartite decomposition of H. We observe that for $Z \in \{X, Y\}$, and any two vertices $v \in D_Z, s \in N_Z$, we have $N(v) \subseteq N(s)$. Thus we may assume that no list contains both s and v. Let Q be the set of vertices of G which have at least one vertex from N in their lists.

▷ Claim 12. If there exists a list homomorphism $h\colon (G, L) \to H$, the image of each $C \in \mathrm{comp}(G - Q)$ is entirely contained either in D or in R.

Proof. By the definition of $\mathrm{comp}(G - Q)$, the image of C is disjoint with N. Suppose there exist $a, b \in C$, such that $h(a) = u \in D$ and $h(b) = r \in R$. Since C is connected, there exists an a-b-path P in C. The image of P is an u-r-walk in H. But since N separates D and R in H, there is a vertex of P, which is mapped to a vertex of N, a contradiction. ◁

Let us define lists L_1 as $L_1(x) := L(x) \cap D$, for every $x \in V(G) - Q$. For each $C \in \mathrm{comp}(G - Q)$, we check if there exists a homomorphism $h_C\colon (C, L_1) \to H_1$. Let $\mathbb{C}_1$ be the set of those $C \in \mathrm{comp}(G - Q)$, for which h_C exists. By Claim 12, we observe that if $C \notin \mathbb{C}_1$, then we can safely remove all vertices from D from the lists of vertices of C.

Now consider the graph H_2. Let $Z \in \{X, Y\}$ and let d_Z be the vertex to which the set D_Z is collapsed (if it exists). Let us define an instance (G, L_2) of the LHOM(H_2) problem, where the lists L_2 are as follows. If $v \in Z_G$ is a vertex from some component of $\mathbb{C}_1$, then $L_2(v) := L(v) - D_Z \cup \{d_Z\}$ (note that in this case d_Z must exist). If v is a vertex from some component of $\mathrm{comp}(G - Q) - \mathbb{C}_1$, then $L_2(v) := L(v) - D_Z$. Finally, if $v \in Q$, then $L_2(v) := L(v)$. Note that the image of each list is contained in $V(H_2)$. Moreover, note that $\bigcup_{x \in X_G} L_2(x) \subseteq R_X \cup N_X \cup \{d_X\}$ and $\bigcup_{y \in Y_G} L_2(y) \subseteq R_Y \cup N_Y \cup \{d_Y\}$.

▷ Claim 13. There is a list homomorphism $h : (G, L) \to H$ if an only if there is a list homomorphism $h' : (G, L_2) \to H_2$.

Proof. First, assume that $h : (G, L) \to H$ exists. Define $h' : V(G) \to V(H_2)$ in the following way:

$$h'(v) = \begin{cases} d_X & \text{if } h(v) \in D_X, \\ d_Y & \text{if } h(v) \in D_Y, \\ h(v) & \text{otherwise.} \end{cases}$$

Clearly h' is a homomorphism from G to H_2, we need to show that it also respects lists L_2. Suppose otherwise and let v be a vertex of G, such that $h'(v) \notin L_2(v)$. By symmetry, assume that $v \in X_G$, and thus $h'(v) \in X$. If $h'(v) \neq d_X$, then $h'(v) = h(v) \in (L(v) - D_X) \subseteq L_2(v)$. So suppose $h'(v) = d_X$ (and thus $h(v) \in D_X$) and $d_X \notin L_2(v)$. Observe that v cannot be a vertex from Q, since h maps v to a vertex of D_X and vertices from Q do not have any vertices of D in their lists. So the only case left is that v belongs to some connected component C of $G - Q$, which cannot be mapped to H_1. But then, by Claim 12, no vertex of C is mapped to any vertex of D, so $h(v) \notin D_X$, a contradiction.

Now suppose there exists a list homomorphism $h' : (G, L_2) \to H_2$. Define the following mapping h from $V(G)$ to $V(H)$. If $h'(v) \notin \{d_X, d_Y\}$, then we set $h(v) := h'(v)$. Otherwise, if $h'(v) \in \{d_X, d_Y\}$, then v is a vertex of some connected component $C \in \mathbb{C}_1$, and we define $h(v) := h_C(v)$. Clearly h preserves lists L: if $h(v) \notin D$, then $h(v) = h'(v) \in L_2(v) - \{d_X, d_Y\} \subseteq L(v)$; otherwise we use h_C, which preserves lists L by the definition.

Now suppose h does not preserve edges, so there are vertices $u \in X_G$ and $v \in Y_G$, such that uv is an edge of G and $h(u)h(v)$ is not an edge of H. If $h'(u) = d_X, h'(v) = d_Y$, or $h'(u) \neq d_X, h'(v) \neq d_Y$, then $h(u)h(v)$ must be an edge of H, otherwise we get a contradiction by the definitions of h_C (as since x and y are neighbors, they belong to the same C) and h', respectively. So, by symmetry, suppose $h'(u) = d_X$ and $h'(v) \neq d_Y$. But then we observe that $h(u) \in D_X$ and $h(v) = h'(v) \in N_Y$, since h' is a homomorphism. And because N_Y is complete to D_X, so $h(u)h(v)$ is an edge – a contradiction. ◁

Computing $\mathrm{comp}(G - Q)$ can be done in time $\mathcal{O}(n \cdot |H| + n^2) = \mathcal{O}((n \cdot |H|)^2)$. Note that given a tree decomposition of G of width at most t, we can easily obtain a tree decomposition of each $C \in \mathrm{comp}(G - Q)$ of width at most t. Computing h_C for all $C \in \mathrm{comp}(G - Q)$ requires time at most

$$\sum_{C \in \mathrm{comp}(G-Q)} T(H_1, |C|, t) \leqslant \sum_{C \in \mathrm{comp}(G-Q)} \alpha \cdot c^t \cdot (|H_1| \cdot |C|)^d \leqslant \alpha \cdot c^t \cdot (|H_1| \cdot n)^d.$$

The estimation follows from the facts that $\sum_{C \in \mathrm{comp}(G-Q)} |C| \leqslant n$, and n^d is superadditive with respect to n, i.e., $n_1^d + n_2^d \leqslant (n_1 + n_2)^d$. Computing lists L_2 can be performed in time $\mathcal{O}(|H| \cdot n)$. Finally, computing h' requires time $T(H_2, n, t) \leqslant \alpha \cdot c^t \cdot (|H_2| \cdot n)^d$. The total running time is therefore bounded by:

$$\mathcal{O}\left((n \cdot |H|)^2\right) + \alpha \cdot c^t \cdot (|H_1| \cdot n)^d + \mathcal{O}(|H| \cdot n) + \alpha \cdot c^t \cdot (|H_2| \cdot n)^d.$$

With a careful analysis one can verify that the above expression is bounded by $\alpha \cdot c^t \cdot (|H| \cdot n)^d$, provided that n is sufficiently large. ◀

2.2 Solving LHom(H) for bipartite targets

Let us define the main combinatorial invariant of the paper, $i^*(H)$:

▶ **Definition 14** ($i^*(H)$ for bipartite H). *Let H be a connected bipartite graph, whose complement is not a circular-arc graph. Define*

$$i^*(H) := \max\{i(H') \colon H' \text{ is an undecomposable, connected, induced subgraph of } H, \text{ whose complement is not a circular-arc graph}\}.$$

Observe that if H' is an induced subgraph of H, then $i(H') \leqslant i(H)$ and $i^*(H') \leqslant i^*(H)$, and thus $i^*(H) = i(H)$ for undecomposable H. Furthermore, we always have $i^*(H) \leqslant i(H) \leqslant |H|$.

Now we are ready to present an algorithm solving LHom(H), note that again we do not assume that $|H|$ is a constant. We present the following, slightly more general variant of Theorem 3 a), where we also do not assume that the tree decomposition of G is optimal.

▶ **Theorem 3' a).** *Let H be a connected bipartite graph (given as an input) and let (G, L) be an instance of LHom(H), where G has n vertices and is given along a tree decomposition of width t. Then there is an algorithm which decides whether $(G, L) \to H$ in time $\mathcal{O}\left(i^*(H)^t \cdot (n \cdot |H|)^{\mathcal{O}(1)}\right)$.*

Proof. Clearly we can assume that n is sufficiently large, as otherwise we can solve the problem in polynomial time by brute-force.

Observe that with H we can associate a *recursion tree* $\mathcal{R}$, whose nodes are labeled with induced subgraphs of H. The root, denoted by $node(H)$ corresponds to the whole graph H. If H is undecomposable or is a complement of a circular-arc graph, then the recursion tree has just one node. Otherwise H has a decomposition with factors H_1 and H_2, and then $node(H)$ has two children, $node(H_1)$ and $node(H_2)$, respectively. Recall that each factor has strictly fewer vertices than H, so we can construct $\mathcal{R}$ recursively. Clearly, each leaf of $\mathcal{R}$ is either the complement of a circular-arc graph (and thus the corresponding problem is polynomial-time solvable), or is an undecomposable induced subgraph of H. Note that a recursion tree may not be unique, as a graph may have more than one decomposition. However, the number of leaves is bounded by $\mathcal{O}(|H|)$ (actually, with a careful analysis we can show that it is at most $|H|-2$), so the total number of nodes is $\mathcal{O}(|H|)$. Furthermore, it can be shown that in time polynomial in H we can check if H is undecomposable, or find a decomposition (we show this in the full version of the paper). Since recognizing circular-arc graphs (and therefore of course their complements) is also polynomial-time solvable, we conclude that $\mathcal{R}$ can be constructed in time polynomial in $|H|$.

If H is the complement of a circular-arc graph, then we solve the problem in polynomial time [17]. If H is undecomposable, we run a standard dynamic programming algorithm on a tree decomposition of G (see [8, 5]). For each bag of the tree decomposition, and every partial list homomorphism from the graph induced by this bag to H we indicate whether this particular partial homomorphism can be extended to a list homomorphism of the graph induced by the subtree rooted at this bag. By Corollary 9, the size of each list $L(x)$ for $x \in V(G)$ is at most $i(H)$, thus the complexity of the algorithm is bounded by $\alpha \cdot i(H)^t \cdot (n \cdot |H|)^d$ for some constants α, d. We can assume that $d > 2$, as otherwise we can always increase it.

So suppose H is decomposable and let us show that we can solve the problem in time $\alpha \cdot i^*(H)^t \cdot (n \cdot |H|)^d$. Let $\mathcal{R}$ be a recursion tree of H and recall that its every leaf corresponds to an induced subgraph of H with strictly fewer vertices. Therefore, for any leaf node of $\mathcal{R}$, corresponding to the subgraph H' of H, we can solve every instance of LHom(H') with n vertices and a tree decomposition of width at most t in time $\alpha \cdot i(H')^t \cdot (n \cdot |H'|)^d \leqslant \alpha \cdot i^*(H)^t \cdot (n \cdot |H'|)^d$. Now, applying Lemma 11 in a bottom-up fashion, we conclude that we can solve LHom(H) in time $\alpha \cdot i^*(H)^t \cdot (n \cdot |H|)^d = \mathcal{O}(i^*(H)^t \cdot (n \cdot |H|)^{\mathcal{O}(1)})$. ◀

3 Hardness for bipartite target graphs

In this section we aim to prove Theorem 3 b). Actually we will show a version, which gives the lower bound parameterized by the pathwidth of G. Clearly such a statement is be stronger, as $\mathrm{pw}(G) \geqslant \mathrm{tw}(G)$. This corresponds to the pathwidth variant of Theorem 1, also shown by Lokshtanov, Marx, Saurabh [33].

▶ **Theorem 1'** (Lokshtanov, Marx, Saurabh [33]). *For any $k \geqslant 3$, there is no algorithm solving k-Coloring on a graph with n vertices and pathwidth t in time $(k-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$, unless the SETH fails.*

Thus we show the following strengthening of Theorem 3 b).

▶ **Theorem 3' b).** *Let H be a fixed bipartite graph, whose complement is not a circular-arc graph. Unless the SETH fails, there is no algorithm that solves the LHom(H) problem on instances with n vertices and pathwidth t in time $(i^*(H)-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$, for any $\varepsilon > 0$.*

In order to prove Theorem 3' b), it is sufficient to show the following.

▶ **Theorem 15.** *Let H be a fixed connected bipartite undecomposable graph, whose complement is not a circular-arc graph. Unless the SETH fails, there is no algorithm that solves the* LHOM*(H) problem on instances with n vertices and pathwidth t in time $(i(H)-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$, for any $\varepsilon > 0$.*

Let us show that Theorem 3' b) and Theorem 15 are equivalent.

(Theorem 15 → Theorem 3' b)). Assume the SETH and suppose that Theorem 15 holds and Theorem 3' b) fails. So there is a bipartite graph H, whose complement is not a circular-arc graph, and an algorithm A that solves LHOM(H) in time $(i^*(H)-\varepsilon)^{\mathrm{pw}(G)} \cdot n^{\mathcal{O}(1)}$ for every input (G, L), assuming that G is given along with its optimal path decomposition.

Let H' be an undecomposable connected induced subgraph of H, whose complement is not a circular-arc graph, and $i(H') = i^*(H)$. Let (G, L') be an arbitrary instance of LHOM(H'). Since H' is an induced subgraph of H, the instance (G, L') can be seen as an instance of LHOM(H), where no vertex from $V(H) - V(H')$ appears in any list. The algorithm A solves this instance in time $(i^*(H)-\varepsilon)^{\mathrm{pw}(G)} \cdot n^{\mathcal{O}(1)} = (i(H')-\varepsilon)^{\mathrm{pw}(G)} \cdot n^{\mathcal{O}(1)}$, contradicting Theorem 15.

(Theorem 3' b) → Theorem 15). Assume the SETH and suppose Theorem 3' b) holds and Theorem 15 fails. So there is a connected bipartite undecomposable graph H, whose complement is not a circular-arc graph, and an algorithm A that solves LHOM(H) in time $(i(H)-\varepsilon)^{\mathrm{pw}(G)} \cdot n^{\mathcal{O}(1)}$ for every input (G, L). But since H is connected, bipartite, and undecomposable, we have $i^*(H) = i(H)$, so algorithm A contradicts Theorem 3' b).

3.1 Hardness reduction

Let H be an undecomposable, bipartite graph, whose complement is not a circular-arc graph. First, we will show that the structure of H is sufficiently rich to express some basic relations.

For a k-ary relation $\mathrm{R}_k \subseteq V(H)^k$, by an R_k-gadget we mean a graph F with H-lists L and k specified vertices $v_1, v_2, \ldots, v_k$, called *interface*, such that for every $i \in [k]$ it holds that

$$\{f(v_1)f(v_2)\ldots f(v_k) \mid f : (F, L) \to H\} = \mathrm{R}_k.$$

In other words, the set of all possible colorings of the interface vertices that can be extended to a list H-coloring of the whole gadget is precisely the relation we are expressing. In the definition of an R_k-gadget we do not insist that interface vertices are pairwise different.

Let (α, β) be a fixed pair of vertices of H. First, we need to express a k-ary relation $\mathrm{OR}_k = \{\alpha, \beta\}^k - \{\alpha^k\}$ and a binary relation $\mathrm{NAND}_2 = \{\alpha\alpha, \alpha\beta, \beta\alpha\}$ for (α, β). To make the definitions more intuitive, note that we can assign logic values to vertices α, β in the following way: α is interpreted as false, and β is interpreted as true. Note that the relations OR_k and NAND_2 are symmetric with respect to interface vertices.

The other type of gadget which we need to introduce is distinguisher.

▶ **Definition 16** (Distinguisher). *Let S be an incomparable set in H and let (α, β) be a fixed pair of vertices of H, such that $\{\alpha, \beta\} \cup S$ is contained in one bipartition class of H. Let $a, b \in S$. A* distinguisher *gadget for (α, β) is a graph $D_{a/b}$ with two specified vertices x (called* input*) and y (called* output*), and H-lists L such that:*

(D1) *$L(x) = S$ and $L(y) = \{\alpha, \beta\}$,*

(D2) *there is a list homomorphism $\varphi_a : (D_{a/b}, L) \to H$, such that $\varphi_a(x) = a$ and $\varphi_a(y) = \alpha$,*

(D3) *there is a list homomorphism* $\varphi_b : (D_{a/b}, L) \to H$*, such that* $\varphi_b(x) = b$ *and* $\varphi_b(y) = \beta$*,*

(D4) *for any* $c \in \mathcal{S} - \{a, b\}$ *there is* $\varphi_c : (D_{a/b}, L) \to H$*, such that* $\varphi_c(x) = c$ *and* $\varphi_c(y) \in \{\alpha, \beta\}$*,*

(D5) *there is no list homomorphism* $\varphi : (D_{a/b}, L) \to H$*, such that* $\varphi(x) = a$ *and* $\varphi(y) = \beta$*.*

The existence of the OR_k-gadget, the NAND_2-gadget, and the distinguisher gadgets follows from the lemma.

▶ **Lemma 17 (♠).** *Let* H *be an undecomposable bipartite graph, whose complement is not a circular-arc graph and let* S *be an incomparable set in* H*,* $|S| \geqslant 2$*. Then there exists a pair of vertices* (α, β)*, such that* $S \cup \{\alpha, \beta\}$ *is contained in one bipartition class of* H *and:*

1. *for every* $k \geqslant 2$ *there exists an* OR_k*-gadget for* (α, β)*,*
2. *there exists a* NAND_2*-gadget for* (α, β)*,*
3. *for every pair* (a, b) *of distinct elements of* S *there exists a distinguisher* $D_{a/b}$ *for* (α, β)*.*

With Lemma 17 in hand we proceed to the construction of the $\mathrm{NEQ}(S)$-gadget.

▶ **Lemma 4** (Constructing a $\mathrm{NEQ}(S)$-gadget)**.** *Let* H *be a connected, bipartite, undecomposable graph, whose complement is not a circular-arc graph. Let* S *be an incomparable set of* $k \geqslant 2$ *vertices of* H*, contained in one bipartition class. Then there exists a* $\mathrm{NEQ}(S)$*-gadget, i.e., a graph* F *with* H*-lists* L *and two special vertices* $x, x' \in V(F)$*, such that* $L(x) = L(x') = S$ *and*

- *for any list homomorphism* $h : (F, L) \to H$*, it holds that* $h(x) \neq h(x')$*,*
- *for any distinct* $s, s' \in S$ *there is* $h : (F, L) \to H$*, such that* $h(x) = s$ *and* $h(x') = s'$*.*

Proof. We denote the vertices of S by $v_1, v_2, \ldots, v_k$. We call Lemma 17 for S, let (α, β) be a pair of vertices given by this lemma. We will construct the $\mathrm{NEQ}(S)$-gadget in three steps.

Step I. In this step we will show that for every $i \in [k]$ we can construct a graph I_i with two special vertices x_i and c_i and H-lists L, satisfying the following properties.

- $L(x_i) = S$ and $L(c_i) = \{\alpha, \beta\}$,
- for every list homomorphism $\varphi : (I_i, L) \to H$, if $\varphi(x_i) = v_i$, then $\varphi(c_i) = \beta$,
- for every $j \neq i$ there exists a list homomorphism $\varphi_j : (I_i, L) \to H$ such that $\varphi_j(x_i) = v_j$ and $\varphi_j(c_i) = \alpha$.

Let us fix any $i \in [k]$. For every $j \in [k] - \{i\}$ we take a distinguisher D_{v_i/v_j} given by Lemma 17 for $a = v_i, b = v_j$ with the input $x_{i,j}$ and the output $y_{i,j}$. We identify the vertices $x_{i,j}$, for all feasible j, to a single vertex x_i, and introduce a new vertex c_i. Then we add to our construction a copy of the OR_k gadget and identify its k interface vertices with distinct elements of $\{c_i\} \cup \{y_{i,j}\}_{j \neq i}$. This completes the construction of I_i (see Figure 2, left).

Recall that the properties of the OR_k-gadget imply that every list homomorphism from I_i to H maps at least one of vertices in $\{c_i\} \cup \{y_{i,j}\}_{j \neq i}$ to β. By the property (D5) in Definition 16, for any $\varphi : (I_i, L) \to H$ with $\varphi(x_i) = v_i$, and for every $j \neq i$ it holds that $\varphi(y_{i,j}) = \alpha$. This in turn forces $\varphi(c_i) = \beta$.

On the other hand, by properties (D2), (D3), and (D4), for any $j \neq i$ there is a homomorphism $\varphi_j : (I_i, L) \to H$, such that $\varphi_j(x_i) = v_j$ and $\varphi(y_{i,j}) = \beta$, which allows us to set $\varphi_j(c_i) = \alpha$. So I_i satisfies all desired properties.

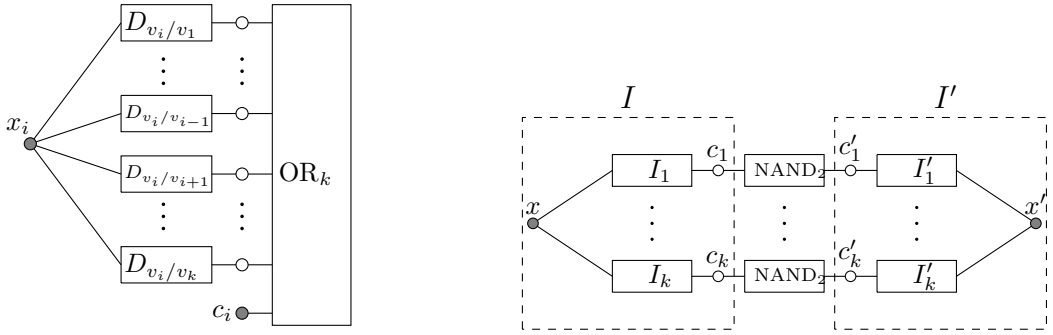


Figure 2 The graph I_i with special vertices x_i and c_i (left) and NEQ(S)-gadget with interface vertices x, x' (right).

Step II. In this step we will construct a graph I with H-lists L and special vertices $x, c_1, \ldots, c_k$, satisfying the following properties.

- $L(x) = S$ and $L(c_i) = \{\alpha, \beta\}$ for every $i \in [k]$,
- for every list homomorphism $\varphi : (I, L) \to H$, if $\varphi(x) = v_i$, then $\varphi(c_i) = \beta$.
- for every $i \in [k]$ there exists a list homomorphism $\varphi_i : (I, L) \to H$, such that $\varphi_i(x) = v_i$ and $\varphi_i(c_i) = \beta$, and $\varphi_i(c_j) = \alpha$ for every $j \in [k] - \{i\}$.

The graph I is constructed by introducing k gadgets $I_1, \ldots, I_k$, and identifying the vertices $x_1, \ldots, x_k$ into a single vertex x (see Figure 2, right). The desired properties of I follow directly from properties of I_i's.

Step III. Finally, we can construct a NEQ(S)-gadget. We introduce two copies of the gadget from the previous step, call them I and I' (we will use primes to denote the appropriate vertices of I'). For each $i \in [k]$, we introduce a NAND_2-gadget and identify its interface vertices with vertices c_i and c'_i (see Figure 2, right). Let us call such constructed graph F. We claim that F is a NEQ(S)-gadget, whose interface vertices are x and x'.

Clearly, $L(x) = L(x') = S$. Suppose that $\varphi : (F, L) \to H$ is a list homomorphism such that $\varphi(x) = \varphi(x') = v_i$. Then, by definition of I, we have $\varphi(c_i) = \varphi(c'_i) = \beta$, but this is impossible due to the properties of the NAND_2-gadget joining c_i and c'_i.

On the other hand, let us choose any distinct $v_i, v_j \in S$. We can color I according to the homomorphism φ_i, and I' according to the homomorphism φ_j (both φ_i and φ_j are defined in Step II). In particular this means that x is mapped to v_i, x' is mapped to v_j, c_i and c'_j are mapped to β, and all other vertices in $\{c_1, \ldots, c_k\} \cup \{c'_1, \ldots, c'_k\}$ are mapped to α. Since $i \neq j$, by the definition of a NAND_2-gadget, we can extend such defined mapping to all vertices of F. This completes the proof of the lemma. ◀

Now, equipped with Lemma 4, we can easily prove Theorem 15.

Proof of Theorem 15. Recall that H is an undecomposable bipartite graph, whose complement is not a circular-arc graph. Let S be the largest incomparable set in H, contained in one bipartition class. Let $k = |S|$, i.e., $k = i(H)$.

Observe that since the complement of H is not a circular-arc graph, we have that $k \geqslant 3$. Indeed, recall that H contains an obstruction, which is either an induced C_6, an induced C_8, or an asteroidal subgraph. Observe that all vertices from one bipartition class of C_6 or C_8 form an incomparable set of size at least 3. On the other hand, recall that a special edge asteroid contains at least three independent edges, so their appropriate endvertices form the desired incomparable set.

We reduce from k-Coloring, let G be an instance. Clearly we can assume that G is connected and has at least 3 vertices. We will construct a graph G^* with H-lists L and the following properties:

- $(G^*, L) \to H$ if and only if G is k-colorable,
- the number of vertices of G^* is at most $g(H) \cdot |E(G)|$ for some function g of H,
- the pathwidth of G^* is at most $g(H) + \mathrm{pw}(G)$,
- G^* can be constructed in time $(|V(G)|)^{\mathcal{O}(1)} \cdot g'(H)$ for some function g'.

Observe that this will be sufficient to prove the theorem. Indeed, suppose that for some $\varepsilon > 0$ we can solve LHom(H) in time $\mathcal{O}^*((k-\varepsilon)^t)$ on instances of pathwidth t. Let us observe that applying this algorithm to G^* gives an algorithm solving the k-Coloring problem on G in time

$$(k-\varepsilon)^{\mathrm{pw}(G^*)} \cdot |V(G^*)|^{\mathcal{O}(1)} \leqslant (k-\varepsilon)^{\mathrm{pw}(G)+g(H)} \cdot (g(H) \cdot |E(G)|)^{\mathcal{O}(1)} = (k-\varepsilon)^{\mathrm{pw}(G)} \cdot |V(G)|^{\mathcal{O}(1)},$$

where the last step follows since $|H|$ is a constant. Recall that by Theorem 1', the existence of such an algorithm for k-Coloring contradicts the SETH.

We start the construction of G^* with the vertex set of G. The lists of these vertices are set to S. Then, for each edge uv of G, we introduce a copy F_{uv} of the NEQ(S)-gadget introduced in Lemma 4. We identify the interface vertices of this gadget with u and v, respectively. This completes the construction of G^*. Let us show that it satisfies the properties (a)–(d).

Note that (a) follows directly from Lemma 4. Indeed, consider an edge uv of G. On one hand, for every list homomorphism $f : (G^*, L) \to H$ we have that $f(u) \neq f(v)$. On the other hand, mapping u and v to any distinct vertices from S can be extended to a homomorphism of the whole graph F_{uv}.

To show (b), recall that the number of vertices of each F_{uv} depends only on H, let it be $g(H)$. Every original vertex of G belongs to some gadget in G^*, so G^* contains at most $g(H) \cdot |E(G)|$ vertices.

Next, consider a path decomposition $\mathcal{T}$ of G of width $\mathrm{pw}(G)$, let the consecutive bags be $X_1, X_2, \ldots, X_\ell$. We extend $\mathcal{T}$ to a path decomposition $\mathcal{T}^*$ of G^* as follows: for every edge uv in G we choose one bag X_i such that $u, v \in X_i$, and we add a new bag $X_i' = X_i \cup V(F)$, which becomes the immediate successor of X_i. We repeat this for every edge, making sure that for X_i we can only choose the original bags coming from $\mathcal{T}$. Note that it might happen that we will insert several new bags in a row, if the same X_i was chosen for different edges, but this is not a problem. Is it straightforward to observe that $\mathcal{T}^*$ is a path decomposition of G^*, and the width of $\mathcal{T}^*$ is at most $g(H) + \mathrm{pw}(G)$. This proves (c).

Finally, it is straightforward to observe that the construction of G^* was performed in time polynomial in G (recall that we treat H as a constant-size graph). ◀

4 Algorithm for general target graphs

In this section we will generalize the invariant $i^*(H)$ and extend Theorem 3' a) to all relevant target graphs H. Let us start with a simple observation, which is an analogue of Observation 7.

▶ **Observation 18 (♠).** *Let (G, L) be an instance of LHom(H). Without loss of generality we might assume the following.*

1. *The graph G is connected,*
2. *for each $x \in V(G)$, the set $L(x)$ is incomparable,*
3. *for each edge $xy \in E(G)$, for every $u \in L(x)$ there is $v \in L(y)$, such that $uv \in E(H)$.*

The high-level idea is to reduce the general case of LHOM(H) to the case, when the target is bipartite, and then use Theorem 3' a). For this, we will consider the so-called *associated instances*, introduced by Feder, Hell, and Huang [18] (see Section 4.1).

We will also separately consider some special graphs that we call *strong split graphs*. A graph H is a *strong split graph*, if its vertex set can be partitioned into two sets B and P, where B is independent and P induces a reflexive clique. We call (B, P) *the partition of* H. Note that the partition is unique: all vertices without loops must be in B and all vertices with loops must be in P.

4.1 Associated instances and clean homomorphisms

For a graph $G = (V, E)$, by G^* we denote the *associated bipartite graph*, defined as follows. The vertex set of G^* is the union of two independent sets: $\{x' : x \in V\}$ and $\{x'' : x \in V\}$. The vertices x' and y'' are adjacent if and only if $xy \in E$. Note that the edges of type $x'x''$ in G^* correspond to loops in G. The vertices x' and x'' are called *twins*. For any $W \subseteq V(G)$, we define two subsets of $V(G^*)$ as follows: $W' := \{x' : x \in W\}$ and $W'' := \{x'' : x \in W\}$.

Let (G, L) be an instance of LHOM(H). An *associated instance* is the instance (G^*, L^*) of LHOM(H^*), where L^* are *associated lists* defined as follows. For $x \in V(G)$, we set $L^*(x') = \{u' : u \in L(x)\}$ and $L^*(x'') = \{u'' : u \in L(x)\}$. Note that in the associated lists, the vertices appearing in the list of x' are precisely the twins of the vertices appearing in the list of x''. A homomorphism $f : (G^*, L^*) \to H^*$ is *clean* if it maps twins to twins, i.e., $f(x') = u'$ if and only if $f(x'') = u''$. The following simple observation was the crucial step of the proof of the complexity dichotomy for list homomorphisms, shown by Feder, Hell, and Huang [18]. We state it using slightly different language, which is more suitable for our purpose.

▶ **Proposition 19** (Feder, Hell, Huang [18]). *Let (G, L) be an instance of LHOM(H). Then it is a yes-instance if and only if (G^*, L^*) admits a clean homomorphism to H^*.*

Let us point out that the restriction to clean homomorphisms is necessary for the equivalence. Indeed, consider for example $G = K_3$, $H = C_6$ and $L(v) = V(H)$ for every $v \in V(G)$. Clearly $(G, L) \not\to H$, however, we have $G^* \simeq C_6$ and $H^* \simeq 2C_6$, so $(G^*, L^*) \to H^*$.

Recall that if H is bipartite, then LHOM(H) is polynomial-time solvable if H is the complement of a circular-arc graph [17], and NP-complete otherwise. Feder, Hell, and Huang [18] proved the following dichotomy theorem.

▶ **Theorem 20** (Feder, Hell, Huang [18]). *Let H be an arbitrary graph (with loops allowed).*
1. *The LHOM(H) problem is in P if H is a bi-arc graph, and NP-complete otherwise.*
2. *The graph H is a bi-arc-graph if and only if H^* is the complement of a circular-arc graph.*

So for our problem the interesting graphs H are those, for which H^* is not the complement of a circular-arc graph. In the observation below we summarize some properties of associated instances.

▶ **Observation 21 (♠).** *Consider an instance (G, L) of LHOM(H) and the associated instance (G^*, L^*) of LHOM(H^*). Suppose that G is given along with a tree decomposition of width t.*
1. *For each $v \in V(G)$ and $u \in V(H)$, we have $u \in L(v)$ if and only if $u' \in L^*(v')$ if and only if $u'' \in L^*(v'')$. In particular, each list is contained in one bipartition class of H^*.*
2. *In polynomial time we can construct a tree decomposition $\mathcal{T}^*$ of G^* of width at most $2t$ with the property that for each $x \in V(G)$, each bag of $\mathcal{T}$ either contains both x', x'' or none of them.*

Observe that for bipartite H, the graph H^* consists of two disjoint copies of H, so clearly $i^*(H) = i^*(H^*)$. This motivates the following definition, generalizing Definition 14.

▶ **Definition 22** ($i^*(H)$). *For a connected non-bi-arc graph H, we define $i^*(H) := i^*(H^*)$.*

4.2 Decompositions of generals target graphs

In this section we generalize the notion of decompositions of bipartite graphs, introduced in Section 2.1, to all graphs (with possible loops). The high-level idea is to define decompositions of H, so that they will correspond to bipartite decompositions of H^*. We consider the following three types of decompositions of a graph H. Note that unless stated explicitly, we do not insist that any of the defined sets is non-empty.

▶ **Definition 23** (F-decomposition). *A partition of $V(H)$ into an ordered triple of sets (F, K, Z) is an* F-decomposition *if the following conditions are satisfied (see Figure 3, left).*

1. *K is non-empty and it separates F and Z,*
2. *$|F| \geqslant 2$,*
3. *K induces a reflexive clique,*
4. *F is complete to K.*

▶ **Definition 24** (BP-decomposition). *A partition of $V(H)$ into an ordered five-tuple of sets (B, P, M, K, Z) is a* BP-decomposition *if the following conditions are satisfied (see Figure 3, middle).*

1. *$K \cup M$ is non-empty and separates $(P \cup B)$ and Z,*
2. *$|P| \geqslant 2$ or $|B| \geqslant 2$,*
3. *$K \cup P$ induces a reflexive clique and B is an independent set,*
4. *M is complete to $P \cup K$ and B is complete to K,*
5. *B is non-adjacent to M.*

▶ **Definition 25** (B-decomposition). *A partition of $V(H)$ into an ordered six-tuple of sets $(B_1, B_2, K, M_1, M_2, Z)$ is a* B-decomposition *if the following conditions are satisfied (see Figure 3, right).*

1. *$K \cup M_1 \cup M_2$ is non-empty and it separates $(B_1 \cup B_2)$ and Z,*
2. *$|B_1| \geqslant 2$ or $|B_2| \geqslant 2$,*
3. *K induces a reflexive clique and each of B_1, B_2 is an independent set,*
4. *K is complete to $M_1 \cup M_2 \cup B_1 \cup B_2$, M_2 is complete to $M_1 \cup B_1$, and M_1 is complete to B_2,*
5. *B_1 is non-adjacent to M_1 and B_2 is non-adjacent to M_2.*

Observe that a graph H can have more than one decomposition: for example, if (B, P, M, K, Z) is an BP-decomposition of H, but $M = \emptyset$, then $(B \cup P, K, Z)$ is an F-decomposition of H.

For each kind of decomposition, we define its *factors* as the following pair of graphs (H_1, H_2).

for an F-decomposition: $H_1 = H[F]$ and H_2 is obtained from H by contracting F to a vertex f. It has a loop if and only if F is not an independent set.

for a BP-decomposition: $H_1 = H[B \cup P]$ and H_2 is obtained from H by contracting P and B respectively (if they are non-empty), to vertices p and b, such that p has a loop and b does not. Also, $pb \in E(H_2)$ if and only if there is any edge between P and B in H.

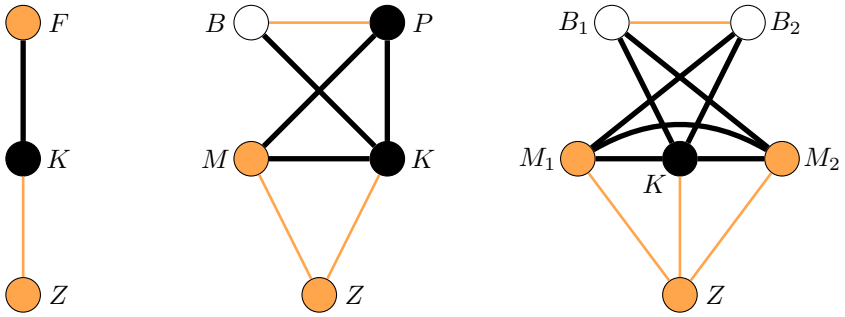


Figure 3 A schematic view of an F-decomposition (left), a BP-decomposition (middle), and a B-decomposition of H (right). Disks correspond to sets of vertices: white ones depict independent sets, black ones depict reflexive cliques, and orange ones depict arbitrary subgraphs. Similarly, thick black lines indicate that all possible edges between two sets exist, and thin orange lines depict edges that might exist, but do not have to. The lack of a line means that there are no edges between two sets.

for a B-decomposition: $H_1 = H[B_1 \cup B_2]$ and H_2 is obtained from H by contracting B_1 and B_2 respectively (if they are non-empty), to vertices b_1 and b_2 (without loops). Also, $b_1b_2 \in E(H_2)$ if and only if there is any edge between B_1 and B_2 in H.

It appears that if H is not a strong split graph, then the three types of decompositions defined above precisely correspond to bipartite decompositions of the associated bipartite graph H^*.

▶ **Lemma 26 (♠).** *Let H be be a connected, non-bi-arc graph, which is not a strong split graph. Then H^* admits a bipartite decomposition if and only if H admits a B-, a BP-, or an F-decomposition.*

Let us point out that one application of a BP-decomposition or a B-decomposition corresponds to two consecutive applications of a bipartite decomposition in H^*. Consider the case of a BP-decomposition and the bipartite decomposition $(B' \cup P'', K' \cup M' \cup P' \cup K'', Z' \cup M'' \cup B'' \cup Z'')$ of H'. Note that after contracting B' to b' and P'' to p'', we still have a decomposition $(P' \cup B'', K' \cup K'' \cup M'' \cup \{p''\}, M' \cup \{b'\} \cup Z' \cup Z'')$ of the second factor $(H^*)_2$ of H^*. Similarly, in the case of a B-decomposition, we still have another bipartite decomposition of $(H^*)_2$, where the new set D is $B_1'' \cup B_2'$.

Note that in a BP-decomposition, a B-decomposition, and an F-decomposition, when F is an independent set or contains a vertex with a loop, the factors are always induced subgraphs of H. Indeed, we can equivalently obtain H_2 by removing certain vertices from H. In the last case of an F-decomposition, when F contains at least one edge and has only vertices without loops, H_2 is not an induced subgraph of H. We can equivalently define H_2 as the graph obtained by removing from F all but two adjacent vertices, and then replacing them with a vertex with a loop.

In the next lemma we consider a graph H' that was obtained from H by a series of decompositions (i.e., it is a factor of H, or a factor of a factor of H etc.). We show that even if H' is not an induced subgraph of H, the associated bipartite graph H'^* is still an induced subgraph of H^*.

▶ **Lemma 27 (♠).** *Let H' be a graph obtained from H by a series of decompositions. Then H'^* is an induced subgraph of H^*.*

Finally, let us show an analogue of Lemma 11 for general graphs. Recall that $T(H, n, t)$ denotes an upper bound for the complexity of LHOM(H) on instances with n vertices, given along with a tree decomposition of width t. Note that we do not assume that $|H|$ is a constant [2].

▶ **Lemma 28** (♠ General decomposition lemma). *Let H be a connected, non-bi-arc graph, and let Γ be a decomposition of H (i.e., Γ is either an F-, a BP-, or a B-decomposition) with factors H_1, H_2. If there exist constants $c \geqslant 1$ and $d > 2$ such that $T(H_1, n, t) = \mathcal{O}\left(c^t \cdot (n \cdot |H_1|)^d\right)$ and $T(H_2, n, t) = \mathcal{O}\left(c^t \cdot (n \cdot |H_2|)^d\right)$, then $T(H, n, t) = \mathcal{O}\left(c^t \cdot (n \cdot (|H| + 2))^d\right)$.*

4.3 Solving LHom(H) for general target graphs

By Lemma 26, it is straightforward to observe the following.

▶ **Observation 29.** *If H is a connected, undecomposable, non-bi-arc graph, then $i^*(H)$ is the size of the largest incomparable set in H.*

In this section we sketch the proof of the following, slightly stronger version of Theorem 5 a), where the input tree decomposition is not assumed to be optimal.

▶ **Theorem 5' a).** *Let H be non-bi-arc graph. Even if H is given as an input, the LHOM(H) problem with instance (G, L) can be solved in time $\mathcal{O}\left(i^*(H)^t \cdot (n \cdot |H|)^{\mathcal{O}(1)}\right)$ for any lists L, provided that G is given with its tree decomposition of width t.*

The main idea is similar to the one in the proof of Theorem 3' a): given an instance of LHOM(H), we recursively decompose H into smaller subgraphs and reduce the initial instance to a number of instances of list homomorphism to these smaller subgraphs. Finally, we solve the problem for leaves of the recursion tree, and then, using Lemma 28 in a bottom-up fashion, we will compute the solution to the original instance. The only thing missing is how to solve the instances corresponding to leaves of the recursion tree. We describe this in the following results.

▶ **Corollary 30** (♠). *Let $\widehat{H}$ be a graph and let H be a strong split graph, which was obtained from $\widehat{H}$ by a series of decompositions. The LHOM(H) problem with instance (G, L) with n vertices, given along with a tree decomposition of width t, can be solved in time $\mathcal{O}(i^*(\widehat{H})^t \cdot (n \cdot |H|)^{\mathcal{O}(1)})$.*

▶ **Lemma 31.** *For any graph H, any n-vertex instance (G, L) of LHOM(H) can be solved in time $\mathcal{O}\left(i(H^*)^t \cdot (n \cdot |H|)^{\mathcal{O}(1)}\right)$, assuming a tree decomposition of G with width at most t is given.*

Proof. Let (G^*, L^*) be the associated instance of LHOM(H^*). By Proposition 19, we know that $(G, L) \to H$ if and only if there is a clean homomorphism $(G^*, L^*) \to H^*$. We will focus on finding such a clean homomorphism.

First, recall that by Observation 18 (2) and Observation 21 (1), the instance (G^*, L^*) is consistent. So, by Corollary 9, the size of each list in L^* is at most $i(H^*)$. Moreover, by Observation 21 (1), for every $x \in V(G)$, the vertices in $L^*(x')$ are precisely the twins

[2] Let us point out that in the full version of the paper we actually prove three variants of Lemma 28, one for each type of decomposition. Furthermore, in the statements of these three lemmas we are more explicit about the constants, to make sure that they do not increase multiplicatively. This is necessary to avoid introducing factors exponential in $|H|$ in the proof of Theorem 5 a).

of vertices in $L^*(x'')$. Finally, by Observation 21 (2), in polynomial time we can obtain a tree decomposition $\mathcal{T}^*$ of G^* with width at most $2t$, in which vertices of G^* come in pairs: whenever any bag contains x', it also contains x''.

Consider the straightforward dynamic programming algorithm for LHOM(H^*), using the tree decomposition $\mathcal{T}^*$ of G^*. We observe that since we are looking for a clean homomorphism, we do not need to remember partial solutions, in which the colors of twins do not agree. Thus, even though the size of each bag of $\mathcal{T}^*$ is at most $2t$, the number of partial colorings we need to consider is bounded by $(\max_{x \in V(G^*)} |L^*(x)|)^t \leqslant i(H^*)^t$. So the claim follows. ◀

Finally, let us wrap everything up and prove Theorem 5' a).

Proof of Theorem 5' a). Let (G, L) be an instance of LHOM(H), where G has n vertices and is given with its tree decomposition of width at most t. Again, we can assume that n is sufficiently large, as otherwise we can solve the problem by brute-force. We proceed as in the Theorem 3' a). We consider a recursion tree $\mathcal{R}$, obtained by decomposing H recursively. For each node corresponding to some graph H', we construct its children recursively, unless none of the following happens (i) H' is a bi-arc graph, (ii) H' is bipartite, (iii) H' is a strong split graph, or (iv) H' is undecomposable.

We compute the solutions in a bottom-up fashion. First, consider a leaf of the recursion tree, let the corresponding target graph for this node of $\mathcal{R}$ be H'. If H' is a bi-arc graph, we can solve the problem in polynomial time. If H' is bipartite, we solve the problem in time $\beta \cdot i^*(H') \cdot (n \cdot |H'|)^{d_1} \leqslant \beta \cdot i^*(H) \cdot (n \cdot |H'|)^{d_1}$ for some constants β and d_1, using the algorithm from Theorem 3' a). If H' is a strong split graph, we can solve the problem in time $\gamma \cdot i^*(H) \cdot (n \cdot |H'|)^{d_2}$, for constants γ, d_2, using the algorithm from Corollary 30.

So finally consider the remaining case, i.e., that H' is connected, non-bi-arc, non-bipartite, undecomposable, which is not a strong split graph. Furthermore we know that H' was obtained from H by a sequence of decompositions. Recall that by Lemma 31, we can solve the instances of LHOM(H') in time $\delta \cdot i(H'^*)^t \cdot (n \cdot |H'|)^{d_3}$, for some constants δ, d_3. Let us consider the graph H'^*, by Lemma 27 we know that H'^* is an induced subgraph of H^*. Also, H'^* is either connected (if H' is non-bipartite), or consists of two disjoint copies of H' (if H' is bipartite). Moreover, by Lemma 26, we observe that H'^* is undecomposable. Thus, by the definition of $i^*(H)$, we observe that

$$i(H'^*) \leqslant \max\{i(H'') \colon H'' \text{ is an undecomposable, connected, induced subgraph of } H^*, \text{ whose complement is not a circular-arc graph}\} = i^*(H).$$

So the algorithm from Lemma 31 solves the instances corresponding to leaves of $\mathcal{R}$ in time $\delta \cdot i^*(H)^t \cdot (n \cdot |H'|)^{d_3}$. Define $\alpha := \max(2\beta, \gamma, \delta)$ and $d := \max(d_1, d_2, d_3, 3)$ [3]. By applying Lemma 28 for every non-leaf node of $\mathcal{R}$ in a bottom-up fashion, we conclude that we can solve LHOM(H) in time $\alpha \cdot i^*(H)^t \cdot (n \cdot |H|)^d = \mathcal{O}\left(i^*(H)^t \cdot (n \cdot |H|)^{\mathcal{O}(1)}\right)$, which completes the proof. ◀

[3] The choice of these constants follows from the full statement of Lemma 28, refined for each of the decompositions.

5 Conclusion

Let us conclude the paper with some side remarks and pointing out several open problems.

5.1 Special cases: reflexive and irreflexive graphs

Recall that the crucial tool for our algorithmic results were the decompositions of a connected graph H, introduced in Section 2.1 (for bipartite graphs H) and in Section 4.2 (for general graphs H). Let us analyze how the general decompositions behave in two natural special cases: if H is either a reflexive or an irreflexive graph. We use the notation introduced in Definition 23, Definition 24, and Definition 25.

First we consider the case that H is reflexive, i.e., every vertex of H has a loop. Let us point out that a B-decomposition cannot occur in this case, as the sets B_1, B_2 are empty. In case of an F-decomposition we obtain exactly the decomposition defined by Egri et al. [15, Lemma 8]. Finally, in the case of a BP-decomposition, note that the set B is empty and therefore each vertex in P has exactly the same neighborhood. Thus the total contribution of the vertices in P to $i^*(H)$ is at most 1. Therefore the only type of decomposition that can be algorithmically exploited in reflexive graph is the F-decomposition, as observed by Egri et al. [15].

Now let us consider the case that H is irreflexive, i.e., no vertex of H has a loop. Observe that the sets K and P are reflexive cliques, so they are empty in our case. Thus BP-decompositions and F-decompositions do not occur in this case (recall that H is connected). Therefore the only possibility left is a B-decomposition, in which the set K is empty. Let us point out that this decomposition is very similar to the bipartite decomposition, in particular, the graph H_1 is bipartite (while H_2 might be non-bipartite). This gives even more evidence that the case of bipartite graphs H is a crucial step to understanding the complexity of the LHom(H) problem. Actually, if H is bipartite, then the B-decomposition turns out to be equivalent to the bipartite decomposition introduced in Section 2.1.

5.2 Generalized algorithm

We believe that the decompositions that we discovered can be used for many problems concerning the complexity of variants of the LHom(H) problem, e.g., for various other parameterizations. Let us point out that in the proofs of Lemma 11 and Lemma 28, we did not really require that the running time is of the form $\mathcal{O}(c^{\mathrm{tw}(G)} \cdot (n \cdot |H|)^d)$, for a constant c.

Moreover, recall that in each variant of the decomposition lemma, all instances for which we computed partial results were induced subgraphs of G, the original instance. This motivates the following, generalized statement. For a graph G, and a graph H, let $T(G, H, n)$ be an upper bound for the complexity of the LHom(H) problem for instances of size n, which are induced subgraphs of G.

▶ **Corollary 32.** *Let H be a connected, non-bi-arc graph. Let $\mathcal{R}$ be its recursion tree and let $\mathcal{H}$ be the set of graphs associated with leaves of $\mathcal{R}$. Consider an instance (G, L) of LHom(H) with n vertices. Suppose that that for each $H' \in \mathcal{H}$ it holds that $T(G, H', n') \leqslant f(n', H')$, where f is superadditive with respect to its first argument. Then (G, L) can be solved in time $\mathcal{O}\left(\sum_{H' \in \mathcal{H}} f(n, H') + n^2 \cdot |H|^3\right)$.*

Sketch of proof. The proof is analogous to the proofs of Theorem 3' a) and Theorem 5' a). Recall that we can compute the instances associated with the leaves of $\mathcal{R}$, and then proceed in a bottom-up fashion. Even though there might be more than one instance associated with a leaf node, we observe that their numbers of vertices sum up to n , so by the superadditivity of f we can bound the running time related to each leaf node, associated with H', by $f(n, H')$.

Now let us consider an internal node of $\mathcal{R}$, it is associated with some subgraph H' of H. Recall that the computation for this node consists of the computations for child nodes and $\mathcal{O}(n^2|H'|^2)$ additional steps. Since the total number of nodes of $\mathcal{R}$ is $\mathcal{O}(|H|)$, we can bound the total running time for the root node (i.e., time needed to solve (G, L)) by $\mathcal{O}\left(\sum_{H' \in \mathcal{H}} f(n, H') + n^2 \cdot |H|^3\right)$. ◀

5.3 Further research directions

In this paper we have shown tight complexity bounds for the list homomorphism problem, parameterized by the treewidth of the instance graph.

A very natural question, mentioned also in [15], is to provide analogous results for the non-list variant of the problem, denoted by $\textsc{Hom}(H)$. As we already mentioned, Okrasa and Rzążewski were able to provide tight bounds for the complexity of $\textsc{Hom}(H)$, assuming two conjectures from algebraic graph theory from early 2000s [35]. It would be very interesting to strengthen these results, either by proving the mentioned two conjectures, or by providing a different reduction.

Another interesting direction is to consider one of many other variants of the graph homomorphism problem. Let us mention one, i.e., *locally surjectve homomorphism*, denoted by $\textsc{LSHom}(H)$. In this problem we ask for a homomorphism from an instance graph G to the target graph H, which is surjective on each neighborhood. In other words, if we map some vertex $x \in V(G)$ to some vertex $v \in V(H)$, then every neighbor of v must appear as a color of some neighbor of x [21, 19, 20, 36]. We believe that it is interesting to show tight complexity bounds for this problem. One of the reasons why this problem is challenging is that the natural dynamic programming runs in time $2^{\mathcal{O}(|H| \cdot \mathrm{tw}(G))} \cdot (n + |H|)^{\mathcal{O}(1)}$. Thus in order to show that this bound is tight, it is not sufficient to design edge gadgets encoding inequality and substitute all edges of the instance of k-Coloring with these edge gadgets, as we did in this paper. Since the number of colors needs to be exponential in H, one should also design some *vertex gadgets*, which will encode the exponential number of possible states.

Finally, instead of changing the problem, we can consider changing the parameter. We believe that an exciting question is to find tight bounds for $\textsc{Hom}(H)$ and $\textsc{LHom}(H)$, parameterized by the *cutwidth* of the instance graph, denoted by $\mathrm{cw}(G)$. Quite recently Jansen and Nederlof showed that the chromatic number of a graph can be found in time $2^{\mathcal{O}(\mathrm{cw}(G))} \cdot n^{\mathcal{O}(1)}$ [30], i.e., the base of the exponential factor does not depend on the number of colors. Jansen [29] asked whether the same is possible for $\textsc{Hom}(H)$ and $\textsc{LHom}(H)$, if the target H is not complete? Note that while the chromatic number of a graph can be found in time $2^n \cdot n^{\mathcal{O}(1)}$ [3], for the $\textsc{Hom}(H)$ and $\textsc{LHom}(H)$ problems the $|H|^{\mathcal{O}(n)}$-time algorithm is essentially best possible, assuming the ETH [12]. We believe that a similar phenomenon might occur if the cutwidth is a parameter.

References

1 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, April 1987. doi:10.1137/0608024.

2 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989. doi:10.1016/0166-218X(89)90031-0.

3 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. doi:10.1137/070683933.

4 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

5 Hans L. Bodlaender, Paul S. Bonsma, and Daniel Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, pages 41–53, 2013. doi:10.1007/978-3-319-03898-8_5.

6 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

7 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

8 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, May 2008. doi:10.1093/comjnl/bxm037.

9 Andrei A. Bulatov. Constraint satisfaction problems: Complexity and algorithms. In Shmuel Tomi Klein, Carlos Martín-Vide, and Dana Shapira, editors, *Language and Automata Theory and Applications*, pages 1–25, Cham, 2018. Springer International Publishing.

10 Rajesh Chitnis, László Egri, and Dániel Marx. List H-coloring a graph by removing few vertices. *Algorithmica*, 78(1):110–146, 2017. doi:10.1007/s00453-016-0139-6.

11 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

12 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight lower bounds on graph embedding problems. *J. ACM*, 64(3):18:1–18:22, 2017. doi:10.1145/3051094.

13 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

14 László Egri, Andrei A. Krokhin, Benoît Larose, and Pascal Tesson. The complexity of the list homomorphism problem for graphs. *Theory Comput. Syst.*, 51(2):143–178, 2012. doi:10.1007/s00224-011-9333-8.

15 László Egri, Dániel Marx, and Paweł Rzążewski. Finding list homomorphisms from bounded-treewidth graphs to reflexive graphs: a complete complexity characterization. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.27.

16 Tomas Feder and Pavol Hell. List homomorphisms to reflexive graphs. *Journal of Combinatorial Theory, Series B*, 72(2):236–250, 1998. doi:10.1006/jctb.1997.1812.

17 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. doi:10.1007/s004939970003.

18 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003. doi:10.1002/jgt.10073.

19 Jirí Fiala and Jana Maxová. Cantor-Bernstein type theorem for locally constrained graph homomorphisms. *Eur. J. Comb.*, 27(7):1111–1116, 2006. doi:10.1016/j.ejc.2006.06.003.

20 Jirí Fiala and Daniël Paulusma. A complete complexity classification of the role assignment problem. *Theor. Comput. Sci.*, 349(1):67–81, 2005. doi:10.1016/j.tcs.2005.09.029.

21 Jirí Fiala, Daniël Paulusma, and Jan Arne Telle. Matrix and graph orders derived from locally constrained graph homomorphisms. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29 - September 2, 2005, Proceedings*, volume 3618 of *Lecture Notes in Computer Science*, pages 340–351. Springer, 2005. doi:10.1007/11549345_30.

22 Fedor V. Fomin, Pinar Heggernes, and Dieter Kratsch. Exact algorithms for graph homomorphisms. *Theory Comput. Syst.*, 41(2):381–393, 2007. doi:10.1007/s00224-007-2007-x.

23 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004. doi:10.1016/j.apal.2004.01.007.

24 Richard Hammack, Wilfried Imrich, and Sandi Klavžar. *Handbook of product graphs*. Discrete Mathematics and its Applications (Boca Raton). CRC Press, Boca Raton, FL, second edition, 2011. With a foreword by Peter Winkler.

25 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*. Oxford University Press, 2004.

26 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

27 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

28 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

29 Bart M. P. Jansen. personal communication.

30 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.

31 Tomasz Kociumaka and Marcin Pilipczuk. Deleting vertices to graphs of bounded genus. *Algorithmica*, 81(9):3655–3691, 2019. doi:10.1007/s00453-019-00592-7.

32 Stefan Kratsch, Daniel Lokshtanov, Dániel Marx, and Peter Rossmanith. Optimality and tight results in parameterized complexity (dagstuhl seminar 14451). *Dagstuhl Reports*, 4(11):1–21, 2014. doi:10.4230/DagRep.4.11.1.

33 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

34 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. *CoRR*, abs/2006.11155, 2020. arXiv:2006.11155.

35 Karolina Okrasa and Paweł Rzążewski. Fine-grained complexity of graph homomorphism problem for bounded-treewidth graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, pages 1578–1590, 2020. doi:10.1137/1.9781611975994.97.

36 Karolina Okrasa and Paweł Rzążewski. Subexponential algorithms for variants of the homomorphism problem in string graphs. *J. Comput. Syst. Sci.*, 109:126–144, 2020. doi:10.1016/j.jcss.2019.12.004.

37 Michał Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *MFCS 2011*, volume 6907, pages 520–531. Springer, 2011.

38 Paweł Rzążewski. Exact algorithm for graph homomorphism and locally injective graph homomorphism. *Inf. Process. Lett.*, 114(7):387–391, 2014. doi:10.1016/j.ipl.2014.02.012.

39 William T. Trotter and John I. Moore. Characterization problems for graphs, partially ordered sets, lattices, and families of sets. *Discrete Mathematics*, 16(4):361–381, 1976. doi:10.1016/S0012-365X(76)80011-8.

40 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

41 Magnus Wahlström. New plain-exponential time classes for graph homomorphism. *Theory Comput. Syst.*, 49(2):273–282, 2011. doi:10.1007/s00224-010-9261-z.

Generalizing CGAL Periodic Delaunay Triangulations

Georg Osang
IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria
georg.osang@ist.ac.at

Mael Rouxel-Labbé
GeometryFactory, Grasse, France
mael.rouxel.labbe@geometryfactory.com

Monique Teillaud
Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
Monique.Teillaud@inria.fr

Abstract

Even though Delaunay originally introduced his famous triangulations in the case of infinite point sets with translational periodicity, a software that computes such triangulations in the general case is not yet available, to the best of our knowledge. Combining and generalizing previous work, we present a practical algorithm for computing such triangulations. The algorithm has been implemented and experiments show that its performance is as good as the one of the CGAL package, which is restricted to cubic periodicity.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Delaunay triangulation, lattice, algorithm, software, experiments

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.75

Supplementary Material `https://members.loria.fr/Monique.Teillaud/CGAL_periodicDT_ESA20/`

Funding *Georg Osang*: This author is partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant no. 788183. *Monique Teillaud*: This author is partially supported by the grants ANR-17-CE40-0033 of the French National Research Agency ANR (project SoS) and INTER/ANR/16/11554412/SoS of the Luxembourg National Research fund FNR (`https://members.loria.fr/Monique.Teillaud/collab/SoS/`).

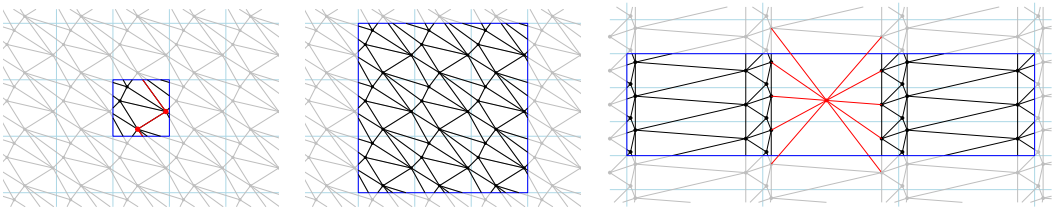
1 Introduction

Delaunay triangulations are one of the most prominent structures in computational geometry. While nowadays many applications use Delaunay triangulations of finite point sets in Euclidean space, Delaunay originally introduced the notion in the context of infinite point sets with translational periodicity [9]. Such periodic point sets are abundant in fields such as crystallography and material sciences; thus their communities would benefit from software that computes Delaunay triangulations of infinite periodic point sets in Euclidean space. Specifically, given a d-dimensional lattice and its associated translation group, the orbits of a given finite point set in $\mathbb{R}^d$ with respect to this translation group define a periodic point set. Our aim is to compute a finite representation of the periodic Delaunay triangulation of such a periodic point set, specifically a projection of the triangulation onto the flat d-torus that is the quotient space of $\mathbb{R}^d$ under the action of the translation group.

The first algorithm for this problem was already outlined in 1997 [12], yet a robust and efficient implementation for this problem does not exist to date as far as we know. The Voro++ library [18] is focused on crystallographic applications in 3 dimensions; however it is

limited to orthogonal lattices. Zeo++ [22] extends its functionality to arbitrary 3-dimensional lattices. Both libraries compute the Voronoi cell of a given input point as an intersection of half-spaces, by searching for other points around it that have an influence on its Voronoi cell. The combinatorics of the Delaunay triangulation cannot be easily accessed. The CGAL library [17] provides packages for periodic Euclidean Delaunay triangulations in 2D and 3D, which currently are limited to the integer lattice, referred to as the square and cubic setting and 2 and 3 dimensions, respectively [15, 6, 5]. We propose an addition to CGAL that extends this functionality to arbitrary lattices.

The algorithm by Dolbilin and Huson [12] creates 3^d copies of each input point and computes their finite Delaunay triangulation, from which a representation of the periodic Delaunay triangulation is extracted. The CGAL algorithm [4, 7] computes the triangulation in a finitely-sheeted covering space of the d-torus. It is based on the classical incremental algorithm by Bowyer and Watson [3, 21], and requires that the triangulation be a simplicial complex at any given time. Let us quickly recall that a triangulation is a simplicial complex, or is *simplicial* for short, if each of its simplices consists of a set of distinct vertices, and the intersection of any two simplices is either empty or a simplex. Operating directly on the d-torus does not guarantee this, see Figure 1a. Thus, in the cubic setting, a 3^d-sheeted cover is used (Figure 1b) until sufficiently many points have been inserted to guarantee that the triangulation in the 1-sheeted cover is a simplicial complex. Unfortunately, a 3^d-sheeted cover is not sufficient for more general periodic point sets: As the 3^d copies of each point have to be inserted iteratively into the 3^d-sheeted cover, simpliciality can be violated in the intermediate stages of point insertion, see Figure 1c. While there always exists a finitely-sheeted covering space [7] that ensures simpliciality, the number of sheets might be prohibitively large. Thus, we propose a different approach.



(a) The intersection of the two red edges is not a simplex but a set of two vertices. Thus the triangulation is not simplicial.

(b) The 9-sheeted cover guarantees a simplicial triangulation in the square setting.

(c) In the non-square setting, incremental point insertion of the 9 copies into the 9-sheeted cover can violate simpliciality. Here, the first of 9 copies is being inserted.

Figure 1 Representation of the projections of the periodic Delaunay triangulation into the 1-sheeted (left) and 9-sheeted cover (middle, right) of the 2-torus.

Overview. After formally defining the problem in Section 2, we propose an algorithm (Section 3) for periodic Delaunay triangulations that combines two different approaches and consists of two phases, both of which use Bowyer-Watson's algorithm. While for 2-dimensional spaces algorithms based on flips circumvent the simpliciality requirement [10], we stick to Bowyer-Watson's algorithm as it easily generalizes to 3 (and higher) dimensions. Furthermore it enables an efficient, clean, and easily maintainable implementation. The first phase of our algorithm (Section 3.2) refines the algorithm by Dolbilin and Huson [12], and its implementation details are based on some new results. It uses 3^d copies of each input point, regardless of the lattice, and computes a finite Euclidean Delaunay triangulation on this point set, from which a representation of the Delaunay triangulation on the d-torus is obtained. Once a simpliciality criterion is met, our algorithm switches to the second phase

(Section 3.4), which conceptually follows the CGAL implementation of the cubic case [4]. It operates directly on the d-torus, maintaining only one copy of each input point, and thus provides better insertion running times than phase 1. A first version of our open-source implementation in 2D and 3D is publicly available.[1] This implementation is expected to be an integral part of CGAL in a near future release. Experiments (Section 4) show similar performances as the CGAL implementation restricted to cubic lattices [5]. We close with a discussion of future extensions in Section 5.

2 Preliminaries

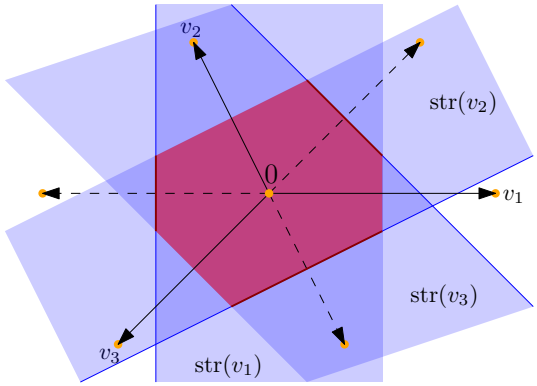
Let us recall various notions [13, 8] that are employed throughout the algorithm. Let $B = \{b_1, b_2, \dots, b_d\}$ be a basis of $\mathbb{R}^d$. The point set $\Lambda := \{\sum_{i=1}^{d} z_i b_i \colon z \in \mathbb{Z}^d\}$ is called a *lattice*, and B is its *lattice basis*. The lattice Λ is associated with the translation group Γ consisting of the translations $\phi_\lambda \colon \mathbb{R}^d \to \mathbb{R}^d$ mapping the origin to λ, for each $\lambda \in \Lambda$. The group Γ acts on $\mathbb{R}^d$ and each translation of Γ maps Λ onto itself. We denote the length of the shortest non-zero lattice vector as $\mathrm{sv}(\Lambda)$. For a given lattice basis B, we call $B_{\mathrm{sup}} := B \cup \{b_0\}$ with $b_0 = -\sum_{i=1}^{d} b_i$ its *superbase*. A superbase is *obtuse* if for any pair b_i and b_j, $\langle b_i, b_j \rangle \leq 0$. A basis is *reduced* if its superbase is obtuse [13, Definition 4.4]. This notion is defined in such a way that we can easily compute $\mathrm{sv}(\Lambda)$ and Dirichlet domains. The *Dirichlet domain* of a lattice point $\lambda \in \Lambda$ is the region of λ in the Voronoi tesselation of Λ, or more formally $\mathrm{dom}(\lambda, \Lambda) := \{p \in \mathbb{R}^d \colon \|p - \lambda\| \leq \|p - \nu\| \ \forall \nu \in \Lambda\}$. It is a convex polytope, and we call the lattice Λ *generic* if each vertex of $\mathrm{dom}(0, \Lambda)$ is incident to the Dirichlet domains of exactly d other lattice points. For 2-dimensional generic lattices the Dirichlet domains are hexagons, for 3-dimensional generic lattices they are combinatorially equivalent to truncated cubes.

For a lattice vector λ, $\mathrm{str}(\lambda) = \{p \in \mathbb{R}^d \colon -0.5 \leq \frac{\langle p, \lambda \rangle}{\langle \lambda, \lambda \rangle} < 0.5\}$ is an infinite half-open strip that contains the subspace orthogonal to λ through the origin. Then $\mathrm{dom}(0, \Lambda)$ is the closure of the intersection of these strips for all non-zero lattice vectors. However, as $\mathrm{dom}(0, \Lambda)$ only has a finite number of facets, there must be a finite subset of strips whose closed intersection yields $\mathrm{dom}(0, \Lambda)$. Let $\mathcal{V}$ be the minimal set of lattice vectors (together with their negates) such that the closure of $\bigcap_{v \in \mathcal{V}} \mathrm{str}(v)$ is $\mathrm{dom}(0, \Lambda)$. The vectors in $\mathcal{V}$ are commonly called *Voronoi-relevant vectors*. Each of them is a normal vector of a facet of the Dirichlet domain of 0, and thus there are at most $2(2^d - 1)$ Voronoi relevant vectors [13, Theorem 3.6]. Let $\mathcal{V}^+ \sqcup \mathcal{V}^-$ be a partition of $\mathcal{V}$ such that if $v \in \mathcal{V}^+$, then $-v \in \mathcal{V}^-$, and vice versa. For a fixed choice of $\mathcal{V}^+$ (and implicitly $\mathcal{V}^-$), we define the *canonical* domain $\mathrm{dom}^{\mathbin{\rotatebox{90}{)}}}(0, \Lambda) := \bigcap_{v \in \mathcal{V}^+} \mathrm{str}(v)$ (Figure 2). Its closure is $\mathrm{dom}(0, \Lambda)$, and its images under Γ, denoted $\mathrm{dom}^{\mathbin{\rotatebox{90}{)}}}(\lambda, \Lambda) := \phi_\lambda(\mathrm{dom}^{\mathbin{\rotatebox{90}{)}}}(0, \Lambda))$ for $\lambda \in \Lambda$, form a partition of $\mathbb{R}^d$. With $k\Lambda$ for $k \in \mathbb{Z}$ referring to the lattice with basis $\{kb_1, \dots, kb_d\}$, we note that $\mathrm{dom}^{\mathbin{\rotatebox{90}{)}}}(0, k\Lambda)$ is $\mathrm{dom}^{\mathbin{\rotatebox{90}{)}}}(0, \Lambda)$ scaled by a factor of k.

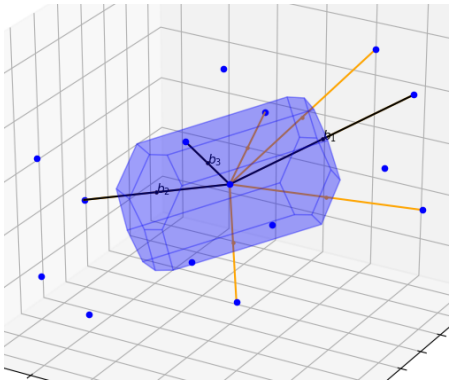
For $d \leq 3$, if we have a reduced lattice basis B with its superbase B_{sup}, then $\mathcal{V}$ is a subset of $\{\sum_{v \in S} v \colon S \subset B_{\mathrm{sup}}, S \neq \emptyset, S \neq B_{\mathrm{sup}}\}$ [8, Theorems 3 and 8], with equality if the lattice is generic. Note that $\mathrm{sv}(\Lambda)$ can be obtained as the length of the shortest vector in $\mathcal{V}$.

Delaunay triangulations. The *Delaunay triangulation* $\mathrm{Del}(X)$ of an input point set $X \subset \mathbb{R}^d$ is a collection of simplices up to dimension d whose vertex set is X and each d-simplex (which we refer to as a *cell*) corresponds to a set of $d + 1$ points whose open circumscribing ball does not contain any other points of X. We call the $(d - 1)$-simplices of $\mathrm{Del}(X)$ its *facets*.

[1] https://members.loria.fr/Monique.Teillaud/CGAL_periodicDT_ESA20/



(a) $\mathrm{dom}^{\mathrm{C}}(0,\Lambda)$ (red) is the intersection of the half-open strips (blue) of $\mathcal{V}^+ = \{v_1, v_2, v_3\}$.



(b) $\mathrm{dom}^{\mathrm{C}}(0,\Lambda)$ of a 3-dimensional lattice (blue). Lattice basis B in black, $\mathcal{V}^+ \supset B$ in orange.

Figure 2 Canonical domains and Voronoi relevant vectors for lattices in 2D and 3D.

Given a lattice Λ and a finite set of points X, we get the *periodic point set* $\Gamma X := \{\phi_\lambda(x) \colon x \in X \text{ and } \lambda \in \Lambda\}$ consisting of the elements of the orbits of X under Γ. ΓX is globally invariant under Γ. Then $\mathrm{Del}(\Gamma X)$ is the *periodic Delaunay triangulation* of the infinite point set ΓX. Note that we can ignore degeneracies in ΓX by using the symbolic perturbation provided by CGAL [11]: it is translation-invariant, so, degeneracies are triangulated in a consistent way, which ensures that the computed $\mathrm{Del}(\Gamma X)$ is actually invariant under Γ. The orbit space $\mathbb{R}^d/\Gamma$ is a *flat torus*, and we denote its projection map as $\pi \colon \mathbb{R}^d \to \mathbb{R}^d/\Gamma$. The *torus triangulation* $\mathrm{Del}(\Gamma X)/\Gamma$ is the projection of $\mathrm{Del}(\Gamma X)$ into $\mathbb{R}^d/\Gamma$. Using $\mathrm{dom}^{\mathrm{C}}(0,\Lambda)$ as a geometric representation of the torus, we can use $X_0 := \Gamma X \cap \mathrm{dom}^{\mathrm{C}}(0,\Lambda)$ as *canonical* representatives of the vertex set of $\mathrm{Del}(\Gamma X)/\Gamma$. While $\mathrm{Del}(\Gamma X)/\Gamma$ gives us a finite representation of $\mathrm{Del}(\Gamma X)$, unlike $\mathrm{Del}(\Gamma X)$ it is not necessarily simplicial (see Figure 1a).

3 Algorithm

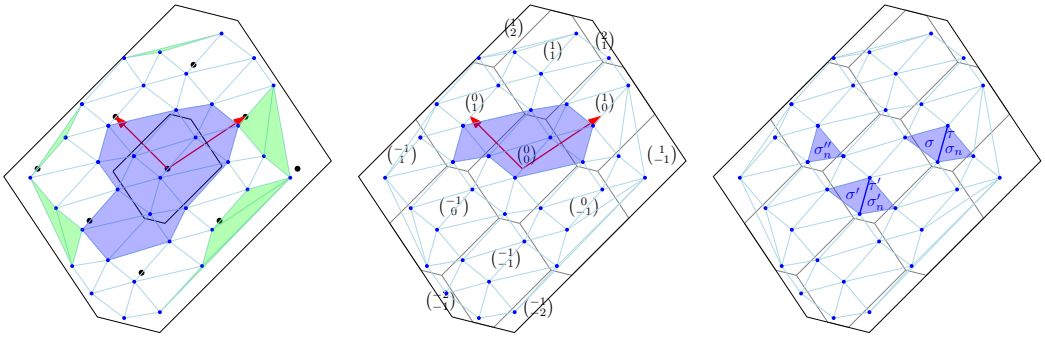
The input to our algorithm is a lattice basis B' for Λ and a set of points X defining the periodic point set. The output is an object representing $\mathrm{Del}(\Gamma X)/\Gamma$. This object provides a uniform interface that, regardless of the internal state of our algorithm, allows the user to access the properties of Λ as well as the torus triangulation $\mathrm{Del}(\Gamma X)/\Gamma$. For Λ this includes the reduced basis $B = \{b_1, \ldots, b_d\}$. For $\mathrm{Del}(\Gamma X)/\Gamma$ this includes the set X_0 of canonical representatives for its vertex set, and its cells. Cells are not solely defined by their vertex set, but have additional geometric information attached. Specifically, each vertex of a cell is represented as a point x from X_0 with an associated offset $o = (o_1, \ldots, o_d)$, which is an integer vector. The geometric location of the vertex then is $x + \sum_{i=1}^{d} o_i b_i$. In alignment with other CGAL triangulations, we also provide access to simplices of lower dimensions, represented with associated vertex offsets akin to cells, as well as neighborhood relations. These include querying a cell for its adjacent cells, or querying a vertex for its incident cells or lower dimensional simplices. While many steps generalize, we restrict our focus to 2- and 3-dimensional triangulations, which are the most widely used.

Internally, our algorithm operates in two phases, which use two different data structures, respectively: The first phase maintains a finite Euclidean Delaunay triangulation while the second phase maintains a triangulation of $\mathrm{Del}(\Gamma X)/\Gamma$.

Let us now recall the result that is crucial to the first phase of our algorithm. For a point set X, let $X_3 := \mathrm{dom}^{\square}(0, 3\Lambda) \cap \Gamma X$, i.e. all periodic copies of X that lie within the Dirichlet domain of 0 scaled by a factor of 3. We call a cell of $\mathrm{Del}(X_3)$ a *periodic cell* if it is also a cell of the periodic triangulation $\mathrm{Del}(\Gamma X)$ (see Figure 3a).

▶ **Proposition 1** ([12, Lemma 3.4]). *Given a point set X, each cell of $\mathrm{Del}(X_3)$ that has at least one vertex in $\mathrm{dom}^{\square}(0, \Lambda)$ is a periodic cell. Furthermore the set of these cells contains at least one periodic copy of each cell of $\mathrm{Del}(\Gamma X)/\Gamma$.*

After some preprocessing that essentially consists in computing the canonical domain, the first phase internally maintains $\mathrm{Del}(X_3)$ using the CGAL packages for Euclidean Delaunay triangulations [14, 23]. For this we need to develop a systematic way of computing X_3 and obtaining the interface for $\mathrm{Del}(\Gamma X)/\Gamma$ from the internal data structure $\mathrm{Del}(X_3)$.



(a) Cells in blue are guaranteed to be periodic cells by Proposition 1. Green: non-periodic cells. Black vertices: Λ.

(b) The set of canonical cells in blue. Each copy of $\mathrm{dom}(0, \Lambda)$ is labeled with its offset.

(c) Finding the periodic neighbor of σ across the edge τ, which is σ''_n, the canonical representative of σ'_n.

Figure 3 In blue, the points X_3 with their Delaunay triangulation $\mathrm{Del}(X_3)$. The large hexagon is $\mathrm{dom}(0, 3\Lambda)$, while the smaller ones are $\mathrm{dom}(0, \Lambda)$ and (middle, right) its periodic copies. The reduced lattice basis is drawn in red (left, middle).

Once we can guarantee that $\mathrm{Del}(\Gamma X)/\Gamma$ is simplicial and will remain so for any future point insertions, we switch to the second phase. Simpliciality guarantees that the Bowyer-Watson algorithm can be used directly on $\mathrm{Del}(\Gamma X)/\Gamma$. For this purpose we leverage the CGAL machinery for periodic triangulations from the cubic setting [15, 5] and enhance its underlying data structures to work for the generic setting. As we keep only one representative for each vertex, inserting points in this phase is more efficient than in the first phase.

The remainder of this section describes the two phases and the transition between them, as well as the preprocessing of the lattice.

3.1 Lattice preprocessing

We first compute the reduced lattice basis B, which allows us to compute $\mathcal{V}$, the face normals of the canonical domain. We use $\mathcal{V}$ to represent $\mathrm{dom}^{\square}(0, \Lambda)$ and check for containment of a point within $\mathrm{dom}^{\square}(0, \Lambda)$, which helps obtaining the canonical copy of each input point. To find all periodic copies of a canonical point x that lie within $\mathrm{dom}^{\square}(0, 3\Lambda)$, it is sufficient to find all points λ such that the domain $\mathrm{dom}^{\square}(\lambda, \Lambda)$ intersects $\mathrm{dom}^{\square}(0, 3\Lambda)$, and then check $\phi_\lambda(x)$ for containment within $\mathrm{dom}^{\square}(0, 3\Lambda)$ for these points. As the set of these points λ is independent of x, we compute it before phase 1.

Lattice reduction

Many lattice related problems, such as the shortest non-zero vector problem (SVP), are believed to be hard in general [20, 2]. However in low dimensions, we can use two classical iterative algorithms to solve lattice reduction. Let B' be a lattice basis and ${B'}_{\text{sup}} = \{b'_0, \ldots, b'_d\}$ its superbase, as defined in Section 2. Define $c_{ij} := \langle b'_i, b'_j \rangle$.

2-dimensional reduction. A 2-dimensional lattice basis is Lagrange-reduced [13, Section 4.2] if $0 \leq 2|c_{12}| \leq c_{11} \leq c_{22}$. We can negate b'_2 if necessary so that $c_{12} \leq 0$. Then $\{b'_1, b'_2, b'_0 := -b'_1 - b'_2\}$ form an obtuse superbase, because $c_{01} = \langle b'_1, -b'_1 - b'_2 \rangle = -c_{11} - c_{12} \leq 0$ due to $2|c_{12}| \leq c_{11}$, and similarly $c_{02} \leq 0$. If a basis is not Lagrange-reduced with $2|c_{12}| > c_{11}$, we exchange b'_2 for a shorter vector that forms a basis with b'_1: Let $b''_2 := b'_2 - sb'_1$, with $s = 1$ if $c_{12} > 0$ and $s = -1$ otherwise. Vector b''_2 is shorter than b'_2 since $c'_{22} = c_{22} + c_{11} - 2sc_{12}$. Since there are only finitely many pairs of lattice vectors within a ball of any given radius, a finite number of applications of this procedure will yield a Lagrange-reduced basis.

3-dimensional reduction. We outline Selling's algorithm [13, Section 4.4], an iterative algorithm that obtains an obtuse superbase in 3 dimensions. In each step of the algorithm, if there is a $c_{ij} > 0$, the algorithm returns a new superbase ${B''}_{\text{sup}}$ defined via $b''_i := -b'_i$, $b''_j := b'_j$, $b''_h := b'_h + b'_i$ and $b''_k := b'_k + b'_i$ where h and k are the remaining two indices different from i and j. For any basis B, let $\sigma(B) := \sum_{b \in B_{\text{sup}}} ||b||^2$. Notice that in each step, $\sigma(B'') = \sigma(B') - 2c_{ij}$. In particular, $\sigma(B'') < \sigma(B')$. This fact, together with the fact that there are only finitely many lattice points in a ball of radius $\sqrt{\sigma(B')}$ (and thus only finitely many quadruplets of vectors whose square magnitudes sum up to at most $\sigma(B')$) guarantees termination of the algorithm.

Intersecting domains

For a given canonical point $x \in X_0$, each of its periodic copies $y \in \Gamma x$ can be obtained as $\phi_\lambda(x)$ for some translation $\phi_\lambda \in \Gamma$. The corresponding lattice point λ can be uniquely written as $\sum_{i=1}^{d} o_i b_i$ for some $o \in \mathbb{Z}^d$ and we write and $\lambda(o) := \lambda$. We then call $o =: o(y)$ the *offset* of y and $\phi_{\lambda(o)}$ the *translation* associated with o. For each point $x \in X_0$ we need to determine the 3^d offsets for which $\phi_{\lambda(o)}(x)$ is within $\text{dom}^{\mathbb{Q}}(0, 3\Lambda)$. Fortunately, these offsets have to come from a fixed set of offsets that only depends on the lattice basis. Notice that if $\phi_{\lambda(o)}(\text{dom}^{\mathbb{Q}}(0, \Lambda))$ does not intersect $\text{dom}^{\mathbb{Q}}(0, 3\Lambda)$, then $\phi_{\lambda(o)}(x)$ cannot be in $\text{dom}^{\mathbb{Q}}(0, 3\Lambda)$ for $x \in X_0$. Thus we only need to check those offsets for which $\phi_{\lambda(o)}(\text{dom}^{\mathbb{Q}}(0, \Lambda)) \cap \text{dom}^{\mathbb{Q}}(0, 3\Lambda) \neq \emptyset$.

▶ **Lemma 2.** *There are at most* $4^d - 2^d + 1$ *translates of* $\text{dom}^{\mathbb{Q}}(0, \Lambda)$ *that have non-empty intersection with* $\text{dom}^{\mathbb{Q}}(0, 3\Lambda)$.

Proof. If $\text{dom}^{\mathbb{Q}}(\lambda, \Lambda)$ intersects $\text{dom}^{\mathbb{Q}}(0, 3\Lambda)$ for some $\lambda \in \Lambda$, then λ must be inside $\text{dom}^{\mathbb{Q}}(0, 4\Lambda)$. There are 4^d lattice points within $\text{dom}^{\mathbb{Q}}(0, 4\Lambda)$. For each $v \in \mathcal{V}^+$, the lattice vectors $-2v$ and $2v$ are on the boundary of $\text{dom}^{\mathbb{Q}}(0, 4\Lambda)$, however only $-2v$ is one of those 4^d points within $\text{dom}^{\mathbb{Q}}(0, 4\Lambda)$. For these lattice points $-2v$ on the boundary however the intersection between $\text{dom}^{\mathbb{Q}}(-2v, \Lambda)$ and $\text{dom}^{\mathbb{Q}}(0, 3\Lambda)$ is empty. Thus only $4^d - |\mathcal{V}^+| = 4^d - (2^d - 1)$ translates of $\text{dom}^{\mathbb{Q}}(0, \Lambda)$ intersect $\text{dom}^{\mathbb{Q}}(0, 3\Lambda)$. ◀

In general, we can find this set of offsets via a breadth-first search: We define a graph on Λ with each lattice point λ connected to $\lambda + v$ for $v \in \mathcal{V}$. We start the search at lattice point 0 and terminate once we have found $4^d - 2^d + 1$ offsets, or in the case of non-generic lattices once all 4^d lattice points inside $\text{dom}^{\mathbb{Q}}(0, 4\Lambda)$ have been reached (with the ones on the boundary being discarded).

In 2 dimensions, there is in fact a fixed set S of 13 offsets such that for any lattice, $\mathrm{dom}^{\square}(\lambda(o), \Lambda)$ only intersects $\mathrm{dom}^{\square}(0, 3\Lambda)$ when $o \in S$ (Figure 3b).

▶ **Lemma 3.** *Given a 2-dimensional lattice with basis B and an offset o, if $o \notin \{$(0, 0), (-1, -1), (0, 1), (1, 0), (-1, 0), (0, -1), (1, 1), (-1, -2), (1, 2), (-2, -1), (2, 1), (-1, 1), (1, -1)$\}$, then $\mathrm{dom}^{\square}(\lambda(o), \Lambda) \cap \mathrm{dom}^{\square}(0, 3\Lambda) = \emptyset$.*

Proof. Each $\mathrm{dom}^{\square}(\lambda, \Lambda)$ adjacent to $\mathrm{dom}^{\square}(0, \Lambda)$ has all but one of its facets in the interior of $\mathrm{dom}^{\square}(0, 3\Lambda)$. Except for those adjacent via one of those facets, all $\mathrm{dom}^{\square}(\lambda, \Lambda)$ at graph distance 2 from 0 intersect $\mathrm{dom}^{\square}(0, 3\Lambda)$. The ones that don't are exactly the 6 domains $\mathrm{dom}^{\square}(2v, \Lambda)$ for $v \in \mathcal{V}$. As there are 19 domains at distance at most 2, we have found 13 domains that intersect $\mathrm{dom}^{\square}(0, 3\Lambda)$. As this is the maximum possible number by Lemma 2, we have found all of them. ◀

In 3 dimensions, the same argument yields a set of 51 fixed offsets. However, by Lemma 2, the total number of domains that intersect $\mathrm{dom}^{\square}(0, 3\Lambda)$ is 57. Fortunately, the remaining 6 offsets come from a fixed set of 24 offsets that is independent of the lattice.

▶ **Lemma 4.** *There is a set $O_{\leq 2}$ of 51 offsets and a set O_3 of 24 offsets such that for any 3-dimensional lattice with basis B, there is a subset S of O_3 of size 6 such that if $o \notin O_{\leq 2} \cup S$, then $\mathrm{dom}^{\square}(\lambda(o), \Lambda) \cap \mathrm{dom}^{\square}(0, 3\Lambda) = \emptyset$.*

Proof. We use the observation that $|\langle v, w\rangle| \leq \langle v, v\rangle$ for any two Voronoi relevant vectors $w, v \in \mathcal{V}$ [13, Section 3.5]. Then the following claim holds:

▷ Claim. Domains $\mathrm{dom}^{\square}(\nu, \Lambda)$ of graph-distance at most $(k-2)$ from some $\lambda = kv$ for an integer $k \geq 2$ and $v \in \mathcal{V}$ cannot intersect $\mathrm{dom}^{\square}(0, 3\Lambda)$.

Proof. We have $\nu = kv + v_1 + \cdots + v_{k-2}$ for some $v_i \in \mathcal{V}, i = 1, \ldots, k-2$. Then $\langle w, v\rangle = k\langle v, v\rangle + \langle v, v_1\rangle + \cdots + \langle v, v_{k-2}\rangle \geq k\langle v, v\rangle - (k-2)\langle v, v\rangle = 2\langle v, v\rangle$ due to the observation above. This means that ν is not in the interior of $\mathrm{str}(4v)$ and thus the interior of the 4-scaled domain. Therefore $\mathrm{dom}^{\square}(\nu, \Lambda) \cap \mathrm{dom}^{\square}(0, 3\Lambda) = \emptyset$. ◁

For a generic 3-dimensional lattice, the Voronoi relevant vectors correspond to the non-empty subsets of B_{sup} of size at most 3, independent of the actual vectors of B_{sup}. Thus the graph we defined on Λ is isomorphic for all generic lattices. For non-generic lattices it is isomorphic to a subgraph of the graph for generic lattices; thus it is sufficient to restrict our attention to generic lattices.

We enumerate all lattice points at graph distance 3 from the origin. For a generic lattice, this yields 110 points, however all but 24 of them can be written as $3v$ or $3v + w$ for Voronoi relevant vectors v and w. The set of the 24 corresponding offsets is O_3, while $O_{\leq 2}$ are those 51 offsets that the argument from Lemma 3 yields. All lattice points at graph distance 4 can be written as $4v + kw$ for $k \leq 2$ and Voronoi relevant vectors v and w, and thus their domains and consequently any domains at higher distances cannot intersect $\mathrm{dom}^{\square}(0, 3\Lambda)$. ◀

3.2 Phase 1

Phase 1 maintains the Euclidean Delaunay triangulation of X_3. For each new point to be inserted, we first find its canonical copy. Then we can compute its periodic copies that are contained within $\mathrm{dom}^{\square}(0, 3\Lambda)$ using Lemmas 3 and 4. These are then inserted into $\mathrm{Del}(X_3)$. To provide user access to the cells of $\mathrm{Del}(\Gamma X)/\Gamma$, we define a notion of canonical cell in $\mathrm{Del}(X_3)$ to get a representative for each cell of $\mathrm{Del}(\Gamma X)/\Gamma$.

Canonical points

For each point of X, finding its periodic copy that lies in $\text{dom}^{\square}(0, \Lambda)$ is equivalent to solving the closest vector problem (CVP), i.e. given $x \in X$, determining the lattice point that is closest to x. For arbitrary dimensions this problem is known to be NP-hard [20]. For the exact version of CVP, various iterative algorithms have been described [1, 19, 16]. As we are only operating in 2 and 3 dimensions, any of them would suffice for us in practice in terms of running time, and we will describe the algorithm by Sommer et al [19] due to its simplicity.

For a real number r, define $\text{round}(r)$ to be the closest integer to r. If this integer is not unique, then it is the one with the smallest absolute value. This definition ensures convergence in cases where x is on the boundary of a Dirichlet domain of the lattice [19]. Recall that $\mathcal{V}^+$ is a set of normals of the facets of the canonical domain, and can be obtained from a reduced basis B (see Section 2). We first sort the vectors of $\mathcal{V}^+$ by their magnitude. As $\text{dom}^{\square}(0, \Lambda)$ is the intersection of the strips $\text{str}(v)$ for $v \in \mathcal{V}^+$, we need to find the periodic copy of x that is in all these $\text{str}(v)$. We loop through the vectors v of $\mathcal{V}^+$ and for each of them perform the following operations: Compute $c := \frac{\langle x,v \rangle}{\langle v,v \rangle}$. If $-0.5 \leq c < 0.5$, then already $x \in \text{str}(v)$. If not, then we subtract $\text{round}(c) \cdot v$ from x. Note that after such a step, it is guaranteed that $x \in \text{str}(v)$. However after modifying x for a longer vector v, it might happen that x is moved outside of $\text{str}(v')$ for some shorter vector v' again. Therefore we need to repeatedly loop through $\mathcal{V}^+$ and perform this operation until $x \in \text{str}(v)$ for all $v \in \mathcal{V}^+$.

Notice that in each step where x is modified, the magnitude of x strictly decreases. Therefore this algorithm terminates in finite time, as the number of lattice vectors within a ball of given radius $||x||$ is finite.

Extracting representative cells

Recall that our interface specifies access to the cells (and lower-dimensional simplices) of $\text{Del}(\Gamma X)/\Gamma$. Thus for each of its cells we have to provide one representative from $\text{Del}(X_3)$. For a cell σ of either $\text{Del}(\Gamma X)$ or $\text{Del}(X_3)$, with $V(\sigma) = \{x_1, \ldots, x_d\}$, we define its offset as the vector $o(\sigma) = \min_{x \in V(\sigma)}\{o(x)\}$ where the minimum is taken lexicographically. Note that this definition differs from [4, Convention 3.3.1] where the coordinate-wise minimum is taken. If the offset of a cell is the 0-vector, we call it a *canonical cell*. Note that due to our definition a canonical cell always has a vertex with offset 0. Therefore by Proposition 1 a canonical cell of $\text{Del}(X_3)$ is also periodic , see Figure 3b. For $\sigma \in \text{Del}(\Gamma X)$, the translated cell $\sigma - \sum_{i=1}^{d} o_i b_i$ is called its *canonical representative*. This means that for each class of cells in $\text{Del}(\Gamma X)$ (or equivalently each cell of $\text{Del}(\Gamma X)/\Gamma$), there is a unique canonical representative in $\text{Del}(X_3)$. Therefore we can get a set of representative cells by iterating over the cells of $\text{Del}(X_3)$ and selecting those that are canonical.

Neighborhood relations

In accordance with our interface, we need to provide neighborhood relations for the vertices and cells of $\text{Del}(\Gamma X)/\Gamma$. These vertices and cells are represented by the canonical vertices and cells of $\text{Del}(X_3)$, whose neighbors in $\text{Del}(X_3)$ (which we have access to) may differ from the neighbors in $\text{Del}(\Gamma X)/\Gamma$ (which we want to find). We outline how to get the neighors of a cell of $\text{Del}(\Gamma X)/\Gamma$ from $\text{Del}(X_3)$, and note that other neighborhood relations work in a conceptually similar way. Note that we do not store these relations explicity, but we compute them upon request and may cache them for future access.

Consider a canonical cell σ of $\text{Del}(X_3)$ and a neighboring cell σ_n. If σ_n is canonical, it is the neighbor of σ in $\text{Del}(\Gamma X)$. If σ_n is not canonical but has a vertex in $\text{dom}^{\square}(0, \Lambda)$, then it is periodic and we return the canonical representative of this cell. However, it is possible that

all vertices of σ_n are outside $\mathrm{dom}^{\lozenge}(0,\Lambda)$, and thus σ_n might not be a periodic cell at all. In that case, we need to consider the facet τ separating σ and its neighbor σ_n. As τ is a facet of a canonical cell, it is also a periodic facet (i.e. a facet of $\mathrm{Del}(\Gamma X)$). We compute τ's offset $o := o(\tau)$. Then $\tau' := \phi_{\lambda(-o)}(\tau)$ is the canonical representative of τ and $\sigma' := \phi_{\lambda(-o)}(\sigma)$ as well as its neighbor σ'_n across τ' are periodic cells because they share τ', which has a vertex in $\mathrm{dom}^{\lozenge}(0,\Lambda)$. If σ'_n is canonical, then is it the canonical representative of the neighbor of σ in $\mathrm{Del}(\Gamma X)$; if not, then its canonical representative is. Figure 3c illustrates this process.

In practice, we store the canonical representative of each vertex of X_3. To obtain the canonical representative of a cell (or facet) σ, we need to choose a vertex x whose offset is minimal (lexicographically) among its vertices. This ensures that the periodic copy x' of x in the canonical version of the cell is inside $\mathrm{dom}^{\lozenge}(0,\Lambda)$. Then one of the cells (or facets) incident to x' in $\mathrm{Del}(X_3)$ is the canonical representative of σ.

3.3 Transition

Phase 2 is more efficient than phase 1 as it directly operates on $\mathrm{Del}(\Gamma X)/\Gamma$; however, we cannot use it from the start as the Bowyer-Watson algorithm [3, 21] comes with some constraints. The Bowyer-Watson algorithm is an incremental algorithm inserting points one by one. For each new point x, it determines the *conflict zone*, which is the set of cells whose circumsphere contains x. All these cells are removed, and all boundary facets of the resulting hole are connected to x to fill in the hole with new cells. The algorithm requires this hole to be a topological d-ball, which is not always guaranteed for $\mathrm{Del}(\Gamma X)/\Gamma$. However the following criterion is a sufficient condition for the Bowyer-Watson algorithm to work [7].

▶ **Lemma 5** ([7, Criterion 3.11]). *If for every cell in* $\mathrm{Del}(\Gamma X)/\Gamma$ *the circumradius is smaller than* $\frac{1}{4}\mathrm{sv}(\Lambda)$*, then* $\mathrm{Del}(\Gamma X')/\Gamma$ *is simplicial for every* $X' \supseteq X$*.*

If this criterion is fulfilled, we can safely switch to phase 2. To detect at which point in phase 1 the criterion is fulfilled, we maintain a set S_{big} of *big* cells, which are canonical cells whose circumradius is larger than or equal to $\frac{1}{4}\mathrm{sv}(\Lambda)$. We update S_{big} during each point insertion.

Assume we wish to insert a new point x into the periodic triangulation of some point set X. On a high level, we need to remove the big cells in the conflict zone of x from S_{big}, and then add the newly created big cells to S_{big}. In practice, we have already computed the triangulation $\mathrm{Del}(X_3)$ and have to insert the periodic copies of x that are within $\mathrm{dom}^{\lozenge}(0,3\Lambda)$, i.e. $\Gamma x \cap \mathrm{dom}^{\lozenge}(0,3\Lambda)$, into $\mathrm{Del}(X_3)$. We first detect the conflict zone of x_0, the canonical copy of x. For each big cell in the conflict zone we check if it has a canonical copy in S_{big}, and remove that copy from S_{big} if it does. While the conflict zone may contain non-periodic cells, Lemma 6 guarantees that we capture a representative of each cell from $\mathrm{Del}(\Gamma X)$ that is in conflict with x. Next we insert all the copies $\Gamma x \cap \mathrm{dom}^{\lozenge}(0,3\Lambda)$ into $\mathrm{Del}(X_3)$. Finally, for each big cell incident to x_0 in the resulting triangulation, we add its canonical copy to S_{big}. Note that all these cells have a canonical copy by Proposition 1 as they have a vertex inside $\mathrm{dom}^{\lozenge}(0,\Lambda)$.

▶ **Lemma 6.** *Let* C *be the conflict zone of some point* $x \in \mathrm{dom}^{\lozenge}(0,\Lambda)$ *with respect to* $\mathrm{Del}(\Gamma X)$*. Then the conflict zone of* x *with respect to* $\mathrm{Del}(X_3)$ *contains at least one periodic copy of each cell from* C*.*

Proof. First observe that if a cell of $\mathrm{Del}(\Gamma X)$ has its circumcenter at the origin, then all its vertices must be within $\mathrm{dom}(0,\Lambda)$, because if the circumsphere contains a point outside $\mathrm{dom}(0,\Lambda)$, then it also contains its canonical copy. Via translation it follows that if a cell has

its circumcenter in $\mathrm{dom}^{\mathrm{D}}(0, 2\Lambda)$, then its vertices are in $\mathrm{dom}^{\mathrm{D}}(0, 3\Lambda)$ and thus it is a cell of $\mathrm{Del}(X_3)$. So assume we have a cell σ in C whose circumcenter c is not within $\mathrm{dom}^{\mathrm{D}}(0, 2\Lambda)$. Then there is a facet of $\mathrm{dom}^{\mathrm{D}}(0, 2\Lambda)$ with respect to which c is outside. Let f be its face normal. Then $x + f$ is also contained in the circumsphere of σ. Reversely, $\sigma - f$ is a cell whose circumsphere contains x, and furthermore its circumcenter is closer to 0 than c, see Figure 4. As there are only finitely many periodic copies of c within a given distance from 0, after

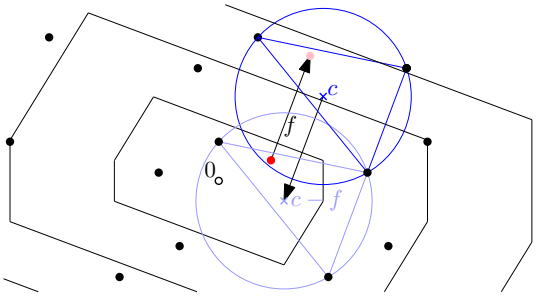


Figure 4 The red point x is in the conflict zone of the blue cell σ, which is not a cell of $\mathrm{Del}(X_3)$ and whose circumcenter c is outside $\mathrm{dom}^{\mathrm{D}}(0, 2\Lambda)$. However the light blue cell $\sigma - f$ is a periodic copy of σ that is in $\mathrm{Del}(X_3)$ and x is in its conflict zone.

applying this process a finite number of times we eventually obtain a cell whose circumsphere contains x and whose circumcenter is in $\mathrm{dom}^{\mathrm{D}}(0, 2\Lambda)$. This cell is a periodic copy of σ, is contained in $\mathrm{Del}(X_3)$ and thus also part of the conflict zone of x with respect to $\mathrm{Del}(X_3)$. ◀

Once S_{big} is empty, the criterion of Lemma 5 is fulfilled, and we can internally convert our triangulation from $\mathrm{Del}(X_3)$ to $\mathrm{Del}(\Gamma X)/\Gamma$, which is maintained in phase 2. We initialize the periodic triangulation data structure with the set of canonical vertices X_0 and canonical cells obtained from $\mathrm{Del}(X_3)$, as well as the adjacency and incidence relations outlined earlier.

3.4 Phase 2

Phase 2 operates directly on the torus triangulation $\mathrm{Del}(\Gamma X)/\Gamma$. Thus it maintains only one copy of each cell and vertex and point insertion is faster than in phase 1. The data structure it uses to represent $\mathrm{Del}(\Gamma X)/\Gamma$ is akin to the one used in CGAL for the cubic case, and closely resembles the interface we defined for our algorithm: Only X_0 is stored as vertex set, and each cell is represented by its vertices, with a vertex encoded as a pair (x, o) of a point $x \in X_0$ and an offset so that its geometric location is $\phi_{\lambda(o)}(x)$. While in the cubic case each offset coordinate either takes the value 0 or 1 and offsets can be encoded in d bits, in our more general setting offsets can take any of the lattice-specific values from Lemma 2. Unlike in phase 1, most neighborhood relations for $\mathrm{Del}(\Gamma X)/\Gamma$ are already stored explicitly in the data structure: For each cell its adjacent cells are stored, and for each vertex one incident cell is stored. The remaining neighborhood relations required by our interface are obtained implicitly from the explicitly stored ones. With each point insertion, all stored neighborhood relations have to be updated accordingly.

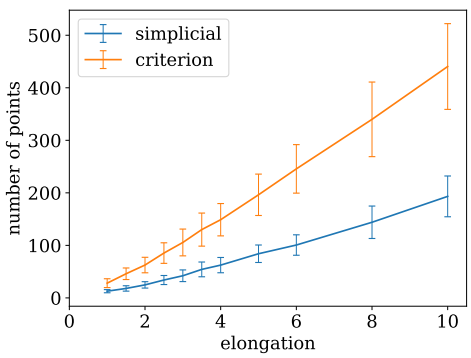
Point insertion. To insert a new point x into $\mathrm{Del}(\Gamma X)/\Gamma$ using the Bowyer-Watson algorithm, first we compute the canonical copy x_0. Then we locate the cell containing x_0, via a traversal starting from an arbitrary cell. The conflict zone is computed via a search starting in the cell containing the point x_0. Whenever we are traversing cells, care has to be taken to maintain the correct offset of the affected cells relative to x_0, and similarly when creating

new cells to fill in the hole left by the deleted conflict zone. For the cubic case the details are described in [4, Section 3.3], and we omit the technical adjustments needed to make these steps work in the more general case.

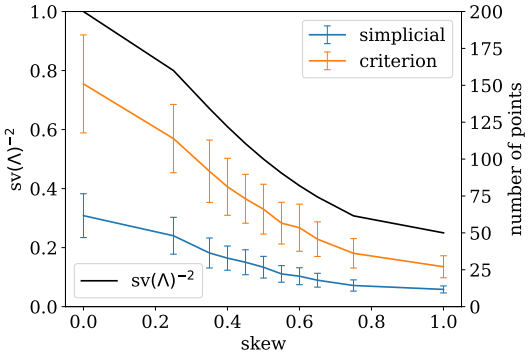
4 Experimental results

Points until transition. We experimentally evaluated the number of points required until the criterion of Lemma 5 is fulfilled and the transition to phase 2 occurs. Lemma 5 requires all of $\mathbb{R}^d/\Gamma$ to be covered by the balls of radius $\frac{1}{4}\mathrm{sv}(\Lambda)$ around X_0. As the volume of the torus equals the volume of the Dirichlet domain of 0, denoted as $\mathrm{vol}(\mathrm{dom}(0,\Lambda))$, we expect the number of points until switching to phase 2 to be roughly proportional to $\mathrm{vol}(\mathrm{dom}(0,\Lambda))/\mathrm{sv}(\Lambda)^d$, assuming the points are sampled uniformly at random.

To investigate this in 2 dimensions, we parametrize a 2-dimensional space of lattices. We call the parameters the elongation ℓ and the skew s. The basis of the lattice with elongation ℓ and skew s is $b_1 = (\ell, 0)$ and $b_2 = (s \cdot \ell/2, 1)$. With $\ell \geq 1$ and s between 0 and 1 we can parametrize all 2-dimensional lattices up to symmetry and scaling. Note that the skew affects $\mathrm{sv}(\Lambda)$ but not $\mathrm{vol}(\mathrm{dom}(0,\Lambda))$, while the elongation is proportional to $\mathrm{vol}(\mathrm{dom}(0,\Lambda))$ but does not affect the $\mathrm{sv}(\Lambda)$. Fixing the skew at 0 and varying the elongation (Figure 5a), we see that the number of random points needed until the phase switch appears to be proportional to the elongation. The same applies to the number of points until the resulting triangulation is simplicial for the first time. Figure 5b shows the same statistics for lattices of fixed area but varying skew. In addition we plot the inverse of $\mathrm{sv}(\Lambda)^2$ for comparison, and observe that it behaves similarly albeit not entirely proportionally.



(a) Results for fixed skew $s = 0$ and varying elongation ℓ.



(b) Results for $\ell = 4$ and varying s. The inverse of the square length of the shortest non-zero lattice vector in black, with its y-axis on the right.

Figure 5 For different lattices, the number of points inserted into a periodic triangulation until (blue) $\mathrm{Del}(\Gamma X)/\Gamma$ is simplicial for the first time, and (orange) $\mathrm{Del}(\Gamma X)/\Gamma$ fulfills Lemma 5. Each data point is the mean of 200 trials, and the bars represent the standard deviation.

In 3-dimensions, a parametrization of lattices up to symmetry and scaling needs 5 parameters, and thus an analysis like in 2D is impractical. As mentioned before, we expect that the number of points until the switch to phase 2 is proportional to $\mathrm{vol}(\mathrm{dom}(0,\Lambda))/\mathrm{sv}(\Lambda)^3$. By comparing the two, Figure 6 confirms this relationship for 182 lattices whose basis vectors have randomly sampled direction and a random magnitude between 1 and 100.

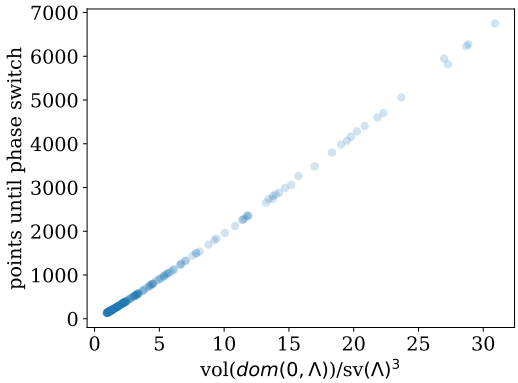


Figure 6 Phase switching in 3D: Each point in the plot represents a lattice, with $\mathrm{vol}(\mathrm{dom}(0,\Lambda))/\mathrm{sv}(\Lambda)^3$ on the x-axis and the number of points inserted until the switch to phase 2 occurred on the y-axis. Each y-value is obtained as the average over 200 point sets, each distributed uniformly at random.

Running times. We evaluate the running times of our algorithm for different 3-dimensional lattices, and compare them to the existing CGAL implementations. The data points collected are from Delaunay triangulations of 10^k uniformly sampled random points for k up to 7. Each data point is an average of 300 trials (10 trials for 10^7 points). The experiments were conducted on a laptop running Fedora 30 64-bits, with two 6-core Intel(R) i9-8950HK CPU clocked at 2.90GHz, and with 32GB of RAM. The CGAL kernel used was CGAL::Exact_predicates_inexact_constructions_kernel and CPU time was measured using CGAL's timer tools. The code was compiled using clang 8.0.0 with compilation flags -O3 and -DNDEBUG.

Table 1 shows a comparison between the CGAL implementation of Euclidean Delaunay triangulations [14] (with random points uniformly sampled in the unit cube), periodic Delaunay triangulations in the cubic setting [5] and our algorithm for various lattices, including the cubic lattice. For each lattice, we also measured the average number of points until the switch to phase 2. As our algorithm and the one for the cubic setting are based on the same code base when operating directly on the torus triangulation, their runtimes are comparable for large point sets. It should be noted that when similar experiments were conducted in 2010 to compare the Euclidean and cubic periodic algorithms [4, Section 3.6.2], both were performing comparably. Since then, Euclidean Delaunay triangulations in CGAL have seen significant optimizations, which were not applied to periodic triangulations. This also explains why our algorithm is faster than the cubic periodic algorithm for point sets where phase 1 takes up a significant portion of the running time, as internally we use a Euclidean rather than a periodic triangulation.

5 Discussion

Weighted points. Phase 1 of our algorithm readily generalizes to weighted point sets. In particular, Proposition 1 still holds for weighted point sets (see Appendix A for a proof). Phase 2 only works for weighted points under additional restrictions because $\mathrm{Del}(\Gamma X)/\Gamma$ can not be guaranteed to remain simplicial, in particular after inserting points with large weights. Such point insertions that break simpliciality can be prevented by requiring points to be inserted in decreasing order of weight, or like in the cubic implementation in CGAL by severely restricting the range of weights a point can have. Thus an implementation is subject to a tradeoff between flexibility (phase 1) and performance (phase 2).

Table 1 Running time (in seconds) of various Delaunay triangulation algorithms on random point sets of different sizes. Our algorithm ("Lattice") is evaluated for different lattices including the cubic lattice, face-centered cubic (FCC) lattice and two other lattices Λ_1 and Λ_2 with bases $B_1 = \{(0.5, -0.5, 0.1), (-0.5, 0.5, 0.1), (0.5, 0.5, -0.1)\}$ and $B_2 = \{(1, 0, 0), (-0.5, \sqrt{3}/2, 0), (0, 0, 0.05)\}$. For each lattice we also record the average number of points until the switch to phase 2 (n_{switch}).

Algorithm	Euclidean [14]	Cubic [5]	Lattice			
Lattice	–	Cubic	Cubic	FCC	Λ_1	Λ_2
$\frac{\text{vol}(\text{dom}(0,\Lambda))}{\text{sv}(\Lambda)^3}$	–	1.00	1.00	0.71	12.50	346.41
n_{switch}	–	145 [4]	141	94	2519	89950
10^0	0.0000	0.0001	0.0002	0.0001	0.0003	0.0004
10^1	0.0000	0.0160	0.0033	0.0026	0.0044	0.0035
10^2	0.0004	0.1848	0.0461	0.0287	0.0460	0.0380
10^3	0.0049	0.5957	0.0858	0.0446	0.9812	0.6372
10^4	0.0487	0.9591	0.3832	0.1642	4.6602	16.5759
10^5	0.5679	4.8119	4.5153	2.7868	10.8139	362.7956
10^6	6.5974	93.9327	95.1447	51.5945	58.1568	517.9715
10^7	59.5152	2314.3618	2317.7867	1215.2648	1799.4515	2983.0943

Higher dimensions. Conceptually, our algorithm generalizes to higher dimensions. The only step that is dimension-dependent and therefore requires significant adjustments is obtaining a representation of $\text{dom}^{\complement}(0, \Lambda)$. While up to 3 dimensions every lattice has an obtuse superbase, this does not hold in higher dimensions. Therefore we would need to find a different way of obtaining the set of Voronoi relevant vectors. Complexity-wise, computing the canonical representative of a point is believed to be hard. Furthermore, the number of copies that we compute of each input point in phase 1 is 3^d. Therefore we expect the runtime of our algorithm to be exponential in the dimension, but still feasible in practice as long as the dimension is not too large.

Dummy points. For best performance, transition from phase 1 to phase 2 should occur as soon as possible. Intuitively, this happens when the input point distribution does not have large gaps. We can achieve this by first inserting additional *dummy points*, which are removed at the end (if possible) [7]. In practice, we can choose these points from a sufficiently fine hexagonal lattice (or body-centered cubic in 3 dimensions), such that the open spheres of radius $\frac{1}{4}\text{sv}(\Lambda)$ around the points cover the entire torus.

Software distribution. We aim to provide a CGAL package for periodic Delaunay triangulations for arbitrary lattices in 2 and 3 dimensions. The current state of our implementation is available online (see Footnote 1). Both the 2- and 3-dimensional implementations have been integrated into the CGAL codebase and only require some additional refactoring and optimizing to adhere to CGAL's quality standards. Automated tests and documentation still have to be produced. An extension to weighted Delaunay triangulations, referred to as regular triangulations in CGAL, is planned for the future.

References

1 Erik Agrell, Thomas Eriksson, Alexander Vardy, and Kenneth Zeger. Closest point search in lattices. *IEEE Trans. Inform. Theory*, 48(8):2201–2214, 2002. doi:10.1109/TIT.2002.800499.

2 Miklós Ajtai. The shortest vector problem in L2 is NP-hard for randomized reductions. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 10–19, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/276698.276705.

3 Adrian Bowyer. Computing Dirichlet tessellations. *The computer journal*, 24(2):162–166, 1981.

4 Manuel Caroli. *Triangulating Point Sets in Orbit Spaces*. PhD thesis, Université Nice Sophia Antipolis, 2010. URL: https://tel.archives-ouvertes.fr/tel-00552215.

5 Manuel Caroli, Aymeric Pellé, Mael Rouxel-Labbé, and Monique Teillaud. 3D periodic triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020. URL: http://doc.cgal.org/latest/Manual/packages.html#PkgPeriodic3Triangulation3.

6 Manuel Caroli and Monique Teillaud. 3D periodic triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.5 edition, 2009.

7 Manuel Caroli and Monique Teillaud. Delaunay triangulations of closed Euclidean d-orbifolds. *Discrete & Computational Geometry*, 55(4):827–853, 2016. URL: https://hal.inria.fr/hal-01294409, doi:10.1007/s00454-016-9782-6.

8 J. H. Conway and N. J. A. Sloane. Low-dimensional lattices VI: Voronoi reduction of three-dimensional lattices. *Proc. R. Soc. Lond. A*, pages 55–68, 1992.

9 B. Delaunay. Sur la sphère vide. À la mémoire de Georges Voronoï. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800, 1934. URL: http://mi.mathnet.ru/eng/izv4937.

10 Vincent Despré, Jean-Marc Schlenker, and Monique Teillaud. Flipping geometric triangulations on hyperbolic surfaces. In *Proceedings of the Thirty-sixth International Symposium on Computational Geometry*, 2020. To appear. Preliminary version: https://hal.inria.fr/hal-02400219.

11 Olivier Devillers and Monique Teillaud. Perturbations for Delaunay and weighted Delaunay 3D triangulations. *Computational Geometry: Theory and Applications*, 44:160–168, 2011. URL: http://hal.inria.fr/inria-00560388/, doi:10.1016/j.comgeo.2010.09.010.

12 Nikolai Dolbilin and Daniel Huson. Periodic Delone tilings. *Periodica Mathematica Hungarica*, 34:1-2:57–64, 1997.

13 Peter Engel. *Geometric crystallography: an axiomatic introduction to crystallography*. Springer, Dordrecht, 1986. doi:10.1007/978-94-009-4760-3.

14 Clément Jamin, Sylvain Pion, and Monique Teillaud. 3D triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020. URL: https://doc.cgal.org/latest/Manual/packages.html#PkgTriangulation3.

15 Nico Kruithof. 2D periodic triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.4 edition, 2014. URL: http://doc.cgal.org/latest/Manual/packages.html#PkgPeriodic2Triangulation2Summary.

16 Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. *SIAM J. Comput.*, 42(3):1364–1391, 2013. doi:10.1137/100811970.

17 The CGAL Project. URL: http://www.cgal.org.

18 Chris Rycroft. Voro++: A three-dimensional Voronoi cell library in c++. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2009. URL: http://math.lbl.gov/voro++/.

19 Naftali Sommer, Meir Feder, and Ofir Shalvi. Finding the closest lattice point by iterative slicing. *SIAM J. Discrete Math.*, 23(2):715–731, 2009. doi:10.1137/060676362.

20 Peter van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical report, Mathematische Instituut, Uni. Amsterdam Report, April 1981.

21 David F Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The computer journal*, 24(2):167–172, 1981.

22 Thomas F Willems, Chris H Rycroft, Michaeel Kazi, Juan C Meza, and Maciej Haranczyk. Algorithms and tools for high-throughput geometry-based analysis of crystalline porous materials. *Microporous and Mesoporous Materials*, 149(1):134–141, 2012. URL: http://zeoplusplus.org/.

23 Mariette Yvinec. 2D triangulation. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020. URL: https://doc.cgal.org/latest/Manual/packages.html#PkgTriangulation2.

A Generalization to weighted points

For a weighted point x, we denote its weight with w_x. For weighted points $x, y \in X$, their power distance is $\text{dist}(x,y) := \|x-y\| - w_x - w_y$. For an arbitrary point $p \in \mathbb{R}$ that is not part of X we define the power distance from x to p as $\text{dist}(x,p) := \|x-p\| - w_x$ (as if the weight of p was 0).

We call two points x and y orthogonal if $\text{dist}(x,y) = 0$. For $d+1$ points $Q \subseteq X$ (if in general position) there is a unique point z with weight w_z that is orthogonal to all points of Q. This point is called the orthocenter or power sphere of Q. If Q is a set of points forming a cell (simplex) in the Delaunay triangulation, then for all points $x \in Q$ and $y \in X \setminus Q$ it holds that $\text{dist}(y,z) > \text{dist}(x,z) = 0$. We call this the empty-sphere property. The converse holds as well. The perpendicular bisector of two weighted points x and y is the set of points p with $\text{dist}(x,p) = \text{dist}(y,p)$.

▶ **Proposition 7** (Generalization of Lemma 3.2–3.4 from [12]). *Given a set X of representatives for our point set, let $X_3 := \text{dom}^{\mathbb{C}}(0, 3\Lambda) \cap \Gamma X$, i.e. all periodic copies of these points that lie within the Dirichlet domain of 0 scaled by a factor of 3. Let T_0 be those cells of $\text{Del}(X_3)$ that have at least one vertex in $\text{dom}^{\mathbb{C}}(0,\Lambda)$. Then the cells of T_0 are all part of the triangulation of ΓX. Furthermore, these cells contain at least one representative of each class of cells from $\text{Del}(\Gamma X)$.*

Proof. We will prove the proposition in 3 steps.

▷ Claim 1. Assume one of our points is $x_0 = 0$ (with arbitrary weight). Then the orthocenter c_T of any Delaunay cell that has x_0 as a vertex is within $\text{dom}(0,\Lambda)$.

Proof. Assume not. Then there is a face F_i of $\text{dom}(0,\Lambda)$ such that F_i separates x_0 from c_T. Let f_i be the corresponding translation lattice vector orthogonal to F_i. Then $x_0 + f_i$ is strictly closer to c_T than x_0 because it has the same weight as x_0. This is a contradiction to the empty-sphere property. ◁

▷ Claim 2. For a cell containing x_0 as a vertex, all vertices are in $\text{dom}(0, 2\Lambda)$.

Proof. Assume some vertex x is not. Consider the perpendicular bisector between x and $x - f_i$ where f_i is the face normal vector orthogonal to the face F_i of $\text{dom}(0,\Lambda)$ with respect to which x is outside of $\text{dom}(0, 2\Lambda)$. This bisector is separated from x_0 and the orthocenter c_T by F_i, the face of $\text{dom}(0,\Lambda)$ that is parallel to this bisector. In particular, it follows from Claim 1 that c_T is (strictly) on the $x - f_i$ side of the perpendicular bisector. So $x - f_i$ is closer to c_T than x and thus the empty-sphere property is violated. See Figure 7 for reference. ◁

▷ Claim 3. All cells σ having a vertex in $\text{dom}^{\mathbb{C}}(0,\Lambda)$ are entirely contained in $\text{dom}^{\mathbb{C}}(0, 3\Lambda)$.

Proof. Let x be a vertex of σ that is within $\text{dom}^{\mathbb{C}}(0,\Lambda)$. Shift the entire point set ΓX and its triangulation by the vector $-x$ so that x now coincides with 0. Now from Claim 2 it follows that the other vertices of the shifted cell are within $\text{dom}(0, 2\Lambda)$. Adding the vector $+x$ to these shifted vertices we get back to the original setting, but because $x \in \text{dom}^{\mathbb{C}}(0,\Lambda)$ we also know now that these vertices are within $\text{dom}^{\mathbb{C}}(0, 3\Lambda)$, as $\text{dom}^{\mathbb{C}}(0, 3\Lambda)$ is the Minkowski sum of $\text{dom}^{\mathbb{C}}(0,\Lambda)$ and $\text{dom}(0, 2\Lambda)$. ◁

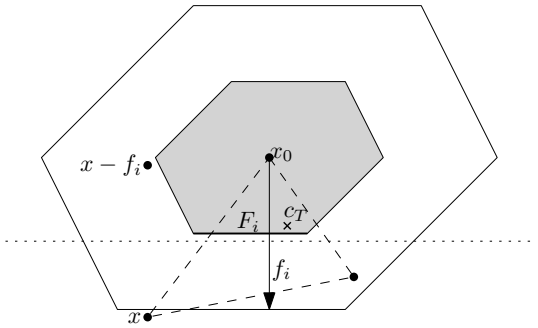


Figure 7 A cell (dashed) with a vertex x that is outside of $\mathrm{dom}(0, 2\Lambda)$. The dotted line is the perpendicular bisector between x and $x - f_i$.

Every cell of $\mathrm{Del}(\Gamma X)$ that has a vertex in $\mathrm{dom}^{\mathbb{Q}}(0, \Lambda)$ has all its vertices in X_3. Furthermore, because it fulfils the empty-sphere property in ΓX, then it also fulfils this property in $X_3 \subset \Gamma X$, and thus is present in $\mathrm{Del}(X_3)$. As every point from X has a representative in $\mathrm{dom}^{\mathbb{Q}}(0, \Lambda)$, also each cell from $\mathrm{Del}(\Gamma X)$ has a representative in T_0, proving the statement. ◀

B Detailed proof of Lemma 4

▶ **Lemma 4** (extended version). *Let* $O_3 := \{$`(3, 2, 1), (2, 1, -1), (3, 1, 2), (2, -1, 1), (1, -1, -2), (1, -2, -1), (2, 3, 1), (1, 2, -1), (1, 3, 2), (-1, 2, 1), (-1, 1, -2), (-2, 1, -1), (2, 1, 3), (1, -1, 2), (1, 2, 3), (-1, 1, 2), (-1, -2, 1), (-2, -1, 1), (-1, -2, -3), (-1, -3, -2), (-2, -1, -3), (-3, -1, -2), (-2, -3, -1), (-3, -2, -1)`$\}$ *and* $O_{\leq 2} := \{$`(0, 0, 0), (-1, -1, -1), (0, 0, 1), (0, 1, 0), (1, 0, 0), (1, 1, 0), (1, 0, 1), (0, -1, -1), (0, 1, 1), (-1, 0, -1), (-1, -1, 0), (1, 1, 1), (0, 0, -1), (0, -1, 0), (-1, 0, 0), (1, 2, 0), (1, 0, 2), (-1, -2, -2), (2, 1, 0), (2, 0, 1), (1, -1, -1), (0, 1, 2), (-2, -1, -2), (0, 2, 1), (-1, 1, -1), (-2, -2, -1), (-1, -1, 1), (1, 1, 2), (-1, -1, -2), (1, 2, 1), (0, 1, -1), (-1, -2, -1), (0, -1, 1), (2, 1, 1), (1, 0, -1), (1, -1, 0), (-2, -1, -1), (-1, 0, 1), (-1, 1, 0), (1, 2, 2), (-1, 0, -2), (-1, -2, 0), (2, 1, 2), (0, -1, -2), (2, 2, 1), (1, 1, -1), (0, -2, -1), (1, -1, 1), (-2, -1, 0), (-2, 0, -1), (-1, 1, 1)`$\}$.

Given a 3-dimensional lattice and an offset o, there is a subset S of size 6 of O_3 such that if $o \notin O_{\leq 2} \cup S$, then $\mathrm{dom}^{\mathbb{Q}}(\lambda(o), \Lambda) \cap \mathrm{dom}^{\mathbb{Q}}(0, 3\Lambda) = \emptyset$.

Proof. As mentioned in the proof of Theorem 3.5 from [13], $\frac{v}{2}$ is on the boundary of $\mathrm{dom}^{\mathbb{Q}}(0, \Lambda)$ for every Voronoi-relevant vector v.

▷ Claim 1. The orthogonal projection of v into the 1-dimensional subspace spanned by another Voronoi relevant vector $w \in \mathcal{V}^+$ has magnitude less than w.

Proof. All facets of $\mathrm{dom}^{\mathbb{Q}}(0, \Lambda)$ are contained in $\mathrm{str}(w)$ for all $w \in \mathcal{V}^+$, so because $\frac{v}{2}$ is on the boundary of $\mathrm{dom}^{\mathbb{Q}}(0, \Lambda)$, it is contained in $\mathrm{str}(w)$. By definition of $\mathrm{str}(w)$ this implies $|\langle \frac{1}{2}v, w\rangle / \langle w, w\rangle| \leq \frac{1}{2}$, from which it follows that $|\langle v, w\rangle| \leq \langle w, w\rangle$. ◁

▷ Claim 2. Domains $\mathrm{dom}^{\mathbb{Q}}(\nu, \Lambda)$ of graph-distance $(k-2)$ from some $\lambda = kv$ for an integer k and $v \in \mathcal{V}$ cannot intersect $\mathrm{dom}^{\mathbb{Q}}(0, 3\Lambda)$.

Proof. We have $\nu = kv + v_1 + \cdots + v_{k-2}$ for some $v_i \in \mathcal{V}$. Then $\langle w, v\rangle = k\langle v, v\rangle + \langle v, v_1\rangle + \cdots + \langle v, v_{k-2}\rangle \geq k\langle v, v\rangle - (k-2)\langle v, v\rangle = 2\langle v, v\rangle$ due to Claim 1. This means that ν is not in the interior of $\mathrm{str}(4v)$ and thus the interior of the 4-scaled domain. Therefore $\mathrm{dom}^{\mathbb{Q}}(\nu, \Lambda) \cap \mathrm{dom}^{\mathbb{Q}}(0, 3\Lambda) = \emptyset$. ◁

Let $B_{\text{sup}} = \{b_0, b_1, b_2, b_3\}$. Note that because $b_0 = -(b_1 + b_2 + b_3)$, every lattice point λ can be written as a non-negative integer combination of three of these extended basis vectors, i.e. $\lambda = c_1 a + c_2 b + c_3 c$ with $c_i \in \mathbb{Z}, c_1 \geq c_2 \geq c_3 \geq 0$ and $a, b, c \in B_{\text{sup}}$. This representation is unique up to permutation of basis vectors with the same coefficient.

Enumerating all $\text{dom}^{\text{◹}}(\lambda, \Lambda)$ at graph distance 3 from $\text{dom}^{\text{◹}}(0, \Lambda)$, we get lattice points λ with non-negative integer combinations of the following types:

$$
\begin{aligned}
&3a \\
&3a + \ \ b \ \ &&= 3a \ \ &&+ b \\
&3a + \ \ b + \ \ c \ &&= 3a \ \ &&+ (b + c) \\
&3a + 2b \ \ &&= 3(a + b) \ \ &&- b \\
&3a + 2b + \ \ c \\
&3a + 2b + 2c \ &&= 3(a + b + c) \ &&- (b + c) \\
&3a + 3b \\
&3a + 3b + \ \ c \ &&= 3(a + b) \ \ &&+ c \\
&3a + 3b + 2c \ &&= 3(a + b + c) \ &&- c \\
&3a + 3b + 3c
\end{aligned}
$$

All of these, except for type $3a + 2b + c$, can be written as $3v$ or $3v + w$ for some Voronoi relevant vectors $v, w \in \mathcal{V}$. This means that they are, or are adjacent to, a domain centered at $3v$, and from Claim 2 it follows that they cannot intersect $\text{dom}^{\text{◹}}(0, 3\Lambda)$. A similar argument shows that none of the domains at graph distance 4 from $\text{dom}^{\text{◹}}(0, \Lambda)$ can intersect $\text{dom}^{\text{◹}}(0, 3\Lambda)$, and therefore also none at higher graph distance.

Now the 51 offsets from $O_{\leq 2}$ are those that the argument from Lemma 3 yields, while O_3 contains the offsets corresponding to type $3a+2b+c$. As by Lemma 2 there are 57 intersecting domains, only 6 of the offsets from O_3 can correspond to intersecting domains. ◀

Engineering Fast Almost Optimal Algorithms for Bipartite Graph Matching

Ioannis Panagiotas
ENS Lyon, France
ioannis.panagiotas@ens-lyon.fr

Bora Uçar
CNRS and LIP, ENS Lyon, France
bora.ucar@ens-lyon.fr

Abstract

We consider the maximum cardinality matching problem in bipartite graphs. There are a number of exact, deterministic algorithms for this purpose, whose complexities are high in practice. There are randomized approaches for special classes of bipartite graphs. Random 2-out bipartite graphs, where each vertex chooses two neighbors at random from the other side, form one class for which there is an $O(m + n \log n)$-time Monte Carlo algorithm. Regular bipartite graphs, where all vertices have the same degree, form another class for which there is an expected $O(m + n \log n)$-time Las Vegas algorithm. We investigate these two algorithms and turn them into practical heuristics with randomization. Experimental results show that the heuristics are fast and obtain near optimal matchings. They are also more robust than the state of the art heuristics used in the cardinality matching algorithms, and are generally more useful as initialization routines.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases bipartite graphs, matching, randomized algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.76

Supplementary Material `https://gitlab.inria.fr/bora-ucar/fast-matching`

1 Introduction

A matching in a graph is a set of edges, such that no two of them share a common vertex. We consider the *maximum cardinality problem* in bipartite graphs which asks for a matching with maximum cardinality. There are a number of exact algorithms for this problem. The best known algorithms [21] run in $O(m\sqrt{n})$ time for a graph with n vertices and m edges. Such complexity can be prohibiting for large instances. For this reason, there is significant interest in algorithms which can find large matchings in linear or near linear time [37]. The practical use of approximate matchings in applications [33] and as an initialization to exact algorithms [30] are well known.

We investigate two randomized algorithms by Karp et al. [22] and Goel et al. [18], both of which run in $O(m + n \log n)$ time. The former algorithm finds, almost surely, maximum cardinality matchings on random graphs formed by allowing each vertex to select two vertices from the other side uniformly at random. The latter algorithm finds maximum cardinality matchings in regular bipartite graphs, where all vertices have equal degree. In both of these classes of graphs, the bipartite graphs have equal number of vertices in each part, and the maximum cardinality matchings cover all vertices (such matchings are called perfect). We investigate these two theoretical algorithms for very special cases of bipartite graphs and convert them to efficient heuristics for general bipartite graphs. We discuss our implementations and investigate the performance of the resulting heuristics in terms of run time and the matching cardinality. Both heuristics run in near linear time and obtain matchings whose cardinality is more than 0.99 of the maximum, even in cases where the current state of the art approaches have difficulties.

The rest of the paper is organized as follows. In Section 2, we give the necessary background. In Sections 3.1 and 3.2 we review the existing randomized algorithms and then discuss how we adapt them. Section 4 contains the experimental results, and Section 5 concludes the paper. Appendices A–D provide some additional results and discussion.

2 Background and notation

Let $G = (R \cup C, E)$ be a bipartite graph, where R and C are two disjoint set of vertices, and E is the set of edges. The bipartite graph G can be represented with a matrix $\mathbf{A}_G$. The vertex $r_i \in R$ corresponds to the ith row, and the vertex $c_j \in C$ corresponds to the jth column, so that $\mathbf{A}_G(i,j) = 1$ if and only if $(r_i, c_j) \in E$. We will refer to vertices of R as *rows* and to those of C as *columns* from this point on, and use $\mathbf{A}$ to refer to $\mathbf{A}_G$.

Let $\mathcal{M}$ be a matching. For $(u,v) \in \mathcal{M}$, the vertices u and v are matched, and they are each other's mate. A vertex is called free if it is not matched by $\mathcal{M}$. If there are no free vertices in R or in C, then $\mathcal{M}$ is called perfect. An augmenting path with respect to $\mathcal{M}$ is a path which starts with a free vertex and ends at another free vertex, where every second edge is in $\mathcal{M}$. A matching is maximum if and only if there are no augmenting paths [7].

A square matrix is called doubly stochastic if the sum of entries in each row and column is equal to one. An $n \times n$ matrix $\mathbf{A}$ has *support* if there is a perfect matching in the associated bipartite graph G. $\mathbf{A}$ is said to have *total support* if each edge in G is used in a perfect matching. A square matrix is fully indecomposable, if it has total support and cannot be permuted into a block diagonal matrix. Any nonnegative matrix $\mathbf{A}$ with total support can be scaled with two positive diagonal matrices $\mathbf{D_R}$ and $\mathbf{D_C}$ such that $\mathbf{A_S} = \mathbf{D_R A D_C}$ is doubly stochastic, and if $\mathbf{A}$ is fully indecomposable, then the matrices $\mathbf{D_R}$ and $\mathbf{D_C}$ are unique. The Sinkhorn–Knopp algorithm [38] is a well-known method for finding such $\mathbf{D_R}$ and $\mathbf{D_C}$ for a given matrix. This is an iterative algorithm, where at each iteration each row is normalized to have unit length, and then each column is normalized to have unit length. If a given matrix $\mathbf{A}$ has total support, then Sinkhorn–Knopp algorithm finds the unique scaling matrices. If $\mathbf{A}$ has support but not total support, then entries that cannot be put into a perfect matching tend to zero. The method converges with an asymptotical convergence rate depending on the second singular value of the final doubly stochastic matrix. There are other iterative, faster converging methods [1, 10, 28], whose iterations are more sophisticated than that of Sinkhorn–Knopp's.

A k-out subgraph G_k of a host graph G is defined by allowing each vertex in G to randomly select uniformly k of its neighbors, and the union of all selections forms the edge set of G_k. Walkup [40] shows that in the pure random k-out setting, where the host graph is the complete bipartite graph, the resulting G_k has a perfect matching with high probability for $k \geq 2$. We do not know any general result about properties of G_2 sampled from any arbitrary host graph. Frieze and Johansson [17] investigate some other properties of G_ks on host graphs where the minimum degree of a vertex is at least $n/2$. Dufossé et al. [16] propose using the doubly stochastic matrix $\mathbf{A_S}$ (scaled version of the matrix representation) for sampling and show an approximation result for G_1, when $\mathbf{A}$ has total support. We give some experiments in which G_2s generated using the same probabilities have perfect matchings in majority of the cases.

Two popular classes of randomized algorithms are *Las Vegas* and *Monte Carlo* algorithms. *Las Vegas* algorithms always return a correct answer, but their run time can depend on random choices, whereas Monte Carlo algorithms can fail with small probability, but their complexity is independent of the random choices made (see for example [34, p. 70]).

There are a number of heuristics for the cardinality matching problem [30, 37] (see Appendix A for a relevant discussion). Among those, that by Karp and Sipser [23] is very well known and widely used. This heuristic eliminates vertices of degree at most two in the following way. It matches any degree-1 vertices with their neighbors (and discards both), or merges the neighbors of a degree-2 vertex (which is then discarded) to a single node, and removes any parallel edges that occur. If neither operation can be done, it matches a pair of vertices randomly.

3 Two heuristics

We describe the original Monte Carlo algorithm [22] for finding perfect matchings in 2-out bipartite graphs in Section 3.1 and the original Las Vegas algorithm [18] for finding perfect matchings in d-regular bipartite graphs in Section 3.2. These two algorithms are based on uniform sampling. We generalize these two algorithms to general bipartite graphs within a common framework. The framework we propose scales the adjacency matrix of the input bipartite graph and uses the nonzero values of the scaled matrix for sampling. We also identify and fix an oversight in the description of the Monte Carlo algorithm, and describe efficient implementations of the two heuristics.

3.1 2outMC: Monte Carlo on 2-out graphs

3.1.1 Description of the algorithm

The Monte Carlo algorithm by Karp et al. [22] finds a perfect matching, with high probability, in a random 2-out bipartite graph, sampled from the complete bipartite graph. A random 2-out bipartite graph B_{2o} is constructed by selecting uniformly at random two row vertices for each column, and two column vertices for each row. These selections form the edges of B_{2o}. Given the edges of B_{2o}, Karp et al. define two multigraphs. The *Column-Graph* (CG) is the multigraph whose vertices are the rows, and whose edges are the choices of the columns. That is, there is an edge in CG for a column vertex in B_{2o}. Parallel edges occur if two columns select the same rows. The *Row-Graph* (RG) is defined similarly. The main idea to show that B_{2o} has a perfect matching is the following. In a component of CG that contains a cycle, it is possible to match all rows (vertices in CG) with one of the columns that have selected them (edges in CG). On the other hand in a tree component of CG, in any matching (pairing of edges with vertices) there will always be a free row vertex. As a consequence, when one or more trees appear in CG, the choices of the columns alone do not suffice to find a perfect matching, and those of the rows must be used. The algorithm thus keeps track of the tree components of CG and tries to identify one row vertex per tree component whose selections should be taken into account. The columns selected by such a row could be used for a set of rows belonging in tree components. Thus one should go back and forth identifying trees in CG and analyzing components in RG. Karp et al.'s algorithm, which is described in Algorithm 1, formalizes this approach.

The algorithm operates on H_1, a copy of CG, and H_2, a copy of RG initially devoid of edges. It furthermore uses two arrays `checked` for columns and `marked` for rows. These two arrays together signal whether a vertex will be matched with one of its two selections or not. More specifically, if a row vertex r is marked (i.e., `marked`[r]=true), then the algorithm will match r with one of its two selections. On the other hand, if a column c is checked (i.e., `checked`[c]=true), then the algorithm will match c with one of the marked row vertices that have selected it.

Initially, all row vertices are unmarked and all column vertices are unchecked. The algorithm at each step picks a tree from H_1 and marks one of its vertices x. This signifies that x can only be matched with one of its choices. Then, the edge of x is inserted in H_2. The algorithm then finds the component Q_x in H_2 containing the edge x, and selects an unchecked column y from Q_x. Column y is checked, which means that it can only be matched with a marked vertex. As y's choices are rendered useless now, the corresponding edge is removed from H_1 upon which new trees can arise. For each tree vertex x identified in H_1, one should be able to find a vertex in the associated component Q_x, so that x can be matched in that component. Otherwise, Q_x has more edges than vertices, and any matching of vertices with edges in Q_x will hence leave some edges unpaired. In other words, Algorithm 1 has decided that all columns that correspond to edges in Q_x should be matched with one of their two selections. However, the union of the rows denoted by these selections has cardinality strictly smaler than the number of such columns, and that is why a column is always left unmatched by the algorithm if this scenario occurs. The algorithm returns failure upon detecting this case (Line 10). The algorithm terminates successfully if all trees have a marked vertex. If this happens, each component in H_1 will have as many edges as unmarked vertices. Likewise, each component in H_2 will have as many edges as checked vertices. It is therefore possible to orient the edges in either H_1 or H_2 such that each vertex (excluding marked rows or unchecked columns) is matched with a unique adjacent edge. This gives a perfect matching in B_{2o}, which can be found by the Karp–Sipser heuristic in linear time. Algorithm 1 finds a perfect matching with probability $1 - O(n^{-\alpha})$, where α is a positive constant.

Algorithm 1 2OUTMC: Monte Carlo on 2-out graphs.

1: $H_1 \leftarrow CG$, $H_2 \leftarrow$ empty graph with columns as vertices;
2: All vertices in H_1 are unmarked, all vertices in H_2 are unchecked;
3: CORE $\leftarrow$ edges in cycles of CG
4: **while** there exists a tree T in H_1 with no marked vertex **do**
5: Let x be a random vertex of T ▶ x is a column vertex
6: `marked`$[x] \leftarrow$ true ▶ x must be matched with one of its choices
7: Add the edge of x in H_2
8: Let Q_x be the component in H_2 containing the edge of x
9: **if** Q_x has no unchecked vertices **then**
10: Return Fail ▶ Q_x has more edges than vertices (no 1-1 pairing possible)
11: **else**
12: Select an unchecked vertex y of Q_x. In case of ties, prefer one from CORE
13: `checked`$[y] \leftarrow$ true ▶ y will be matched with a row that selected it
14: delete y in H_1 ▶ The algorithm forgets y's choices
15: Create B'_{2o} from B_{2o} by keeping only edges between marked rows and checked columns (edges in H_2) or unmarked rows and unchecked columns (edges in H_1)
16: Apply Karp–Sipser on B'_{2o} to find a perfect matchin

The authors then describe how to efficiently implement the algorithm such that it runs in $O(n \log n)$ worst case time. They identify two main tasks:

- **Task A**: Keep track of the tree components during edge deletions in H_1.
- **Task B**: Keep track of the connected components during edge insertions in H_2, and the single unchecked vertex in each component.

Task B can be efficiently done in amortized near linear time (over the course of the algorithm) by using a union-find data-structure and keeping the identity of the single unchecked vertex in a component of H_2 at the root of the component. For Task A, Karp

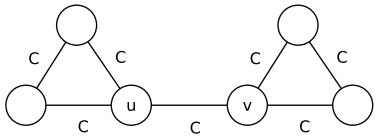


Figure 1 Algorithm 1 does not recognize new trees, if another edge is deleted after (u, v).

et al. propose the following. In the beginning, the edges of CG are labeled as $\mathcal{F}$, if their deletion creates a tree; $\mathcal{T}$, if they belong to a tree component; and $\mathcal{C}$ otherwise. Let c-degree of a vertex v be the number of $\mathcal{C}$ edges incident on v. During deleting the edge (u, v) from H_1, one of the following is performed depending on the label of (u, v).

- **Case 1**: (u, v) is $\mathcal{C}$: The c-degrees of u and v are decreased by one. Then, while there is a vertex with a single $\mathcal{C}$ edge; its $\mathcal{C}$ edge is relabeled as $\mathcal{F}$.
- **Case 2**: (u, v) is $\mathcal{F}$: Using a dove-tailed depth-first search, where depth-first searches from u and v are interleaved, the tree component created can be found in time proportional to its size. One then changes the labels of all edges in this tree from $\mathcal{F}$ to $\mathcal{T}$.
- **Case 3**: (u, v) is $\mathcal{T}$: Deleting (u, v) creates two trees. As in the previous case, a dove-tailed DFS is used to find these two trees in time proportional to the size of the smaller one. The new trees are to be examined by the algorithm.

We identify an oversight in this procedure, where the algorithm fails to keep track of some trees in H_1. We demonstrate this by an example. In Figure 1, if the edge between vertices u and v gets deleted, then the connected component is split into two triangles. The c-degree of both u and v decreases to two, and as both are greater to one, the deletion procedure stops without any action. However, both triangles are unicylic. If an edge is deleted from either triangle, then Case-1 does not recognize that the remaining edges should be relabeled as $\mathcal{T}$ not $\mathcal{F}$.

If Algorithm 1 is not able to keep track of all the trees in H_1, then it can exit the loop of Line 4 prematurely. As a consequence Karp–Sipser in Line 16 will return a suboptimal matching. We propose a fix for this oversight in Lemma 1.

▶ **Lemma 1.** *Let u be an endpoint of a deleted edge (u, v) with label $\mathcal{C}$. Apply the procedure of Case-1 until we arrive at a vertex p with* c-degree$[p] \neq 1$. *If* c-degree$[p] = 0$, *then u's component has become a tree.*

Proof. We claim that if c-degree$[p] = 0$, then p and v are the same vertex. Each vertex on the path from u to p had its c-degree affected twice (from 2 to 0), except p. Hence for p to become 0, its c-degree must have been equal to 1. If $p \neq v$, then p should had its $\mathcal{C}$ edge relabeled during another deletion process. Therefore, prior to the deletion of (u, v), there was a cycle on H_1 with all vertices having c-degree equal to 2, and both their $\mathcal{C}$ edges participated in the cycle. Any outgoing edges from vertices of the cycle therefore were labeled $\mathcal{F}$ and by definition, their deletion led to a tree being formed. The component was hence unicyclic before. ◀

Case 1-continuation is therefore as follows:

- Once there are no vertices with c-degree equal to 1, take the last vertex v whose c-degree was reduced. If c-degree$[v] = 0$, then relabel all edges in vs component from $\mathcal{F}$ to $\mathcal{T}$.

This addition has overall $O(n)$ cost, because each edge can change label at most twice.

3.1.2 Conversion to an efficient general heuristic

Algorithm 1 works well when the random 2-out graph is sampled from $K_{n,n}$. However, in the case of an arbitrary host graph, the underlying theory is not shown to hold, and the algorithm can make erroneous decisions. Here we discuss how to turn Algorithm 1 into a general heuristic. Apart from the aim of obtaining a practical heuristic for bipartite matching, there is another reason to investigate the matching problem in 2-out bipartite graphs. We show in Appendix D that an $O(f(n,m))$ time algorithm to find a maximum cardinality matching in a 2-out bipartite graph can be used to find a maximum cardinality matching in any bipartite graph with m edges in $O(f(m,m))$ time, where f is a function on the number of vertices n and edges m. Such a reduction is important because it shows that an algorithm for finding maximum cardinality matchings in 2-out graphs with similar complexity to 2OUTMC can be used to obtain an $O(m \log m)$ algorithm for matchings in general bipartite graphs.

If the algorithm reaches Line 10 during execution, it quits immediately before examining all trees in H_1. We instead propose to continue with the execution of the algorithm to make the returned matching as large as possible. To achieve this efficiently, we keep for each tree T a list L_T of unmarked vertices. At Line 5 we randomly sample x from L_T and discard it from L_T. Contrary to Algorithm 1, we neither mark x nor insert it in H_2 yet. Instead, we examine first whether the component in H_2 of either of the two choices of x has an unchecked column y. If y exists, we mark x, insert it to H_2 and continue by deleting y from H_1. Otherwise, we perform the same set of actions with another randomly sampled vertex from L_T. If L_T becomes empty, and no vertex was marked, we abandon T and proceed to another tree. Each such tree in the final state of H_1 decreases the cardinality of the returned matching by one, as a row is left free. If T is split into two trees, the lists of unmarked vertices for the new trees contain only those vertices still inside L_T at the moment of splitting. This is necessary to avoid sampling vertices more than once.

The overall algorithm 2OUTMC is as follows. It takes the matrix representation of the given bipartite graph and scales it with a few steps of the Sinkhorn–Knopp algorithm to obtain $\mathbf{A_S}$. It then chooses two random neighbors for each column and row using their respective probability distributions in the corresponding row and column of $\mathbf{A_S}$, which are given as input to Algorithm 1. Then, the auxiliary graph B_{2o} is constructed and Karp–Sipser is run on this graph to retrieve a maximum cardinality matching in B_{2o}. If one allows vertices to choose neighbors uniformly, then there are no guarantees on the maximum cardinality of a matching in B_{2o}. As an example, consider the graph where the ith row and ith column are connected for $i = 1, \ldots, n$, and additionally the first ℓ rows and columns are connected with every vertex on the opposite side. Then, in expectation $O(\frac{\ell-1}{\ell+1} \cdot n)$ rows (resp. columns) make both choices from the first ℓ columns (resp. rows), such that in the generated B_{2o} the maximum cardinality matching is of size $O(\frac{n}{\ell} + \ell)$. Using $\mathbf{A_S}$'s values to perform the random choices spreads the choices so that the maximum cardinality of the matching in the subgraph increases (see Theorem 2 and Lemmas 6–8 in [16] that examines the 1-out subgraph model).

In Appendix B we describe two heuristics for 2OUTMC which can lead to an increase in the cardinality of the returned matching. The main idea of both heuristics is to reduce the chance that an edge deletion in H_1 creates a new tree.

3.2 TruncRW: Truncated random walk with nonuniform sampling

3.2.1 Description of the algorithm for regular bipartite graphs

Goel et al. [18] propose a randomized algorithm (of the Las Vegas type) that finds a perfect matching in a d-regular bipartite graph with n vertices in each side in $O(n \log n)$ time in expectation. This algorithm starts a random walk from a randomly chosen free column-vertex.

At a column vertex c, the algorithm selects uniformly at random one of the row-vertices that are not matched to c, and goes to the chosen row vertex r. If r is free, then an augmenting path is obtained by removing possible loops from the walk. If r is matched, then the random walk goes to the mate of r. Goel et al. show that the total length of the random walks is $O(n \log n)$ in expectation, and thus the algorithm obtains a perfect matching in the stated time [18, Theorem 4]. They also show that one can obtain a Monte Carlo-type algorithm by truncating the random walks. The expected length of an augmenting path with respect to a given matching of cardinality j is $2(4 + 2n/(n - j))$, and the random walks could be truncated at this length to obtain near optimal matchings in $O(n \log n)$ time.

A random walk is easy to implement for d-regular bipartite graphs. At a column vertex c, one can create a random number between 1 and d in $O(1)$ time and choose the neighbor at that position, and repeat the experiment if the mate of c is chosen. This will take $O(1)$ time in expectation for each step of the walk, and the run time bound of $O(n \log n)$ is maintained.

Goel et al. show that the random-walk based algorithm will work for finding perfect matchings in the bipartite graph representation of a doubly stochastic matrix. They also suggest using an existing data structure [20] when the row and column sums are constant with nonnegative integer entries bounded by a polynomial in n, to attain an $O(n \log n)$ run time bound. A more recent paper [32] removes the restriction on the entries, and obtains an expected constant time per update and sampling. Further investigations and a careful implementation are necessary to apply the mentioned sampling approaches in our context. Instead, for general doubly stochastic matrices without any bound on the entries, Goel et al. propose an augmented binary search tree with which each selection step of the random walk can be implemented in $O(\log n)$ time, and obtain a run time of $O(m + n \log^2 n)$ in expectation, with a total of $O(m)$ preprocessing time.

3.2.2 Conversion to an efficient general heuristic

Let c be a free column vertex with respect to a given matching of cardinality j. Assuming there is a perfect matching, one can find an augmenting path to match c, and a random walk can find it. The $O(\frac{n}{n-j})$ bound on the expected length of such a path will not hold if the bipartite graph is not regular. One may perform more than m steps, which is the worst case time complexity of deterministically finding an augmenting path starting from a free vertex. We propose two methods to make the random walks more useful and to sample efficiently in a random walk. We also discuss an efficient implementation of the whole approach.

The first proposed method is to scale the matrix representation $\mathbf{A}$ of a given bipartite graph to obtain a doubly stochastic matrix $\mathbf{A_S}$ for random selections. The expected length of a random walk to find an augmenting path holds when $\mathbf{A_S}$ has bounded nonzero entries. In general, ones does not have any bound on the entries of $\mathbf{A_S}$. Consider the matrix $\mathbf{A}$ associated with an upper Hessenberg matrix of size n. $\mathbf{A}$ has a full lower triangular part, and additional $n - 1$ entries $\mathbf{A}(i - 1, i) = 1$ for $i = 2, \ldots, n$, and fully indecomposable. The 4×4 example along with its unique scaling matrices are shown in Fig. 2. In the resulting scaled matrix $\mathbf{A_S}(n, 1) = 1/2^{n-1}$ whose inverse is not bounded polynomially in n.

As highlighted at the end of Section 3.2, one needs an $O(\log n)$ time algorithm to select a row vertex randomly from a given column vertex. The second proposed method is a simple yet efficient algorithm for this purpose, rather than a sophisticated augmented tree. The main components of the proposed sampling method are as follows. For each column vertex c, with d_c neighbors, we have:

$$\begin{pmatrix} \sqrt{2} & & & \\ & \frac{1}{\sqrt{2}} & & \\ & & \frac{1}{\sqrt{8}} & \\ & & & \frac{1}{\sqrt{8}} \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{8}} & & & \\ & \frac{1}{\sqrt{8}} & & \\ & & \frac{1}{\sqrt{2}} & \\ & & & \sqrt{2} \end{pmatrix} = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/4 & 1/4 & 1/2 & 0 \\ 1/8 & 1/8 & 1/4 & 1/2 \\ 1/8 & 1/8 & 1/4 & 1/2 \end{pmatrix}$$

Figure 2 The matrix $\mathbf{A}$ associated with a 4×4 Hessenberg matrix, the scaling matrices $\mathbf{D_R}$ and $\mathbf{D_C}$, and the resulting doubly stochastic matrix $\mathbf{A_S} = \mathbf{D_R A D_C}$. In general, $\mathbf{A_S}(n, 1) = 1/2^{n-1}$.

- $\mathsf{adj}_c[1, \ldots, d_c]$: an array keeping the neighbors of c.
- $\mathsf{wghts}_c[1, \ldots, d_c]$: the weight of the edges incident on c. This array is parallel to the first one so that the weight of the edge $(c, \mathsf{adj}_c[i])$ is $\mathsf{wghts}_c[i]$.
- $\mathsf{medge}[c]$: the position of the mate of c in the array adj_c, or -1 if c is not matched.

At the beginning, we compute the prefix sum of $\mathsf{wghts}_c[1, \ldots, d_c]$. After this operation, the total weight of the edges incident on c is $\mathsf{wghts}_c[d_c]$, and the weight of the edge $(c, \mathsf{mate}[c])$ is $\mathsf{wghts}_c[\mathsf{medge}[c]] - \mathsf{wghts}_c[\mathsf{medge}[c] - 1]$, assuming that $\mathsf{wghts}_c[0]$ signifies zero.

Given the prefix sums in $\mathsf{wghts}_c[1, \ldots, d_c]$, the position of the mate of c at $\mathsf{medge}[c]$, we can choose a random neighbor (which is not equal to $\mathsf{mate}[c]$) as shown in Algorithm 2. We use a binary search function, binSearch, which takes an array, the array's start and end positions, a target value, and returns the smallest index of an array element which is larger than the given value with binary search (we skip the details of this search function). At Line 5, since c does not have a mate, we search in the whole list. At Line 8, since the prefix sum just before $\mathsf{medge}[c]$ is larger than the target value, we search in the first part of wghts_c until the current mate located at $\mathsf{medge}[c]$. At Line 10, we search on the right of $\mathsf{medge}[c]$, by a modified target value. This last part is the gist of the algorithm's efficiency as it avoids updating the prefix sums when the mate changes.

Algorithm 2 Sampling a random neighbor of the column vertex c with d_c neighbors.

Require: $\mathsf{adj}_c[1, \ldots, d_c]$, $\mathsf{wghts}_c[1, \ldots, d_c]$, and $\mathsf{medge}[c]$.
1: mwght $\leftarrow \mathsf{wghts}_c[\mathsf{medge}[c]] - \mathsf{wghts}_c[\mathsf{medge}[c] - 1]$ if $\mathsf{medge}[c] \neq -1$, otherwise 0
2: totalW $\leftarrow \mathsf{wghts}_c[d_c] -$ mwght ▶ The total weight of the edges that can be sampled
3: create a random value rv between 0 and totalW
4: **if** $\mathsf{medge}_c = -1$ **then**
5: **return** binarySearch($\mathsf{wghts}_c[1, \ldots, d_c], rv$)
6: **else**
7: **if** $\mathsf{wghts}_c[\mathsf{medge}_c] -$mwght $\geq rv$ **then**
8: **return** binSearch($\mathsf{wghts}_c[1, \ldots, \mathsf{medge}[c] - 1], rv$)
9: **else**
10: **return** binSearch($\mathsf{wghts}_c[\mathsf{medge}[c] + 1, \ldots, d_c], rv +$ mwght$) + \mathsf{medge}_c$

The sampling algorithm returns the index of the neighbor in adj_c different from the current mate in time $O(\log d_c)$, independent of the values of the edges. It thus respects the required run time bound. If we were to apply the rejection sampling (as discussed before for the regular bipartite graphs), the run time would depend on the value of the matching edge that we want to avoid. This could of course lead to an expected run time of more than $O(n)$.

There are two key components of Algorithm 2. The first one is the prefix sum, which is computed once before the random walks start and does not change. The second one is $\mathsf{medge}[c]$, the position of $\mathsf{mate}[c]$ in adj_c. The value $\mathsf{medge}[c]$ changes and needs to be updated when we perform an augmentation. We handle this update as follows. We keep the random walk in a stack by storing only the column vertices, as the row vertices direct the walk to their mate, or terminate the walk if not matched. We discard the cycles from the random

walk as soon as they arise – this way we only store a path on the stack, and its length can be at most n. Storing a path also enables keeping the $\mathsf{medge}[\cdot]$ up-to-date. Every time we sample an outgoing edge from a column vertex c, we assign the location of the sampled row vertex in adj_c to a variable $\mathsf{nmedge}[c]$. When we find a free row, the stack contains the column vertices of the corresponding augmenting path, whose new mates' locations are in $\mathsf{nmedge}[\cdot]$ and thus can be used to update $\mathsf{medge}[\cdot]$.

The described procedure will work gracefully in expected $O(m + n \log n)$ time for regular bipartite graphs and for doubly stochastic matrices where the nonzero values do not differ by large. On the other hand, when there are large differences in edge weights, a random walk can get stuck in a cycle. That is why truncating the long walks is necessary to make the algorithm work for any given doubly stochastic matrix. Furthermore, such a truncation is necessary with the proposed matrix scaling approach for defining random choices. For the overall approach to be practical, we should not apply the scaling algorithms until convergence. As in the previous approaches [15, 16], we allot a linear time of $O(m + n)$ for scaling. Applying Sinkhorn–Knopp algorithm for a few iterations will thus be allowable. The known convergence bounds for the Sinkhorn–Knopp algorithm [27, Thm. 4.5] apply asymptotically, therefore we do not have any bounds on the error after a few iterations; it can be large. That is why truncation makes the random walk based augmenting path search practical.

The overall algorithm TruncRW is thus as follows. It takes the matrix representation of the given bipartite graph and scales it with a few steps of the Sinkhorn–Knopp algorithm. Then for $j = 0$ to $n - 1$, it uniformly at random picks a free column vertex, and starts a random walk starting from that column, for at most $2(4 + 2n/(n - j))$ steps, after which the walk is truncated. Some follow discussion and experiments with different parameters for TruncRW may be found in Appendix C.

4 Experiments

We implemented 2outMC and TruncRW in C/C++, and the codes are accessible from `https://gitlab.inria.fr/bora-ucar/fast-matching`. The codes, all are sequential, were compiled with "-O3" and run on a machine with 2 x Intel Xeon CPU Gold 6136 CPUs and 187 GB RAM. We evaluate 2outMC and TruncRW both on real-life and synthetic bipartite graphs with equal number of vertices in each side. We compared the two algorithms against KaSi, the widely used version of Karp–Sipser which applies degree-1 reduction (own implementation), and KaSi2, the original version of Karp–Sipser with both reduction rules. We use a publicly available implementation of KaSi2 (`https://gitlab.inria.fr/bora-ucar/karp--sipser-reduction`) which is the fastest of recent implementations [26, 29]. We note that there are other heuristics (a short summary and further references are in Appendix A) which deliver very good results in practice. For most of these heuristics, especially for those based on vertex degree, there are known worst case upper bounds close to 1/2. We therefore restrict the focus on KaSi and KaSi2, which are efficient and very effective in practice [14, 25, 30]. We also investigated if random 2-out bipartite graphs of a general host graph have perfect matchings if rows and columns select neighbors with the probabilities in the scaled matrix representation. The quality of a matching refers to the ratio of the cardinality of the matching to the maximum cardinality of a matching in a given graph. The practical version of Sinkhorn-Knopp is referred to as SK-t, where t is the number of allowed iterations. All run times are reported in seconds.

Table 1 We divide the real-life graphs into five groups. The ith group consists of graphs whose $\frac{m}{n}$ ratio is between $10(i-1)$ and $10i$. For each group, we give the number of instances in which a 2-out graph built using the models M_1 and M_2 has a perfect matching and the largest difference from the maximum cardinality of a matching.

$\frac{m}{n}$	[0,10)		[10,20)		[20,30)		[30,40)		[40,50)	
#Instances	27		5		5		1		1	
	#PM	deficiency	#PM	deficiency	#PM	deficiency	#PM	deficiency	#PM	deficiency
Model M_1	0	223	0	8	1	20	0	2	1	0
Model M_2	27	0	3	3	1	10	0	1	0	1

4.1 Investigation of perfect matchings in 2-out graphs

Here, we investigate the claim that G_2 will likely have a perfect matching for G, if created with the probabilities in the scaled matrix. We used a set of 39 large sparse square matrices from the SuiteSparse Collection [12], whose bipartite graphs have perfect matchings. These matrices are automatically selected from all square matrices available at the collection with $10^6 \leq n \leq 28 \times 10^6$, and with at least two nonzeros per row or column.

We consider two different models to create G_2. In the model M_1, row choices are independent of the column choices. Under this model, a row and a column can select each other resulting in parallel edges – only one of them is kept. The model M_2 tries to avoid parallel edges. In this model, all columns perform their selections. Then, each row r attempts to randomly choose two columns, only from those that did not select r. These selections again are based on the scaled matrix. In this model, parallel edges can arise (and be discarded) only when a vertex v is connected in the 2-out graph with all of its neighbors in G, because it is impossible for v to select otherwise. We experimented three times with each real-life graph. M_i's result is the maximum of those three experiments. In each test, we first created the choices of all columns. Then we allowed the two models to generate the choices of the rows accordingly.

The results are shown in Table 1 for the 39 real-life graphs and are with SK-5. As seen in this table, the random G_2 graphs generated with the model M_1 have near perfect matchings, but they do not contain perfect matchings in most cases. In contrast, the random G_2 graphs generated by M_2 in many cases contain a perfect matching. In only a few graphs this does not hold true, and in these cases the deficiency is no more than 10.

4.2 On synthetic graphs

In Table 2, we give results with a synthetic family $\mathcal{I}$ of graphs from literature [16], whose matrix representations do not have total support. To create a member of $\mathcal{I}$, we separate the vertex set R into $R_1 = \{r_1, \ldots, r_{n/2}\}$ and $R_2 = \{r_{n/2+1}, \ldots, r_n\}$ and likewise for C. All vertices of R_1 are connected to all vertices of C_1. Edges $(r_i, c_{n/2+i})$ and $(r_{n/2+i}, c_i)$ for $i = 1, \ldots, n/2$ are added to introduce a perfect matching. A parameter h is used to connect h vertices from R_1, and h vertices from C_1 to every vertex on the opposite side.

As seen in Table 2, KaSi and KaSi2 have more and more difficulty with increasing h. The matching quality drops over 30% between $h = 2$ and $h = 512$ for KaSi and almost 40% for KaSi2. On the contrary, 2outMC and TruncRW both obtain a near perfect matching, with SK-5. Even though the matrices associated with the graphs of $\mathcal{I}$ lack total support, SK-5 sufficed to obtain near optimal matchings. We notice the effect of scaling: if vertices select without scaling (Uniform), the matching quality reduces. This is particularly true for 2outMC, which exhibits the worst overall performance with uniform selection. Family

Table 2 Average quality of the matchings found by the algorithms on graphs from the synthetic family $\mathcal{I}$ for $n = 30000$ and various values of h.

			2OUTMC		TRUNCRW	
h	KASI	KASI2	Uniform	SK-5	Uniform	SK-5
2	0.93	1.00	0.78	0.99	0.88	0.99
8	0.80	0.85	0.59	0.99	0.91	0.99
32	0.69	0.72	0.52	0.99	0.83	0.99
128	0.64	0.65	0.51	0.99	0.78	0.99
512	0.61	0.63	0.52	0.99	0.76	0.99

Table 3 Average quality of the matchings found by the algorithms on graphs from the synthetic family $\mathcal{J}$ for $n \in \{10000, 20000, 30000\}$.

	KASI	KASI2	2OUTMC			TRUNCRW		
n	quality	quality	uniform	SK-5	SK-20	uniform	SK-5	SK-20
10000	0.76	0.84	0.81	0.92	0.95	0.97	0.97	0.97
20000	0.73	0.83	0.81	0.92	0.95	0.97	0.97	0.97
30000	0.73	0.83	0.81	0.92	0.95	0.97	0.97	0.97

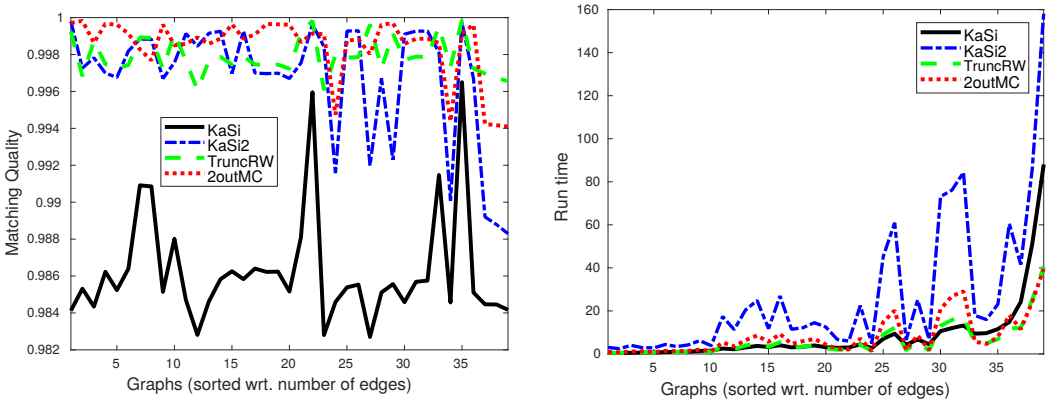
$\mathcal{I}$ shows the importance of scaling, and more importantly highlights the robustness of the proposed methods. An adversary can create graphs which make degree-based randomized approaches lose quality – some of those heuristics are briefly mentioned in Appendix A, and the full details including negative results on KASI2 can be found elsewhere [9]. On the other hand, the use of scaling helps to avoid such cases for 2OUTMC and TRUNCRW.

We now discuss another synthetic family of graphs $\mathcal{J}$ in which the proposed approaches obtain matchings of much higher quality than KASI and KASI2. A bipartite graph with n vertices per side belonging to $\mathcal{J}$ contains the following edges: (r_i, c_j) for all $i \leq j$; (r_2, c_1), (r_n, c_{n-1}); (r_3, c_1), (r_3, c_2), (r_n, c_{n-2}); and (r_{n-1}, c_{n-2}). The graphs in $\mathcal{J}$ are hard for Karp–Sipser-based heuristics because only few of the edges participate in a perfect matching, the deterministic rules do not apply, and hence they resort to multiple suboptimal random decisions. Likewise, due to the large number of entries without support in the matrix representation, Sinkhorn–Knopp will take many iterations to properly scale the matrix.

In Table 3, we give results of the algorithms for a few graphs from this family. In the table, we also show the effects of scaling on 2OUTMC and TRUNCRW by showing results without scaling (under column "uniform", in which a column vertex chooses a neighbor uniformly at random), with SK-5, and with SK-20. As can be seen, despite the lack of total support, both 2OUTMC and TRUNCRW obtain matchings whose cardinality is more than 0.92 of the maximum, when SK-5 or SK-20 is used. TRUNCRW in particular is nearly optimal. These results are always better than that of KASI and KASI2, with the difference in matching quality being about 20–25% for the former, and 10–15% for the latter. With increased iterations of Sinkhorn–Knopp, 2OUTMC increases the cardinality of its matchings by 3%. If we do not use scaling ("uniform"), while there's no noticeable effect on TRUNCRW's matchings, 2OUTMC matchings decrease by roughly 10%. Even so, its results remain better than KASI's and on par with those of KASI2.

4.3 On real-life graphs

We compared TRUNCRW and 2OUTMC with KASI and KASI2 on all 39 real-life graphs from Section 4.1. Figure 3a and Figure 3b present the high level picture. For the experiments, we did not permute the matrices randomly, which generally increases the experimentation time.



(a) Quality results. **(b)** Run time results.

Figure 3 Quality (left) and run time (right) results for all 39 graphs from Section 4.1.

The results for matching quality can be seen in Figure 3a, where we plot the ratio of the cardinality of the matchings found by different algorithms to the maximum cardinality of the matching. The graphs are indexed in nondecreasing number of edges. 2OUTMC and TRUNCRW use SK-3 for scaling. As can be observed, both 2OUTMC and TRUNCRW obtain near perfect matchings. The average matching quality obtained by 2OUTMC is 0.9979 and that obtained by TRUNCRW is 0.9984. Both algorithms never drop below 0.9900 in any of the 39 cases.

Figure 3a also shows the matching quality of KASI2 and KASI. KASI obtains matchings of quality 0.9862 on average, with always smaller cardinality than TRUNCRW and 2OUTMC. KASI2 fares better and its average quality is 0.9968. Even so, in the majority of cases, it obtains matchings that are inferior quality-wise to both TRUNCRW and 2OUTMC.

While all algorithms obtain matchings of high quality, the absolute different is remarkable in some cases. For example, the largest difference observed between the matching cardinalities obtained by 2OUTMC and KASI was 346577, in favor of 2OUTMC.

Figure 3b shows the run time of all examined heuristics, where the graphs are again indexed in nondecreasing number of edges. KASI is in general the fastest of these four algorithms when there are not too many edges. TRUNCRW and 2OUTMC are close run-time wise to KASI and in some instances faster than it. This is especially true in instances with many edges because KASI depends more on m. KASI2 has the slowest performance overall.

For a detailed study, we show results on the five largest graphs from the mentioned dataset and `Circuit5M`, which was identified as a challenging instance in earlier work [25]. Degree-1 vertices from `Circuit5M` are removed by applying Rule-1 of KASI2 as a preprocessing step – this is without loss of generality of the heuristics. For each graph we relabeled its row-vertices randomly and executed five tests with each algorithm.

Table 4 shows the matching quality and the run time of the four heuristics. 2OUTMC and TRUNCRW used SK-3 for this set of experiments for speed. For each graph, we give the minimum, maximum, and averages over five runs. As already discussed, all heuristics obtain high quality matchings. On a closer look, we see that TRUNCRW, on average, matched 158410 more edges than KASI, and 50847 more edges than KASI2. Similarly 2OUTMC matched 139220 more edges than KASI on average, and 31652 more edges than KASI2. Interestingly, on graph `Channel-500` TRUNCRW was able to find the maximum matching.

Table 4 Full run time comparisons with heuristics for the graphs of Section 4.3. The run time of SK-3 should be added to TruncRW and 2outMC. For each instance we give the minimum, the average, and the maximum of five runs for all columns regarding the quality and the run time. The number of vertices n per side is in the order of millions. `Hugebub-20` stands for `Hugebubbles-0020`.

			KaSi		KaSi2		SK-3	2outMC		TruncRW	
name	n	statistics	quality	time	quality	time	time	quality	time	quality	time
`cage15`	5.15	min.	0.99	12.67	0.99	26.89	4.59	0.99	8.82	0.99	8.27
		avg.	0.99	12.81	0.99	27.08	4.68	0.99	8.88	0.99	9.32
		max.	0.99	13.17	0.99	27.27	4.83	0.99	8.96	0.99	10.23
`Channel-500`	4.80	min.	0.99	10.12	0.99	20.63	2.74	0.99	7.63	1.00	3.86
		avg.	0.99	10.16	0.99	20.94	2.75	0.99	7.66	1.00	4.48
		max.	0.99	10.18	0.99	21.87	2.75	0.99	7.70	1.00	5.11
`Circuit5M`	5.55	min.	0.99	6.57	0.99	24.74	2.45	0.99	4.40	0.99	2.07
		avg.	0.99	6.76	0.99	24.93	2.84	0.99	4.56	0.99	2.19
		max.	0.99	7.03	0.99	25.33	4.16	0.99	4.81	0.99	2.35
`Delaunay_24`	16.00	min.	0.99	11.58	0.99	65.97	4.32	0.99	23.34	0.99	11.21
		avg.	0.99	11.61	0.99	68.30	4.44	0.99	23.58	0.99	11.31
		max.	0.99	11.66	0.99	72.47	4.48	0.99	24.38	0.99	11.37
`Hugebub-20`	21.19	min.	0.99	14.97	0.99	91.42	6.26	0.99	30.96	0.99	14.25
		avg.	0.99	15.04	0.99	97.77	6.29	0.99	31.28	0.99	14.38
		max.	0.99	15.15	0.99	106.78	6.31	0.99	31.59	0.99	14.57
`nlpkkt240`	27.99	min.	0.98	98.58	0.99	182.08	29.77	0.99	52.34	0.99	34.34
		avg.	0.98	98.66	0.99	183.10	29.92	0.99	52.53	0.99	34.50
		max.	0.98	98.76	0.99	186.08	30.27	0.99	52.76	0.99	34.70

Concerning run time, as KaSi is a linear time heuristic it is expected to be the fastest. Surprisingly, TruncRW even with the scaling time added is faster than KaSi in three instances. This is due to the fact that each iteration of the scaling algorithm takes linear time with small constants. As an algorithm on its own (without scaling time), TruncRW becomes the fastest one, thanks to its run time not depending on m after the initialization. 2outMC, though slower, also exhibits good behavior, except in `nlpkkt240`. KaSi2 has the worst run time overall. Its initialization takes more time, and its implementation is more involved. SK-3 is fast except for `nlpkkt240` where it requires about 30 seconds. The reason that SK-3 requires 30 seconds for this particular graph is due to the random permutation of its rows, which is not cache-friendly (if SK-3 is run on `nlpkkt240` using the initial ordering of rows, it finishes in less than 10 seconds). In the other cases and despite the large size of the graphs, scaling finishes in less than seven seconds. Table 4 additionally shows that TruncRW and 2outMC's run time performance does not seem to be affected by their random decisions. The largest difference between the result of the minimum, and the maximum run is no more than two seconds for both of these algorithms.

Combined with the results in the previous section, we conclude thus that (i) 2outMC and TruncRW always obtain near perfect matchings, while KaSi and KaSi2 are not as robust; (ii) 2outMC and TruncRW are nearly as fast as the linear time algorithm KaSi, and are much faster than KaSi2.

Next, we consider the impact of our heuristics as initialization to an exact algorithm for finding a maximum cardinality matching. We first run the heuristics to obtain an initial matching, then call an exact algorithm to augment the initial matchings for maximum cardinality. We consider three different exact algorithms MC21, PR, and PF+ for the augmentation steps. MC21 [13] from `mmaker` [14, 25] visits free vertices one by one and tries to match the visited vertex with a depth-first search, and hence is closely related to TruncRW. In this setting, differences among the qualities of initial matchings should

be observable while computing an exact matching. PR [25] is based on the Push-Relabel method [19], and PF+ which is a depth-first search based method [14, 36]. The last two algorithms are more elaborate than MC21, and the cardinality difference between two different initial matchings does not necessarily correlate with the run time.

The statistics of five runs with MC21 are given in Table 5. In this table, the time spent in augmentations is given in column "augment.". The overall time to compute a maximum cardinality matching is given in column "overall', which includes the time spent in heuristics – in case of 2OUTMC and TRUNCRW it includes the scaling time as well. The runs on `nlpkkt240` did not finish within an hour and are not presented. As seen in the table, the overall time to obtain a maximum cardinality matching is always the smallest with TRUNCRW initialization. 2OUTMC is usually competitive with the faster of KASI2 and KASI, without a clear winner. It is also interesting to note that in all graphs the worst behavior of TRUNCRW is better than the best behavior of KASI2 and KASI and in some cases (see `cage15` or `Channel-500`) significantly so. The same is almost true for 2OUTMC as well except for graphs `Delaunay_24` and `Hugebbubles-0020` where 2OUTMC's worst result is only a few seconds slower than KASI's best result, or `cage15` versus KASI2.

In Table 6, we observe the behavior of the heuristics when used for initializing the PF+ algorithm. The table shows the minimum, average, and maximum time over the five runs. As can be observed, TRUNCRW exhibits the best overall behavior. TRUNCRW has the fastest performance in four out of six instances, and in the remaining two instances it is very close to KASI. The largest difference between the two can be observed in `nlpkkt240` where KASI is overall almost 50 seconds slower. The total run time with KASI2 is never better than that with TRUNCRW. It roughly takes the same amount of time for PF+ to augment 2OUTMC's initial matching, as it takes for it to augment the matching of TRUNCRW. Therefore, when 2OUTMC has a run time similar to TRUNCRW their overall run times are similar. In the largest of instances 2OUTMC's and TRUNCRW's performance diverge, but 2OUTMC's overall behavior is superior to KASI2 and competitive with that of KASI.

In Table 7, we observe the behavior of the heuristics when used for initializing the PR algorithm. The behavior of KASI in `Circuit5M` demonstrates the robustness of our approaches. The average behavior of PR initialized with KASI is 339 seconds with the maximum run time exceeding 500 seconds. In stark contrast, PR with TRUNCRW's input never needs more than 25 seconds, whereas with 2OUTMC it never surpasses 150 seconds. In the remaining instances, the proposed algorithms are competitive with KASI or even faster.

In summary, the effects of the proposed methods as an initialization routine are more observable with MC21 on all instances. With PF+, we see that the augmentations take less time on average with 2OUTMC and TRUNCRW, but the overall time with KASI can be sometimes better than that of TRUNCRW slightly thanks to KASI being faster. When PR is used, the augmentations take less time with KASI in three instances compared to TRUNCRW; and in four instances compared to 2OUTMC. When 2OUTMC and TRUNCRW serve better than KASI as an initialization to PR, the difference is more significant. The above results with three different algorithms demonstrate the merits of the two proposed algorithms for use as initialization routines in exact matching algorithms.

5 Conclusions

We have examined two randomized algorithms for the maximum cardinality matching problem in bipartite graphs. These algorithms originally were designed for two very special classes of bipartite graphs. We have discussed how to convert them into efficient and effective heuristics.

Table 5 Detailed run times when MC21 is used for augmentations on the graphs described in Section 4.3. The quality of heuristics are in Table 4. We have omitted graph `nlpkkt240` for which MC21 did not finish within a reasonable amount of time. For each instance we give the minimum, the average, and the maximum run time of five runs. `Hugebub-20` stands for `Hugebubbles-0020`.

		KaSi		KaSi2		2outMC		TruncRW	
name	statistics	augment	overall.	augment	overall.	augment	overall.	augment	overall
cage15	min.	133.85	146.52	7.42	34.47	27.29	40.75	0.22	14.07
	avg.	140.13	152.94	8.81	35.90	31.44	45.00	1.85	15.84
	max.	144.42	157.28	10.70	37.59	37.84	51.47	2.46	16.84
Channel-500	min.	64.29	74.46	9.15	29.81	12.18	22.62	0.04	6.65
	avg.	71.61	81.76	10.93	31.86	15.28	25.68	0.14	7.36
	max.	78.81	88.98	11.71	33.58	18.84	29.25	0.25	8.11
Circuit5M	min.	14.33	20.94	10.51	35.32	4.38	12.21	0.50	5.02
	avg.	15.26	22.01	13.11	38.04	5.70	13.09	0.77	5.80
	max.	16.00	22.72	14.42	39.43	6.81	13.68	1.31	7.79
Delaunay_24	min.	49.95	61.54	26.93	94.02	35.10	63.71	26.77	42.49
	avg.	54.79	66.40	29.99	98.29	36.68	64.70	31.06	46.81
	max.	61.23	72.81	32.70	104.13	40.30	68.11	34.09	49.77
Hugebub-20	min.	68.17	83.14	55.79	148.64	44.83	82.31	42.02	62.56
	avg.	73.15	88.20	58.95	156.72	50.65	88.21	44.54	65.21
	max.	75.99	91.10	61.18	166.98	54.35	91.60	47.11	67.68

Table 6 Detailed run times when PF+ is used for augmentations on the graphs described in Section 4.3. The quality of heuristics are in Table 4. For each instance we give the minimum, the average, and the maximum run time of five runs. `Hugebub-20` stands for `Hugebubbles-0020`.

		KaSi		KaSi2		2outMC		TruncRW	
name	statistics	augment.	overall	augment.	overall	augment.	overall	augment.	overall
cage15	min.	2.19	14.89	2.11	29.18	1.90	15.46	0.73	14.22
	avg.	2.51	15.33	2.59	29.67	1.97	15.53	1.16	15.15
	max.	2.98	16.15	3.16	30.43	2.01	15.69	1.55	15.63
Channel-500	min.	1.70	11.84	1.82	22.50	1.19	11.60	0.04	6.66
	avg.	1.91	12.06	2.07	23.01	1.30	11.71	0.04	7.27
	max.	2.60	12.77	2.89	23.69	1.40	11.84	0.05	7.90
Circuit5M	min.	0.63	7.20	0.45	25.28	0.45	7.34	0.48	5.01
	avg.	0.77	7.53	0.62	25.55	0.53	7.93	0.58	5.61
	max.	0.97	7.97	0.90	25.92	0.67	9.55	0.64	7.04
Delaunay_24	min.	18.47	30.06	13.88	80.75	14.24	42.05	14.20	29.92
	avg.	20.83	32.44	14.89	83.19	15.47	43.49	17.67	33.41
	max.	22.33	33.91	16.17	86.35	17.12	44.98	20.40	36.09
Hugebub-20	min.	23.09	38.09	14.99	106.41	23.27	60.75	21.97	42.54
	avg.	28.13	43.17	19.63	117.40	26.97	64.53	24.49	45.16
	max.	34.11	49.26	23.00	127.49	30.38	68.17	29.65	50.53
nlpkkt240	min.	27.01	125.69	28.19	210.27	14.91	97.26	13.76	77.87
	avg.	27.09	125.76	29.63	212.73	17.56	100.01	13.96	78.38
	max.	27.24	125.83	30.27	216.15	20.99	103.47	14.09	79.06

Table 7 Detailed run times when PR is used for augmentations on the graphs described in Section 4.3. The quality of heuristics are in Table 4. For each instance we give the minimum, the average, and the maximum run time of five runs. `Hugebub-20` stands for `Hugebubbles-0020`.

		KaSi		KaSi2		2outMC		TruncRW	
name	statistics	augment.	overall	augment.	overall	augment	overall	augment	overall
cage15	min.	2.15	14.85	3.63	30.52	1.19	14.67	1.10	14.03
	avg.	2.41	15.22	3.80	30.88	1.39	14.95	1.28	15.28
	max.	2.68	15.85	4.01	31.08	1.69	15.32	1.69	16.59
Channel-500	min.	1.57	11.75	2.83	23.47	1.63	12.03	0.04	6.68
	avg.	1.66	11.81	2.92	23.86	1.75	12.16	0.06	7.28
	max.	1.70	11.85	3.01	24.88	2.02	12.44	0.08	7.92
Circuit5M	min.	116.67	123.24	107.51	132.34	2.02	8.89	0.74	5.26
	avg.	332.29	339.05	235.54	260.47	37.11	44.51	5.37	10.40
	max.	559.09	566.09	378.31	403.12	139.61	148.58	18.30	24.78
Delaunay_24	min.	40.52	52.15	32.09	98.89	41.66	69.52	48.63	64.32
	avg.	45.48	57.09	36.90	105.20	46.94	74.96	52.48	68.23
	max.	52.47	64.06	43.74	110.18	53.19	81.04	58.07	73.91
Hugebub-20	min.	41.01	56.16	55.22	146.78	44.71	81.96	49.46	70.34
	avg.	47.53	62.58	58.56	156.33	51.59	89.15	53.16	73.84
	max.	52.59	67.56	61.17	166.57	58.54	96.15	54.82	75.36
nlpkkt240	min.	13.98	112.59	22.87	205.18	15.49	97.63	19.74	84.26
	avg.	14.13	112.80	24.17	207.27	17.34	99.79	28.70	93.13
	max.	14.51	113.27	25.77	211.10	19.01	101.46	47.31	112.28

Our experimental results show that these approaches obtain near perfect matchings in real-life and synthetic instances and have a near linear time run time. The two approaches are also shown to be more robust than the state of the art heuristics used in the cardinality matching algorithms, and are generally more useful as initialization routines.

Our adaptation of 2outMC is based on the premise that 2-out graphs sampled from a host graph have perfect matchings, assuming that the matrix representation of the host graph have total support. We showed evidence that this may be true and even if not, the sampled graphs have close to perfect matchings. A proof or the disproof of such 2-out graphs having perfect matchings is certainly welcome. Furthermore, this was the first attempt to implement 2outMC, and there is room for improved performance.

References

1 Z. Allen-Zhu, Y. Li, R. Mendes de Oliveira, and A. Wigderson. Much faster algorithms for matrix scaling. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 890–901, Berkeley, CA, USA, October 2017.

2 J. Aronson, M. Dyer, A. Frieze, and S. Suen. Randomized greedy matching II. *Random Structures & Algorithms*, 6(1):55–73, 1995.

3 S. Assadi, M. Bateni, A. Bernstein, V. Mirrokni, and C. Stein. Coresets meet edcs: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1616–1635. SIAM, 2019.

4 S. Assadi and A. Bernstein. Towards a unified theory of sparsification for matching problems. *arXiv preprint*, 2018. `arXiv:1811.02009`.

5 S. Behnezhad, S. Brandt, M. Derakhshan, M. Fischer, M. Hajiaghayi, R.M. Karp, and J. Uitto. Massively parallel computation of matching and mis in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 481–490, 2019.

6 S. Behnezhad, J. Łącki, and V. Mirrokni. Fully dynamic matching: Beating 2-approximation in δ^{ϵ} update time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2492–2508. SIAM, 2020.

7 C. Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the USA*, 43:842–844, 1957.

8 A. Bernstein and C. Stein. Fully dynamic matching in bipartite graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 167–179. Springer, 2015.

9 B. Besser and M. Poloczek. Greedy matching: Guarantees and limitations. *Algorithmica*, 77(1):201–234, 2017.

10 M. B. Cohen, A. Madry, D. Tsipras, and A. Vladu. Matrix scaling and balancing via box constrained newton's method and interior point methods. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 902–913, Berkeley, CA, USA, October 2017.

11 A. Czumaj, J. Łącki, A. Madry, S. Mitrovic, K. Onak, and P. Sankowski. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 471–484. Association for Computing Machinery, 2018.

12 T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, 2011.

13 I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Transactions on Mathematical Software*, 7(3):315–330, 1981.

14 I. S. Duff, K. Kaya, and B. Uçar. Design, implementation, and analysis of maximum transversal algorithms. *ACM Transactions on Mathematical Software*, 38:13:1–13:31, 2011.

15 F. Dufossé, K. Kaya, I. Panagiotas, and B. Uçar. Approximation algorithms for maximum matchings in undirected graphs. In *2018 Proceedings of the Seventh SIAM Workshop on Combinatorial Scientific Computing*, pages 56–65, 2018.

16 F. Dufossé, K. Kaya, and B. Uçar. Two approximation algorithms for bipartite matching on multicore architectures. *Journal of Parallel and Distributed Computing*, 85:62–78, 2015.

17 A. Frieze and T. Johansson. On random k-out subgraphs of large graphs. *Random Structures & Algorithms*, 50(2):143–157, 2017.

18 A. Goel, M. Kapralov, and S. Khanna. Perfect matchings in $O(n \log n)$ time in regular bipartite graphs. *SIAM Journal on Computing*, 42(3):1392–1404, 2013.

19 A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

20 T. Hagerup, K. Mehlhorn, and J. I. Munro. Maintaining discrete probability distributions optimally. In A. Lingas, R. Karlsson, and S. Carlsson, editors, *20th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 253–264, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

21 J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

22 R. M. Karp, A. H. G. Rinnooy Kan, and R. V. Vohra. Average case analysis of a heuristic for the assignment problem. *Mathematics of Operations Research*, 19(3):513–522, 1994.

23 R. M. Karp and M. Sipser. Maximum matching in sparse random graphs. In *22nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 364–375, Los Alamitos, CA, USA, 1981. IEEE Computer Society.

24 R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, STOC '90, pages 352–358, New York, NY, USA, 1990. ACM.

25 K. Kaya, J. Langguth, F. Manne, and B. Uçar. Push-relabel based algorithms for the maximum transversal problem. *Computers & Operations Research*, 40(5):1266–1275, 2013.

26 K. Kaya, J. Langguth, I. Panagiotas, and B. Uçar. Karp–Sipser based kernels for bipartite graph matching. In *SIAM Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 134–145, Salt Lake City, Utah, US, January 2020.

27 P. A. Knight. The Sinkhorn–Knopp algorithm: Convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 30(1):261–275, 2008.

28 P. A. Knight and D. Ruiz. A fast algorithm for matrix balancing. *IMA Journal of Numerical Analysis*, 33(3):1029–1047, 2013.

29 V. Korenwein, A. Nichterlein, R. Niedermeier, and P. Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. In *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112, pages 53:1–53:13, Dagstuhl, Germany, 2018.

30 J. Langguth, F. Manne, and P. Sanders. Heuristic initialization for bipartite matching problems. *Journal of Experimental Algorithmics (JEA)*, 15:1–22, 2010.

31 J. Magun. Greedy matching algorithms, an experimental study. *Journal of Experimental Algorithmics*, 3:6, 1998.

32 Y. Matias, J. S. Vitter, and W.-C. Ni. Dynamic generation of discrete random variates. *Theory of Computing Systems*, 36(4):329–358, 2003.

33 N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 7:188–201, 1999.

34 M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 1st edition, 2005.

35 M. Poloczek and M. Szegedy. Randomized greedy algorithms for the maximum matching problem with new analysis. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 708–717, 2012.

36 A. Pothen and C.-J. Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16(4):303–324, 1990.

37 A. Pothen, S. M. Ferdous, and F. Manne. Approximation algorithms in combinatorial scientific computing. *Acta Numerica*, 28:541–633, 2019.

38 R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

39 G. Tinhofer. A probabilistic analysis of some greedy cardinality matching algorithms. *Annals of Operations Research*, 1(3):239–254, 1984.

40 D. Walkup. Matchings in random regular bipartite digraphs. *Discrete Mathematics*, 31(1):59–64, 1980.

A Other heuristics for bipartite matching and recent work

In the main text, we compared the proposed heuristics with KaSi and KaSi2. There are a few other effective heuristics, which we briefly review here (see a recent survey [37]).

Hopcroft and Karp's original algorithm [21] proceeds in phases. At each phase, it finds shortest augmenting paths, and augments the current matching along a maximal set of disjoint such paths, where each phase runs in $O(n+m)$ time. Stopping when the shortest augmenting paths is of length $2k+1$ at a phase no larger than k results in an $1-1/(k+1)$ approximate matching in $O(k(m+n))$ time in the worst case. Greedy [39] chooses a random edge and matches the two endpoints and discards both vertices and the edges incident on them. Modified Greedy [39] chooses a free vertex and then randomly matches it to one of the available neighbors. MinGreedy [39] (see also Magun [31] and Langguth et al. [30] for related algorithms) improves upon Modified Greedy by selecting a random vertex with the minimum degree at the first step. The Greedy-like algorithms obtain maximal matchings and therefore are $1/2$ approximate. Slight improvements in the form of $1/2+\varepsilon$ are shown for these algorithms [2, 35], but there are theoretical bounds in the same vicinity [9]. Duff et al. [14] and Langguth et al. [25, 30] compare these algorithms for initialization in maximum cardinality matching algorithms and suggest using KaSi as initialization for general problems especially with the push-relabel based algorithms.

Another class of heuristics use randomization for breaking the 1/2 barrier. RANKING [24] algorithm achieves an approximation ratio of $1 - 1/e$, where e is the base of the natural logarithm. The same approximation ration is also achieved by a very simple parallel algorithm [16] whose most involved step is the application of a matrix scaling algorithm. This last paper also proposes an algorithm based on sampling 1-out subgraphs of a general bipartite graph (as we did in this paper) to obtain matchings of size about 0.86 times the maximum cardinality.

Matching has stirred some recent interest in the theoretical computer science community, with works focusing on parallel and distributed settings [4, 5, 11, 3] or on the fully dynamic version [6, 8] among others. Among the recent work, a method by Assadi et al. [4] shares similarities with the 2OUTMC algorithm. Their approach similarly sparsifies a given graph G to produce a subgraph with some approximation guarantees for the maximum cardinality matching. A detailed experimentation with this sparsification approach will reveal useful.

B Further comments on 2outMC

As demonstrated in the experiments in Section 4, 2OUTMC obtains matchings of very high cardinality. We can improve its matching quality by the following two heuristics. These two heuristics are not used in the given experiments. We plan to improve their run time.

B.1 Heuristic 1: Delayed tree vertex selection during Line 5

The ideal case at Line 5 of Algorithm 1 is to select an x such that x's insertion as an edge to H_2 does not lead to a new tree in H_1 after the deletion of the edge corresponding to the unchecked vertex of the connected component Q_x. This is only possible if Q_x contains an unchecked column labeled as $\mathcal{C}$ in H_1. Otherwise, a new tree will be created in H_1, and the algorithm will have to process it in a future step. For the first heuristic, we greedily select an x such that, if possible, the creation of a tree in H_1 is avoided.

We replace L_T is with two lists L_T^1 and L_T^2. The lists L_T^1 contains those unmarked vertices of T whose insertion in H_2 leads to a new tree; L_T^2 contains all other L_T vertices that have not been tried yet. At first, we sample x from L_T^2 and see whether the components of x's choices in H_2 have an unchecked vertex of type $\mathcal{C}$ in H_1. If they have, x is marked and inserted to H_2. Otherwise, x is inserted in L_T^1, and we consider another random vertex of L_T^2. If L_T^2 becomes empty, we start sampling from L_T^1.

With the union-find data structure, this heuristic requires constant amortized time per sample and each vertex can be sampled at most twice. Therefore the overhead associated with this heuristic is almost linear in n.

B.2 Heuristic 2: Online creation of the RG multigraph

In this heuristic, the decisions of the rows are not given as input, but are instead defined during the course of the algorithm. Similar to the previous idea, this heuristic aims to reduce the possibility that a tree in H_1 gets created following an edge insertion into H_2.

More specifically, consider a vertex x randomly chosen at Line 5. In this heuristic, x has not picked its two choices yet, and we let x choose them at this point, in the way that benefits the algorithm the most. This is done as follows. Initially, we iterate over all of x's neighbors in the host graph G. Let c be one of x's neighbors and c^* be the sole unchecked vertex in c's connected component in H_2, or $c^* = -1$ if no unchecked vertices exist. We assign values to x's neighbors to classify them. If c^* is equal to -1, c's value is 0. If c^* has

label $\mathcal{F}$ or $\mathcal{T}$ in H_1, c's value is 1. Otherwise, c's value is 2. Based on these assigned values, we partition the neighbors of x in G into three disjoint sets C_0, C_1 and C_2 such that C_i contains all neighbors of x with value equal to i. Selecting columns from C_2 is preferred, as they can avoid creating a tree in H_1. Vertex x will attempt to sample first from C_2, and if needed from C_1 or C_0, with a preference for C_1 over C_0. The sets C_0, C_1 and C_2 are kept implicitly, and each vertex x requires amortized $O(d_x)$ to make its choices, where d_x is its degree. Hence, the overhead associated with this heuristic is almost linear in m.

B.3 Comparison with 2outMC

Here, we briefly discuss the effects that the above two heuristics have on the performance of the 2OUTMC algorithm. Since 2OUTMC obtains high quality results, the two heuristics can only yield a relatively small improvement. When they are enabled and used with SK-5 2OUTMC finds matchings with average quality of 0.9997 for the real-world graphs from Section 4.3 for which 2OUTMC obtained matchings of quality 0.9983. This difference corresponds to about 13113 additionally matched edges, and hence signals that 13113 augmentations are avoided.

It is also interesting to consider the effects that these heuristics can have on cases where 2OUTMC did not deliver near-optimal matchings. As an example, we consider the synthetic family $\mathcal{J}$ from Section 4.2. When scaling was not enabled, 2OUTMC found matchings of average cardinality $0.80 - 0.81\%$ of the maximum. If however one uses the two heuristics proposed in this section, then there is a significant improvement in performance, and 2OUTMC finds matchings of cardinality 0.89 of the maximum.

C Further comments on TruncRW

We incorporated a known heuristic called look-ahead [13, 14] for speeding up the augmenting path search in practice. All our experiments with TRUNCRW in Section 4 were with the look-ahead approach. In this heuristic, before sampling an arbitrary row-vertex from a column-vertex c, we check if there is a free row vertex in the adjacency list of c. If so, such a row is returned, and the random walk terminates. The implementation of this heuristic has a total overhead of $O(m)$ for the whole course of the algorithm [13, 14]. We note that the look-ahead technique trades the quality of TRUNCRW with run time. In our experiments, the look-ahead heuristic reduced the run time significantly; it interferes with the randomization though.

We can easily apply TRUNCRW to bipartite graphs with different number of vertices in each side. This is based on the fact that we can scale a rectangular $n_1 \times n_2$ matrix (say $n_1 \geq n_2$) so that all columns have sum of 1, and all rows have equal sum of n_2/n_1, if there is matching covering all columns, and all entries can be put in such a matching. Then, all components of TRUNCRW work without any change.

If there is no total support, then Sinkhorn–Knopp works in such a way that the entries that cannot put into a perfect matching tend to zero. This is helpful in TRUNCRW's context, as the corresponding edges will not likely be selected in a random walk. If there is no perfect matching, then little is known about scaling. It is our experience that the Sinkhorn–Knopp iterations tend to zero out entries that cannot be put into a maximum cardinality matching. Therefore, in this case again, scaling, random selection, and truncation should help. We present some experiments to support this observation and leave the question of showing this theoretically as an open problem.

We experimented with bipartite graphs without total support which correspond to square (10000×10000) and rectangular matrices (12000×10000) with a uniform nonzero distribution. These matrices are generated with `sprand` command of Maltab and have about $d \times 10000$

Table 8 The quality of TruncRW on bipartite graphs without perfect matchings.

	10000 × 10000		12000 × 10000	
d	sprank	TruncRW	sprank	TruncRW
2	7787	0.9888	8724	0.9919
3	9266	0.9697	9667	0.9958
4	9761	0.9828	9899	0.9995
5	9918	0.9922	9973	1.0000

nonzeros for $d = 2, 3, 4, 5$. The matrix representation of the bipartite graphs were scaled with 10 iterations of SK. For each d, we created five random matrices and ran TruncRW on the corresponding five instances. We report the worst quality of the five instances in Table 8. As seen in this table, TruncRW works just fine for this case. We did not report in the table but with increased SK iterations, the results improve, which is in accordance with earlier work [16].

C.1 Engineering TruncRW

The experiments here are on real-life instances from Subsection 4.3 and with SK-5.

Recall that TruncRW tries to find an augmenting path starting from a column vertex a certain number of times before giving up and moving to the next column vertex. When we allowed TruncRW just a single attempt, it was unable to find a perfect matching in any of the cases, and its average matching quality was 0.9984. When we allowed five attempts, TruncRW found a perfect matching for 13 graphs, and its average matching quality was 0.9999. With 10 attempts, it managed to find a perfect matching in 5 additional graphs. This verifies that allowing more attempts indeed improves the performance of the algorithm. The drawback, however, was the increased run time, which we did not think worth. That is why our implementation of TruncRW starts a random walk from a vertex only once.

We also test the effects of the look-ahead mechanism. Let us define the walk efficiency of TruncRW as the ratio of the cardinality of the matching found to the total length of the random walks. The higher this ratio, the more useful the random walks are. We evaluate the walk efficiency on a set of seven instances (real-life instances having at most 10000000 edges). We test both with and without scaling and report the results of the 14 tests. In 13 cases, the look-ahead mechanism improved the walk efficiency. The geometric mean (of 14 cases) of the ratios of walk efficiencies with look-ahead to that of without was 1.37. In the case where the look-ahead did not help (ratio was 0.71 in an instance named `Hamrle3`), the maximum deviation of a row or column sum from one after SK-5 was 0.28, which is high. We conclude that the look-ahead mechanism is very helpful.

Finally we test the effects that the length of the augmenting walk has on TruncRW. We doubled the allowed length of a random walk to $4(4 + 2n/(n - j))$. On average, the matching quality rose from 0.9984 to 0.9998. This modification was not able to find a perfect matching in any of the 39 instances. This led to an increase in the run time, which we deemed too large. We therefore keep $2(4 + 2n/(n - j))$ as the truncation length.

D Reducing bipartite graph matching to matching on 2-out graphs

Here, we prove our claim in Section 3.1 that bipartite matching can be reduced to matching on a 2-out bipartite graph. Let $G = (V_G, E_G)$, with be a graph with minimum degree at least two. If G's minimum degree is one, we can apply the first deterministic rule of Karp–Sipser to match degree-1 vertices with their neighbors and consider as G the resulting graph.

We produce a new graph G' from G in the following way. For any edge $e = (a, b) \in E$ we add edges $e' = (a, a_e)$, $e'' = (a_e, b_e)$, and $e''' = (b_e, b)$ to G'. We hence introduce two new vertices a_e, b_e s.t $d_{G'}(a_e) = d_{G'} = 2$ for each edge $e \in E_G$. The degree of nodes in V_G remains unchanged in G'.

▶ **Lemma 2.** *Let H be a random 2-out subgraph G'. Then $H = G'$.*

Proof. The added vertices a_e, b_e have degree two and will select both neighbors, hence no edge will remain unpicked. ◀

In what follows, we refer to the second reduction rule of Karp–Sipser which merges the neighbors of a degree-2 vertex, which is then discarded, as a degree-2 reduction.

▶ **Lemma 3.** *It is possible to obtain G by doing only degree-2 reductions on G'.*

Proof. Let a_e be a vertex of G', introduced due to the edge $e = (a, b)$. Since $d_{G'}(a_e) = 2$ we can apply a degree-2 reduction which will merge a with b_e to create a single node ab_e. As a consequence of this merge, the edge (ab_e, b) will be created and edges $(a, a_e), (a_e, b_e), (b_e, b)$ will be erased. We simply relabel ab_e to a again. The proof then follows similarly by applying degree-2 reduction for all a_e corresponding to $e \in E$ until we obtain G. ◀

Now we show that maximum matchings in G' are related to those on G and vice versa.

▶ **Lemma 4.** *Any maximum cardinality matching M' on G' corresponds to a maximum cardinality matching M on G.*

Proof. Let M' be a maximum cardinality matching on G'. A matching M for G can be generated in the following way: If both (a, a_e) and (b_e, b) appear in M', e is added to M. Hence it suffices to show that any maximum cardinality matching M' in G' necessarily contains $|M|$ pair of matched edges (a, a_e) and (b, b_e).

First, we have that $|M'| = |E_G| + |M|$. To see this, note that per Lemma 2 we perform $|E_G|$ degree-2 reductions, and result in G. Each of this reductions corresponds with a matched edge in M'. Then, we only need to find the maximum cardinality on G which is $|M|$.

Let S_a contain all indices e such that (a, a_e) is in M' and (b_e, b) is not in M'. Set S_b is defined similarly. Set $S_\emptyset$ contains all indices e such that (a_e, b_e) appears in M'. Finally, S_{ab} contains all indices e such that (a, a_e) and (b, b_e) are matched together in M'. Then, since M' is a maximum cardinality matching we have

$$|S_a| + |S_b| + |S_\emptyset| + 2 \cdot |S_{ab}| = |E_G| + |M| \ .$$

This is true because of the fact that for each edge e exactly one matched edge appears in M' in case $e \in S_a \cup S_b \cup S_\emptyset$ and two edges are added if $e \in S_{ab}$.

However, $|S_a| + |S_b| + |S_\emptyset| + |S_{ab}| = |E_G|$, since each edge e must appear in one of those sets and there exist exactly $|E_G|$ of them.

Hence, $|S_{ab}| = |M|$ necessarily. As they define a matching in G and their cardinality is $|M|$, the matching is maximum. ◀

Using the above lemma, we can prove Theorem 5 below.

▶ **Theorem 5.** *Assume there is an algorithm* `ALG` *working in $O(f(n, m))$ time for finding a maximum cardinality matching in a 2-out graph. Then we can find a maximum cardinality matching in $O(f(m, m))$ time for any given graph.*

Proof. Let G be any bipartite graph without degree-1 vertices and $m = |E_G|$. In $O(m)$ time we generate G'. By Lemma 2, the 2-out subgraph of G' corresponds to G' itself. In addition $|E_{G'}|, |V_{G'}| \in O(m)$. Using `ALG`, we can find a maximum cardinality M' for G' in $O(f(m, m))$ time. By Lemma 4 then, we can convert M' to a maximum cardinality matching M for G in $O(m)$ time. ◀

As a byproduct of Lemma 4, we observe that the transformation of G to G' also eliminates the need to perform SK as a preprocessing step. We briefly experimented with this method on the real-world graphs of Section 4.3. For each graph G of the test-set, we generated its extension G' and executed the 2OUTMC algorithm on 2-out graphs sampled from G', with uniform selections. The behavior of 2OUTMC was similar with that of the previous experiments. It was not able to obtain a perfect matching in G' (and consequently G), but it always returned near-optimal matchings of quality over 0.99. These matchings, when converted into matchings of G (following the idea in Lemma 4) yielded also near-optimal matchings with quality over 0.99.

Efficient Computation of 2-Covers of a String

Jakub Radoszewski
Institute of Informatics, University of Warsaw, Poland
Samsung R&D Poland, Warsaw, Poland
jrad@mimuw.edu.pl

Juliusz Straszyński
Institute of Informatics, University of Warsaw, Poland
jks@mimuw.edu.pl

Abstract

Quasiperiodicity is a generalization of periodicity that has been researched for almost 30 years. The notion of cover is the classic variant of quasiperiodicity. A cover of a text T is a string whose occurrences in T cover all positions of T. There are several algorithms computing covers of a text in linear time. In this paper we consider a natural extension of cover. For a text T, we call a pair of strings a 2-cover if they have the same length and their occurrences cover the text T. We give an algorithm that computes all 2-covers of a string of length n in $\mathcal{O}(n \log n \log\log n + \mathsf{output})$ expected time or $\mathcal{O}(n \log n \log^2 \log n / \log\log\log n + \mathsf{output})$ worst-case time, where output is the size of output.

If (X, Y) is a 2-cover of T, then either X is a prefix and Y is a suffix of T, in which case we call (X, Y) a ps-cover, or one of X, Y is a border (that is, both a prefix and a suffix) of T, and then we call (X, Y) a b-cover. A string of length n has up to n ps-covers; we show an algorithm that computes all of them in $\mathcal{O}(n \log\log n)$ expected time or $\mathcal{O}(n \log^2 \log n / \log\log\log n)$ worst-case time. A string of length n can have $\Theta(n^2)$ non-trivial b-covers; our algorithm can report one b-cover per length (if it exists) or all shortest b-covers in $\mathcal{O}(n \log n \log\log n)$ expected time or $\mathcal{O}(n \log n \log^2 \log n / \log\log\log n)$ worst-case time. All our algorithms use linear space.

The problem in scope can be generalized to $\lambda > 2$ equal-length strings, resulting in the notion of λ-cover. Cole et al. (2005) showed that the λ-cover problem is NP-complete. Our algorithms generalize to λ-covers, with (the first component of) the algorithm's complexity multiplied by $n^{\lambda-2}$.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases quasiperiodicity, cover of a string, 2-cover, lambda-cover

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.77

Funding Supported by the "Algorithms for text processing with errors and uncertainties" project carried out within the HOMING program of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund, project no. POIR.04.04.00-00-24BA/16, and by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Acknowledgements The authors thank Patryk Czajka for helpful discussions on the initial version of the algorithm.

1 Introduction

Identifying repetitive structure of a string is one of the key research areas of text algorithms, with applications to computational biology; see e.g. the books [19, 28]. Processing of a string that has a regular structure can be performed more efficiently, be it for pattern matching or for data compression.

The most elementary notion that grasps repetitiveness is *periodicity*. If a string can be generated by repeated concatenation of its smaller piece, then we say that it is periodic. The field of periodicity has been expanded upon by allowing not only concatenation, but also superpositions, which resulted in the introduction of *quasiperiodicity* by Apostolico and Ehrenfeucht [6].

The basic terms of quasiperiodicity are the notions of *cover* and *seed*. A cover of a text T is a string whose occurrences in T cover all positions of T, while a seed of T is a cover of some superstring of T. An $\mathcal{O}(n)$-time algorithm for computing the shortest cover of a text of length n was presented by Apostolico et al. [7]. Moore and Smyth showed that all the covers of a string can be computed in $\mathcal{O}(n)$ time [43, 44, 45]. Moreover, $\mathcal{O}(n)$-time algorithms for computing covers of all prefixes of a string were shown [13, 41]. Seeds were introduced by Iliopoulos et al. [33] who showed an algorithm for finding a representation of all seeds of a string in $\mathcal{O}(n \log n)$ time. The majority of these classic algorithms were developed in the 1990s. It was not until many years later that an $\mathcal{O}(n)$-time algorithm for computing seeds was found [36, 37]. Various approximate variants of covers and seeds were studied – see e.g. [3, 4, 16, 25, 38, 39] – as well as covers in other models of computation [10, 14, 26], in non-standard stringology [1, 9, 20, 31, 32] and in 2-dimensional texts [21, 47].

We consider *2-covers* which are a natural generalization of covers. A 2-cover of a text T is a pair of equal-length strings whose occurrences in T cover all positions of T; see Figure 1. A yet more general notion of a λ-*cover*, that is, a λ-tuple of strings whose occurrences cover the whole text T, was introduced by Iliopoulos et al. in [27, 49]. Unfortunately, the authors' $\mathcal{O}(n^2)$ time algorithm for finding all λ-covers under fixed λ and constant-size alphabet has been proven to be faulty. In reality, the algorithm has, at worst, exponential runtime [23].

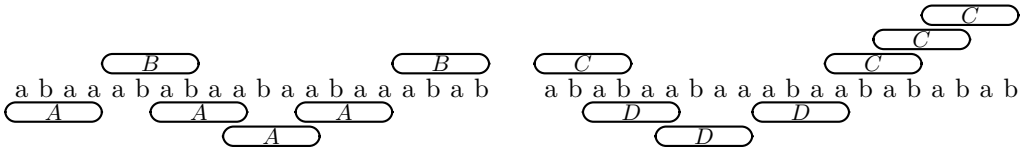


Figure 1 Two examples of a 2-cover of a string: a ps-cover (left) and a b-cover (right). Note that none of these strings has a proper cover.

In this paper, we present an $\mathcal{O}(n \log n \log\log n + \mathtt{output})$ time algorithm for finding all 2-covers of a text of length n. Each string from a 2-cover in the output is represented by giving endpoints of its sample occurrence. Our algorithm can compute a 2-cover of each length or all shortest 2-covers in $\mathcal{O}(n \log n \log\log n)$ time. The complexities show the expected running time of the algorithms; they can be made worst-case at a cost of an additional $\log\log n / \log\log\log n$ factor. The space complexities of the algorithms are $\mathcal{O}(n)$. We assume the standard word-RAM model of computation.

In the case of previously mentioned seeds and covers, the input text is generated by concatenations and superpositions of a single string. However, in our problem, we need to check if the text can be generated by two strings of equal length. This alone suggests that the problem is computationally harder than its original counterpart. Intuitively, to find all covers of a string we need to check only $\mathcal{O}(n)$ candidates, i.e. all prefixes. This is not the case with 2-covers, because a text of length n can have up to $\Theta(n^2)$ different non-trivial 2-covers. (A simple example $T = \mathtt{a}^m\mathtt{ba}^m\mathtt{ba}^m$ of such a text was shown in [23].) The general λ-cover problem was shown to be NP-hard by Cole et al. [17].

There are two types of 2-cover of a text T, as shown in Figure 1: a *ps-cover* (X, Y) that is composed of a prefix X and a suffix Y of T and a *b-cover* (X, Y) in which one of the strings, let us say X, is a border of T. (2-covers (X, Y) in which X is actually a cover of T are considered to be trivial and can be ignored.) Our main result consists of two algorithms, one for each of the types.

The first algorithm finds ps-covers. This is the easier type and for it, we propose an $\mathcal{O}(n \log\log n)$ expected time algorithm. It iterates over all possible candidates (there are $\mathcal{O}(n)$ of them) and maintains a set of *gaps*, that is, parts of the text that are not covered yet. There, we exploit locality of changes in coverage between consecutive lengths by using a predecessor data structure [5, 48]. Secondly, we efficiently express the dynamics of the gaps by storing linear functions.

The remaining, harder algorithm, finds b-covers (X, Y). In this case there are significantly more candidates to consider (up to $\mathcal{O}(n^2)$ [23]). For each length ℓ we use string periodicity to compute a set of $\mathcal{O}(n/\ell)$ positions in T, called *anchors*, that implies all non-redundant occurrences of any string Y in a b-cover of length ℓ. This set is computed using Internal Pattern Matching [40]. Finally our algorithm forms a set of constraints on Y based on the anchors and finds all strings that satisfy these constraints in $\mathcal{O}(n \log\log n/\ell + \mathsf{output})$ expected time using predecessor queries.

Our algorithms easily generalize to the λ-covers problem, achieving $\mathcal{O}(n^{\lambda-1}\mathrm{polylog}\, n + \mathsf{output})$ time.

▶ Remark 1. "String cover" is also used to describe a different notion that should not be confused with the one studied in this work. Namely, a string cover C of a set of strings S is a set of factors of strings from S such that every string in S can be written as a concatenation of the strings in C; see [12, 15, 30, 46].

2 Preliminaries

By $[i\,..\,j]$ we denote the integer interval $\{i, \ldots, j\}$; we use a round bracket if the interval does not contain one of its ends. For a set S of integers and integer a, by $S \oplus a$ and $S \ominus a$ we denote the sets $\{s + a : s \in S\}$ and $\{s - a : s \in S\}$, respectively, and by $\mathit{intervals}_k(S)$ we denote the set $\{[i\,..\,i + k) : i \in S\}$.

A string T is a sequence of letters from a given alphabet. The length of string T is denoted by $|T|$. We assume that the positions in T are numbered 1 through $|T|$, with letter at position i denoted as $T[i]$. By $T[i\,..\,j]$ we denote the string $T[i] \ldots T[j]$ that is called a *factor* of T (the same notation is used for open intervals of positions). A factor $T[i\,..\,j]$ is called a *prefix* if $i = 1$ and a *suffix* if $j = |T|$.

For a string X, by $\mathit{Occ}_T(X)$ we denote the set of starting positions of occurrences of X in T and by $\mathit{Cov}_T(X)$ the set of positions that are covered by occurrences of X in T, i.e.,

$$\mathit{Cov}_T(X) = \bigcup \mathit{intervals}_{|X|}(\mathit{Occ}_T(X)).$$

We omit the subscript T when it is clear from the context. We say that a set of strings $\mathcal{S}$ is a *λ-cover of length ℓ* of T if the following conditions hold:

- $|\mathcal{S}| = \lambda$
- $|X| = \ell$ for all $X \in \mathcal{S}$
- $\bigcup_{X \in \mathcal{S}} \mathit{Cov}(X) = [1\,..\,|T|]$

Periodicity of strings. We say that string S has *period* p (for $p \in [1\,..\,|S|]$) if $S[i] = S[i + p]$ for all $i \in [1\,..\,|S| - p]$.

▶ **Fact 2** (Periodicity lemma; Fine and Wilf [24]). *If string S has periods p and q such that $p + q \le |S|$, then it has a period* $\gcd(p, q)$.

A string is called *periodic* if it has a period that is at most a half of its length and *aperiodic* otherwise. Moreover, a string is called *$\frac{1}{4}$-periodic* if it has a period that is at most a quarter of its length.

▶ **Fact 3** (Folklore; see [2]). *If S is periodic and S' is a string of length $|S|$ that differs from S at exactly one position, then S' is aperiodic.*

In particular, if S is periodic with smallest period p and the letter c is different from $S[|S|-p+1]$, then Sc, i.e., S concatenated with c, is aperiodic.

String B is called a *border* of string S if B is a prefix and a suffix of S. String S has a period p if and only if it has a border of length $n-p$. In particular, this implies the following.

▶ **Observation 4.** *If string S is not periodic, then $|Occ_T(S)| = \mathcal{O}(|T|/|S|)$.*

A string S is *primitive* if $S = V^k$ for a string V and positive integer k implies that $k = 1$.

▶ **Fact 5** (Synchronization property; [19, Lemma 1.11]). *A primitive string S has exactly two occurrences in S^2.*

PREF table. The table $PREF$ over a length-n string T stores, as $PREF[i]$, the length of the longest common prefix of T and $T[i\,..\,n]$. Let $PREF^R[i]$ denote the length of the longest common suffix of T and $T[1\,..\,i]$. Both arrays can be computed in $\mathcal{O}(n)$ time by a classical comparison-based algorithm, as in the Main-Lorentz algorithm [42]; see also the book [22].

Longest Common Extension (LCE) queries. Assume that string T is over an integer alphabet $[1\,..\,n^{\mathcal{O}(1)}]$. A longest common prefix (longest common suffix) query on T, given indices $i, j \in [1\,..\,n]$, returns the length of the longest common prefix of suffixes $T[i\,..\,n]$ and $T[j\,..\,n]$ (the length of the longest common suffix of $T[1\,..\,i]$ and $T[1\,..\,j]$, respectively). Both types of queries are often referred to as LCE queries. It is well-known that after $\mathcal{O}(n)$-time preprocessing, one can answer LCE queries for T in $\mathcal{O}(1)$ time using the suffix array [34] and range minimum queries [11]. Moreover, we use the inverse suffix array that gives, for each suffix, its position in the sorted list of suffixes.

Assume that $T[i\,..\,j]$ is periodic with smallest period p. A position $j' > j$ ($i' < i$) is said to *break the periodicity* of $T[i\,..\,j]$ if $j' = \min\{k > j \,:\, T[k] \neq T[k-p]\}$ ($i' = \max\{k < i \,:\, T[k] \neq T[k+p]\}$, respectively). We set $i' = 0$ and $j' = n+1$ if the respective position does not exist. One can use LCE queries to compute the positions breaking periodicity of a given factor $T[i\,..\,j]$, if they exist, in $\mathcal{O}(1)$ time.

Internal Pattern Matching (IPM) queries. Again assume that T is over an integer alphabet $[0\,..\,n^{\mathcal{O}(1)}]$. The IPM problem requires one to preprocess a text T of length n so that one can efficiently compute the occurrences of a factor of T in another factor of T. An $\mathcal{O}(n)$-sized data structure, with $\mathcal{O}(n)$ expected time construction, that answers IPM queries in $\mathcal{O}(1)$ time when the ratio between the lengths of the two factors is at most 2 was presented in [40]. The set of occurrences is returned as a single arithmetic sequence. Moreover, if the sequence contains at least three elements, then its difference equals the smallest period of the pattern factor. A deterministic version of this data structure can be found in [35]. This data structure can also be used to answer in $\mathcal{O}(1)$ time so-called *two-period queries*, in which we are asked to find the smallest period of a given factor of T if this factor is periodic (an alternative data structure was proposed in [8]).

Predecessor data structures. For a set of integers A, by $pred(x, A)$ and $succ(x, A)$ we denote the predecessor and successor of x in A, that is, $\max\{a \in A \,:\, a < x\}$ and $\min\{a \in A \,:\, a > x\}$, respectively. (We assume that $\max\emptyset = -\infty$ and $\min\emptyset = \infty$.) We use the following known efficient dynamic predecessor data structures. A collection $A \subseteq [1\,..\,n]$ can be maintained

under insertions and deletions and can answer predecessor and successor queries in $\mathcal{O}(\log\log n)$ expected time per operation using a y-fast trie [48] or in $\mathcal{O}(\log^2 \log n/\log\log\log n)$ worst-case time using an exponential search tree [5]. Below by τ_n we denote the time complexity of an operation on a predecessor data structure. Moreover, we use Han's deterministic algorithm [29] to sort n numbers in $\mathcal{O}(n \log\log n)$ time.

3 Computing ps-covers

Let T be a string of length n. Let us start with a simpler but less efficient approach for computing ps-covers. For each length ℓ we would like to check if there is a ps-cover (X, Y) of length ℓ of T. We aim at $\mathcal{O}(n/\ell)$ time complexity after linear-time preprocessing. In the preprocessing phase we compute the data structures for LCE-queries [11, 34] and IPM queries [35, 40] in T. If T has a ps-cover (X, Y) of length ℓ, then $X = T[1\,..\,\ell]$ and $Y = T[n-\ell+1\,..\,n]$. We apply IPM queries to compute the sets of occurrences $Occ(X)$ and $Occ(Y)$, represented as unions of $\mathcal{O}(n/\ell)$ of arithmetic sequences, in $\mathcal{O}(n/\ell)$ time. This lets us compute the sets $Cov(X)$ and $Cov(Y)$, represented as unions of $\mathcal{O}(n/\ell)$ maximal intervals, sorted left-to-right. Then we need to check if $Cov(X) \cup Cov(Y) = [1\,..\,n]$, which can be done in linear time w.r.t. to the sizes of the representations of these sets by merging the sorted lists of intervals. Thus we have shown the following result.

▶ **Lemma 6.** *Let T be a string of length n over an integer alphabet. After $\mathcal{O}(n)$-time and space preprocessing, one can compute a ps-cover of T of a given length ℓ, if it exists, in $\mathcal{O}(n/\ell)$ time.*

Let us note that Lemma 6 applied for all lengths $\ell = 1, \ldots, n$ allows us to compute all ps-covers in $\mathcal{O}(n \log n)$ time. However, there is a more efficient approach that does not involve the intricate technique of IPM queries and also works for strings over any alphabet. We will use the lemma when computing λ-covers in Section 5.

Let P_ℓ be the prefix of length ℓ of T and S_ℓ be the suffix of length ℓ of T. For each length ℓ there is only one candidate for a ps-cover, that is, (P_ℓ, S_ℓ). Furthermore, the set of positions of the text T that are covered by $Cov(P_\ell) \cup Cov(S_\ell)$ does not change much when ℓ is incremented.

The idea is to iterate over increasing values of ℓ and check whether occurrences of P_ℓ and S_ℓ cover the entire text. We are going to maintain a set of *gaps*, that is, parts of the text that are covered by occurrences of neither the prefix nor the suffix.

First, let us identify an occurrence of a prefix P_ℓ with the index of its leftmost character and an occurrence of a suffix S_ℓ with the index of its *rightmost* character. In this way, when the length ℓ is incremented, some occurrences persist and get their length increased by one and other occurrences disappear. Specifically, occurrences of the prefix extend to the right and, respectively, occurrences of the suffix extend to the left. As a result, some gaps shrink or disappear and some other gaps are created. For an example, see Figure 2. Because of the way how joint occurrences of the prefix and the suffix affect the sizes of gaps, we will refer to these occurrences as the *pressing factors*.

We will iterate over subsequent $\ell = 1, \ldots, n$ and observe the set of gaps. If for some ℓ the set of gaps is empty, then (P_ℓ, S_ℓ) is a ps-cover. We track the following data:

- length ℓ
- the set $\triangleright_\ell$ of left endpoints of occurrences of P_ℓ
- the set $\triangleleft_\ell$ of right endpoints of occurrences of S_ℓ
- a set of pairwise disjoint gaps and an expiration time (value of ℓ) for each of them.

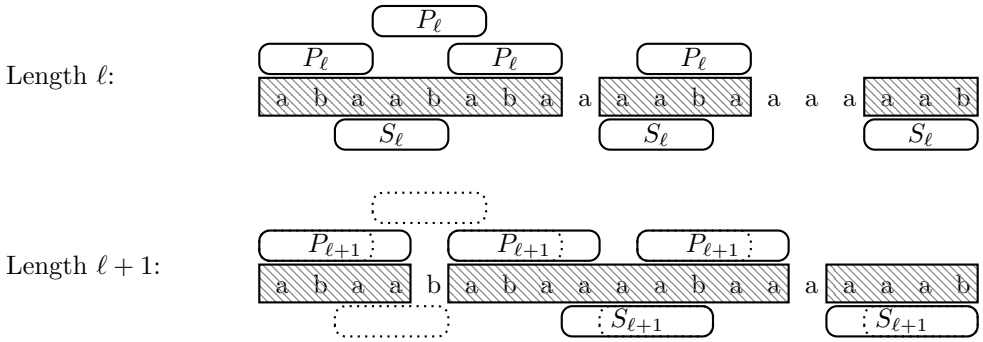


Figure 2 Illustration of gap dynamics. After incrementation of ℓ, a gap `b` was created, a gap `a` disappeared, and a gap `aaa` shrunk to `a`.

The sets will be maintained using predecessor data structures, which allow to perform predecessor/successor queries in τ_n time. Using the aforementioned data, the outline of the algorithm is as follows:

Algorithm 1 Outline of the algorithm for computing ps-covers.

```
pressing_factors := Occurrences of P_1 and S_1 in T;
gaps := Gaps between pressing_factors;
for ℓ := 1 to n do
    to_remove := Expired pressing_factors;
    Remove to_remove from pressing_factors;
    foreach expired_factor in to_remove do
        Recalculate elements of gaps around expired_factor;
```

An occurrence $i \in \triangleright_\ell$ ($i \in \triangleleft_\ell$) persists as long as $\ell \leq PREF[i]$ ($\ell \leq PREF^R[i]$, respectively). Therefore, for $\ell = PREF[i] + 1$ ($\ell = PREF^R[i] + 1$), we consider that occurrence as expired. In conclusion, the $PREF$ arrays allow us to compute expiration times of every prefix and suffix. This allows us to efficiently compute expired pressing factors in amortized $\mathcal{O}(1)$ time by precomputing a list of factors to expire for each moment of time in $\mathcal{O}(n)$ time.

Now let us simulate gap dynamics. Incrementations of ℓ successively get a gap increasingly covered (by occurrences of a prefix and/or suffix) until it expires completely. Assuming that none of the relevant pressing factors disappears, a gap expiration depends on the closest prefix occurrence to the left and the closest suffix occurrence to the right of the gap. If we know that some position p belongs to a gap, we would like to know the following:

- $L_\triangleright = \max\{a : a \in \triangleright_\ell, a < p\}$ and $L_\triangleleft = \max\{a : a \in \triangleleft_\ell, a < p\}$
- $R_\triangleright = \min\{b : b \in \triangleright_\ell, b > p\}$ and $R_\triangleleft = \min\{b : b \in \triangleleft_\ell, b > p\}$.

Unfortunately, this is too much to maintain. One factor that expires might influence many gaps. Let us analyze it further. Let us fix some prefix occurrence, i.e. pressing factor that extends to the right. It might influence expiration time of many gaps to the right. On the other, hand we can safely note this exclusively in the closest gap to the right. This is because the pressing factor won't reach other gaps before closing the immediate gap. When the gap closes, we can propagate the information to neighbouring gaps. Therefore, in a gap we only take into consideration pressing factors whose immediate neighbour is this gap and ignore them otherwise. We can easily check for this and compute all these values in τ_n time. If the gap initially covers the interval $[i\,..\,j]$, then it can expire in two ways:

- it can close on one boundary by a single opposing pressing factor, so the gap will close no later than $\ell = \min(R_\triangleleft - i + 1, j - L_\triangleright + 1)$, or

- it can close in the middle of the gap, by both pressing factors simultaneously, at $\ell = \lceil \frac{R_\triangleleft - L_\triangleright + 1}{2} \rceil$.

  The endpoints of a gap at moment ℓ can be computed using the formulas:

  $i = \max(L_\triangleright + \ell,\, L_\triangleleft + 1)$ and $j = \min(R_\triangleleft - \ell,\, R_\triangleright - 1)$.

When a gap is created or its neighbouring pressing factors are altered, we use these formulas to recompute the gap boundaries. The predecessor data structure that stores gaps uses, for each gap, its recently computed left boundary for comparison. It is sufficient since the left-to-right order of gaps is never changed.

Thus we can recompute the expiration moment of a single gap given at least one position belonging to the gap. The remaining issue is to know which gaps need to be updated. Note that each expired factor can affect at most two existing neighbouring gaps and possibly introduce a new one. We can find the neighbouring gaps via predecessor/successor queries. Positions that were not covered will still not be covered after removing the expired factor, so we can pick an arbitrary position from this gap and recalculate its boundaries.

Now, we need to check if some new gap was created in the boundaries of the expired factor. In this case we have some intervals of length ℓ, representing the set $Cov(P_\ell) \cup Cov(S_\ell)$, and we would like to know if removing one interval creates a gap in coverage. Thanks to the fact that all intervals are of the same length, if the expired factor is $[i\,..\,j]$, we only need to find the last interval ending at most at j and the first interval starting at least at i. If found intervals do not cover the entirety of $[i\,..\,j]$, we have at least one position of the gap and we are able to calculate its boundaries. Otherwise, removing the factor did not change the coverage, so no new gap was created. All of this can be performed using the predecessor data structures in τ_n time.

In conclusion, the entire computation of ps-covers takes $\mathcal{O}(n\tau_n)$ time and $\mathcal{O}(n)$ space. We obtain the following result.

▶ **Theorem 7.** *Let T be a string of length n over any alphabet that allows $\mathcal{O}(1)$-time checking of letter equality. One can compute a ps-cover of T of every possible length in $\mathcal{O}(n\tau_n)$ time and $\mathcal{O}(n)$ space.*

4 Computing b-covers

Let T be a string of length n. Our goal in this section is, given a length ℓ, to check if there is a b-cover (X, Y) of length ℓ of T. We aim at $\mathcal{O}(n\tau_n/\ell)$ time complexity after linear-time preprocessing. In the preprocessing phase we compute the data structures for LCE-queries [11, 34] and IPM queries [35, 40] in T.

Let $X = T[1\,..\,\ell]$. We apply IPM queries to compute the set $Occ(X)$, represented as a union of $\mathcal{O}(n/\ell)$ of arithmetic sequences, and the set $Cov(X)$, represented as a union of $\mathcal{O}(n/\ell)$ maximal intervals, in $\mathcal{O}(n/\ell)$ time. If $n - \ell + 1 \notin Occ(X)$, there is no b-cover of length ℓ, and if $Cov(X) = [1\,..\,n]$, we skip this length since we have the trivial case of a 2-cover containing a cover. Henceforth we assume that X is a border of T whose occurrences do not cover the whole string T.

Our goal is to find all strings Y for which (X, Y) is a b-cover of T. We start by building up some intuition. We have $|Y| = \ell$, so in order for Y to cover all positions from the set $Cov(Y)$, it suffices to use $\mathcal{O}(n/\ell)$ occurrences of Y (instead of, potentially, $\Theta(n)$ occurrences). Let P_Y be a set of starting positions of such a set of occurrences. We will compute $t = \mathcal{O}(1)$ sets $\Gamma_1, \ldots, \Gamma_t$, each of size $\mathcal{O}(n/\ell)$, that contain information about all (Y, P_Y), for every Y that can form a b-cover with X. In each set Γ_i we will select an element $\gamma_i \in \Gamma_i$ and consider only length-ℓ factors Y starting at positions $\gamma_i - a$ for $a \in [0\,..\,\ell)$.

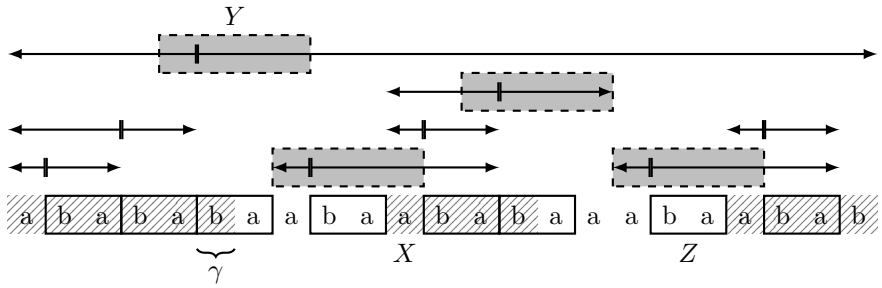


Figure 3 This string has a b-cover ($X = abab, Y = abaa$). The sets $Cov(X)$ and $Cov(Y)$ are shown in gray. We have $Z = ba$, $\Gamma = Occ_T(Z)$, and γ is the position of the occurrence of Z that ends at the first position that is not covered by $Cov(X)$. The set P_Y of occurrences of Y that is generated by $(\Gamma, \gamma, 1)$ is shown. For the meaning of arrows, see Section 4.3.

In particular, for every such (Y, P_Y) we would like to have $P_Y \subseteq (\Gamma_i \ominus a)$ and $Y = T[\gamma_i - a \,..\, \gamma_i - a + \ell)$ for some $i \in [1\,..\,t]$ and $a \in [0\,..\,\ell)$. We then say that (Y, P_Y) is *generated* by (Γ_i, γ_i, a). Moreover, for each set Γ_i we will provide an interval $J_i \subseteq [0\,..\,\ell)$ such that for every Y that forms a b-cover with X, the factor Y is generated by (Γ_i, γ_i, a) for just a constant number of $a \in J_i$. This will allow us to report each sought factor Y a constant number of times and filter out repetitions in the end.

In the algorithm we first compute a constant number of factors $Z_1, \ldots, Z_t$ of T length $z = \lceil \ell/2 \rceil$ such that if (X, Y) is a b-cover, then Y contains at least one of $Z_1, \ldots, Z_t$ as a factor. Let Z be a factor of Y such that $a+1 \in Occ_Y(Z)$. If $i \in Occ_T(Z)$ and $i - a \in Occ_T(Y)$, then we say that the occurrence i of Z *a-anchors* the occurrence $i - a$ of Y and that the latter is *a-anchored* at the former. If Z_i is aperiodic, by Observation 4, we have $|Occ_{Z_i}(T)| = \mathcal{O}(n/\ell)$ and $|Occ_{Z_i}(Y)| = \mathcal{O}(1)$ for any length-ℓ string Y. In this case we will take $\Gamma_i = Occ_{Z_i}(T)$ and $J_i = [0\,..\,\ell - z]$. If Z_i is periodic with period p, we will only be interested in factors Y that are periodic with the same period. In this case we will take as Γ_i a sufficient subset of $Occ_{Z_i}(T)$ and set $J_i = [0\,..\,p)$. See Figure 3 for an example.

Formally, we reduce computing a b-cover of a given length to a constant number of instances of the following problem.

Positioned Cover of Length ℓ
Input: A factor Z of T, a set of positions $\Gamma \subseteq Occ_T(Z)$, its element $\gamma \in \Gamma$, and an interval $J \subseteq [0\,..\,\ell)$.
Output: Report all $a \in J$ such that $Cov_T(X) \cup (\bigcup intervals_\ell(P_Y)) = [1\,..\,n]$ for (Y, P_Y) that is generated by (Γ, γ, a), with $|Y| = \ell$.

In Section 4.3 we show how to solve this problem efficiently if $|\Gamma| = \mathcal{O}(n/\ell)$. Clearly:

▶ **Observation 8.** *If an instance of Positioned Cover of Length ℓ for any Γ, γ, J has a solution (X, Y) for some $a \in J$, then (X, Y) is a b-cover of T.*

Let i be the first position of T that is not covered by occurrences of X. Hence, i has to be covered by the second string Y of the b-cover. Let us denote

$$z = \lceil \ell/2 \rceil, \quad Z_1 = T[i - z + 1\,..\,i], \quad Z_2 = T[i\,..\,i + z).$$

▶ **Observation 9.** *If (X, Y) is a b-cover of length ℓ of T, then Z_1 or Z_2 is a factor of Y.*

Proof. Let $T[j\,..\,j+\ell)$ be an occurrence of Y that covers the position i. If $j \le i-z+1$, then it covers the factor Z_1. Otherwise, $j+\ell-1 \ge i+z$ and $j \le i$, so it covers Z_2. ◀

We will consider as Z each of the two factors Z_1, Z_2 and denote by i_Z the starting position of the occurrence of Z mentioned in the definition. We can ask a two-period query [8, 35, 40] to check if Z is periodic and, if so, compute its smallest period.

▶ **Observation 10.** *If Z is aperiodic, then Y is not 4-periodic. If Z is periodic, then either Y is 4-periodic with the same period, or Y is not 4-periodic.*

Proof. Assume that Z is aperiodic. If Y was 4-periodic with period p, i.e., $4p \le \ell$, then p would also be a period of its factor Z and $2p \le z$, so Z would be periodic.

Assume now that Z is periodic. Let p be the smallest period of Z; we have $2p \le z$. Assume to the contrary that Y is 4-periodic with smallest period p' such that $p' \ne p$. We have $4p' \le \ell$, so $2p' \le z$. Then p' is not a multiple of p, since otherwise p would have been a period of Y. By the periodicity lemma (Fact 2), Z has period $\gcd(p,p') < p$, a contradiction. ◀

In the remainder of the reduction we consider two cases depending on if Y is 4-periodic.

4.1 Reduction for non-4-periodic Y

If Z is periodic, then we try two ways of substituting it with a string that is not periodic.

▶ **Observation 11.** *Assume that Z is periodic with smallest period p, Y is not 4-periodic and an occurrence $T[i\,..\,i+\ell)$ of Y contains $T[i_Z\,..\,i_Z+z)$. Let $i' < j'$ be the positions that break the periodicity of $T[i_Z\,..\,i_Z+z)$. Then $T[i\,..\,i+\ell)$ contains at least one of the fragments $T[i'\,..\,i'+z)$, $T[j'-z+1\,..\,j']$.*

We denote the fragments in the conclusion of the observation by Z' and Z'', respectively. Let us recall that if Z is periodic, the positions breaking the periodicity can be computed using LCE queries. Hence, Z' and Z'' can be computed in $\mathcal{O}(1)$ time. By Fact 3, if Z' or Z'' exists, it is aperiodic. If Z is periodic, we try replacing it by Z' or Z'' (and redefine the occurrence i_Z).

In total we obtain up to four aperiodic strings Z such that if Y is not 4-periodic, its occurrence contains the occurrence $T[i_Z\,..\,i_Z+z)$ for at least one of them. We have $|Occ(Z)| = \mathcal{O}(n/\ell)$ (Observation 4) and all the occurrences can be found in $\mathcal{O}(n/\ell)$ time using IPM queries. The following lemma summarizes the above argument.

▶ **Lemma 12.** *If T has a b-cover (X,Y) of length ℓ with non-4-periodic Y, then (Y,P_Y) is generated by (Γ,γ,a) where $\Gamma = Occ(Z)$, $\gamma = i_Z$ and $a \in [0\,..\,\ell-z]$, for one of up to four aperiodic strings Z. We have $|\Gamma| = \mathcal{O}(n/\ell)$ and Γ, γ can be computed in $\mathcal{O}(n/\ell)$ time.*

4.2 Reduction for 4-periodic Y

By Observation 10, in this case Z is necessarily periodic with the same smallest period as Y. If we used the same reduction as in Lemma 12, we could unfortunately have $|\Gamma| = |Occ(Z)| = \Theta(n)$. We deal with this problem by choosing the first occurrence of Z in Y as an anchor and selecting only some of the occurrences of Z in T to the set Γ that are sufficient for Y to cover all positions in $Cov(Y)$; see Figure 4.

▶ **Lemma 13.** *If T has a b-cover (X,Y) of length ℓ with 4-periodic Y, then (Y,P_Y) is generated by (Γ,γ,a) where $\Gamma \subseteq Occ(Z)$ and $a \in [0\,..\,p)$, for one of up to two periodic strings Z with smallest period p and one of up to two positions γ. We have $|\Gamma| = \mathcal{O}(n/\ell)$ and Γ, γ, p can be computed in $\mathcal{O}(n/\ell)$ time.*

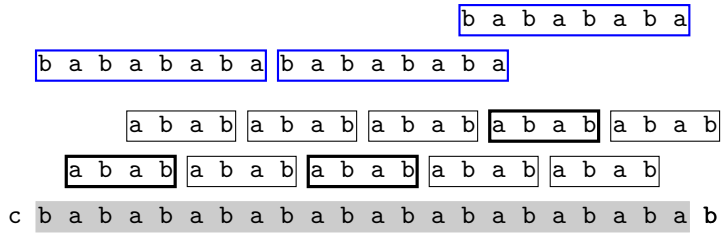


Figure 4 $Z = \texttt{abab}$ (black rectangles), $Y = \texttt{bababababa}$ (blue rectangles); gray color shows $Cov(Y)$. The occurrences of Y that are 1-anchored at marked occurrences of Z are shown and cover $Cov(Y)$.

Proof. Let $p = \mathsf{per}(Z)$. We apply IPM queries to compute the set $Occ(Z)$, represented as a union of $\mathcal{O}(n/\ell)$ arithmetic sequences with difference p. Let us further merge these arithmetic sequences into maximal sequences with difference p, that we denote as $S_1, \ldots, S_k$. We note that $Z[1\,..\,p]$ is primitive, since otherwise Z would have a smaller period. By the synchronization property (Fact 5) for $Z[1\,..\,p]$, we can assume that $\max(S_i) + p < \min(S_{i+1})$ for $i = 2, \ldots, k$, so $\sum_{i=1}^{k} |S_i| = \mathcal{O}(n/p)$. Initially let $\Gamma = Occ(Z)$. We will show how to prune Γ by leaving $\mathcal{O}(|S_i|p/\ell)$ elements in each of the sequences S_i. This will indeed give $|\Gamma| = \mathcal{O}(n/\ell)$.

The set $Occ_Y(Z)$ is an arithmetic sequence with difference p and first element $t \in [1\,..\,p)$. Let $m = |Occ_Y(Z)|$; we have $2 \le m \le \ell/p$. An occurrence of Y in T implies a subsequence of length m of consecutive elements in one of the sequences S_i. Moreover, any arithmetic sequence $j, j+p, \ldots, j+(m+1)p$ of $m+2$ occurrences of Z in T implies an occurrence of Y in T at position $j + p - t + 1$. (Note that a difference-p arithmetic sequence of $m+1$ occurrences of Z in T does not have to imply an occurrence of Y, e.g. if $T = \texttt{abababab}$, $Z = \texttt{abab}$ and $Y = \texttt{bababa}$.)

We can now construct the pruned set Γ' as follows. Let us consider $S_i = \{j, j+p, \ldots, j+(w-1)p\}$. If $w + 1 < m$, then we can ignore S_i. Otherwise we insert to Γ':

- the elements j and $j + p$;
- all elements $j + m \cdot p \cdot t \in S_i$ for positive integer t;
- the elements $j + (w - m - 1)p$ and $j + (w - m)p$.

This way $\mathcal{O}(w/m) = \mathcal{O}(|S_i|p/\ell)$ elements are inserted to Γ', so indeed $|\Gamma'| = \mathcal{O}(n/\ell)$.

Finally, let S_b be the arithmetic sequence that contains the position i_Z. Then we have two choices for γ: $\min(S_b)$ or $\min(S_b) + p$. ◀

4.3 Solution to Positioned Cover problem

Let us recall the problem statement.

Positioned Cover of Length ℓ
Input: A factor Z of T, a set of positions $\Gamma \subseteq Occ_T(Z)$, its element $\gamma \in \Gamma$, and an interval $J \subseteq [0\,..\,\ell)$.
Output: Report all $a \in J$ such that $Cov_T(X) \cup (\bigcup intervals_\ell(P_Y)) = [1\,..\,n]$ for (Y, P_Y) that is generated by (Γ, γ, a), with $\lvert Y\rvert = \ell$.

▶ **Lemma 14.** *After $\mathcal{O}(n)$ time and space preprocessing, assuming that $|\Gamma| = \mathcal{O}(n/\ell)$, Positioned Cover of Length ℓ over an integer alphabet can be solved in $\mathcal{O}(n\tau_n/\ell + \mathsf{output})$ time and $\mathcal{O}(n/\ell)$ space.*

Proof. Let $A = Cov(X)$, $A' = [1\,..\,n] \setminus A$, and $\mathcal{A} \subseteq [1\,..\,n]^2$ be the set of maximal intervals of A. We have $|\mathcal{A}| \leq n/\ell$ and $\mathcal{A}$ can be computed in $\mathcal{O}(n/\ell)$ time. Then the Positioned Cover problem can be solved with the following Claim 15 for

$$S = \{(i,\, \mathsf{lcs}(T[1\,..\,i), T[1\,..\,\gamma)),\, \mathsf{lcp}(T[i\,..\,n], T[\gamma\,..\,n])) \,:\, i \in \Gamma\},$$

where lcp and lcs is the length of the longest common prefix and the longest common suffix, respectively. Intuitively, if $(i, x, y) \in S$, then there is an occurrence of a length-ℓ factor Y a-anchored at $i \in Occ(Z)$ if and only if $a \leq x$ and $|Z| \leq \ell - a \leq y$. See the arrows in Figure 3.

▷ Claim 15. In $\mathcal{O}(n\tau_n/\ell + \mathsf{output})$ time and $\mathcal{O}(n/\ell)$ space one can report all $a \in J$ such that

$$A \cup \bigcup intervals_\ell(S'_a \ominus a) = [1\,..\,n], \tag{1}$$

where $S'_a = \{i \,:\, (i, x, y) \in S,\, a \leq x,\, \ell - a \leq y\}$.

Proof. In the algorithm we store $\mathcal{A}$ in a predecessor data structure $D_{\mathcal{A}}$ sorted by the left endpoints of intervals. We will consider all $a \in J$ in a decreasing order and store the current set S'_a in a predecessor data structure D_S. However, we will only consider values of a for which $S'_a \neq S'_{a+1}$. Let us note that $(i, x, y) \in S$ contributes to $i \in S'_a$ for $a \in [\ell - y\,..\,x]$. Hence, if this interval is non-empty, we will insert i to S'_a for $a = x$ and remove it for $a = \ell - y - 1$. We have $|S| = \mathcal{O}(n/\ell)$, so all events of insertion and deletion to D_S can be precomputed and sorted in $\mathcal{O}(n \log\log n/\ell)$ time using Han's algorithm [29].

Assume that D_S is the data structure that stores S'_a for all a in an interval $J_0 \subseteq J$. Let $i \in D_S$ and $i' = succ(i, D_S)$. We can observe that:

- If $K = [i\,..\,i') \setminus A$ is non-empty and (1) holds for some $a \in J_0$, then $K \subseteq [i - a\,..\,i - a + \ell) \cup [i' - a\,..\,i' - a + \ell)$.

Hence, if $[i\,..\,i')$ is to be covered by the left hand side of (1) for some $a \in J_0$, we have the following set $C(i, i')$ of constraints on a (see Figure 5 in the appendix):

(a) If $i' - i \leq \ell$ or $[i\,..\,i') \subseteq A$, no constraints are imposed. If there are at least two intervals from $\mathcal{A}$ that are fully contained in $[i\,..\,i')$, then there is no such a.

(b) Otherwise, if no interval in $\mathcal{A}$ is a subset of $[i\,..\,i')$, then $a \geq i' - j$ or $\ell - a \geq j' - i + 1$, where $j = succ(i, A')$ and $j' = pred(i' - 1, A')$.

(c) Otherwise, if there is an interval $[u\,..\,v] \in \mathcal{A}$ such that $[u\,..\,v] \subseteq [i\,..\,i')$, then $a \geq i' - v - 1$ and $\ell - a \geq u - i$.

The respective cases can be checked and $C(i, i')$ can be constructed in τ_n time using $D_{\mathcal{A}}$. A similar set of conditions can be stated for the left hand side of (1) to contain all elements of $[1\,..\,\min D_S)$ and $[\max D_S\,..\,n]$; we denote the resulting constraints by $C(0, \min D_S)$ and $C(\max D_S, n + 1)$, respectively, and insert 0 and $n + 1$ to S'_a.

Let us note that each of the constraints from the set $C(i, i')$ is a conjunction of at most two constraints of the form $a \notin I$ for some interval I. Indeed,

$$(a \geq x) \vee (a \leq y) \Leftrightarrow a \notin (y\,..\,x), \quad (a \geq x) \wedge (a \leq y) \Leftrightarrow (a \notin [0\,..\,x)) \wedge (a \notin (y\,..\,\ell)).$$

When i is inserted to D_S, we remove the constraints $C(i', i'')$ imposed by the pair $i' = pred(i, D_S)$ and $i'' = succ(i, D_S)$ and insert the constraints $C(i', i)$ and $C(i, i'')$. For every constraint $a \notin I$, we will retain the value a_1 of a for which it is inserted and the value a_2 for which it is removed. If $I' = [a_1\,..\,a_2)$, the constraint imposes a constraint $a \notin (I \cap I')$ on values of a that satisfy (1).

Overall we obtain $\mathcal{O}(n/\ell)$ constraints of the form $a \notin I$ for (1) to hold. Our goal is to report all $a \in J$ that satisfy all the constraints, i.e., all a in the complement of the union of the $\mathcal{O}(n/\ell)$ intervals from the constraints. This task can be completed by a classic 1d sweep algorithm if the endpoints of intervals from the constraints are sorted [29].

The data structure $D_{\mathcal{A}}$ takes $\mathcal{O}(n\tau_n/\ell)$ time to construct since $|\mathcal{A}| = \mathcal{O}(n/\ell)$ and admits $\mathcal{O}(n/\ell)$ queries. The data structure D_S admits $\mathcal{O}(n/\ell)$ operations. Additional sorting takes $\mathcal{O}(n\tau_n/\ell)$ time. Finally, all values of a for which (1) is satisfied are reported in $\mathcal{O}(\mathsf{output})$ time. The complexity follows. ◁

This concludes the solution to Positioned Cover problem. ◀

A single string Y can be generated by (Γ, γ, a) with $a \in J$ from Lemma 12 a constant number of times because Z is aperiodic, and a constant number of times from Lemma 13 because of the synchronization property. By combining Lemma 14 with Observation 8 and the reductions of Lemmas 12 and 13, we obtain the following result and its corollary.

▶ **Lemma 16.** *Let T be a string of length n over an integer alphabet. After $\mathcal{O}(n)$-time and space preprocessing, one can report all b-covers of T of a given length ℓ, each of them $\mathcal{O}(1)$ times, in $\mathcal{O}(n\tau_n/\ell + \mathsf{output})$ time.*

▶ **Theorem 17.** *Let T be a string of length n over any ordered alphabet. All b-covers of T can be computed in $\mathcal{O}(n\tau_n \log n + \mathsf{output})$ time and $\mathcal{O}(n)$ space.*

Proof. Let us sort and renumber letters in T so that they are in $[1\,..\,n]$. This takes $\mathcal{O}(n \log n)$ time. Then we apply Lemma 16 for every possible length ℓ of a b-cover. Apart from the time to report the output, the complexity becomes $\sum_{\ell=1}^{n} \mathcal{O}(n\tau_n/\ell) = \mathcal{O}(n\tau_n \log n)$.

Finally, we need to make sure that each b-cover is reported only once. We can use the inverse suffix array to sort all factors Y of a given length in the lexicographic order. The sorting is performed globally, across all lengths, using radix sort. We can then iterate over length-ℓ strings Y in the sorted order and remove duplicates using LCE-queries. ◀

5 Computation of 2-covers and λ-covers

We summarize the results of Theorems 7 and 17 and use efficient predecessor data structures [5, 48] to obtain the following result.

▶ **Theorem 18.** *Let T be a string of length n over any ordered alphabet. All 2-covers of T can be computed in $\mathcal{O}(n \log n \log\log n + \mathsf{output})$ expected time or $\mathcal{O}(n \log n \log^2 \log n / \log\log\log n + \mathsf{output})$ worst-case time and $\mathcal{O}(n)$ space.*

Let us recall that there are up to n ps-covers. Moreover, the algorithm behind Lemma 16 allows one to generate as many b-covers of a given length as one requires. This shows that indeed one can compute a 2-cover of each possible length or all the shortest 2-covers in $\mathcal{O}(n\tau_n \log n)$ time.

Theorem 19 extends Theorem 18 to λ-covers for any $\lambda \geq 2$. As in the case of 2-covers, we are only interested in computing λ-covers of lengths for which T does not have a $(\lambda-1)$-cover.

▶ **Theorem 19.** *Let T be a string of length n over any ordered alphabet. For any $\lambda \geq 2$, all λ-covers of T can be computed in $\mathcal{O}(n^{\lambda-1} \log n \log\log n + \mathsf{output})$ expected time or $\mathcal{O}(n^{\lambda-1} \log n \log^2 \log n / \log\log\log n + \mathsf{output})$ worst-case time and $\mathcal{O}(n)$ space.*

Proof. It suffices to give a proof for $\lambda \geq 3$. Similarly as in the case of 2-covers, we classify λ-covers $\mathcal{S} = (X_1, \ldots, X_\lambda)$ into ps-λ-covers, for which X_1 is a prefix and X_2 is a suffix of T, and b-λ-covers, for which X_1 is a border of T. (Formally, in order to compute all λ-covers, in case of ps-λ-covers in the end we need to generate all tuples where the prefix and suffix of T are not the first two respective elements of the tuple, and similarly for b-λ-covers.) The two cases are handled similarly as ps-covers and b-covers, respectively. The number of ps-λ-covers is upper bounded by $n^{\lambda-1}$, whereas the number of b-λ-covers can be $\Theta(n^\lambda)$; see [23].

Let us show how to compute all ps-λ-covers of a given length $\ell \in [1\,..\,n]$. First we use IPM queries to compute $Cov(X_1) \cup Cov(X_2)$, represented as a union of $\mathcal{O}(n/\ell)$ maximal intervals, as in Lemma 6. We LCE-queries on suffixes of the suffix array of T to partition positions of T into classes $C_1, \ldots, C_m$ such that positions i, j belong to the same class if and only if $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$. This could be also done in $\mathcal{O}(n \log n)$ total time using Crochemore's partitioning [18]. For each of the $\binom{m}{\lambda-2}$ choices of $\lambda - 2$ classes $C_{i_1}, \ldots, C_{i_{\lambda-2}}$, if none of them corresponds to X_1 or X_2, we compute their union B. The sets B are computed simultaneously for several choices containing $\Theta(n)$ elements in total using radix sort in order to achieve $\mathcal{O}(|C_{i_1}| + \cdots + |C_{i_{\lambda-2}}|)$ amortized time per choice. Within the same time complexity we can compute the set $\bigcup \mathit{intervals}_\ell(B)$ represented as a union of maximal intervals. Finally, we merge this set with $Cov(X_1) \cup Cov(X_2)$ and check if their union is $[1\,..\,n]$. The time complexity for a given choice of classes is $\mathcal{O}(|C_{i_1}| + \cdots + |C_{i_{\lambda-2}}| + n/\ell)$.

Over all choices, the running time is proportional to

$$\sum_{1 \leq i_1 \leq \ldots \leq i_{\lambda-2} \leq m} \left(|C_{i_1}| + \cdots + |C_{i_{\lambda-2}}| + \frac{n}{\ell} \right) = \binom{m-1}{\lambda-3} \sum_{i=1}^{m} |C_i| + \binom{m}{\lambda-2} \frac{n}{\ell} \leq \frac{2n^{\lambda-1}}{\ell}. \quad (2)$$

The total cost of computing all classes C_i, over all $\ell \in [1\,..\,n]$, is $\mathcal{O}(n^2)$ (or $\mathcal{O}(n \log n)$), and the other preprocessing (LCE and IPM) takes $\mathcal{O}(n)$ time. Thus the overall cost of computing all ps-λ-covers is $\mathcal{O}(n^{\lambda-1} \log n)$.

Computation of b-λ-covers is a similar adjustment to the computation of b-covers of a given length. Recall that X_1 is a length-ℓ border of T. We iterate over all $\binom{m}{\lambda-2}$ choices of $\lambda - 2$ classes $C_{i_1}, \ldots, C_{i_{\lambda-2}}$ which corresponds to selecting factors $X_2, \ldots, X_{\lambda-1}$ from the b-λ-cover. A selection for which $X_i = X_1$ for some $i > 1$ is discarded. The set $\mathcal{C} := Cov(X_1) \cup \cdots \cup Cov(X_{\lambda-1})$ can be expressed as a union of $\mathcal{O}(n/\ell)$ maximal intervals in $\mathcal{O}(|C_{i_1}| + \cdots + |C_{i_{\lambda-2}}| + n/\ell)$ time, which is $\mathcal{O}(n^{\lambda-1}/\ell)$ overall by (2).

In order to compute X_λ, we we make a reduction to a generalization of Positioned Cover of Length ℓ in which we take $\mathcal{C}$ instead of $Cov_T(X)$. The factors Z_1 and Z_2 are computed as in Observation 9, by setting i to the first position in T that is not covered by $\mathcal{C}$. This allows us to compute Z depending on if X_λ is 4-periodic, as in Sections 4.1 and 4.2, in $\mathcal{O}(1)$ time. The solution of the general Positioned Cover of Length ℓ is the same as the one given in Lemma 14, but using $\mathcal{C}$ instead of $Cov_T(X)$. The time complexity of the solution is $\mathcal{O}(n\tau_n/\ell)$ plus the time needed to output b-λ-covers. These steps need to be performed for each of the $\binom{m}{\lambda-2} \leq n^{\lambda-2}$ choices of classes, which gives $\mathcal{O}(n^{\lambda-1}\tau_n/\ell)$ for the given length ℓ, and $\mathcal{O}(n^{\lambda-1}\tau_n \log n)$ in total (plus output).

The complexity follows by summing the complexities of computing ps-λ-covers and b-λ-covers and using efficient predecessor data structures [5, 48]. ◀

6 Conclusions and open problems

We presented quasi-linear time algorithms (plus the time to report the output) for computing 2-covers of a string. One could ask if a shortest 2-cover can be computed in linear time. A further problem is to check if the general λ-cover problem parameterized by λ is fixed-parameter tractable.

One could also consider alternative definitions of 2-covers (and λ-covers) in which the factors that are to cover the text need not to be of the same length. Efficient computation of such covers seems to be an interesting open problem. In particular, under this alternative definition, there can be $\Theta(n^4)$ candidates for a 2-cover (every pair of factors).

References

1 Ali Alatabbi, M. Sohel Rahman, and William F. Smyth. Computing covers using prefix tables. *Discrete Applied Mathematics*, 212:2–9, 2016. doi:10.1016/j.dam.2015.05.019.

2 Amihood Amir, Costas S. Iliopoulos, and Jakub Radoszewski. Two strings at Hamming distance 1 cannot be both quasiperiodic. *Information Processing Letters*, 128:54–57, 2017. doi:10.1016/j.ipl.2017.08.005.

3 Amihood Amir, Avivit Levy, Moshe Lewenstein, Ronit Lubin, and Benny Porat. Can we recover the cover? *Algorithmica*, 81(7):2857–2875, 2019. doi:10.1007/s00453-019-00559-8.

4 Amihood Amir, Avivit Levy, Ronit Lubin, and Ely Porat. Approximate cover of strings. *Theoretical Computer Science*, 793:59–69, 2019. doi:10.1016/j.tcs.2019.05.020.

5 Arne Andersson and Mikkel Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM*, 54(3):13, 2007. doi:10.1145/1236457.1236460.

6 Alberto Apostolico and Andrzej Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Theoretical Computer Science*, 119(2):247–265, 1993. doi:10.1016/0304-3975(93)90159-Q.

7 Alberto Apostolico, Martin Farach, and Costas S. Iliopoulos. Optimal superprimitivity testing for strings. *Information Processing Letters*, 39(1):17–20, 1991. doi:10.1016/0020-0190(91)90056-N.

8 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

9 Carl Barton, Tomasz Kociumaka, Chang Liu, Solon P. Pissis, and Jakub Radoszewski. Indexing weighted sequences: Neat and efficient. *Information and Computation*, 270:104462, 2020. doi:10.1016/j.ic.2019.104462.

10 Amir M. Ben-Amram, Omer Berkman, Costas S. Iliopoulos, and Kunsoo Park. The subtree max gap problem with application to parallel string covering. In Daniel Dominic Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 501–510. ACM/SIAM, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314633.

11 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

12 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, Michael T. Hallett, and Harold T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, 11(1):49–57, 1995. doi:10.1093/bioinformatics/11.1.49.

13 Dany Breslauer. An on-line string superprimitivity test. *Information Processing Letters*, 44(6):345–347, 1992. doi:10.1016/0020-0190(92)90111-8.

14 Dany Breslauer. Testing string superprimitivity in parallel. *Information Processing Letters*, 49(5):235–241, 1994. doi:10.1016/0020-0190(94)90060-4.

15 Stefan Canzar, Tobias Marschall, Sven Rahmann, and Chris Schwiegelshohn. Solving the minimum string cover problem. In *Proceedings of the 14th Meeting on Algorithm Engineering & Experiments, ALENEX 2012*, pages 75–83. SIAM / Omnipress, 2012. doi:10.1137/1.9781611972924.8.

16 Manolis Christodoulakis, Costas S. Iliopoulos, Kunsoo Park, and Jeong Seop Sim. Approximate seeds of strings. *Journal of Automata, Languages, and Combinatorics*, 10(5/6):609–626, 2005. doi:10.25596/jalc-2005-609.

17 Richard Cole, Costas S. Iliopoulos, Manal Mohamed, William F. Smyth, and Lu Yang. The complexity of the minimum k-cover problem. *Journal of Automata, Languages, and Combinatorics*, 10(5/6):641–653, 2005.

18 Maxime Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981. doi:10.1016/0020-0190(81)90024-7.

19 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007. doi:10.1017/cbo9780511546853.

20 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Covering problems for partial words and for indeterminate strings. *Theoretical Computer Science*, 698:25–39, 2017. doi:10.1016/j.tcs.2017.05.026.

21 Maxime Crochemore, Costas S. Iliopoulos, and Maureen Korda. Two-dimensional prefix string matching and covering on square matrices. *Algorithmica*, 20(4):353–373, 1998. doi:10.1007/PL00009200.

22 Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific, 2003. doi:10.1142/4838.

23 Patryk Czajka and Jakub Radoszewski. Experimental evaluation of algorithms for computing quasiperiods. *CoRR*, abs/1909.11336, 2019 (accepted to *Theoretical Computer Science*). arXiv:1909.11336.

24 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

25 Tomás Flouri, Costas S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Simon J. Puglisi, William F. Smyth, and Wojciech Tyczyński. Enhanced string covering. *Theoretical Computer Science*, 506:102–114, 2013. doi:10.1016/j.tcs.2013.08.013.

26 Paweł Gawrychowski, Jakub Radoszewski, and Tatiana A. Starikovskaya. Quasi-periodicity in streams. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, volume 128 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.22.

27 Qing Guo, Hui Zhang, and Costas S. Iliopoulos. Computing the λ-covers of a string. *Information Sciences*, 177(19):3957–3967, 2007. doi:10.1016/j.ins.2007.02.020.

28 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

29 Yijie Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *Journal of Algorithms*, 50(1):96–105, 2004. doi:10.1016/j.jalgor.2003.09.001.

30 Danny Hermelin, Dror Rawitz, Romeo Rizzi, and Stéphane Vialette. The minimum substring cover problem. *Information and Computation*, 206(11):1303–1312, 2008. doi:10.1016/j.ic.2008.06.002.

31 Costas S. Iliopoulos, Christos Makris, Yannis Panagis, Katerina Perdikuri, Evangelos Theodoridis, and Athanasios K. Tsakalidis. The weighted suffix tree: An efficient data structure for handling molecular weighted sequences and its applications. *Fundamenta Informaticae*, 71(2-3):259–277, 2006. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi71-2-3-07.

32 Costas S. Iliopoulos, Manal Mohamed, Laurent Mouchard, Katerina Perdikuri, William F. Smyth, and Athanasios K. Tsakalidis. String regularities with don't cares. *Nordic Journal on Computing*, 10(1):40–51, 2003.

33 Costas S. Iliopoulos, Dennis W. G. Moore, and Kunsoo Park. Covering a string. *Algorithmica*, 16(3):288–297, 1996. doi:10.1007/BF01955677.

34 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.

35 Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, 2018. URL: https://mimuw.edu.pl/~kociumaka/files/phd.pdf.

36 Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear-time algorithm for seeds computation. *ACM Transactions on Algorithms*, 16(2):Article 27, 2020. doi:10.1145/3386369.

37 Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear time algorithm for seeds computation. In Yuval Rabani, editor, *23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 1095–1112. SIAM, 2012. doi:10.1137/1.9781611973099.

38 Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Efficient algorithms for shortest partial seeds in words. *Theoretical Computer Science*, 710:139–147, 2018. doi:10.1016/j.tcs.2016.11.035.

39 Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Fast algorithm for partial covers in words. *Algorithmica*, 73(1):217–233, 2015. doi:10.1007/s00453-014-9915-3.

40 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In Piotr Indyk, editor, *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.

41 Yin Li and William F. Smyth. Computing the cover array in linear time. *Algorithmica*, 32(1):95–106, 2002. doi:10.1007/s00453-001-0062-2.

42 Michael G. Main and Richard J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984. doi:10.1016/0196-6774(84)90021-X.

43 Dennis Moore and W. F. Smyth. Computing the covers of a string in linear time. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, page 511–515, USA, 1994. Society for Industrial and Applied Mathematics.

44 Dennis W. G. Moore and William F. Smyth. An optimal algorithm to compute all the covers of a string. *Information Processing Letters*, 50(5):239–246, 1994. doi:10.1016/0020-0190(94)00045-X.

45 Dennis W. G. Moore and William F. Smyth. A correction to "An optimal algorithm to compute all the covers of a string". *Information Processing Letters*, 54(2):101–103, 1995. doi:10.1016/0020-0190(94)00235-Q.

46 Jean Néraud. Elementariness of a finite set of words is co-NP-complete. *RAIRO Theoretical Informatics and Applications*, 24:459–470, 1990. doi:10.1051/ita/1990240504591.

47 Alexandru Popa and Andrei Tanasescu. An output-sensitive algorithm for the minimization of 2-dimensional string covers. In T. V. Gopal and Junzo Watada, editors, *Theory and Applications of Models of Computation - 15th Annual Conference, TAMC 2019*, volume 11436 of *Lecture Notes in Computer Science*, pages 536–549. Springer, 2019. doi:10.1007/978-3-030-14812-6_33.

48 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\theta(n)$. *Information Processing Letters*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

49 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. Algorithms for computing the lambda-regularities in strings. *Fundamenta Informaticae*, 84(1):33–49, 2008. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi84-1-04.

A Supplementary Figure

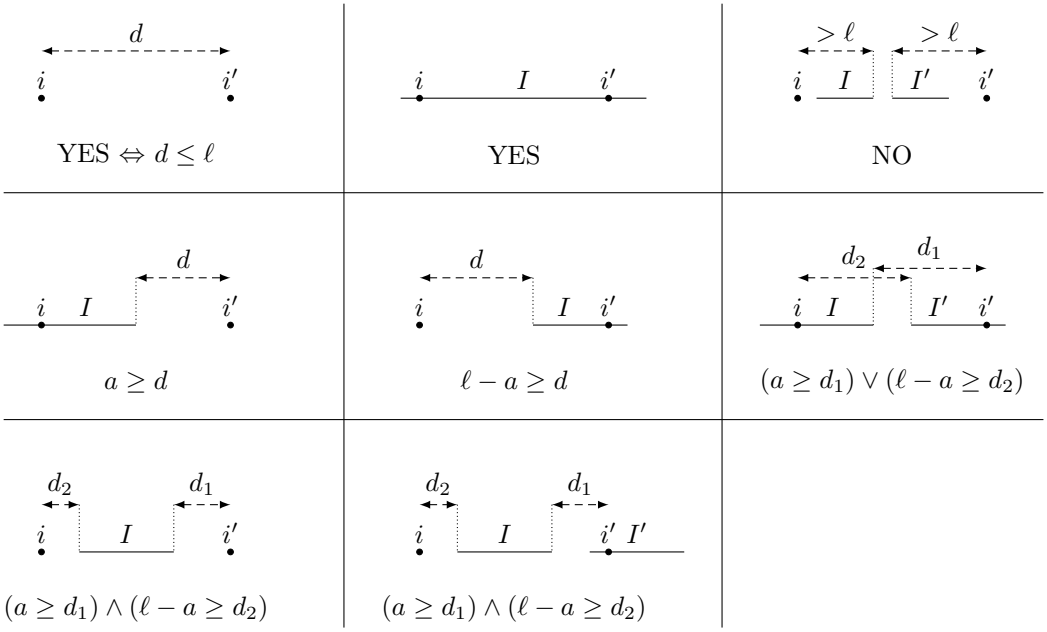


Figure 5 Sets of constraints $C(i, i')$ generated depending on the interactions with intervals $I, I' \in \mathcal{A}$. The respective rows correspond to items (a)–(c).

Improved Approximation Algorithm for Set Multicover with Non-Piercing Regions

Rajiv Raman
IIIT Delhi, India
rajiv@iiitd.ac.in

Saurabh Ray
NYU Abu Dhabi, United Arab Emirates
saurabh.ray@nyu.edu

Abstract

In the Set Multicover problem, we are given a set system $(X, \mathcal{S})$, where X is a finite ground set, and $\mathcal{S}$ is a collection of subsets of X. Each element $x \in X$ has a non-negative demand $d(x)$. The goal is to pick a smallest cardinality sub-collection $\mathcal{S}'$ of $\mathcal{S}$ such that each point is covered by at least $d(x)$ sets from $\mathcal{S}'$. In this paper, we study the set multicover problem for set systems defined by points and non-piercing regions in the plane, which includes disks, pseudodisks, k-admissible regions, squares, unit height rectangles, homothets of convex sets, upward paths on a tree, etc.

We give a polynomial time $(2+\epsilon)$-approximation algorithm for the set multicover problem $(P, \mathcal{R})$, where P is a set of points with demands, and $\mathcal{R}$ is a set of non-piercing regions, as well as for the set multicover problem $(\mathcal{D}, P)$, where $\mathcal{D}$ is a set of pseudodisks with demands, and P is a set of points in the plane, which is the hitting set problem with demands.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Computational geometry

Keywords and phrases Approximation algorithms, geometry, Covering

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.78

Acknowledgements The authors are grateful to Nabil H. Mustafa for ideas and comments.

1 Introduction

The Set Cover problem and its variants are central problems in Computer Science. For general set systems, tight results are known - there is an $O(\log n)$ approximation algorithm [32] and this is tight under standard complexity assumptions [18]. Over the last decade, significant progress has been on these problems for geometric set systems in the plane and in low dimensions. There are broadly two approaches that have been successful in the geometric setting, *viz.*, LP-rounding based algorithms and local-search. The LP-rounding approach relies on the existence of small ϵ-nets, which exist whenever the *VC-dimension* of the set system is bounded. Set systems with VC-dimension at most d have ϵ-nets of size $O(d/\epsilon \log 1/\epsilon)$ [22], and this leads to an $O(\log |\textsc{Opt}|)$-approximation algorithm [5, 17], that holds even in the *weighted setting*[1]. Smaller ϵ-nets are known when the *union complexity* of the set system is *small* [13], leading to algorithms with better approximation factors, though not in the weighted setting. Varadarajan [31] showed via the *quasi-uniform* sampling technique how these results can be made to work in the weighted setting. His technique was optimized by Chan, et al. [7] who also introduced the notion of *shallow cell complexity* generalizing the notion of union complexity to abstract set systems. Chekuri, et al. [11] extended the LP-based techniques for set cover to set multicover obtaining an $O(\log |\textsc{Opt}|)$

[1] In the weighted setting, the sets have non-negative weights, and the goal is to find a minimum weight feasible sub-collection, as opposed to the minimum cardinality.

approximation when the VC dimension is bounded, and better bounds in the case where the union complexity is bounded. In particular, they obtain $O(1)$-approximation algorithms when union complexity is linear. However, their results only hold in the unweighted setting. Bansal and Pruhs [3] extended the approach based on shallow cell complexity and quasi-uniform sampling [31, 7] to work for the weighted set multicover problem. The main weakness of these LP-rounding techniques is that the approximation factor obtained is at least as large as the integrality gap, which is often large. Furthermore, the constants gained in the approximation factor during the rounding process is often large. For instance, [11] uses *shallow cuttings*, which involves large constants.

A second approach that has been effective for fundamental geometric packing and covering problems, albeit in the unweighted setting [8, 28, 21, 25, 4, 30] is Local Search. Besides packing and covering problems, Local Search has also been remarkably successful for several clustering problems (See [20, 19, 14, 15], and references therein). A drawback of the Local Search approach is that the running time of the algorithms are prohibitive. In particular, Mustafa and Jartoux [23] showed that to obtain a $(1+\epsilon)$-approximation for the Set Cover problem with disks, the local search algorithm takes $n^{\Omega(1/\epsilon^2)}$ time.

The analysis of local search for most of the geometric packing and covering problems relies on showing the existence of a graph with desired characteristics which is problem specific, and this is usually the challenging part of the analysis. In [30], Raman and Ray gave a unified method to obtain such graphs for several packing and covering problems. While the techniques in [30] can be extended to *packing problems* with bounded capacities as was shown in [4], it was not clear how to extend them to the set multicover problem - even with bounded demands.

Obtaining approximation algorithms that run fast, while simultaneously guaranteeing small approximation factors is a challenging research direction. Recently, Chekuri et al. [12], and Chan and He [10], building on the work of Agarwal and Pan [1] have improved the running times of the LP based algorithms for both packing and covering problems via the multiplicative weights update framework. In this work, we improve the approximation factor, but we do not improve the running times of the algorithms. We give a polynomial time $(2+\epsilon)$-approximation algorithm for the Set Multicover problem for non-piercing regions in the unweighted setting. We obtain the same approximation factor for the multi-hitting set problem for pseudodisks. Our key observation is that even if the LP relaxation has a large integrality gap, we can round it without losing more than a $(1+\delta)$ factor so that it meets all demands with only a constant deficit which depends on the parameter δ. This yields a problem with low demands for which a PTAS can be obtained via local search. Note that even the second part of our approach - PTAS for multihitting set problems with bounded demands, is non-trivial, and builds on tools developed by Raman and Ray [30].

2 Preliminaries

The set multicover problem is defined by a set system $(X, \mathcal{S})$ and a demand function $d : X \mapsto \mathbb{R}$. The task is to select the smallest cardinality subset $\mathcal{S}'$ of $\mathcal{S}$ such that each $x \in X$ is contained in at least $d(x)$ subsets of $\mathcal{S}'$. We refer to the set X as the *ground set* and the elements of $\mathcal{S}$ as *ranges*. For any subset $\mathcal{S}' \subseteq \mathcal{S}$ and any $x \in X$, we denote by $\mathcal{S}'(x)$ the set of elements in $\mathcal{S}'$ containing x and we refer to $|\mathcal{S}'(x)|$ as the *depth* of x in $\mathcal{S}'$.

We require the notion of *shallow-cell complexity* defined by Chan et al. [7]. A *cell* in the set system $(X, \mathcal{S})$ is a maximal subset $X' \subseteq X$ such that the elements of X' are contained in the same collection of ranges in $\mathcal{S}$. We say that a range $S \in \mathcal{S}$ contains a cell C if S contains

the elements in C. The depth of a cell C is the number of ranges in $\mathcal{S}$ containing C. A set system has shallow-cell complexity $f(n,k)$ if for any subset $\mathcal{S}' \subseteq \mathcal{S}$ of size n, the number of cells of depth at most k in $(X, \mathcal{S}')$ is at most $f(n,k)$. We focus on set systems whose shallow cell complexity is linear in n and polynomial in k. We say that a set system is *c-linear*, for some constant c, if its shallow-cell complexity is $O(nk^c)$.

The set systems we study in this paper are defined by a set of points and a set of regions in the plane. A *Jordan region* is a compact, simply connected set in the plane whose boundary is a simple jordan curve. We say that a set $\mathcal{R}$ of Jordan regions is a non-piercing family if for any $\gamma, \gamma' \in \mathcal{R}$, $\gamma \setminus \gamma'$ and $\gamma' \setminus \gamma$ are both path connected sets. A set of Jordan regions $\mathcal{R}$ is said to be a family of *pseudodisks* if the boundaries of every pair of regions either do not intersect or cross (i.e. intersect non-tangentially) at exactly two points. Note that a family of pseudodisks is also a family of non-piercing regions but not vice-versa.

In this paper, we study two set multicover problems defined by a set of points and a set of regions in the plane. The first is the set multicover problem in which the ground set is a finite set P of points in the plane and the ranges are obtained by intersecting P with the regions in a family $\mathcal{R}$ of non-piercing regions. Abusing notation, we denote such set systems by $(P, \mathcal{R})$. In the second set multicover problem we study, the ground set is a family of pseudodisks $\mathcal{D}$ and each range is the subset of $\mathcal{D}$ containing a particular point p in a set of points P. Again, for simplicity, we denote such a set system by $(\mathcal{D}, P)$. This variant of the set cover problem is usually called the *hitting set* problem.

3 Our Results

The results of Chekuri et al. [11] imply an $O(1)$-approximation algorithm for the set multicover problem for set systems with linear union complexity. Such set systems are c-linear for some constant c. Bansal and Pruhs [3] guarantee an $O(1)$-approximation factor for the weighted set multicover problem defined by a c-linear set system.

Since both the methods above are based on LP-rounding, the approximation factor is at least as large as the integrality gap. Even for set cover problems (i.e., all demands are 1) defined by very simple geometric regions in the plane, the integrality gap is not known to be small. For halfspaces in the plane, the integrality gap is 2 [24] which implies that it is at least 2 for disks as well. Even though no larger lower bound is known, the best upper bound currently known on the integrality gap for disks is significantly higher: 13.4 [6]. The integrality gap is probably higher when the demands are allowed to be more than 1. However, we are not aware of any results regarding this. Apart from the integrality gap, there are additional constant factors that are gained in the process of rounding. While the exact constants are not analyzed in [11] or [3], they seem to be large. For instance, since the main tool is used in [3] is the quasi-sampling technique [7, 31], the constant seems to be at least $e^{4.34362} > 76$ (see Claims 2 and 4 of [7]). We do not have accurate estimates for the constants involved in shallow cuttings used in [11] but we believe that they are not significantly smaller (and likely to be much larger since they use ϵ-nets and approximations for which the known constants are quite large). Our main result is a $(2+\epsilon)$-approximation algorithm for the set multicover problem for points and non-piercing regions in the plane, in the unweighted setting.

▶ **Theorem 1.** *The set multicover problem defined by a set system $(P, \mathcal{R})$, where P is a finite set of points and $\mathcal{R}$ is a set of non-piercing regions in the plane with an arbitrary demand function $d : P \to \mathbb{R}$ admits a polynomial time $(2+\epsilon)$-approximation algorithm for any $\epsilon > 0$.*

The result follows from the following results that we prove in Sections 4 and 5. First, we show that the set system $(P, \mathcal{R})$ is 2-linear (Lemma 25) and for any set system that is c-linear for some constant c, we obtain the following result via a simple modification of the technique of Bansal and Pruhs [3].

▶ **Theorem 2.** *Let $(X, \mathcal{S})$ be a c-linear set system for some constant c and let $m = |\mathcal{S}|$. Consider the set multicover problem defined by $(X, \mathcal{S})$ in which each element $x \in X$ has a demand $d(x)$. Let y be any feasible solution to the linear programming relaxation of this problem i.e., y satisfies the constraints: $\forall x \in X$, $\sum_{S \ni x} y_S \geq d(x)$, and let $\delta \in (0, 1)$ be a given parameter. Then, we can obtain a subset of ranges $\mathcal{S}' \subseteq \mathcal{S}$ s.t.*

(i) $\forall x \in X,\ |\mathcal{S}'(x)| \geq d(x) - 3\tau$, *where $\tau = C\delta^{-4} \log \delta^{-1}$ with a large enough C*

(ii) $|\mathcal{S}'| \leq (1 + O(\delta)) \sum_{S \in \mathcal{S}} y_S$

This shows that the LP solution can be "rounded" without increasing the objective value much and causing only a constant *deficit* in the demands. Since the *residual* demands are $O(1)$ for any constant δ, we obtain a set multicover problem with bounded demands. We then show that the set multicover problem with non-piercing regions and bounded demands has a PTAS via local search.

▶ **Theorem 3.** *Local Search yields a PTAS for the set multicover problem defined by a set system $(P, \mathcal{R})$ where P is a set of points and $\mathcal{R}$ is a family of non-piercing regions in the plane and where each point $p \in P$ has a demand bounded above by some constant Θ.*

We also obtain the same approximation factor in the dual setting, though here we are currently only able to prove the result when the regions are pseudodisks.

▶ **Theorem 4.** *Let $\mathcal{P}$ be a Set Multicover problem for the set system $(\mathcal{D}, P)$ defined by a set of pseudodisks $\mathcal{D}$ and a set of points P in the plane with demand function $d : \mathcal{D} \to \mathbb{R}$. For any $\epsilon > 0$, there is a polynomial time $(2 + \epsilon)$-approximation algorithm for $\mathcal{P}$.*

The result is obtained by showing that the set system $(\mathcal{D}, P)$ is 2-linear, which implies that Theorem 2 can be used to obtain an instance of the set multicover problem where the demands of the pseudodisks are bounded above by a constant Θ. For these instances, we show that local search yields a PTAS.

▶ **Theorem 5.** *Local Search yields a PTAS for the set multicover problem defined by a set system $(\mathcal{D}, P)$ where P is a set of points and $\mathcal{D}$ is a family of pseudodisks in the plane and where each pseudodisk $D \in \mathcal{D}$ has a demand bounded above by some constant Θ.*

Even though the PTASes obtained in Theorems 3 and 5 are not surprising, the proofs are not trivial. In fact, it would not be surprising if local search yields a PTAS for the set multicover problem with arbitrary demands. However, we are currently unable to prove such a result.

The paper is organized as follows. In Section 4, we prove Theorem 2. In Subsection 5.1, we prove Theorem 3, and we prove Theorem 5 in Subsection 5.2. Theorems 1 and 4 are proved in Section 6.

4 LP Rounding with bounded deficits

In this section we prove Theorem 2. Consider the set multicover problem defined by a c-linear set system $(X, \mathcal{S})$ and a demand function $d : X \mapsto \mathbb{R}$. The natural linear programming relaxation for this problem is the following:

$$\min \sum_{S \in \mathcal{S}} y_S \quad \text{s.t.} \quad \forall S \in \mathcal{S},\ y_S \in [0,1] \quad \text{and} \quad \forall x \in X,\ \sum_{S \ni x : S \in \mathcal{S}} y_S \geq d(x) \tag{1}$$

Our goal is to show that given any feasible solution y to the above LP and any $\delta > 0$, we can find a subset $\mathcal{S}' \subseteq \mathcal{S}$ of size at most $(1 + O(\delta)) \sum_{S \in \mathcal{S}} y_S$ and $\forall x \in X, |\mathcal{S}'(x)| \geq d(x) - O(\delta^{-4} \log \delta^{-1})$. We start with a few technical results. The lemma below follows from the techniques in [7, 31] where similar statements are proved.

▶ **Lemma 6.** *Let $(X, \mathcal{S})$ be a c-linear set system and let $m = |\mathcal{S}|$. Then, there exists an ordering $S_1, \cdots, S_m$ of the ranges in $\mathcal{S}$ s.t. in the set system $(X, \mathcal{S}_i)$ where $\mathcal{S}_i = \{S_1, \cdots, S_i\}$, the range S_i contains at most $O(k^{c+3})$ cells of depth at most k, for any k.*

Proof. We assign a weight to each cell in $(X, \mathcal{S})$ depending on its depth. A cell of depth k is assigned a weight of $1/k^{c+3}$. We define the weight of a range to be the total weight of all the cells it contains. Since there are at most $O(mk^c)$ cells of depth k (since $(X, \mathcal{S})$ is c-linear), the total weight of all cells of depth k is at most $O(m/k^3)$. Since each cell of depth k contributes to the weight of k ranges in $\mathcal{S}$, the contribution of the depth k cells to the total weight of all ranges is $O(m/k^2)$. The total weight of all ranges is therefore at most $O(m) \sum_{k=1}^{m} k^{-2} = O(m)$. This implies that there is a range $S \in \mathcal{S}$ whose weight is $O(1)$ which in turn implies that for any k, the number of depth k cells in S is at most $O(k^{c+3})$. We recursively find the ordering of $(X, \mathcal{S} \setminus \{S\})$. The ordering for $(X, \mathcal{S})$ is obtained by appending S to the ordering for $(X, \mathcal{S} \setminus \{S\})$. The lemma follows. ◀

▶ **Definition 7** (Weighted Depth). *Let $(X, \mathcal{S})$ be a set system. Let $w(S) \geq 0$ be a weight associated with each range $S \in \mathcal{S}$. Then, for any element $x \in X$, we denote by $w_{\mathcal{S}}(x)$ the total weight of the ranges in $\mathcal{S}$ containing x. We call this the weighted depth of the element x with respect to the ranges in $\mathcal{S}$ and the weight function w.*

▶ **Lemma 8** (Weighted Sampling Procedure). *Let $(X, \mathcal{S})$ be a c-linear set system for some constant c and let $m = |\mathcal{S}|$. Let $w(S) \in [0.5, 1.0]$ be a weight associated with each $S \in \mathcal{S}$ and let $\delta \in (0, 1)$ be a parameter. Then, there is a polynomial time procedure to pick a subset $\mathcal{S}' \subseteq \mathcal{S}$ s.t.*

- **(i)** *$|\mathcal{S}'| \leq (1 + O(\delta))\, W$ where $W = \sum_{S \in \mathcal{S}} w(S)$, and*
- **(ii)** *for any element $x \in X$: $|\mathcal{S}'(x)| \geq w_{\mathcal{S}}(x) - \tau$ where $\tau = C\delta^{-3} \log \delta^{-1}$ for a large enough constant C.*

Proof. The set $\mathcal{S}'$ is unweighted but we can think of each range in $\mathcal{S}'$ as having weight 1. This means that any range $S \in \mathcal{S}$ with weight $\geq 1 - \delta$ can be safely included in $\mathcal{S}'$ since this way we are increasing its weight by at most a factor of $1 + O(\delta)$. We will thus assume without loss of generality that all ranges in $\mathcal{S}$ have weight at most $1 - \delta$. We will show that a procedure similar to the quasi-uniform sampling procedure [31, 7] can be used to pick each range $S \in \mathcal{S}$ into $\mathcal{S}'$ with probability at most $(1 + O(\delta))\, w(S)$ s.t. the second condition in the lemma is satisfied. Then, the expected number of ranges in $\mathcal{S}'$ is at most $(1 + O(\delta))\, W$.

Now, by Markov inequality, the probability that $|\mathcal{S}'|$ exceeds its expectation by a factor of more than $(1 + \delta)$ is at most $1/(1 + \delta)$ which means that the with probability $\Omega(\delta)$, $|\mathcal{S}'|$ is $(1 + O(\delta))W$. We can therefore repeat the process $O(1/\delta)$ times in expectation to obtain the desired collection $\mathcal{S}'$ of sets.

Our sampling procedure has two stages. In the first stage, we pick a sample $\mathcal{T} \subseteq \mathcal{S}$ by picking each range $S \in \mathcal{S}$ independently with probability $w(S)/(1-\delta) \in [0,1]$. In the second stage, we pick another sample $\mathcal{T}' \subseteq \mathcal{S}$ s.t. each range is picked with a probability $O(\delta) = O(w(S) \cdot \delta)$ but whether or not a range is picked depends on the outcome of the first stage. We set $\mathcal{S}' = \mathcal{T} \cup \mathcal{T}'$. The probability that a particular range $S \in \mathcal{S}$ is included in $\mathcal{S}'$ is at most $w(S)/(1-\delta) + O(w(S) \cdot \delta) = w(S)(1 + O(\delta))$ as required.

Let $S_1, \cdots, S_m$ be the ordering of the ranges in $\mathcal{S}$ given by Lemma 6. Let $\mathcal{S}_i = \{S_1, \cdots, S_i\}$ and let $\mathcal{T}_i = \mathcal{T} \cap \mathcal{S}_i$ denote the subset of the ranges in $\mathcal{S}_i$ picked in the first stage. The range S_i is picked in the second stage if it is *forced* by an element $x \in X$ which happens if $k = w_{\mathcal{S}}(x) \geq \tau$, $k_i = w_{\mathcal{S}_i}(x) \geq \delta k$ and $s_i = |\mathcal{T}_i(x)| < k_i$. In words, S_i is forced by x if i) x has high weighted depth (at least τ) w.r.t. the ranges in $\mathcal{S}$, ii) the ranges in $\mathcal{S}_i$ contribute at least δ fraction of the total weight of the ranges in $\mathcal{S}$ containing x and iii) fewer than $w_{\mathcal{S}_i}(x)$ ranges among the ranges in $\mathcal{S}_i$ containing x are sampled in the first stage.

Note that the second stage guarantees that the second condition in the lemma is satisfied for all elements in X. We now bound the probability that S_i is forced by x. Since each range $S \in \mathcal{S}$ is picked independently with probability $w(S)/(1-\delta)$ in the first stage, the expected value of s_i is $\mu = k_i(1-\delta)$ and by Chernoff bound, $\Pr(s_i < k_i) = \Pr(s_i < (1-\delta)\mu) \leq \exp(-\mu\delta^2/2) \leq \exp(-k_i\delta^2/2) \leq k_i^{-(c+5)}$ since $k_i \geq \delta k \geq \delta\tau = C\delta^{-2}\log\delta^{-1}$ which for large enough C implies that $k_i\delta^2/2 \geq (c+5)\log k_i$.

Thus, we have shown that the probability that the range S_i is forced by a particular element x having weighted depth k_i w.r.t. $\mathcal{S}_i$ is at most $k_i^{-(c+5)}$. However S_i may be forced by many elements in X. To bound the probability that S_i is forced (by some element), first note that all elements lying in the same cell of $(X, \mathcal{S}_i)$ behave identically i.e., they all either force S_i or don't force S_i. Thus, we only need to consider elements in distinct cells of $(X, \mathcal{S}_i)$. By Lemma 6, for any $t \geq 1$, S_i contains at most $O(t^{c+3})$ cells of depth t in $(X, \mathcal{S}_i)$. Since each range $S \in \mathcal{S}$ has a weight $w(S) \geq 0.5$, it follows that if an element of X has (unweighted) depth t, then its weighted depth is at least $t/2$. Thus, the probability that S_i is forced by elements in X of depth t w.r.t. $\mathcal{S}_i$ is at most $O(t^{c+3}) \cdot O(t^{-(c+5)}) = O(t^{-2})$. Since the second condition in the lemma is trivially satisfied for elements having weighted depth at most τ in $(X, \mathcal{S})$, we are concerned only with elements having weighted depth $k \geq \tau$ in $(X, \mathcal{S})$, and any such an element can force S_i only if its weighted depth in $(X, \mathcal{S}_i)$ is at least $\delta k \geq \delta\tau$, the probability that S_i is forced (by some element) is at most $\sum_{t=\delta\tau}^{\infty} O(t^{-2}) = O(1/(\delta\tau)) = O(\delta)$. The lemma follows. ◀

The following is an *unweighted* version of the above lemma.

▶ **Lemma 9** (Unweighted Sampling Procedure). *Let $(X, \mathcal{S})$ be a c-linear set system and let $m = |\mathcal{S}|$. Let Δ be a large enough parameter. Then, there is a procedure to pick a subset $\mathcal{S}' \subseteq \mathcal{S}$ s.t.*

(i) $|\mathcal{S}'| \leq (1 + O(\sqrt[4]{(\log\Delta)/\Delta})\,|\mathcal{S}|/2$ *and*

(ii) *any element $x \in X$ having depth $k \geq \Delta$ in $\mathcal{S}$, has depth at least $k/2$ in $\mathcal{S}'$.*

Proof. The lemma follows from a straightforward application of Lemma 8 with parameter $\delta = \alpha \sqrt[4]{(\log\Delta)/\Delta}$ for a small enough constant α s.t. $\Delta \geq \frac{2}{\delta}\, C\, \delta^{-3} \log\delta^{-1}$ where C is the constant in Lemma 8 and setting the weight of each range in $\mathcal{S}$ to $0.5(1+\delta)$. ◀

We now restate and prove Theorem 2.

▶ **Theorem 2.** *Let $(X, \mathcal{S})$ be a c-linear set system for some constant c and let $m = |\mathcal{S}|$. Consider the set multicover problem defined by $(X, \mathcal{S})$ in which each element $x \in X$ has a demand $d(x)$. Let y be any feasible solution to the linear programming relaxation of this problem i.e., y satisfies the constraints: $\forall x \in X$, $\sum_{S \ni x} y_S \geq d(x)$, and let $\delta \in (0, 1)$ be a given parameter. Then, we can obtain a subset of ranges $\mathcal{S}' \subseteq \mathcal{S}$ s.t.*

(i) $\forall x \in X$, $|\mathcal{S}'(x)| \geq d(x) - 3\tau$, *where* $\tau = C\delta^{-4} \log \delta^{-1}$ *with a large enough* C

(ii) $|\mathcal{S}'| \leq (1 + O(\delta)) \sum_{S \in \mathcal{S}} y_S$

Proof. Since condition (i) in the lemma is trivially satisfied for elements with demand at most 3τ, we can assume without loss of generality that each element has demand more than 3τ. We associate a *value* v_S with each range $S \in \mathcal{S}$. Initially $v_S = y_S$. The proof is constructive, and consists of two phases. The first phase consists of several rounds in which we modify the values of the ranges so that they are either 0 or lie in the interval $[0.5, 1]$. In a second phase we pick each range $S \in \mathcal{S}$ into $\mathcal{S}'$ with probability equal to the value v_S at the end of the first phase.

In the first phase we maintain a partition of the set of ranges $\mathcal{S}$ into two sets $\mathcal{F}$ and $\mathcal{N}$ where $\mathcal{F}$ is the set of *frozen* ranges whose values are either 0 or in the interval $[0.5, 1]$ and $\mathcal{N} = \mathcal{D} \setminus \mathcal{F}$ is the set of *non-frozen* ranges. We do not modify the value of any range in $\mathcal{F}$ which means that once a range is in $\mathcal{F}$, it remains in $\mathcal{F}$. We continue to modify the values of the ranges in $\mathcal{N}$ until all of them move to $\mathcal{F}$.

The first phase has several rounds. At the beginning of the first round, each range S has value y_S. The total value of all ranges at this time is $\sum_{S \in \mathcal{S}} y_S$. We replace each range $S \in \mathcal{N}$ by $\lfloor m \cdot y_S \rfloor$ replicas where each replica has value $\lambda_1 = 1/m$. The value of a range S is now the product of the number of replicas and the replica value. Note that the total value of any particular range decreases by at most $1/m$ due to this and therefore for each element $x \in X$, we have $\sum_{S \ni x : S \in \mathcal{S}} v_S \geq d(x) - 1$ i.e., v satisfies the demands with a *deficit* of ≤ 1.

At the beginning of round i, each replica has value $\lambda_i = 2^{i-1}/m$. Note that the replica values are uniform in any round and we double the replica value in each round. In round i, we use the unweighted sampling procedure (Lemma 9) with the parameter Δ set to $\Delta_i = \tau/\lambda_i$ to obtain a subset of the replicas that go to the round $i + 1$. Note that the value of a frozen range is not modified by this procedure. The value of a non-frozen range changes according to the number of its replicas that make it to the next round. In particular, if a range has lost all its replicas, then its value becomes 0 and is frozen. Similarly, a range is frozen if the total value of its replicas becomes at least 0.5. The number of rounds is less than $\log_2 m$ since after so many rounds each replica has value at least 0.5.

Let v_S be the value of range S after the first phase. We claim that v satisfies $\forall x \in X$: $\sum_{S \in \mathcal{S}} v_S \geq d(x) - 2\tau$. To see this, consider any element $x \in X$ and let us say that x is *satisfied* at any point in time if the total value of the ranges containing it is at least its demand minus 1 i.e., $\sum_{S \ni x} v_S = \sum_{S \ni x : S \in \mathcal{N}} v_S + \sum_{S \ni x : S \in \mathcal{F}} v_S \geq d(x) - 1$. Note that x is satisfied at the beginning of round 1. Furthermore, if x is satisfied at the beginning of round i and in addition $\sum_{x \ni S : S \in \mathcal{N}} v_S \geq \tau$, then it is also satisfied after round i. This follows since every replica has value λ_i in round i, which means that if x's depth in the arrangement of the replicas is k then $k \geq \Delta_i = \tau/\lambda_i$ and Lemma 9 guarantees that its depth with respect to the replicas after round i is at least $k/2$. Recall that replica weights are doubled in every round which compensates for the decrease in the number of replicas covering x. Thus, if x is satisfied at the beginning of round i, then the only way it can be dissatisfied at the end of the round is if at the beginning of the round, $\sum_{S \ni x : S \in \mathcal{N}} v_S < \tau$ which means that $\sum_{S \ni x : S \in \mathcal{F}} v_S \geq d(x) - \tau - 1 \geq d(x) - 2\tau$. Since we don't modify the value of any range once it enters the set $\mathcal{F}$, this inequality is satisfied at the end of the first phase too.

We now argue that at the end of the first phase: $\sum_{S\in\mathcal{S}} v_S \leq (1+O(\delta))\sum_{S\in\mathcal{S}} y_S$. To see this note that, by Lemma 9, the total value of the ranges in $\mathcal{S}$ increases, by a factor of at most $1+O(\sqrt[4]{(\log\Delta_i)/\Delta_i})$ in round i. Therefore, the total value of all ranges increases in the first phase by a factor of at most

$$\prod_{i=1}^{t}\left(1+O\left(\sqrt[4]{\frac{\log\Delta_i}{\Delta_i}}\right)\right) \leq \exp\left[O\left(\sum_{i=1}^{t}\sqrt[4]{\frac{\log\Delta_i}{\Delta_i}}\right)\right] \leq \exp\left[O\left(\sqrt[4]{\frac{\log\Delta_t}{\Delta_t}}\right)\right] \leq (1+O(\delta))$$

where $t<\log_2 m$ is the number of rounds. The second last inequality above follows from the fact that $\sqrt[4]{(\log\Delta_i)/\Delta_i}$ increases geometrically with i since $\Delta_i=\tau m/2^{i-1}$ decreases geometrically with i. The last inequality follows from the fact that $t<\log_2 m$, which means that $\Delta_t<\tau$.

Recall that at the end of the first phase, the value of each range is either 0 or lies in the range $[0.5,1]$. In the second phase, we apply Lemma 8 to the ranges having a non-zero value $v_S\in[0.5,1]$ at the end of the first phase with the weight function $w(S)=v_S$ and the parameter δ. Since every element $x\in X$ is assumed to have demand at least 3τ, its weighted depth k with respect to the weight function w is least $d(x)-2\tau\geq\tau$. Lemma 8 then guarantees that its unweighted depth with respect to the set of ranges $\mathcal{S}'$ returned by Lemma 8 is also at least k. Thus $\mathcal{S}'$ satisfies the first condition in the lemma. It also satisfies the second condition in the lemma since Lemma 8 guarantees that $|\mathcal{S}'|\leq(1+O(\delta))\sum_{S\in\mathcal{S}} v_S$ which is at most $(1+O(\delta))\sum_{S\in\mathcal{S}} y_S$ since we had established earlier that $\sum_{S\in\mathcal{S}} v_S\leq(1+O(\delta))\sum_{S\in\mathcal{S}} y_S$. ◀

5 Set Multicover with bounded demands

Consider the set multicover problem defined by a set system $(X,\mathcal{S})$ and a demand function d s.t. for each element $x\in X$, $d(x)$ is bounded above by a constant Θ. We will show that a standard local search algorithm (see e.g. [2, 9, 28, 30] for details) yields a PTAS for this problem when X is a set of points and $\mathcal{S}$ is a set of non-piercing regions in the plane, or when X is a set of pseudodisks and $\mathcal{S}$ is a set of points in the plane. The algorithm takes as input a parameter k and starting with any feasible solution it tries to improve the solution by making *swaps* of size at most k and stops when no such improvement is possible.

In order to show that such a local search algorithm yields a PTAS, we consider an optimal solution R to the problem and a solution B returned by the local search algorithm. We want to show that $|B|\leq(1+\epsilon(k))\,|R|$ where $\epsilon(k)\to 0$ as $k\to\infty$. We can assume without loss of generality that R and B are disjoint by removing the set $S=R\cap B$ from both and considering the problem with residual demands where the residual demand of any element $x\in X$ is $d(x)-|S(x)|$. Note that the residual demands are still bounded above by Θ and if $|B\setminus S|\leq(1+\epsilon(k))\,|R\setminus S|$ then it follows that $|B|\leq(1+\epsilon(k))\,|R|$.

By standard arguments [2, 9, 28, 30], it then suffices to show that there exists a graph $G=(R\cup B,E)$ satisfying the following ***local search conditions***:

1. For any $B'\subseteq B$, $(B\setminus B')\cup N(B')$ is a feasible solution, where $N(B')$ is the set of neighbors of B' in the graph G. We call this the *local exchange property.*
2. The graph has the *sublinear separator property* (see Definition 10).

▶ **Definition 10** (Sublinear Separator Property)**.** *We say that a graph H satisfies the sublinear separator property if there exist $0<\alpha,\delta<1$ s.t. for any induced subgraph H' of H with vertex set $V(H')$, there is a vertex separator $S\subset V(H')$ so that $|S|=O(|V(H')|^{\delta})$, and each connected component of $H'\setminus S$ is of size at most $\alpha|V(H')|$.*

In all applications of this technique so far, the critical part has been showing the existence of such a graph, which is what we now focus on. Instead of the local exchange property, we will use the following *local expansion* property which implies the local exchange property.

▶ **Definition 11** (Local Expansion). *Let R and B be two feasible solutions for a set multicover problem defined by a set system $(X, \mathcal{S})$ and a demand function d. We say that a graph $G = (R \cup B, E)$ satisfies the local expansion property with respect to $(X, R \cup B)$ if for every $x \in X$ at least one of the following statements hold:*

- *There are at least $d(x)$ elements in $R(x)$ that have $d(x)$ or more neighbors in $B(x)$.*
- *There are at least $d(x)$ elements in $B(x)$ that have $d(x)$ or more neighbors in $R(x)$.*

Here, by "neighbors" we mean neighbors in the graph G.

▶ **Proposition 12.** *Let R and B be two feasible solutions to the set multicover problem defined by a set system $(X, \mathcal{S})$ and a demand function d. Then, any graph $G = (R \cup B, E)$ satisfying the local expansion property with respect to $(X, R \cup B)$ also satisfies the local exchange property (i.e., local search condition 1).*

Proof. Consider any $x \in X$. The local expansion property implies that the subgraph of G induced on $R(x) \cup B(x)$ contains a matching of size $d(x)$. Let $U \subseteq R(x)$ and $V \subseteq B(x)$ be the matched elements. Suppose that we replace B by $(B \setminus B') \cup N(B')$ for some $B' \subseteq B$. Then we may lose $k \leq d(x)$ of elements of V but we gain at least an equal number of elements from U due to the matching. This implies that $(B \setminus B') \cup N(B')$ still satisfies x's demand. Since this is true for any $x \in X$, $(B \setminus B') \cup N(B')$ is a feasible solution. ◀

Observe that the intersection graph $I(R \cup B)$ of the ranges in $R \cup B$ in which two ranges are adjacent iff they share an element of X, has the local expansion property and therefore satisfies the local exchange property. However, the intersection graph may not satisfy the sublinear separator property. The next two lemmas show that for the geometric sets systems we consider, the intersection graph does satisfy the sublinear separator property if the depth of each cell in the set system is bounded above by a constant.

▶ **Lemma 13.** *Let $\mathcal{R}$ be a family of n non-piercing regions. Then, the number of pairs of regions in $\mathcal{R}$ that intersect at a point of depth 2 in the arrangement $\mathcal{A}(\mathcal{R})$ of $\mathcal{R}$ is at most $3n - 6$.*

Proof. Let $G = (\mathcal{R}, E)$ be the planar graph obtained from part 1 of Theorem 20. There must be an edge in G between any two regions in $\mathcal{R}$ that intersect at a point of depth 2 in $\mathcal{A}(\mathcal{R})$. The lemma follows from the fact that a planar graph with n vertices has at most $3n - 6$ edges. ◀

▶ **Lemma 14.** *Let $\mathcal{R}$ be a family of n non-piercing regions whose arrangement has depth at most Δ. Then, there exists a set Q of $O(\Delta n)$ points in the plane s.t. any intersecting pair of regions in $\mathcal{R}$ also intersect at one the points in Q.*

Proof. We assume that $\Delta \geq 2$ as otherwise no pair of regions in $\mathcal{R}$ intersect. Let Q be a minimal set of points so that any pair of intersecting regions in $\mathcal{R}$ also intersect at some point in Q. Since Q is minimal, for each $q \in Q$, there must be two regions A_q and B_q in $\mathcal{R}$ which intersect at q but don't intersect at any other point in Q. Suppose now that we take a sample $\mathcal{R}'$ of $\mathcal{R}$ by picking each region in $\mathcal{R}$ independently with probability $p = 1/\Delta$. Since any point $q \in Q$ is contained in at most Δ regions in $\mathcal{R}$, the probability that we pick A_q and B_q in our sample but do not pick any other regions containing q is at least $p^2(1-p)^{\Delta-2} \geq 1/(e\Delta)^2$. The expected number of points in Q for which this happens is at least $|Q|/(e\Delta)^2$. Each such

point q corresponds to a distinct pair of regions in $\mathcal{R}'$ intersecting at a point of depth 2 (in the arrangement of $\mathcal{R}'$). Since the expected number of regions in $\mathcal{R}'$ is $pn = n/\Delta$, and there can be at most $3|\mathcal{R}'|$ pairs of regions in $\mathcal{R}'$ that intersect at a point of depth 2, by Lemma 13, we conclude that $|Q|/(e\Delta)^2 \leq 3n/\Delta$ which implies that $|Q| \leq 3e^2\Delta n$. ◀

▶ **Lemma 15.** *Let $H = (V, E)$ be a planar graph and let $\mathcal{T}$ be a set of subsets of V s.t. i) for each $T \in \mathcal{T}$ the subgraph $H[T]$ of H induced by T is connected and ii) each vertex $v \in V$ is contained in at most Δ of the sets in $\mathcal{T}$, where Δ is a constant. Then, the intersection graph of the sets in any $\mathcal{S} \subseteq \mathcal{T}$ has a balanced separator of size $O(\Delta\sqrt{|W|})$, where $W = \cup_{S\in\mathcal{S}} S$. In other words, the intersection graph of the sets in $\mathcal{T}$ satisfies the sublinear separator property.*

Proof. Fix any $\mathcal{S} \subseteq \mathcal{T}$. We assign a weight $w(v)$ to each vertex $v \in W$ as follows: each set $S \in \mathcal{S}$ has a weight of 1 and it distributes it equally among all the vertices it contains. More precisely, $w(v) = \sum_{S\in\mathcal{S}:v\in S} 1/|S|$. Note that the total weight of all vertices is $|\mathcal{S}|$. Let $X \subseteq W$ be a balanced vertex separator of $G = H[W]$ of size $O(\sqrt{|W|})$ so that the total weight of the vertices in each connected component of $G[W \setminus X]$ is at most $\alpha\,|\mathcal{S}|$ for some $\alpha < 1$. The fact that such a separator exists follows from the fact that $H[W]$ is planar and the planar graph separator theorem [27].

Let $\mathcal{X} = \{S \in \mathcal{S} : S \cap X \neq \emptyset\}$. As each vertex in X is contained in at most Δ sets in $\mathcal{S}$, $|\mathcal{X}| \leq \Delta|X| = O(\Delta\sqrt{|W|})$. Since each set $S \in \mathcal{S}$ induces a connected subgraph of H, $\mathcal{X}$ is a separator for the intersection graph I of the sets in $\mathcal{S}$. Furthermore, it is balanced since each connected component of $I[\mathcal{S} \setminus \mathcal{X}]$ corresponds to a connected component of $H[W \setminus X]$ and since the total weight of all vertices in the latter is at most αn, the number of sets in the former is also at most αn. ◀

▶ **Lemma 16.** *Let $\mathcal{R}$ be a family of n non-piercing regions in the plane whose arrangement has depth at most Δ, where Δ is a constant. Then the intersection graph of the regions in $\mathcal{R}$ has the sublinear separator property.*

Proof. Let Q be a set of $O(\Delta n)$ points obtained by applying Lemma 14. By Theorem 20, there is a planar graph $H = (Q, F)$, such that for any region $R \in \mathcal{R}$, the induced subgraph $H[Q_R]$, where $Q_R = R \cap Q$, is connected. Since every pair of intersecting regions in $\mathcal{R}$ intersect at some point in Q, the intersection graph of the sets in $\mathcal{S} = \{Q_R : R \in \mathcal{R}\}$ is isomorphic to the intersection graph of the regions in $\mathcal{R}$. Further, since the arrangement of the regions has depth Δ, no vertex of H is contained in more than Δ sets in $\mathcal{S}$. Therefore, the Lemma follows from the application of Lemma 15 to H and $\mathcal{S}$. ◀

▶ **Lemma 17.** *Let $(\mathcal{D}, P)$ be a set system defined by a set P of n points and a set $\mathcal{D}$ of pseudodisks in the plane. Then, the number of pairs of points (p, q) that lie in a common cell of depth 2 in $(\mathcal{D}, P)$ (i.e., there is a pseudodisk $D \in \mathcal{D}$ s.t. $D = \{p, q\}$) is at most $3n - 6$.*

Proof. The proof is the same as the proof of Lemma 13 except that we use part 2 of Theorem 20 instead of part 1. ◀

▶ **Lemma 18.** *Let P be a set of n points and let $\mathcal{D}$ be a family of pseudodisks in the plane s.t. for each $D \in \mathcal{D}, |D \cap P| \leq \Delta$. Then, there exists a subset $\mathcal{D}' \subseteq \mathcal{D}$ of size $O(\Delta n)$ s.t. any pair of points in P that belong to a common pseudodisk in $D \in \mathcal{D}$ also belong to a common pseudodisk $D' \in \mathcal{D}'$.*

Proof. The proof is identical to the proof of Lemma 14 except that we use Lemma 17 instead of Lemma 13. ◀

▶ **Lemma 19.** *Let P be a set of points and let $\mathcal{D}$ be a family of pseudodisks in the plane such that for any $D \in \mathcal{D}$, $|D \cap P| \leq \Delta$ for some constant Δ. Then, the graph $G(P, E)$ in which two points $p, q \in P$ are adjacent iff there is a pseudodisk $D \in \mathcal{D}$ containing both p and q, has the sublinear separator property.*

Proof. The proof is identical to the proof of Lemma 16, except that we use Lemma 18 instead of Lemma 14. ◀

In order to use the above lemmas, we need to modify a given set system $(X, R \cup B)$ so that each cell in it has bounded depth. We also require the following result which follows from Theorem 1 in [30].

▶ **Theorem 20** ([30])**.** *Let $\mathcal{R}$ be any family of non-piercing regions in the plane, and let P be any finite set of points in the plane. Then,*

1. *There exists a planar graph $G = (\mathcal{R}, E)$ s.t. for each point $p \in \mathbb{R}^2$, the subgraph of G induced by the regions containing p is connected.*
2. *There exists a planar graph $H = (P, E')$ s.t. for each $\gamma \in \mathcal{R}$, the subgraph of H induced by $\gamma \cap P$ is connected.*

5.1 Points and Non-Piercing regions

In this subsection, we prove Theorem 3. Consider the set multicover problem defined by a set system $(P, \mathcal{R})$ where P is a set of points and $\mathcal{R}$ is a family of non-piercing regions in the plane, and in which the demand $d(p)$ of each point is at most Θ. As before, $R, B \subseteq \mathcal{R}$ represent an optimal solution and a solution returned by the local search algorithm respectively. Furthermore, as argue before, we can assume without loss of generality that $R \cap B = \emptyset$. We will show the existence of a graph $G = (R \cup B, E)$ satisfying the local search conditions, implying a PTAS for this problem.

We first need a technical tool from [30] for modifying the regions so that the arrangement of the modified regions has bounded depth. A *maximal cell* in an arrangement of regions in the plane is a cell whose depth is higher than all neighboring cells.

▶ **Definition 21** (Cell Bypassing [30])**.** *Let $\mathcal{R}$ be a non-piercing family of regions. Let $\gamma \in \mathcal{R}$ be one of the regions and let C be a maximal cell contained in γ so that that the boundary of γ contributes exactly one arc to the boundary of C. Then, if we modified γ to $\gamma' = \gamma \setminus (C \oplus K_\epsilon)$, the resulting set of regions remains a non-piercing family. Here $\oplus$ denotes Minkowski sum and K_ϵ is a disk of arbitrarily small radius ϵ.*

We refer to the modification of γ to γ' in the above theorem as the "bypassing of C by γ". Note that the modified region $\gamma' \approx \gamma \setminus C$ since ϵ is arbitrarily small.

▶ **Lemma 22.** *There exists a graph $G = (R \cup B, E)$ that satisfies the local search conditions with respect to the set system $(P, \mathcal{R})$.*

Proof. We will construct a graph that satisfies the local expansion property at every point $p \in P$ and has the sublinear separator property. In fact, we will make the first condition stronger: we require the local expansion property to be satisfied at every point in the plane. In other words, we replace P by $\mathbb{R}^2$ and for each point $q \in \mathbb{R}^2$, we define the demand $d(q)$ as $\min(|R(q)|, |B(q)|, \Theta)$. Note that R and B are still feasible solutions to this modified problem and a graph $G = (R \cup B, E)$ satisfying the local expansion property with respect to $(\mathbb{R}^2, R \cup B)$ also satisfies the property with respect to $(P, R \cup B)$.

If the depth of the arrangement of $R \cup B$ is at most 2Θ, then we can simply use Lemma 16 to obtain the required graph G. Otherwise, consider a cell C of maximum depth. By the results in [30], there exists a region $\gamma \in R \cup B$ that contributes exactly one arc to the boundary of C. Our plan is to modify γ so that it bypasses C in order to reduce the depth of the cell C.

For convenience, we will refer to the regions in R and B as *red* and *blue* respectively. We will use the following slightly modified version of the cell bypassing procedure: The process remains the same as before except when the cell C to be bypassed consists of just one side i.e., it is identical to one of the regions. In such a case, the usual cell bypassing procedure removes the region. Instead, we do the following: if the points in the region are contained in more than Θ regions of its color, then we remove the region as before. Otherwise, if the point is contained in exactly Θ regions of its color (including itself), we keep the region but make it *inactive* so that this region does not participate in any further cell bypassing. Initially, all regions are defined to be active. Note that since an inactive region lies in the interior of a cell in the arrangement of the remaining active regions, it does not affect further cell bypassing. Only a cell defined by active regions is bypassed. All inactive regions lie in the interior of some cell in the arrangement of the active regions.

Let C be a cell of maximum depth in the arrangement of the active regions in $R \cup B$ and suppose that this cell is contained in at least $\Theta + 1$ active regions of some color (either red or blue). Let us assume that it is contained in at least $\Theta + 1$ active blue regions. The other case is analogous. We will argue that a graph satisfying the local expansion property for the modified red and blue regions obtained after bypassing C by one of the active regions also satisfies the condition for the original regions. To see this, consider any point p in the cell C that we are about to bypass in the arrangement of the active regions. The point p may be contained in some of the inactive regions contained in C apart from the active regions containing C. However, p cannot be contained in any inactive blue region β since that would mean that p was contained in at most Θ blue regions when β became inactive and is now contained in more than Θ blue regions (by assumption). Since cell bypassing only contracts regions, this is impossible.

Let γ be the region bypassing C. If γ is blue, note that p is still contained in at least $\Theta \geq d(p)$ blue regions after bypassing. Therefore, a graph satisfying the local expansion property for p in the arrangement of the modified regions also satisfies the condition in the arrangement of the original regions.

Consider now the case when γ is red. Let C' be a cell contained in γ and adjacent to C in the original arrangement of the active regions. Let q be a point in C' that is not contained in any of the inactive regions. Note that the set of red regions that contain at least one of the points in $\{p, q\}$ after bypassing is the same as the set of regions containing p before bypassing. Furthermore, both p and q are contained in the same set of blue regions before and after bypassing. Thus, a graph satisfying the locality preserving condition for the points p and q in the modified arrangement also satisfies the condition for p in the original arrangement.

Each cell-bypassing operation either decreases the maximum depth of the arrangement of the active regions, or decreases the number of cells of maximum depth in that arrangement. Therefore, we eventually obtain an arrangement where the maximum number of blue or red regions containing any cell in the arrangement of active regions is at most Θ. At this point, the number of active or inactive regions of the same color containing any point in the plane is at most Θ. Thus the depth of the arrangement of all active and inactive regions is at most 2Θ. We can now obtain the required graph using Lemma 16. ◀

Theorem 3 now follows from the discussions above.

5.2 Pseudodisks and Points

In this subsection, we prove Theorem 5. Consider a set multicover problem defined by the set system $(\mathcal{D}, P)$ where P is set of points and $\mathcal{D}$ is a family of non-piercing regions in the plane, and in which each pseudodisk $D \in \mathcal{D}$ has a demand bounded above by Θ. As earlier, let $R, B \subseteq P$ represent an optimal solution and a solution yielded by the local search algorithm respectively. Also, without loss of generality, $R \cap B = \emptyset$.

▶ **Lemma 23.** *There exists a graph $G = (R \cup B, E)$ that satisfies the local search conditions with respect to the set system $(\mathcal{D}, P)$.*

In order to prove Lemma 23, we will use the following result of Pinchasi [29].

▶ **Lemma 24** ([29])**.** *Let $\mathcal{D}$ be a set of pseudodisks in the plane. For any specified $D \in \mathcal{D}$ and any point $q \in D$, we can continuously shrink D to q so that the arrangement at any time remains an arrangement of pseudodisks.*

Proof of Lemma 23. If every pseudodisk in $\mathcal{D}$ contains at most 2Θ points of P, then we can use Lemma 19 to obtain the required graph. Now, consider any pseudodisk $D \in \mathcal{D}$ containing more than 2Θ points of P. Then, $\mathcal{D}$ must contain either $> d(D)$ points of B or $> d(D)$ points of R. Using Lemma 24, we first shrink D to D' as follows: we pick any point $p \in D \cap (R \cup B)$ and imagine continuously shrinking D to the point p. During the shrinking, we stop as soon as $|D \cap R| = d(D)$ or $|D \cap B| = d(D)$. We call this modified region D'. Lemma 24 guarantees that the arrangement obtained by replacing D by D' is still a family of pseudodisks.

Note that both $|D' \cap B|$ and $|D' \cap R|$ are at least $d(D)$ and one of them is equal to $d(D)$. Assume that $|D' \cap B| = d(D)$, the other case being analogous. Then, one by one, for every $b \in D' \cap B$, we imagine shrinking a copy of D' continuously to the point b and stop when the region contains exactly $d(D)$ points of R. We call this region D_b and add it to the collection $\mathcal{D}$. Again by Lemma 24 we can do this so that the arrangement remains an arrangement of pseudodisks. Finally, we remove the pseudodisk D from the collection $\mathcal{D}$. Observe that each of the regions added to the collection contain at most $d(D)$ points of R and at most $d(D)$ points of B.

This entire procedure is repeated for each pseudodisk $D \in \mathcal{D}$ containing more than 2Θ points. Let $\mathcal{D}'$ be the collection of regions obtained in the end. We now claim that a graph $G = (R \cup B, E)$ satisfying the local expansion property with respect to $(\mathcal{D}', R \cup B)$ also satisfies the local expansion property with respect to $(\mathcal{D}, R \cup B)$.

To see this, observe that if a pseudodisk $D \in \mathcal{D}$ is still in $\mathcal{D}'$, then a graph that satisfies the local expansion property with respect to $(\mathcal{D}, R \cup B)$ also satisfies the condition for D. On the other hand, if D was removed i.e., $D \notin \mathcal{D}'$, then one of the following holds:

- There are at least $d(D)$ points $b \in B \cap D$ for each of which we have added a pseudodisk D_b contained in D s.t. D_b contains $|d(D)|$ points of $R \cap D$
- There are at least $d(D)$ points $r \in R \cap D$ for each of which we have added a pseudodisk D_r contained in D s.t. D_r contains $|d(D)|$ points of $B \cap D$.

Assume the first case holds for a pseudodisk D removed from $\mathcal{D}$. The second case is analogous. Then, since each region D_b added has exactly $d(D)$ points of R, in any graph G that satisfies the local expansion property with respect to $(\mathcal{D}', R \cup B)$, the point b must be adjacent to all of the points in $R \cap D_b \subseteq D$. Since we have added such a pseudodisk D_b for at least $d(D)$ points in $B \cap D$, we have at least $d(D)$ points $B \cap D$ each of which has at least

$d(D)$ neighbors in $R \cap D$. Thus, G satisfies the local expansion property with respect to $(\mathcal{D}, R \cup B)$. Now, since each $D \in \mathcal{D}'$ has at most 2Θ points of $R \cup B$, Lemma 19 yields the required graph G that satisfies the local search conditions with respect to $(\mathcal{D}, R \cup B)$. ◀

Theorem 5 follows from the discussions above.

6 $(2+\epsilon)$-Approximation Algorithms

In this section, we prove Theorems 1 and 4, which require the following lemmas,

▶ **Lemma 25.** *The set system $(P, \mathcal{R})$ where P is a set of points and $\mathcal{R}$ is a non-piercing family is 2-linear.*

Proof. By Theorem 20 of [30], there is a planar support for the dual set system $(\mathcal{R}, P)$ i.e., there is a planar graph $H = (\mathcal{R}, E)$ s.t. for any $p \in P$, the set of regions $\mathcal{R}(p)$ induce a connected subgraph of H. Since H has $O(m)$ edges where $m = |\mathcal{R}|$, the number of cells of depth 2 in $(P, \mathcal{R})$ is $O(m)$. By a standard Clarkson-Shor type argument, this shows that the number of cells of depth at most k is $O(mk^2)$. ◀

Note that a family of non-piercing regions may not have linear union complexity but the set system $(P, \mathcal{R})$ still has a low shallow cell complexity.

▶ **Lemma 26.** *The set system $(\mathcal{D}, P)$ where $\mathcal{D}$ is a set of pseudodisks and P is a set of points in the plane is 2-linear.*

Proof. By Theorem 20 of [30], there is a planar support for the dual set system $(P, \mathcal{D})$ i.e., there is a planar graph $H = (P, E)$ s.t. for any $D \in \mathcal{D}$, the induced subgraph on the points in D is connected. Since H has $O(m)$ edges where $m = |P|$, the number of cells of depth 2 in $(P, \mathcal{D})$ is $O(m)$. By a standard Clarkson-Shor type argument, this shows that the number of cells of depth at most k is $O(mk^2)$. ◀

Proof of Theorem 1. We solve the LP-relaxation (1) for the multicover problem, and obtain a solution y, with objective value $\sum_{D \in \mathcal{R}} y_D \leq |\textsc{Opt}|$ where Opt is an optimal solution to the given set multicover problem. By Lemma 25, the set system $(P, \mathcal{R})$ is 2-linear, and therefore we can use Theorem 2 to obtain a subset $\mathcal{R}' \subseteq \mathcal{R}$ of size at most $(1 + O(\delta)) \sum_{D \in \mathcal{R}} y_D \leq (1 + O(\delta)) |\textsc{Opt}|$, which satisfies the demands of the points in p with a deficit of at most $\Theta = O(\delta^{-4} \log \delta^{-1})$. We then consider the problem with the residual demands in which all the demands are at most Θ. For this problem, we obtain a solution $\mathcal{R}'' \subseteq \mathcal{R}$ of size at most $(1 + O(\delta)) |\textsc{Opt}|$ using Theorem 3. Then, $\mathcal{R}' \cup \mathcal{R}''$ is a solution to the set multicover and has size at most $(2 + O(\delta)) |\textsc{Opt}|$. By taking $\delta = \alpha\epsilon$ for some suitably small constant $\alpha > 0$, we get a $(2+\epsilon)$-approximation to the optimal solution. ◀

Proof of Theorem 4. The proof is exactly the same as the proof of Theorem 1 except that we use Lemma 26 instead of Lemma 25 and Theorem 5 instead of Theorem 3. ◀

7 Conclusion

In this paper, we made progress on the Set Multicover in the geometric setting. We obtained a $(2+\epsilon)$ approximation by combining LP-rounding and local search. We believe that local search itself yields a PTAS for the problem with arbitrary demands. Unfortunately we are currently able to prove that local search yields a PTAS only when the demands bounded above by a constant. Our approach does not work for the weighted setting since local search

thus far works only for the unweighted setting. It is likely that a dynamic programming approach similar to [26, 16] can give a PTAS for the weighted set multicover problem for unit size disks and squares in the plane. Obtaining a PTAS for the problem with arbitrary size squares or disks seems challenging even for the set cover problem. Even with the improved approximation factor, our algorithm remains infeasible in practice due to the prohibitive running time ($n^{\text{poly}(1/\epsilon)}$) of the local search algorithm.

References

1. Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. *Discret. Comput. Geom.*, 63(2):460–482, 2020. doi:10.1007/s00454-019-00099-6.
2. Rom Aschner, Matthew J. Katz, Gila Morgenstern, and Yelena Yuditsky. Approximation schemes for covering and packing. In *WALCOM: Algorithms and Computation*, volume 7748 of *Lecture Notes in Computer Science*, pages 89–100. Springer Berlin Heidelberg, 2013.
3. Nikhil Bansal and Kirk Pruhs. Weighted geometric set multi-cover via quasi-uniform sampling. *JoCG*, 7(1):221–236, 2016.
4. Aniket Basu Roy, Sathish Govindarajan, Rajiv Raman, and Saurabh Ray. Packing and covering with non-piercing regions. *Discrete & Computational Geometry*, 2018.
5. Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.
6. Norbert Bus, Shashwat Garg, Nabil H. Mustafa, and Saurabh Ray. Tighter estimates for ϵ-nets for disks. *Comput. Geom.*, 53:27–35, 2016. doi:10.1016/j.comgeo.2015.12.002.
7. Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1576–1585, 2012.
8. Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.
9. Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.
10. Timothy M. Chan and Qizheng He. Faster approximation algorithms for geometric set cover. *CoRR*, abs/2003.13420, 2020. arXiv:2003.13420.
11. Chandra Chekuri, Kenneth L. Clarkson, and Sariel Har-Peled. On the set multicover problem in geometric settings. *ACM Trans. Algorithms*, 9(1):9:1–9:17, 2012. doi:10.1145/2390176.2390185.
12. Chandra Chekuri, Sariel Har-Peled, and Kent Quanrud. Fast lp-based approximations for geometric packing and covering problems. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1019–1038. SIAM, 2020. doi:10.1137/1.9781611975994.62.
13. Kenneth L. Clarkson and Kasturi R. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007. doi:10.1007/s00454-006-1273-8.
14. Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. *SIAM J. Comput.*, 48(2):644–667, 2019. doi:10.1137/17M112717X.
15. Vincent Cohen-Addad and Claire Mathieu. Effectiveness of local search for geometric optimization. In *Proceedings of the Thirty-first International Symposium on Computational Geometry*, SoCG '15, pages 329–343, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
16. Thomas Erlebach and Erik Jan van Leeuwen. PTAS for weighted set cover on unit squares. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, pages 166–177, 2010.

17 Guy Even, Dror Rawitz, and Shimon Shahar. Hitting sets when the vc-dimension is small. *Inf. Process. Lett.*, 95(2):358–362, 2005. doi:10.1016/j.ipl.2005.03.010.

18 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

19 Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R. Salavatipour. Approximation schemes for clustering with outliers. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 398–414, 2018.

20 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. *SIAM J. Comput.*, 48(2):452–480, 2019. doi:10.1137/17M1127181.

21 Matt Gibson and Imran A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the $\log n$ barrier. In *Algorithms – ESA 2010 - 18th Annual European Symposium, Liverpool, United Kingdom, September 6–8, 2010, Proceedings*, pages 243–254, 2010.

22 David Haussler and Emo Welzl. epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987. doi:10.1007/BF02187876.

23 Bruno Jartoux and Nabil H. Mustafa. Optimality of Geometric Local Search. In *34th International Symposium on Computational Geometry (SoCG 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 48:1–48:15, 2018.

24 János Komlós, János Pach, and Gerhard J. Woeginger. Almost tight bounds for epsilon-nets. *Discret. Comput. Geom.*, 7:163–173, 1992. doi:10.1007/BF02187833.

25 Erik Krohn, Matt Gibson, Gaurav Kanade, and Kasturi Varadarajan. Guarding terrains via local search. *Journal of Computational Geometry*, 5(1):168–178, 2014.

26 Jian Li and Yifei Jin. A PTAS for the weighted unit disk cover problem. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 898–909, 2015.

27 Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

28 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

29 Rom Pinchasi. A finite family of pseudodiscs must include a "small" pseudodisc. *SIAM J. Discret. Math.*, 28(4):1930–1934, 2014. doi:10.1137/130949750.

30 Rajiv Raman and Saurabh Ray. Planar support for non-piercing regions and applications. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 69:1–69:14, 2018.

31 Kasturi Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 641–648, 2010.

32 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001. URL: http://www.springer.com/computer/theoretical+computer+science/book/978-3-540-65367-7.

Improved Distance Sensitivity Oracles with Subcubic Preprocessing Time

Hanlin Ren
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
rhl16@mails.tsinghua.edu.cn

Abstract

We consider the problem of building *Distance Sensitivity Oracles* (DSOs). Given a directed graph $G = (V, E)$ with edge weights in $\{1, 2, \ldots, M\}$, we need to preprocess it into a data structure, and answer the following queries: given vertices $u, v, x \in V$, output the length of the shortest path from u to v that does not go through x. Our main result is a simple DSO with $\tilde{O}(n^{2.7233}M^2)$ preprocessing time and $O(1)$ query time. Moreover, if the input graph is undirected, the preprocessing time can be improved to $\tilde{O}(n^{2.6865}M^2)$. Our algorithms are randomized with correct probability $\geq 1 - 1/n^c$, for a constant c that can be made arbitrarily large. Previously, there is a DSO with $\tilde{O}(n^{2.8729}M)$ preprocessing time and polylog(n) query time [Chechik and Cohen, STOC'20].

At the core of our DSO is the following observation from [Bernstein and Karger, STOC'09]: if there is a DSO with preprocessing time P and query time Q, then we can construct a DSO with preprocessing time $P + \tilde{O}(Mn^2) \cdot Q$ and query time $O(1)$. (Here $\tilde{O}(\cdot)$ hides polylog(n) factors.)

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Data structures design and analysis

Keywords and phrases Graph theory, Failure-prone structures

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.79

Related Version A full version of this paper is available at `https://arxiv.org/abs/2007.11495`.

Acknowledgements I would like to thank Zihao Li for introducing this problem to me, and Ran Duan and Yong Gu for helpful discussions. I would also like to thank Hongxun Wu for reading and commenting an early draft version of this paper, and pointing out the issue with non-unique shortest paths. I am also grateful to the anonymous referees of ESA for their helpful comments that improves the presentation of this work.

1 Introduction

Suppose we are given a directed graph $G = (V, E)$, and we want to build a data structure that, given any three vertices $u, v, x \in V$, outputs the length of the shortest path from u to v that does not go through x. Such a data structure is called a *Distance Sensitivity Oracle* (or DSO for short).

The problem of constructing DSOs is motivated by the fact that real-life networks often suffer from failures. Suppose we have a network with n nodes and m (directed) links, and we want to route a package from a node u to another node v. Normally, it suffices to compute the shortest path from u to v. However, if some node x in this network fails, then our route cannot use x, and our task becomes to find the shortest path from u to v that does not go through x. Usually, there is only a very small number of failures. In this paper, we consider the simplest case, in which there is only one failed node.

The problem of constructing a DSO is well-studied: Demetrescu et al. [6] showed that given a directed graph $G = (V, E)$, there is a DSO which occupies $O(n^2 \log n)$ space, and can answer a query in $O(1)$ time. Duan and Zhang [8] improved the space complexity to $O(n^2)$, which is optimal for dense graphs (i.e. $m = \Theta(n^2)$).

Unfortunately, the oracle in [6] requires a large preprocessing time ($O(mn^2 + n^3 \log n)$). In real-life applications, the preprocessing time of the DSO is also very important. Bernstein and Karger [2, 3] improved this time bound to $\tilde{O}(mn)$. Note that the All-Pairs Shortest Paths (APSP) problem, which only asks the distances between each pair of vertices u, v, is conjectured to require $mn^{1-o(1)}$ time to solve [13]. Since we can solve the APSP problem by using a DSO, the preprocessing time $\tilde{O}(mn)$ is optimal in this sense.

However, if the edge weights are small positive integers (that do not exceed M), then the APSP problem can be solved in $\tilde{O}(n^{2.58}M^{0.7})$ time [20], which is significantly faster than $O(mn)$ for dense graphs with small weights (e.g. $M = O(1)$). Thus it might be possible to obtain better results than [3] in the regime of small integer edge weights. Weimann and Yuster [19] showed that for any constant $\alpha \in (0, 1)$, we can construct a DSO in $\tilde{O}(n^{1-\alpha+\omega}M)$ time. Here $\omega < 2.3728639$ is the exponent of matrix multiplication [10]. However, the query time for this oracle is $\tilde{O}(n^{1+\alpha})$, which is *superlinear*. Later, Grandoni and Williams [12] showed that for every constant $\alpha \in [0, 1]$, we can construct a DSO in $\tilde{O}(n^{\omega+1/2}M + n^{\omega+\alpha(4-\omega)}M)$ time, which answers each query in $\tilde{O}(n^{1-\alpha})$ time.

Recently, in an independent work, Chechik and Cohen [4] showed that a DSO with polylog(n) query time can be constructed in $\tilde{O}(Mn^{2.873})$ time, achieving *both* subcubic preprocessing time and polylogarithmic query time. The space complexity for their DSO is $\tilde{O}(n^{2.5})$.

1.1 Our Results

In this work, we show improved and simplified constructions of DSOs. We start with an observation.

▶ **Observation 1** (informal version). *If we have a DSO with preprocessing time P and query time Q, then we can build a DSO with preprocessing time $P + \tilde{O}(M \cdot n^2) \cdot Q$ and query time $O(1)$.*

For $\alpha = 0.2$, the oracle in [12] already achieves $\tilde{O}(n^{2.8729}M)$ preprocessing time and $O(n^{0.8})$ query time. Observation 1 implies that this query time can be brought down to $O(1)$.

Observation 1 can be proven by a close inspection of [3]: The algorithm in [3] for constructing a DSO picks $\tilde{O}(n^2)$ carefully chosen queries (u, v, x), such that the answers of all these queries can be computed in $\tilde{O}(mn)$ time. Then from these answers, we can easily compute a DSO with $O(1)$ query time. If, instead of computing these answers in $\tilde{O}(mn)$ time, we use the given DSO to answer these queries, the preprocessing time becomes $P + \tilde{O}(n^2) \cdot Q$. As the proof essentially repeats the arguments in [3], we leave it in Appendix A.

Our main result is a simple construction of DSOs with preprocessing time $\tilde{O}(n^{2.7233}M^2)$ and query time $O(1)$. If the input graph is undirected, we can achieve a better preprocessing time of $\tilde{O}(n^{2.6865}M^2)$.

▶ **Theorem 2.** *We can construct a DSO with $\tilde{O}(n^{2.7233}M^2)$ preprocessing time and $O(1)$ query time. Moreover, if the input graph is* undirected*, then we can construct a DSO with $\tilde{O}(n^{(3+\omega)/2}M^2) = \tilde{O}(n^{2.6865}M^2)$ preprocessing time and $O(1)$ query time. The construction algorithms are randomized and yield valid DSOs w.h.p.*[1]

[1] We say that an event happens *with high probability* (w.h.p.), if it happens with probability $1 - 1/n^C$, for some constant C that can be made arbitrarily large.

We remark that two drawbacks of our results compared to [4, 12, 19] are that it does not handle negative edge weights, and it has a worse (quadratic) dependence on M. However, our results have the currently best dependence on n, and is conceptually simple. Also, the space complexity of our DSO is $\tilde{O}(n^2)$, which is better than [4].

Non-unique shortest paths. A subtle technical issue is that the shortest paths in G may not be unique. Normally, if we perturb every edge weight by a small random value, then we can ensure that shortest paths are unique w.h.p. by the isolation lemma [15, 18]. However, the subcubic-time algorithms for shortest paths [16, 17, 20] depend crucially on the assumption that edge weights are small integers, so we cannot perform the random perturbation.

Therefore, we have to work without assuming the uniqueness of shortest paths. It turns out that Observation 1 is affected. Actually, if we assume the shortest paths are unique, we can build a DSO with preprocessing time $P + \tilde{O}(n^2) \cdot Q$ in the conclusion of Observation 1, regardless of the edge weights. However, without this assumption, Observation 1 somehow needs $\tilde{O}(Mn^2)$ queries to the slower DSO, instead of $\tilde{O}(n^2)$. See Remark 5 and Remark 6 for more details.

1.2 Notation

We mainly stick to the notation used in [7], namely:

- If p is a path, then $|p|$ denotes the number of edges in it, and $\|p\|$ denotes its length (i.e. the total weight of its edges).
- We use uv to denote the shortest path from u to v in the original graph, and $uv \diamond x$ the shortest path from u to v that does not go through x. In the case that there are multiple shortest paths, we will use e.g. "a path of the form $uv \diamond x$" to denote an arbitrary shortest path from uv in $G - x$ (i.e. the graph G with vertex x deleted). Note that if x is not in the original path uv, then $\|uv \diamond x\| = \|uv\|$.
- Let p be a path from u to v. For two vertices $a, b \in p$ such that a appears earlier than b, we say the interval $p[a, b]$ is the subpath from a to b, and $p(a, b)$ is the path $p[a, b]$ without its endpoints (a and b). If the path p is known in the context, then we may omit p and simply write $[a, b]$ or (a, b).

We define $\mathsf{MM}(n_1, n_2, n_3)$ as the complexity of multiplying an $n_1 \times n_2$ matrix and an $n_2 \times n_3$ matrix. Let a, b, c be real numbers, we define $\omega(a, b, c)$ be the infimum over all real numbers α such that $\mathsf{MM}(n^a, n^b, n^c) = O(n^\alpha)$. For any real number r, we have $\omega(1, 1, r) = \omega(1, r, 1) = \omega(r, 1, 1)$ [14], and we denote $\omega(r) = \omega(1, 1, r)$.

We also need the following adaptation of Zwick's APSP algorithm [20] (see also [12, Corollary 1]):

▶ **Theorem 3.** *Given a directed graph $G = (V, E)$ with edge weights in $\{1, 2, \ldots, M\}$, and an integer r, we can compute the distances between every pair of nodes whose shortest path uses at most r edges, in $\tilde{O}(rM \cdot \mathsf{MM}(n, n/r, n))$ time.*

Proof Sketch. We simply run the first $\lceil \log_{3/2} r \rceil$ iterations of the algorithm **rand-short-path** in [20]. The correctness of this algorithm is guaranteed by [20, Lemma 4.2]. ◀

2 Constructing a DSO in $\tilde{O}(n^{2.7233}M^2)$ Time

In this section, we prove Theorem 2.

▶ **Theorem 2.** *We can construct a DSO with $\tilde{O}(n^{2.7233}M^2)$ preprocessing time and $O(1)$ query time. Moreover, if the input graph is* undirected*, then we can construct a DSO with $\tilde{O}(n^{(3+\omega)/2}M^2) = \tilde{O}(n^{2.6865}M^2)$ preprocessing time and $O(1)$ query time. The construction algorithms are randomized and yield valid DSOs w.h.p.*

Given an integer r and a graph $G = (V, E)$, we define an *r-truncated DSO* to be a data structure that, when given a query (u, v, x), outputs the value $\min\{\|uv \diamond x\|, r\}$. In other words, an r-truncated DSO is a DSO which only needs to be correct when the path $uv \diamond x$ has length at most r. If this length is greater than r, it outputs r instead.

Inspired by Zwick's APSP algorithm [20], our main idea is to compute an r-truncated DSO for every $r = (3/2)^i$. Our strategies for small r and large r are completely different.

When r is small, the sampling approach in [12, 19] already suffices. Fix a particular query (u, v, x), we assume that $\|uv \diamond x\| \le r$. In particular, if we fix any path of the form $uv \diamond x$, then this path contains at most $r + 1$ vertices. Suppose we sample a graph by discarding each vertex w.p. $1/r$. With probability $\Omega(1/r)$, the resulting graph would "capture" this query in the sense that x is not in it but $uv \diamond x$ is completely included in it. Therefore, if we take $\tilde{O}(r)$ independent samples, and compute APSP for each sampled subgraph, we can deal with all queries w.h.p.

For large r, our idea is to compute a $(3/2)r$-truncated DSO from an r-truncated DSO. More precisely, given an r-truncated DSO with $O(1)$ query time, we can compute a $(3/2)r$-truncated DSO with $\tilde{O}(Mn/r)$ query time. First we sample a bridging set (cf. [20]) H of size $\tilde{O}(Mn/r)$. Let (u, v, x) be a query such that $r \le \|uv \diamond x\| \le (3/2)r$, then w.h.p. there is a "bridging vertex" $h \in H$ such that h is on some path of the form $uv \diamond x$, and both of the queries (u, h, x) and (h, v, x) are captured by the r-truncated DSO. If we iterate through H, we can answer the query (u, v, x) in $\tilde{O}(Mn/r)$ time. Then we use an "r-truncated" version of Observation 1 to transform this $(3/2)r$-truncated DSO into a new one with $O(1)$ query time.

2.1 Case I: r is Small

We make $\tilde{r} = \lceil 4Cr \ln n \rceil$ independent samples of graphs $G_1, G_2, \ldots, G_{\tilde{r}}$, where C is a large enough constant. The vertex set $V(G_i)$ of each graph is sampled by including each vertex independently w.p. $1 - 1/r$. The graph G_i is simply the induced subgraph of G on vertices $V(G_i)$. Then, for each $1 \le i \le \tilde{r}$, we compute all-pairs shortest paths of the graph G_i, but we only compute the shortest paths that use at most r edges. By Theorem 3, this step can be done in $\tilde{O}(rM \cdot \mathsf{MM}(n, n/r, n))$ time for each graph G_i. Alternatively, if the input graph is undirected, then this step can be done in $\tilde{O}(Mn^\omega)$ time [16, 17] for each G_i.

Consider a query (u, v, x), assume that $\|uv \diamond x\| \le r$, and fix any path of the form $uv \diamond x$. Let $1 \le i \le \tilde{r}$, we say i is *good* for the query (u, v, x), if both of the following hold.

- The graph G_i does not contain the failed vertex x.
- The graph G_i contains the path $uv \diamond x$ entirely.

For every i $(1 \le i \le \tilde{r})$, the probability that i is good for the particular query (u, v, x) is at least

$$(1/r) \cdot (1 - 1/r)^r \ge 1/4r \text{ (if } r \ge 2).$$

Given a query (u, v, x), we iterate through every i's such that $x \notin V(G_i)$, and take the smallest value among the distances from u to v in these graphs G_i. With high probability, there are only $\tilde{O}(1)$ valid i's such that $x \notin V(G_i)$, and we can preprocess this set of i's for every $x \in V$. Therefore the query time is $\tilde{O}(1)$.

The algorithm succeeds at a query (u, v, x) if there is an i that is good for (u, v, x). Since the G_i's are independent, the probability that there is an i good for (u, v, x) is at least

$$1 - (1 - 1/4r)^{\tilde{r}} \geq 1 - 1/n^C.$$

By a union bound over all n^3 triples of possible queries (u, v, x), it follows that our data structure is correct w.p. at least $1 - 1/n^{C-3}$, which is high probability.

In conclusion, there is an r-truncated DSO with $\tilde{O}(1)$ query time, whose preprocessing time is $\tilde{O}(\tilde{r} \cdot rM \cdot \mathsf{MM}(n, n/r, n))$ for directed graphs, and $\tilde{O}(\tilde{r} \cdot Mn^\omega)$ for undirected graphs.

2.2 An Observation

We need the following observation ("r-truncated" version of Observation 1), which roughly states that given an r-truncated DSO with preprocessing time P and query time Q, we can build an r-truncated DSO with preprocessing time $P + \tilde{O}(Mn^2) \cdot Q$ and query time $O(1)$. More formally, we have:

▶ **Observation 4.** *Let r be an integer, $G = (V, E)$ be an input graph whose edge weights are in $\{1, 2, \ldots, M\}$, and $\mathcal{D}$ be an arbitrary r-truncated DSO. We can construct* Fast($\mathcal{D}$)*, which is an r-truncated DSO with $O(1)$ query time and a preprocessing algorithm as follows.*

- *It first computes the distance matrix of G, and the outgoing shortest path trees rooted at each vertex.*
- *Then it invokes the preprocessing algorithm of $\mathcal{D}$ on the input graph G.*
- *At last, it makes $\tilde{O}(Mn^2)$ queries to $\mathcal{D}$, and spends $\tilde{O}(Mn^2)$ additional time to finish the preprocessing algorithm.*

We emphasize the following technical details that are not reflected in the informal statement of Observation 1. First, we need to compute the distance matrix and outgoing shortest path trees of G (henceforth the "APSP data" of G) before using $\mathcal{D}$. The APSP data can be computed in $\tilde{O}(Mn^{2.58})$ time [20], and in particular, the **wit-to-suc** algorithm in [20] describes how to compute the shortest path trees efficiently. Second, the preprocessing algorithm of $\mathcal{D}$ is called *only once*, and on the same graph G (on which we have already computed the APSP data). The reason that the second detail is important is: Suppose we have another routine that given an r-truncated DSO $\mathcal{D}$, constructs Extend($\mathcal{D}$) which is a $(3/2)r$-truncated DSO with a possibly large query time. Then given an 1-truncated DSO $\mathcal{D}^{\mathsf{start}}$, we can construct a (normal) DSO as follows:

$$\mathcal{D}^{\mathsf{final}} = \underbrace{\text{Fast}(\text{Extend}(\text{Fast}(\text{Extend}(\ldots \text{Extend}(\mathcal{D}^{\mathsf{start}})))))}_{O(\log n) \text{ times}}.$$

However, even if the preprocessing algorithm of Fast($\mathcal{D}$) invokes the preprocessing algorithm of $\mathcal{D}$ *twice*, the preprocessing algorithm of $\mathcal{D}^{\mathsf{final}}$ would invoke *a polynomial times* the preprocessing algorithm of $\mathcal{D}^{\mathsf{start}}$, which is too many. In contrast, if the preprocessing algorithm of both Fast($\mathcal{D}$) and Extend($\mathcal{D}$) only invokes the preprocessing algorithm of $\mathcal{D}$ once, then the preprocessing algorithm of $\mathcal{D}^{\mathsf{final}}$ would also invoke the preprocessing algorithm of $\mathcal{D}^{\mathsf{start}}$ only once, which is okay.

2.3 Case II: r is Large

Suppose we have an r-truncated DSO $\mathcal{D}$, which has preprocessing time P and query time $O(1)$. We show how to construct a $(3/2)r$-truncated DSO, which we name as Extend($\mathcal{D}$), with preprocessing time $P + O(n^2)$ and query time $\tilde{O}(nM/r)$. This is done by the following bridging set argument.

Let $\mathcal{P}$ be a set of paths that contains exactly one path of the form $uv \diamond x$, for every u, v, x such that $r \le \|uv \diamond x\| < (3/2)r$. This corresponds to the paths that $\mathcal{D}$ cannot deal with, but Extend($\mathcal{D}$) has to output the correct answer. Let $p = uv \diamond x \in \mathcal{P}$, $\mathsf{mid}(p)$ be the set of vertices $y \in p$ such that $\|p[u, y]\| < r$ and $\|p[y, v]\| < r$. (See Figure 1.) For every $y \in \mathsf{mid}(p)$, as $p[u, y]$, $p[y, v]$ are of the form $uy \diamond x$ and $yv \diamond x$ respectively, it follows that $\mathcal{D}$ can find $\|uy \diamond x\|$ and $\|yv \diamond x\|$ correctly. Moreover, $|\mathsf{mid}(p)| \ge r/3M$.

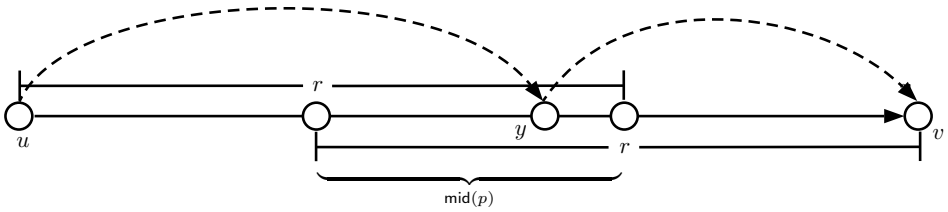


Figure 1 Example of a path $p = uv \diamond x$. If we can find a vertex $y \in \mathsf{mid}(p)$, then we can use $\mathcal{D}$ to compute $\|uy \diamond x\|$ and $\|yv \diamond x\|$, thus to compute the length of p.

Fix a large enough constant C, the preprocessing algorithm of Extend($\mathcal{D}$) is as follows: We preprocess $\mathcal{D}$, and then randomly sample a set H of vertices, where every vertex $v \in V$ is in H with probability $\min\{1, 3CM \ln n/r\}$ independently. We have $|H| = \tilde{O}(nM/r)$ w.h.p.

Fix $u, v, x \in V$, suppose $p = uv \diamond x$ and $r \le \|p\| < (3/2)r$. Then the probability that H hits $\mathsf{mid}(p)$ (i.e. $H \cap \mathsf{mid}(p) \ne \varnothing$) is at least

$$1 - (1 - 3CM \ln n/r)^{r/3M} \ge 1 - 1/n^C.$$

By a union bound over $O(n^3)$ paths in $\mathcal{P}$, it follows that w.h.p. H hits $\mathsf{mid}(p)$ for every path $p \in \mathcal{P}$.

The query algorithm for Extend($\mathcal{D}$) is as follows: Given a query (u, v, x), if $\mathcal{D}(u, v, x) < r$, then we output $\mathcal{D}(u, v, x)$; otherwise we output

$$\min\left\{(3/2)r, \min_{h \in H}\{\mathcal{D}(u, h, x) + \mathcal{D}(h, v, x)\}\right\}.$$

It is easy to see that Extend($\mathcal{D}$) is a correct $(3/2)r$-truncated DSO, has preprocessing time $P + O(n^2)$ and query time $\tilde{O}(nM/r)$.

2.4 Putting it Together

Let $a \in [0, 1]$ be a constant that we pick later, and $r = n^a$. To start, we invoke Section 2.1 to build an r-truncated DSO for $r = n^a$, which costs $\tilde{O}(r^2 M \cdot \mathsf{MM}(n, n/r, n))$ time for directed graphs or $\tilde{O}(r \cdot Mn^\omega)$ for undirected graphs. Then for every $1 \le i \le \lceil \log_{3/2}(Mn/r) \rceil$, suppose we have an $r(3/2)^{i-1}$-truncated DSO $\mathcal{D}^{i-1}$, we can construct $\mathcal{D}^i = \text{Fast}(\text{Extend}(\mathcal{D}^{i-1}))$ which is an $r(3/2)^i$-truncated DSO. This step costs $\tilde{O}(n^3 M^2/(r(3/2)^i))$ time. The preprocessing algorithm terminates when $i = i_\star = \lceil \log_{3/2}(Mn/r) \rceil = O(\log n)$, and we obtain an $r(3/2)^{i_\star}$-truncated DSO which is a (normal) DSO.

Case 1: the input graph is undirected. The total preprocessing time is

$$\tilde{O}\big(r \cdot Mn^{\omega} + n^3M^2/r\big) = \tilde{O}\Big(n^{\max\{\omega+a,3-a\}}M^2\Big).$$

When $a = (3-\omega)/2$, this time complexity is $\tilde{O}(n^{(3+\omega)/2}M^2) = \tilde{O}(n^{2.6865}M^2)$.

Therefore, given an undirected graph $G = (V, E)$, there is a DSO with $\tilde{O}(n^{2.6865}M^2)$ preprocessing time and $O(1)$ query time.

Case 2: the input graph is directed. The total preprocessing time is

$$\tilde{O}\big(r^2M \cdot \mathsf{MM}(n, n/r, n) + n^3M^2/r\big) = \tilde{O}\Big(n^{2a+\omega(1-a)}M + n^{3-a}M^2\Big).$$

(Recall that $\omega(1-a)$ is the exponent of multiplying an $n \times n^{1-a}$ matrix and an $n^{1-a} \times n$ matrix.)

Let $a = 0.276724$, then $1 - a = 0.723276$. By convexity of the function $\omega(\cdot)$ [14], we have

$$\omega(1-a) \leq \frac{(a-0.25)\omega(0.7) + (0.3-a)\omega(0.75)}{0.75-0.7}.$$

We substitute $\omega(0.7) \leq 2.154399$ and $\omega(0.75) \leq 2.187543$ [11], and obtain:

$$\omega(1-a) \leq 20 \cdot ((a-0.25) \cdot 2.154399 + (0.3-a) \cdot 2.187543) \leq 2.169829.$$

Therefore, given a directed graph $G = (V, E)$, there is a DSO with

$$\tilde{O}\Big(n^{\max\{2a+\omega(1-a),3-a\}}M^2\Big) = \tilde{O}(n^{2.723277}M^2).$$

preprocessing time and $O(1)$ query time.

3 Open Questions

The main open problem after this work is to improve the preprocessing time for DSOs. We believe it should be possible to overcome the issue that shortest paths are not unique, and improve the preprocessing time of the oracle specified in Observation 1 and Observation 4 to $P + \tilde{O}(n^2) \cdot Q$ (dropping the M factor). Then, we can build DSOs with $O(1)$ query time, and $\tilde{O}(n^{2.7233}M)$ preprocessing time (for directed graphs), or $\tilde{O}(n^{(3+\omega)/2}M)$ preprocessing time (for undirected graphs).

Can we improve the preprocessing time for directed graphs to $\tilde{O}(n^{2.58}M)$, matching the current best algorithm for APSP in directed graphs [20]? Can we improve the preprocessing time for undirected graphs to $\tilde{O}(n^{\omega}M)$, matching the nearly-optimal algorithm for APSP in undirected graphs [16, 17]?

References

1 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

2 Aaron Bernstein and David R. Karger. Improved distance sensitivity oracles via random sampling. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 34–43. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347087.

3 Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 – June 2, 2009*, pages 101–110. ACM, 2009. doi:10.1145/1536414.1536431.

4 Shiri Chechik and Sarel Cohen. Distance sensitivity oracles with subcubic preprocessing time and fast query time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1375–1388. ACM, 2020. doi:10.1145/3357713.3384253.

5 Erik D. Demaine, Gad M. Landau, and Oren Weimann. On Cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014. doi:10.1007/s00453-012-9683-x.

6 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008. doi:10.1137/S0097539705429847.

7 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 506–515. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496826.

8 Ran Duan and Tianyi Zhang. Improved distance sensitivity oracles via tree partitioning. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures – 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 – August 2, 2017, Proceedings*, volume 10389 of *Lecture Notes in Computer Science*, pages 349–360. Springer, 2017. doi:10.1007/978-3-319-62127-2_30.

9 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 165–169. IEEE Computer Society, 1982. doi:10.1109/SFCS.1982.39.

10 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

11 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1029–1046. SIAM, 2018. doi:10.1137/1.9781611975031.67.

12 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 748–757. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.17.

13 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252. SIAM, 2018. doi:10.1137/1.9781611975031.80.

14 Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theor. Comput. Sci.*, 23:171–185, 1983. doi:10.1016/0304-3975(83)90054-3.

15 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

16 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995. doi:10.1006/jcss.1995.1078.

17 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 605–615. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814635.

18 Noam Ta-Shma. A simple proof of the isolation lemma. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:80, 2015. URL: https://eccc.weizmann.ac.il/report/2015/080.

19 Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Trans. Algorithms*, 9(2):14:1–14:13, 2013. doi:10.1145/2438645.2438646.

20 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

A Proof of Observation 4 and Observation 1

In this section, we prove Observation 4. Note that Observation 1 follows from Observation 4 by setting $r = +\infty$.

▶ **Observation 4.** *Let r be an integer, $G = (V, E)$ be an input graph whose edge weights are in $\{1, 2, \ldots, M\}$, and $\mathcal{D}$ be an arbitrary r-truncated DSO. We can construct* Fast($\mathcal{D}$)*, which is an r-truncated DSO with $O(1)$ query time and a preprocessing algorithm as follows.*

- *It first computes the distance matrix of G, and the outgoing shortest path trees rooted at each vertex.*
- *Then it invokes the preprocessing algorithm of $\mathcal{D}$ on the input graph G.*
- *At last, it makes $\tilde{O}(Mn^2)$ queries to $\mathcal{D}$, and spends $\tilde{O}(Mn^2)$ additional time to finish the preprocessing algorithm.*

Let $T_{\mathsf{out}}(u)$ be the outgoing shortest path tree rooted at u. In this section, the notation uv denotes the shortest path from u to v *in the tree* $T_{out}(u)$. In particular, for any $x \in uv$, the path ux is guaranteed to be a subpath of uv.

A.1 The Preprocessing Algorithm

We review and slightly modify the preprocessing algorithm of [3]. For convenience, we denote $\|p\|_r = \min\{\|p\|, r\}$ for any path p and number r.

We define a path to be *good* if it is a subpath of some shortest path in the shortest path trees. In other words, for any vertices $u, v, x \in V$ such that $x \in uv$, we say the subpath $(uv)[x, v]$ is *good*. Note that if the shortest paths in G are unique, then the set of good paths coincides with the set of shortest paths in G. Also note that there are only $O(n^3)$ good paths even if the shortest paths are not unique.

Assigning priorities. We assign each vertex a *priority*, which is independently sampled from the following distribution: for any positive integer c, each vertex has priority c w.p. $1/2^c$. Denote $c(v)$ the priority of the vertex v. With high probability, all of the following are true:

- The maximum priority is $O(\log n)$.
- For every $c \le O(\log n)$, there are $\tilde{O}(n/2^c)$ vertices with priority c.
- Let C be a large enough constant. For every good path p with at least $C \cdot 2^c \log n$ edges, there is a vertex on p whose priority is greater than c.

In the following discussions, we will assume that all of the above assumptions hold.

Fix a pair $u, v \in V$, let s_i be the first vertex in uv with priority $\geq i$, and t_i be the last such vertex. Then we can write the path uv as

$$u \rightsquigarrow s_1 \rightsquigarrow s_2 \rightsquigarrow \ldots \rightsquigarrow s_{O(\log n)} \rightsquigarrow t_{O(\log n)} \rightsquigarrow \ldots \rightsquigarrow t_1 \rightsquigarrow v.$$

We say that the vertices u, v, s_i, t_i are *key vertices*, and the i-th key vertex is denoted as k_i. Then the path uv can also be written as

$$u = k_0 \rightsquigarrow k_1 \rightsquigarrow \ldots \rightsquigarrow k_{O(\log n)} = v.$$

It is important to see that

$$|(uv)[k_i, k_{i+1}]| \leq C \cdot 2^{\min\{c(k_i), c(k_{i+1})\}} \log n \tag{1}$$

for every valid i, as otherwise there will be another key vertex between k_i and k_{i+1}.

Data structure for quick location. Suppose we are given a query (u, v, x), the first thing we should do is to "locate" x, i.e. find the key vertices $k_i, k_{i+1} \in uv$ such that $x \in (uv)[k_i, k_{i+1}]$. We will utilize the following data (cf. [2]).

For every $u, v \in V$, we compute

- $\mathsf{CL}[u, v, c]$ (for "center left"): the first vertex in uv with priority at least c; and
- $\mathsf{CR}[u, v, c]$ (for "center right"): the last vertex in uv with priority at least c.

It is easy to compute these numbers in $\tilde{O}(n^2)$ time: for each priority c and each vertex $u \in V$, we simply perform a depth-first search on the outgoing shortest path tree $T_{\mathsf{out}}(u)$ rooted at u to compute all $\mathsf{CL}[u, \cdot, c]$ and $\mathsf{CR}[u, \cdot, c]$.

We also compute a structure called BCP (for "biggest center priority"). In this structure, for each vertex $u \in V$, we preprocess the outgoing shortest path tree $T_{\mathsf{out}}(u)$ rooted at u in $\tilde{O}(n)$ time such that given any vertices $x, y \in V$, we can find the biggest priority of any vertex on the path from x to y in $T_{\mathsf{out}}(u)$ in $O(1)$ time [5]. Then, given any good path p (p is specified by vertices u, v, x and $p = (uv)[x, v]$), we can find the biggest priority of any vertex on p in $O(1)$ time.

In addition, for every $u, v \in V$, we store the key vertices on uv into a hash table of size $O(\log n)$. Given a vertex x, we can output whether x is among these key vertices on uv in $O(1)$ worst-case time [9].

Data structure for avoiding a failure. We use $\mathcal{D}$ to preprocess the input graph. Then we compute the following data:

(Data a) For every $u, v \in V$, and every $1 \leq i \leq \min\{M \cdot C \cdot 2^{c(u)} \log n, |uv|\}$, let x_i be the i-th vertex in the path uv. (Here u is the 0-th vertex.) We compute and store the value $\|uv \diamond x_i\|_r$. Symmetrically, let x_{-i} be the *last* i-th vertex in the path vu (not uv!), for every $1 \leq i \leq \min\{M \cdot C \cdot 2^{c(u)} \log n, |vu|\}$, we compute and store $\|vu \diamond x_{-i}\|_r$.

(Data b) For every $u, v \in V$ and consecutive key vertices $k_i, k_{i+1} \in uv$, let y be the vertex in the portion $k_i \rightsquigarrow k_{i+1}$ that maximizes $\|uv \diamond y\|_r$. We compute and store $\|uv \diamond y\|_r$.

(Data c) For every $u, v \in V$ and key vertex $k_i \in uv$, we compute and store $\|uv \diamond k_i\|_r$.

For each priority $c \leq \tilde{O}(1)$, there are $\tilde{O}(n/2^c)$ vertices u whose priority is exactly c. In (Data a), we make $\tilde{O}(nM2^c)$ queries for each such u ($\tilde{O}(M2^c)$ queries for each $v \in V$). Therefore in total, we make $\tilde{O}(Mn^2)$ queries in (Data a). We will show in Appendix A.3 that we can compute (Data b) using $\tilde{O}(n^2)$ queries to $\mathcal{D}$ and $\tilde{O}(n^2)$ additional time. (Data c) can be computed in $\tilde{O}(n^2)$ queries easily.

▶ Remark 5. If shortest paths in G are unique, then in (Data a) we only need to store $\|uv \diamond x_i\|_r$ and $\|vu \diamond x_{-i}\|_r$ for $i \le C \cdot 2^{c(u)} \log n$ (shaving off a factor of M). The reason will be evident when we see the query algorithm.

A.2 The Query Algorithm

Let (u, v, x) be a query. We check whether $x \in uv$ in $O(1)$ time on the shortest path tree $T_{\mathsf{out}}(u)$. If $x \notin uv$, then it is easy to see that $\|uv \diamond x\|_r = \|uv\|_r$.

We check whether x is a key vertex on uv (that is, $x = k_i$ for some i), using the hash tables. If this is the case, we return $\|uv \diamond x\|_r$ stored in (Data c) immediately.

Otherwise, we start by finding two consecutive key vertices $k_i, k_{i+1} \in uv$ such that $x \in [k_i, k_{i+1}]$. Recall that, if ℓ is the biggest priority of any vertex on uv, then the key vertices on uv are

$$(u =)\mathsf{CL}[u,v,1] \rightsquigarrow \mathsf{CL}[u,v,2] \rightsquigarrow \ldots \rightsquigarrow \mathsf{CL}[u,v,\ell] \rightsquigarrow \mathsf{CR}[u,v,\ell] \rightsquigarrow \ldots \rightsquigarrow \mathsf{CR}[u,v,2] \rightsquigarrow \mathsf{CR}[u,v,1](= v).$$

Therefore, we can find k_i in $O(1)$ time using the following procedure. Let c_1 be the biggest priority of any vertex in ux (which coincides the $[u, x]$ portion of uv), and c_2 be the biggest priority of any vertex in the $[x, v]$ portion of uv. We can compute c_1, c_2 from the structure BCP in $O(1)$ time. If $c_2 = \ell$, then x is in the range $(u, \mathsf{CR}[u, v, \ell])$, so $k_i = \mathsf{CL}[u, v, c_1]$. Otherwise, x is in the range $(\mathsf{CR}[u, v, \ell], v)$, so $k_i = \mathsf{CR}[u, v, c_2 + 1]$. We can find k_{i+1} similarly.

By (1), if $k_i = u$, then $|(uv)[u, x]| \le C \cdot 2^{c(u)} \log n$, and we can look up the value $\|uv \diamond x\|_r$ from (Data a) directly. Similarly, if $k_{i+1} = v$ then we can also look up $\|uv \diamond x\|_r$ from (Data a).

Now we assume that $k_i \ne u$ and $k_{i+1} \ne v$. A crucial observation is that

$$\|uv \diamond x\| = \min\{\|uk_{i+1} \diamond x\| + \|k_{i+1}v\|, \|uk_i\| + \|k_iv \diamond x\|, \|uv \diamond y\|\}, \tag{2}$$

where y is the vertex in $[k_i, k_{i+1}]$ that maximizes $\|uv \diamond y\|$. The proof of (2) is as follows:

- **(i)** If there is a path of the form $uv \diamond x$ that goes through k_i, then $\|uv \diamond x\| = \|uk_i\| + \|k_iv \diamond x\|$.
- **(ii)** If there is a path of the form $uv \diamond x$ that goes through k_{i+1}, then $\|uv \diamond x\| = \|uk_{i+1} \diamond x\| + \|k_{i+1}v\|$.
- **(iii)** If every path of the form $uv \diamond x$ does not through either k_i or k_{i+1}, then every path of the form $uv \diamond x$ avoids the entire portion of $k_i \rightsquigarrow k_{i+1}$, thus also avoids y. We have $\|uv \diamond x\| \ge \|uv \diamond y\|$. But $\|uv \diamond y\| \ge \|uv \diamond x\|$ by definition of y, thus $\|uv \diamond x\| = \|uv \diamond y\|$.

It is easy to see that a similar equation holds for r-truncated DSOs:

$$\|uv \diamond x\|_r = \min\{\|uk_{i+1} \diamond x\|_r + \|k_{i+1}v\|, \|uk_i\| + \|k_iv \diamond x\|_r, \|uv \diamond y\|_r, r\}, \tag{3}$$

where y is any vertex in $[k_i, k_{i+1}]$ that maximizes $\|uv \diamond y\|_r$.

Recall that we already know the values $\|uk_i\|$ and $\|k_{i+1}v\|$. To compute $\|uk_{i+1} \diamond x\|_r$, we note that if x is the last j-th vertex in uk_{i+1}, then $j \le C \cdot 2^{c(k_{i+1})} \log n$. Therefore we can look up the value of $\|uk_{i+1} \diamond x\|_r$ from (Data a). Similarly, to compute $\|k_iv \diamond x\|_r$, we note that if x is the j-th vertex in the $k_i \rightsquigarrow v$ portion of the path uv, then $j \le C \cdot 2^{c(k_i)} \log n$. However, k_iv may be different from the $k_i \rightsquigarrow v$ portion of the path uv. Nevertheless, since $\|k_ix\| = \|(uv)[k_i, x]\| \le M \cdot C \cdot 2^{c(k_i)} \log n$, we also have $|k_ix| \le M \cdot C \cdot 2^{c(k_i)} \log n$, so we can still look up the value $\|k_iv \diamond x\|_r$ in (Data a). Finally, we can look up $\|uv \diamond y\|_r$ from (Data b).

We can see that the query time is $O(1)$.

▶ Remark 6. If shortest paths in G are unique, then the path $k_i v$ actually coincides with the $k_i \rightsquigarrow v$ portion of the path uv. Therefore, $|k_i x| = |(uv)[k_i, x]| \le C \cdot 2^{c(k_i)} \log n$. In this case we do not need to "sacrifice" a factor of M in (Data a): We can look up $\|k_i v \diamond x\|_r$ even if we only stored the values $\|k_i v \diamond x_{i'}\|_r$ for $i' \le C \cdot 2^{c(k_i)} \log n$, as in Remark 5.

A.3 Computing (Data b)

We will use the following notation. Let p be a path from u to v which is fixed in context, and a, b be two vertices in p. We will say that $a < b$ if $|p[u, a]| < |p[u, b]|$, i.e. a appears strictly before b on the path p. Similarly, $a > b$, $a \le b$, $a \ge b$ mean $|p[u, a]| > |p[u, b]|$, $|p[u, a]| \le |p[u, b]|$, $|p[u, a]| \ge |p[u, b]|$ respectively.

Let $u, v \in V$ and $s < t$ be two vertices on the path uv. Let $y \in (uv)[s, t]$ be the vertex in $(uv)[s, t]$ which maximizes $\|uv \diamond y\|_r$. We first show that *assuming we have built some oracles*, we can find this vertex y in $O(\log n)$ oracle calls and $O(\log n)$ additional time. The idea is to use a binary search described in [3, Section 6].

▶ **Lemma 7.** *Let r be an integer, $u, v \in V$, p be the path uv, and s, t be two vertices on p such that $u < s < t < v$. Suppose we have the following oracles, each with $O(1)$ query time:*

- *an oracle that given a vertex $x \in p[s, t]$, outputs $\|ut \diamond x\|_r$;*
- *an oracle that given a vertex $x \in p[s, t]$, outputs $\|sv \diamond x\|_r$;*
- *an oracle that given an interval $p[s', t']$ such that $s \le s' \le t' \le t$, outputs a vertex $x \in p[s', t']$ that maximizes the value $\|ut \diamond x\|_r$.*

Then we can find a vertex $y \in p[s, t]$ which maximizes $\|uv \diamond y\|_r$ in $O(\log n)$ time.

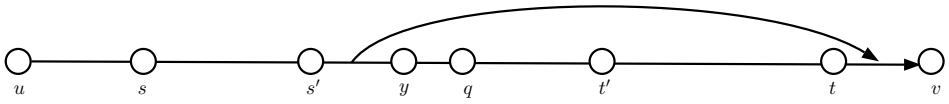


Figure 2 If every path of the form $sv \diamond y$ does not go through t, then every path of the form $sv \diamond y$ does not go through the whole interval $p[q, t']$.

Proof. For any $y \in p[s, t]$, we denote

$$h(y) = \min\{\|ut \diamond y\|_r + \|tv\|, \|us\| + \|sv \diamond y\|_r, r\}.$$

By (3), we have $\|uv \diamond y\|_r = \min\{h(y), \|uv \diamond y^\star\|_r\}$ where $y^\star$ is some vertex independent of y. Thus it suffices to find some $y \in p[s, t]$ that maximizes $h(y)$.

We use a binary search. Assume that we know the optimal y is in some interval $p[s', t']$, where $s \le s' < t' \le t$. (Initially we set $s' = s$ and $t' = t$.) If $|p[s', t']| = O(1)$ then we can use brute force to find a vertex $y \in p[s', t']$ that maximizes $h(y)$. Otherwise let q be the middle point of $p[s', t']$, and we use the third oracle to find a vertex $y \in p[s', q]$ that maximizes $\|ut \diamond y\|_r$. There are two cases:

- If $\min\{\|ut \diamond y\|_r + \|tv\|, r\} = h(y)$, then we can restrict our attention to the interval $p[q, t']$. This is because for every vertex $x \in p[s', q]$,

$$h(x) \le \min\{\|ut \diamond x\|_r + \|tv\|, r\} \le \min\{\|ut \diamond y\|_r + \|tv\|, r\} \le h(y).$$

- Otherwise, $h(y) = \|us\| + \|sv \diamond y\|_r$, and every path of the form $sv \diamond y$ does not go through t. Therefore every path of the form $sv \diamond y$ avoids every vertex in $p[q, t']$. (See Figure 2.) For every vertex $x \in p[q, t']$,

$$h(x) \le \|us\| + \|sv \diamond y\| \le h(y).$$

It follows that we can restrict our attention to the interval $[s', q]$ now.
Therefore, we can always shrink the length of our candidate interval $p[s', t']$ by a half. It follows that we can find the desired vertex y in $O(\log n)$ time. ◀

Now we show how to compute (Data b) in $\tilde{O}(n^2)$ time (assuming that (Data a) is ready). The most crucial ingredient is the following Range Maximum Query (RMQ) structures (used in the third item of Lemma 7).

For every $u, v \in V$, consider the following sequence (of length $\ell = \min\{|uv| - 1, C \cdot 2^{c(v)} \log n\}$):

$$(\|uv \diamond x_{-1}\|_r, \|uv \diamond x_{-2}\|_r, \ldots, \|uv \diamond x_{-\ell}\|_r),$$

where x_{-i} denotes the last i-th vertex in the path uv (v is the last 0-th). We build an RMQ structure of this sequence, which given a query (s, t) $(1 \le s \le t \le \ell)$, outputs a number $i \in [s, t]$ that maximizes $\|uv \diamond x_{-i}\|_r$. After we compute the above sequence, this data structure can be preprocessed in $O(\ell)$ time, and each query costs $O(1)$ time [1].

For every priority $c \le O(\log n)$, there are $\tilde{O}(n/2^c)$ vertices v of this priority, and for each vertex v we construct n RMQ structures (one for each $u \in V$) on length-$\tilde{O}(2^c)$ sequences. The total size of these RMQ structures is

$$\sum_{c=1}^{O(\log n)} \tilde{O}(n/2^c) \cdot n \cdot \tilde{O}(2^c) = \tilde{O}(n^2).$$

Therefore, these RMQ structures can be preprocessed in $\tilde{O}(n^2)$ time. (Note that every element $\|uv \diamond x_{-i}\|_r$ is already computed in (Data a).)

To compute (Data b), we enumerate u, v, k_i, k_{i+1} where k_i, k_{i+1} are consecutive key vertices in uv. There are $\tilde{O}(n^2)$ possible combinations of (u, v, k_i, k_{i+1}). As argued in Appendix A.2, we know that the following data are already computed in (Data a):

- $\|uk_{i+1} \diamond x\|_r$, for any $x \in (uv)[k_i, k_{i+1}]$;
- $\|k_i v \diamond x\|_r$, for any $x \in (uv)[k_i, k_{i+1}]$.

Since uk_{i+1} is a prefix of uv (as defined in the outgoing shortest path trees), we also have the following RMQ oracles constructed above:

- An oracle that given any interval $[s', t']$ on the path uv such that $k_i \le s' \le t' \le k_{i+1}$, finds the vertex $y \in [s', t']$ that maximizes $\|uk_{i+1} \diamond y\|_r$ in $O(1)$ time.

It follows from Lemma 7 that we can find a vertex $y \in (uv)[k_i, k_{i+1}]$ that maximizes $\|uv \diamond y\|_r$ in $O(\log n)$ time. The total time for computing (Data b) is thus $\tilde{O}(n^2)$.

Fine-Grained Complexity of Regular Expression Pattern Matching and Membership

Philipp Schepper
CISPA – Helmholtz Center for Information Security, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, Saarland Informatics Campus, Saarbrücken, Germany
philipp.schepper@cispa.saarland

Abstract

The currently fastest algorithm for regular expression pattern matching and membership improves the classical $\mathcal{O}(nm)$ time algorithm by a factor of about $\log^{3/2} n$. Instead of focussing on general patterns we analyse homogeneous patterns of bounded depth in this work. For them a classification splitting the types in easy (strongly sub-quadratic) and hard (essentially quadratic time under SETH) is known. We take a very fine-grained look at the hard pattern types from this classification and show a dichotomy: few types allow super-poly-logarithmic improvements while the algorithms for the other pattern types can only be improved by a constant number of log-factors, assuming the Formula-SAT Hypothesis.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Fine-Grained Complexity, Regular Expression, Pattern Matching, Dichotomy

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.80

Related Version A full version of this paper is available at `https://arxiv.org/abs/2008.02769`. All presented lower bounds and an alternative proof of the upper bounds for pattern matching using the polynomial method are contained in the author's Master's thesis.

Funding Supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.

Acknowledgements I thank Karl Bringmann for the supervision during the research for my Master's Thesis which this paper is based on and especially the pointer to Batch-OV which simplified the upper bounds extremely.

1 Introduction

Regular expressions with the operations alternative $|$, concatenation $\circ$, Kleene Plus $+$, and Kleene Star $\star$ are used in many fields of computer science. For example to search in texts and files or to replace strings by other strings as the unix tool `sed` does. But they are also used to analyse XML files [17, 18], for network analysis [12, 24], human computer interaction [13], and in biology to search for proteins in DNA sequences [16, 20].

The most intuitive problem for regular expressions is the *membership* problem. There we ask whether a given text t can be generated by a given regular expression p, i.e. is $t \in \mathcal{L}(p)$? We also call p a pattern in the following. A similar problem is the *pattern matching* problem, where we are interested whether some substring of the given text t can be matched by p. To simplify notation we define the matching language of p as $\mathcal{M}(p) \coloneqq \Sigma^* \mathcal{L}(p) \Sigma^*$. Then we want to check whether $t \in \mathcal{M}(p)$. The standard algorithm for both problems runs in time $\mathcal{O}(nm)$ where n is the text length and m the pattern size [22].

Based on the "Four Russians" trick Myers showed an algorithm with running time $\mathcal{O}(nm/\log n)$ [19]. This result was improved to an $\mathcal{O}(nm \log\log n/\log^{3/2} n)$ time algorithm by Bille and Thorup [5]. Although for several special cases of pattern matching and

Table 1 Hard pattern types that have to be considered.

Pattern matching / Membership	$\circ\star$	$\circ\vert\circ$	$\circ\vert+$	$\circ+\circ$	$\circ+\vert$	$\vert\circ\vert$ / $+\vert\circ\vert$	$\vert\circ+$ / $+\vert\circ+$	$\vert+\vert\circ$

membership improved sub-quadratic time algorithms have been given [3, 11, 14], it remained an open question whether there are truly sub-quadratic time algorithms for the general case. The first conditional lower bounds were shown by Backurs and Indyk [4]. They introduced so-called homogenous patterns and classified their hardness into easy, i.e. strongly sub-quadratic time solvable, and hard, requiring essentially quadratic time assuming the Strong Exponential Time Hypothesis (SETH). This classification of Backurs and Indyk was completed by a dichotomy for all homogeneous pattern types by Bringmann, Grønlund, and Larsen [8]. They reduced the hardness of all hard pattern types to the hardness of few pattern types of bounded depth. By this it was sufficient to check few cases instead of infinitely many.

To understand what a homogeneous pattern is, we observe that one can see patterns as rooted and node labeled trees where the inner nodes correspond to the operations of the pattern. Then a pattern is homogenous if the operations on each level of the tree are equal. The type of the pattern is the sequence of operations from the root to the leaves. See Section 2 for a formal introduction.

But as SETH rules out only polynomial improvements, super-poly-logarithmic runtime improvements are still feasible. Such improvements are know for Orthogonal Vectors (OV) [2, 9], for example, although there is a known conditional lower bound based on SETH. But for pattern matching and membership no faster algorithms are known. By a reduction from Formula-SAT Abboud and Bringmann showed that in general pattern matching and membership cannot be solved in time $\mathcal{O}(nm/\log^{7+\epsilon} n)$ under the Formula-SAT Hypothesis [1].

For Formula-SAT one is given a De Morgan formula F over n inputs and size s, i.e. the formula is a tree where each inner gate computes the AND or OR of two other gates and each of the s leaves is labeled with one of the n variables or their negation. The task is to find a satisfying assignment for F. While the naive approach takes time $\mathcal{O}(2^n s)$ to evaluate F on all possible assignments, there are polynomial improvements for formulas of size $s = o(n^3)$ [10, 15, 21]. But despite intense research there is currently no faster algorithm known for $s = n^{3+\Omega(1)}$. Thus it seem reasonable to assume the following hypothesis:

▶ **Hypothesis 1.1** (Formula-SAT Hypothesis (FSH) [1]). *There is no algorithm that can solve Formula-SAT on De Morgan formulas of size $s = n^{3+\Omega(1)}$ in $\mathcal{O}(2^n/n^\epsilon)$ time, for some $\epsilon > 0$, in the Word-RAM model.*

Although the new lower bound of $\mathcal{O}(nm/\log^{7+\epsilon} n)$ is quite astonishing since before only polynomial improvements have been ruled out, the bound is for the general case. It remained an open question whether it also holds for homogeneous patterns of bounded depth. Using the results by Bringmann, Grønlund, and Larsen [8] relating the hardness of different pattern types to each other, it suffices to check the pattern types in Table 1 for the corresponding problem.

We answer this last question and give a dichotomy for these hard pattern types: For few pattern types we give the currently fastest algorithm for pattern matching and membership. For the remaining patterns we show improved lower bounds of the form $\Omega(nm/\log^c n)$.

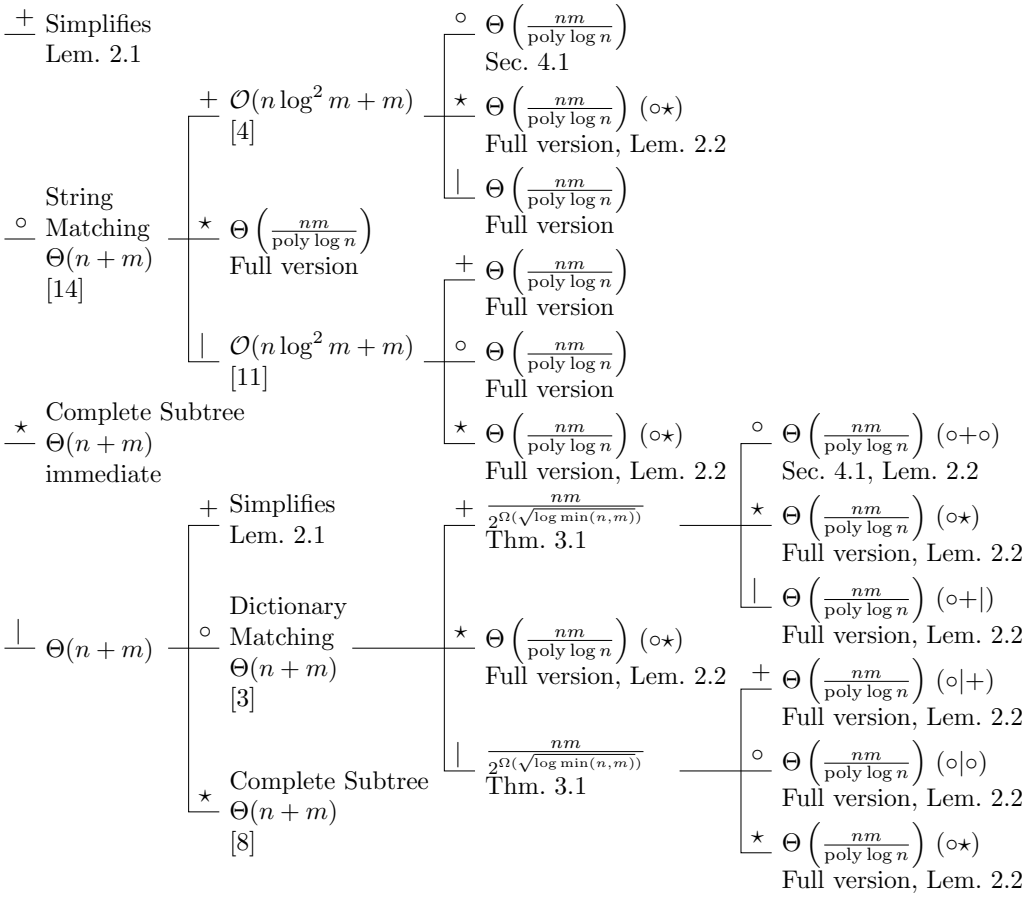


Figure 1 The classification of the patterns for pattern matching.

▶ **Theorem 1.2.** *For texts of length n and patterns of size m we have the following time bounds for the stated problems:*

- $nm/2^{\Omega(\sqrt{\log \min(n,m)})}$ *for* $|\circ|$*- and* $|\circ+$*-pattern matching, and* $+|\circ|$*- and* $+|\circ+$*-membership*
- $\Theta(nm/\operatorname{poly}\log n)$ *for pattern matching and membership with types* $\circ+|$*,* $\circ|+$*,* $\circ+\circ$*,* $\circ|\circ$*, and* $\circ\star$ *and for* $|+|\circ$*-membership, unless FSH is false.*

This dichotomy result gives us a simple classification for the hard pattern types. Depending on the pattern type one can decide if there is super-poly-logarithmic algorithm, or if even the classical algorithm is optimal up to a constant number of log-factors. See Figure 1 for an overview of the results for pattern matching. Further, the dichotomy shows that the type of a pattern has a larger impact on the hardness than the depth. The alternative as outer operation of the "easier" patterns allows us to split the pattern into independent sub-patterns. This is crucial for the speed-up since pattern matching for $\circ+$ and $\circ|$ is near-linear time solvable [4, 11]. Contrary almost all hard pattern types have a concatenation as outer operation which does not allow this decomposition into independent problems. Further, the length of the matched texts can vary largely. The pattern $(a \mid aba)(b \mid bca)(a \mid ab)$, for example, can match strings of length 3 to 8. We exploit both properties in our reductions, especially to encode a boolean OR.

In Section 2 we give a formal definition of homogeneous patterns and state the problems we start reducing from and the ones we reduce to. We show the algorithms for the upper bounds in Section 3. In Section 4 we give the improved lower bounds for pattern matching while the ones for membership are given in Section 5.

2 Preliminaries

Regular expressions. Recall, that patterns over a finite alphabet Σ are build recursively from other patterns using the operations $|$, $\circ$, $+$, and $\star$. We construct the patterns and the language of each pattern (i.e. the set of words matched by the pattern) as follows. Each symbol $\sigma \in \Sigma$ is a pattern representing the language $\mathcal{L}(\sigma) = \{\sigma\}$. Let in the following p_1 and p_2 be two patterns. For the alternative operation we define $\mathcal{L}(p_1 \mid p_2) = \mathcal{L}(p_1) \cup \mathcal{L}(p_2)$. For the concatenation we define $\mathcal{L}(p_1 \circ p_2) = \{w_1 w_2 \mid w_1 \in \mathcal{L}(p_1) \wedge w_2 \in \mathcal{L}(p_2)\}$. For the Kleene Plus we set $\mathcal{L}(p_1^+) = \{w \mid \exists k \geq 1 : \exists w_1, \ldots, w_k \in \mathcal{L}(p_1) : w = w_1 \ldots w_k\}$. With ε as the empty word we have $\mathcal{L}(p_1^\star) = \mathcal{L}(p_1^+) \cup \{\varepsilon\}$ for the Kleene Star.

Based on this construction it is easy to see patterns as rooted and node-labeled trees where each inner node is labeled by an operation and the leaves are labeled by symbols. We call this tree the *parse tree* of a pattern in the following. Then each node is connected to the node representing the sub-pattern p_1 and also for p_2 in the case of the binary operations $\circ$ and $|$. We extend the definition of the alternative and the concatenation in the natural way to more than two sub-patterns. To simplify notation we omit the symbol $\circ$ from the patterns in the following. We define the *size* of a pattern to be the number of inner nodes plus the number of leaves in the parse tree.

We call a pattern *homogeneous* if for each level of the parse tree, all inner nodes are labeled with the same operation. We define the *type* of a homogeneous pattern p to be the sequence of operations from the root of the parse tree of p to the deepest leaf. The *depth* of a pattern is the depth of the tree, which is equal to the number of operations in the type. For example, the pattern $[(abc \mid c)(a \mid dc)c(db \mid c \mid bd)]^+$ is of type $+\circ|\circ$ and has depth 4.

Relations between pattern types. Backurs and Indyk showed in [4] the first quadratic time lower bound for several homogeneous patterns based on SETH. This classification was completed by the dichotomy result of Bringmann, Grønlund, and Larsen in [8]. As there are infinitely many homogeneous pattern types, they showed linear-time reductions between different pattern types. By these reductions lower bounds also transfer to other (more complicated) pattern types and faster algorithms also give improvements for other (equivalent) pattern types.

▶ **Lemma 2.1** (Lemma 1 and Lemma 8 in the full version of [8]). *For any type T, applying any of the following rules yields a type T' such that both are equivalent for pattern matching and membership under linear-time reductions, respectively:*

- *For pattern matching: remove prefix $+$ and replace prefix $|+$ by $|$.*
- *For membership: replace any substring $+|+$ by $+|$ and replace prefix $r\star$ by $r+$ for any $r \in \{+, |\}^*$.*
- *For both problems: replace any substring pp, for any $p \in \{\circ, |, \star, +\}$, by p.*

We say that T simplifies *if one of these rules applies. Applying these rules in any order will eventually lead to an unsimplifiable type.*

▶ **Lemma 2.2** (Lemma 6 and Lemma 9 in the full version of [8]). *For types T and T', there is a linear-time reduction from T-pattern matching/membership to T'-pattern matching/membership if one of the following sufficient conditions holds:*

- *T is a prefix of T',*
- *we may obtain T' from T by replacing a $\star$ by $+\star$,*
- *we may obtain T' from T by inserting a $|$ at any position,*
- *only for membership: T starts with $\circ$ and we may obtain T' from T by prepending a $+$ to T.*

Together with the already known sub-quadratic time algorithms for various pattern types [3, 4, 8, 11, 14], it suffices to check the remaining cases in Table 1 to get a fine-grained dichotomy for the hard pattern types (i.e. the ones requiring essentially quadratic time under SETH).

Hypothesis. As mentioned in the introduction, we follow the ideas of Abboud and Bringmann in [1] and show reductions from Formula-SAT to pattern matching to prove lower bounds. Likewise as in their result, we also start from the intermediate problem Formula-Pair: Given a *monotone* De Morgan formula F with size s, that is a De Morgan formula where each leaf is labeled with a variable, i.e. no negation allowed, and each variable is used only once. Further, one is given two sets A, B of half-assignments to $s/2$ variables of F with $|A| = n$ and $|B| = m$. The task is to find a pair $a \in A, b \in B$ such that $F(a, b) = \mathsf{true}$.

There is an intuitive reduction from Formula-SAT to Formula-Pair as shown in [1]. Thus, FSH implies the following hypothesis, which we prove in Appendix A:

▶ **Hypothesis 2.3** (Formula-Pair Hypothesis (FPH)). *For all $k \geq 1$, there is no algorithm that can solve Formula-Pair for a monotone De Morgan formula F of size s and sets $A, B \subseteq \{0,1\}^{s/2}$ of size n and m, respectively, in time $\mathcal{O}(nms^k/\log^{3k+2} n)$ in the Word-RAM model.*

Batch-OV. For the upper bounds we transform texts and patterns into bit-vectors such that they are orthogonal if and only if the text is matched by the pattern. This gives us a reduction from pattern matching to Orthogonal Vectors (OV) ([9, 23]). But to improve the runtime we process many text simultaneously using the following lemmas.

▶ **Lemma 2.4** (Batch-OV, cf. [9]). *Let $A, B \subseteq \{0,1\}^d$ with $|A| = |B| = n$ and $d \leq 2^{c^{-1}\sqrt{\log n}}$ for some constant $c > 0$. We can decide for all vectors $a \in A$ whether there is a vector $b \in B$ such that $\langle a, b\rangle = 0$ in time $n^2/2^{\epsilon c\sqrt{\log n}}$ for sufficiently small $\epsilon > 0$.*

We generalize this balanced case to the unbalanced case which we use later:

▶ **Lemma 2.5** (Unbalanced Batch-OV). *Let $A, B \subseteq \{0,1\}^d$ with $|A| = n$ and $|B| = m$ and $d \leq 2^{c^{-1}\sqrt{\log \min(n,m)}}$ for some constant $c > 0$. We can decide for all vectors $a \in A$ whether there is a vector $b \in B$ such that $\langle a, b\rangle = 0$ in time $nm/2^{\epsilon c\sqrt{\log \min(n,m)}}$ for sufficiently small $\epsilon > 0$.*

Proof. If $n \leq m$, partition B into $\lceil m/n \rceil$ sets of size n and run the algorithm from Lemma 2.4 on every instance in time $\lceil m/n \rceil n^2/2^{\epsilon c\sqrt{\log n}} \approx nm/2^{\epsilon c\sqrt{\log n}}$. Analogously for $n > m$. ◀

3 Upper Bounds

For patterns p of type $|\circ|$ and $|\circ+$ let $p = (p_1 \mid p_2 \mid \ldots \mid p_k)$ be the pattern of size m. Likewise for the patterns with a Kleene Plus as additional outer operation. Let further $t = t_1 \cdots t_n$ be the text of length n. The main idea of the fast algorithm is to compute a set of matched substrings: $M = \{(i, j) \mid \exists \ell \in [k] : t_i \cdots t_j \in \mathcal{L}(p_\ell)\} \subseteq [n] \times [n]$. From M we construct a graph where the nodes correspond to different prefixes that can be matched. The tuples in M represent edges between these nodes. Then it remains to check whether the node corresponding to t is reachable.

▶ **Theorem 3.1** (Upper Bounds). *We can solve in time* $nm/2^{\Omega\sqrt{\log\min(n,m)}}$*:*

1. $|\circ|$*-pattern matching and* $+|\circ|$*-membership.*
2. $|\circ+$*-pattern matching and* $+|\circ+$*-membership.*

To compute M we split the patterns into large and small ones. For the large patterns we compute the corresponding values of M sequentially while for the small patterns we reduce to unbalanced Batch-OV and use the fast algorithm for this problem shown in Lemma 2.5.

3.1 Patterns of Type $+|\circ|$ and $|\circ|$

As mentioned in the beginning of this section, we compute the set M of matched substring by partitioning the sub-patterns into large and small ones.

▶ **Lemma 3.2.** *Given a text* t *of length* n *and patterns* $\{p_i\}_i$ *of type* $\circ|$ *such that* $\sum_i |p_i| = m$*. We can compute* M *in time* $nm/2^{\Omega(\sqrt{\log\min(n,m)})}$*.*

▶ **Lemma 3.3** (Large Sub-Patterns). *Given a text* t *of length* n *and patterns* $p_1, \ldots, p_\ell$ *of type* $\circ|$ *such that* $\sum_{i=1}^{\ell} |p_i| \leq m$*. We can compute* M *in time* $\mathcal{O}(\ell n \log^2 \min(n,m) + m)$*.*

Proof. From a result by Cole and Hariharan [11] we know that there is a $\mathcal{O}(n\log^2 \hat{m} + \hat{m})$ time algorithm for $\circ|$-pattern matching with patterns of size $\hat{m}$. We run this algorithm sequentially for every pattern. We can ignore all p_i with $|p_i| > |\Sigma|n$ since they match more than n symbols. We get $|p_i| \leq \min(|\Sigma|n, m) \leq \min(n^2, m) \leq \min(n^2, m^2)$. Since $\log\min(n^2, m^2) = 2\log\min(n,m)$, each iteration takes time $\mathcal{O}(n\log^2\min(n,m) + |p_i|)$ and the claim follows. ◀

▶ **Lemma 3.4** (Small Sub-Patterns). *Given a text* t *of length* n *and patterns* $p_1, \ldots, p_m$ *of type* $\circ|$*. There is a* $f \in 2^{\Omega(\sqrt{\log\min(n,m)})}$ *such that the following holds: If* $|p_i| \leq f$ *for all* $i \in [m]$*, then we can compute* M *in time* $nm/2^{\Omega(\sqrt{\log\min(n,m)})}$ *with small error probability.*

We postpone the proof of this lemma and first combine the results for small and large patterns to proof the main theorem.

Proof of Lemma 3.2. Choose $f \in 2^{\Omega(\sqrt{\log\min(n,m)})}$ as in Lemma 3.4 and split the patterns into large patterns of size $> f$ and small patterns of size $\leq f$.

For the at most m/f large patterns compute $M_>$ by Lemma 3.3 in time $\mathcal{O}(m/f \cdot n\log^2\min(n,m) + m) \in nm/2^{\Omega(\sqrt{\log\min(n,m)})}$. Duplicate the ℓ small-patterns m/ℓ times and compute $M_\leq$ for the m small patterns by Lemma 3.4 in the claimed running time. ◀

Proof of Theorem 3.1 Item 1. Construct M by Lemma 3.2. Check for $|\circ|$-pattern matching whether $M = \emptyset$ since any matched substring is sufficient.

For $+|\circ|$-membership we construct a graph G with nodes $v_0, \ldots, v_n$ where we put an edge from v_{i-1} to v_j if $(i,j) \in M$. Then v_n is reachable from v_0 iff there is a decomposition of t into substrings which can be matched by the p_is. This reachability check can be performed in time $\mathcal{O}(n + |M|)$ by a depth-first search starting from v_0. ◀

For the proof of Lemma 3.4 we proceed as follows. For the construction of M for small sub-patterns we define some threshold f and check for every substring of t of length at most f whether there is a pattern that matches this substring. This check is reduced to Batch-OV by encoding the substrings and patterns as bit-vectors.

For small alphabets with $|\Sigma| < f$ this encoding is rather simple since we can use a one-hot encoding of the alphabet. But for larger alphabets this does not work as the dimension of the vectors would increase too much and the fast algorithm for Batch-OV could not be used

anymore. Therefore, we define a randomised encoding χ to ensure that the final bit-vectors are not to large. For simplicity we can assume $|\Sigma| = \Theta(\min(n,m))$ by padding Σ with fresh symbols. The construction in the following lemma is based on the idea of Bloom-Filters [6].

▶ **Lemma 3.5** (Randomised Characteristic Vector). *For a finite universe Σ and a threshold $f \leq 2^{\mathcal{O}(\sqrt{\log|\Sigma|})}$ there is a randomised $\chi : \mathcal{P}(\Sigma) \to \{0,1\}^d$ with $d \in \mathcal{O}(f \log|\Sigma|)$ such that for all $\sigma \in \Sigma$ and $S \subseteq \Sigma$ with $|S| \leq f$ the following holds:*

- *If $\sigma \in S$, then $\chi(\sigma) \coloneqq \chi(\{\sigma\}) \subseteq \chi(S)$, i.e. $\forall i \in [d] : \chi(\{\sigma\})[i] = 1 \implies \chi(S)[i] = 1$.*
- *If $\chi(\sigma) \subseteq \chi(S)$, then $\sigma \in S$ with high probability, i.e. $\geq 1 - 1/\operatorname{poly}(|\Sigma|)$.*

Proof. We define χ element-wise and set for $S \subseteq \Sigma$: $\chi(S)[i] \coloneqq \bigvee_{s\in S} \chi(s)[i]$, i.e. the bitwise OR over $\chi(s)$ for $s \in S$. Hence, the first claim already holds by definition. For each $\sigma \in \Sigma$ we define $\chi(\sigma)$ independently by setting $\chi(\sigma)[i] = 1$ with probability $1/f$ for all $i \in [d]$. Let $S \subseteq \Sigma$ with $|S| \leq f$ and $\sigma \in \Sigma \setminus S$. For all $i \in [d]$:

$$\begin{aligned}
\Pr[\chi(\sigma)[i] \not\subseteq \chi(S)[i]] &= \Pr[\chi(\sigma)[i] = 1 \wedge \chi(S)[i] = 0] \\
&= \tfrac{1}{f}\Big(1 - \tfrac{1}{f}\Big)^{|S|} \geq \tfrac{1}{f}\Big(1 - \tfrac{1}{f}\Big)^{f} \geq \tfrac{\mathrm{e}^{-2}}{f} \\
\Pr[\chi(\sigma) \subseteq \chi(S)] &= \prod_{i=1}^{d} \Pr[\chi(\sigma)[i] \subseteq \chi(S)[i]] = \prod_{i=1}^{d} (1 - \Pr[\chi(\sigma)[i] \not\subseteq \chi(S)[i]]) \\
&\leq \prod_{i=1}^{d} \Big(1 - \tfrac{\mathrm{e}^{-2}}{f}\Big) = \Big(\Big(1 - \tfrac{\mathrm{e}^{-2}}{f}\Big)^{d}\Big).
\end{aligned}$$

Setting $d = fc\ln|\Sigma|$ for some arbitrary $c > \mathrm{e}^2$, we get:

$$= \Big(1 - \tfrac{\mathrm{e}^{-2}}{f}\Big)^{f\cdot c\ln|\Sigma|} \leq \mathrm{e}^{-1/\mathrm{e}^2 \cdot c\ln|\Sigma|} = |\Sigma|^{-c/\mathrm{e}^2} = 1/\operatorname{poly}|\Sigma|. \qquad \blacktriangleleft$$

Proof of Lemma 3.4. Define $f = 2^{\sqrt{\epsilon}/3\cdot\sqrt{\log\min(n,m)}}$ with ϵ as in Lemma 2.5 and let a be some fresh symbol we add to Σ. Let $\chi : \mathcal{P}(\Sigma) \to \{0,1\}^{f^2}$ be as in Lemma 3.5. For simplicity one can think of χ as the one-hot encoding of alphabet Σ.

We define $T_j \coloneqq \{t_i \cdots t_{i+j-1} \mid 1 \leq i \leq n-j+1\}$ and $P_j \coloneqq \{p_i \mid \mathcal{L}(p_i) \subseteq \Sigma^j\}$ for all $j \in [f]$. Then replace all symbols and sub-patterns of type $|$ by bit-vectors by applying χ. Finally, pad every vector in T_j and P_j by $f - j$ repetitions of $\chi(a)$ and flip all values of P_j bit-wise such that 1s become 0s and vice versa. Let T be the set of all $\leq nf$ modified texts and P be the set of all m transformed patterns.

We observe that a text-vector in T is orthogonal to a pattern-vector in P iff the original text was matched by the original pattern. Since $f \cdot f^2 \leq 2^{\sqrt{\epsilon}\sqrt{\log\min(n,m)}} \leq 2^{\sqrt{\epsilon}\sqrt{\log\min(nf,m)}}$, we can apply Lemma 2.5 for T and P:

$$\frac{nfm}{2^{(\epsilon/\sqrt{\epsilon})\sqrt{\log\min(nf,m)}}} \leq \frac{nm}{2^{(\sqrt{\epsilon}-\sqrt{\epsilon}/3)\cdot\sqrt{\log\min(n,m)}}} \in \frac{nm}{2^{\Omega(\sqrt{\log\min(n,m)})}}. \qquad \blacktriangleleft$$

3.2 Patterns of Type $+|\circ+$ and $|\circ+$

First observe that even for small patterns M can be too large to be computed explicitly. For $t = 0^n1^n$ and $p = 0^+1^+$ we have $M = [1,n] \times [n+1,2n]$ and thus cannot write down M explicitly in time $o(nm)$.

To get around this problem we first define the *run-length encoding* $r(u)$ of a text u as in [4]: We have $r(\varepsilon) = \varepsilon$. For a non-empty string starting with σ, let ℓ be the largest integer such that the first ℓ symbols of u are σ. Append the tuple (σ, ℓ) to the run-length encoding

and recurse on u after removing the first ℓ symbols. We use the same approach for patterns of type $\circ+$. But if there occurs a σ^+ during these ℓ positions, we add $(\sigma, \geq \ell)$ to the encoding, otherwise $(\sigma, = \ell)$. We write σ^ℓ for the tuple (σ, ℓ) and similar for $\sigma^{=\ell}$ and $\sigma^{\geq\ell}$ to shorten notation. For example, $r(aaa^+b^+bc) = a^{\geq 3}b^{\geq 2}c^{=1}$.

The idea is to compute a subset of M which only contains those (i, j) such that there is no distinct (i', j') in the subset with $i' \leq i$ and $j' \geq j$ and both substrings of t are matched by the same pattern p_ℓ. We augment each tuple with two boolean flags, indicating whether the first and last run of the pattern p_ℓ contains a Kleene Plus. From this set $M' \subseteq \{0, 1\} \times [n] \times [n] \times \{0, 1\}$ we can fully recover M. For our above example we get $M' = \{(1, n, n + 1, 1)\}$.

▶ **Lemma 3.6.** *Given a text t of length n and patterns $\{p_i\}_i$ of type $\circ+$ such that $\sum_i |p_i| = m$. We can compute M' in time $nm/2^{\Omega(\sqrt{\log \min(n,m)})}$.*

▶ **Lemma 3.7** (Large Sub-Patterns)**.** *Given a text t of length n and patterns $p_1, \ldots, p_\ell$ of type $\circ+$ such that $\sum_{i=1}^{\ell} |p_i| \leq m$. We can compute M' in time $\mathcal{O}(\ell n \log^2 \min(n, m) + m)$.*

Proof. We modify all patterns such that their first and last run is of the form $\sigma^{=\ell}$, i.e. we remove every Kleene Plus from these two runs. There is a $\mathcal{O}(n \log^2 \hat{m} + \hat{m})$ time algorithm for $\circ+$-pattern matching with patterns of size $\hat{m}$ shown in [4]. We run this algorithm sequentially for each altered pattern. For every tuple (i, j) the algorithm outputs, we add (f, i, j, e) to M' where f and e are set to 1 iff the first and last run of the pattern contain a Kleene Plus, respectively.

We can ignore all p_i with $|p_i| > |\Sigma|n$ because they match more than n symbols. Since $|p_i| \leq \min(|\Sigma|n, m) \leq \min(n^2, m) \leq \min(n^2, m^2) = 2 \log \min(n, m)$, each iteration takes time $\mathcal{O}(n \log^2 \min(n, m) + |p_i|)$ and the claim follows. ◀

▶ **Lemma 3.8** (Small Sub-Patterns)**.** *For a text t of length n and patterns $p_1, \ldots, p_m$ of type $\circ+$, there is a $f \in 2^{\Omega(\sqrt{\log \min(n,m)})}$ such that the following holds: If $|p_i| \leq f$ for all $i \in [m]$, then we can compute M' in time $nm/2^{\Omega(\sqrt{\log \min(n,m)})}$.*

We postpone the proof of this lemma and first show the final upper bound as the proof of Lemma 3.2 also works for Lemma 3.6.

Proof of Theorem 3.1 Item 2. Use Lemma 3.6 to construct M' and check for $|\circ+$-pattern matching whether $M' = \emptyset$.

For $+|\circ+$-membership we define a graph $G = (V, E)$. Instead of having nodes $v_0, \ldots, v_n$ as for $+|\circ|$-membership we have for each node v_i three versions, $V \coloneqq \{v_i^0, v_i^1, v_i^2 \mid 0 \leq i \leq n\}$. The versions correspond to the different ways a suffix or prefix of a run can be matched. For node v_i^0 we need that all symbols are explicitly matched by a pattern. For v_i^1 we need that the suffix of the run containing t_i has to be matched by a pattern starting with t_i^+. For v_i^2 we say that the prefix has to be matched by a pattern ending with t_{i-1}^+. Hence, we add edges for the runs simulating the σ^+ of a pattern: For each run σ^ℓ from position i to j in t with $\ell > 1$ we add the edges (v_{k-1}^1, v_k^1) and (v_k^2, v_{k+1}^2) to the graph for $i \leq k < j$. Further, we add edges (v_i^2, v_i^0) and (v_i^0, v_i^1) to change between the states for all $0 \leq i \leq n$. While this construction solely depends on the text, we add for each $(f, i, j, e) \in M'$ the edge (v_{i-1}^f, v_j^{2e}) to the graph. We claim that there is a path from v_0^0 to v_n^0 if and only if $t \in \mathcal{L}((p_1 \mid \cdots \mid p_k)^+)$. We prove this claim in Appendix B. See Figure 2 for an example of the construction.

The time for the construction is linear in the output size. The graph has $\Theta(n)$ nodes and $|M'| + \mathcal{O}(n)$ edges. As the DFS runs in linear time, the overall runtime follows. ◀

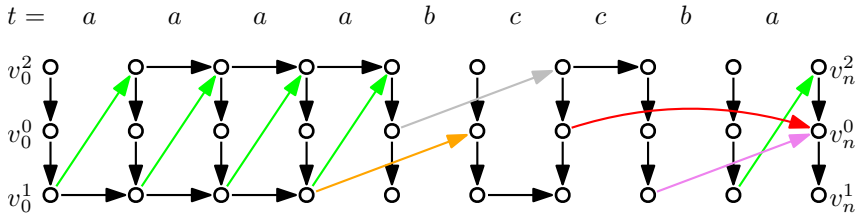


Figure 2 Graph for the pattern $(a^+ \mid a^+b \mid bc^+ \mid cba \mid b^+a)^+$ and text $aaaabccba$.

It remains to show how the set M' is constructed for small patterns.

Proof of Lemma 3.8. Set $f := 2^{\sqrt{\epsilon}/5\sqrt{\log \min(n,m)}}$ with ϵ as in Lemma 2.5 and consider all $\leq n/f^3$ many long runs of length $\geq f^3$ in t. Check for each long run by an exhaustive search whether there is a p_i such that the following holds: The run in the text is matched by one of the $\leq |p_i|$ runs in p_i and the remaining runs of p_i can match the contiguous parts of the text. This check can be performed in the following time for all large runs:

$$\frac{n}{f^3}\sum_{i=1}^{m}|p_i|^2 \leq \frac{n}{f^3}\sum_{i=1}^{m}f^2 \leq \frac{nm}{f}$$

Since a pattern can have at most f runs and each run matches now at most f^3 symbols, it remains to check substrings of t of length at most f^4. Hence, define $T = \{t_i \cdots t_{i+j-1} \mid \forall j \in [f^4], i \in [n-j+1]\}$ and ignore all substrings with more than f runs or runs longer than f^3. Convert these substrings and the patterns into bit-vectors by replacing the runs by the following bit-vectors of length $2\log|\Sigma| + 2f^3$:

$$c^r \mapsto \langle c\rangle\overline{\langle c\rangle}0^r1^{f^3-r}1^r0^{f^3-r} \qquad c^{=r} \mapsto \overline{\langle c\rangle}\langle c\rangle1^r0^{f^3-r}0^r1^{f^3-r} \qquad c^{\geq r} \mapsto \overline{\langle c\rangle}\langle c\rangle1^r0^{f^3-r}0^{f^3}$$

$\langle c\rangle$ denotes the unique binary representation of symbol c and $\overline{\langle c\rangle}$ its bit-wise negation. One can easily see that two such vectors are orthogonal if and only if the runs match each other. Thus, a text and a pattern vector resulting from this transformation are orthogonal iff the text is matched by the pattern. By padding the vectors with 1s we normalise their length but still preserve orthogonality between text and pattern vectors with the same number of runs. Let T' and P' be the resulting sets with $\leq nf^4$ and m elements, respectively.

From $\log|\Sigma| \leq \log \min(n,m) \leq f$ we get $f(2\log|\Sigma| + 2f^3) \leq f^5 \leq 2^{\sqrt{\epsilon}\sqrt{\log \min(nf^4,m)}}$ and hence can apply Lemma 2.5 for T' and P'. Actually we have to partition P' depending on whether a pattern has a Kleene Plus in its first and last run. Thus, we need four iterations but we can always duplicate patterns such that there are m patterns in each group.

$$\frac{nf^4m}{2^{\epsilon/\sqrt{\epsilon}\sqrt{\log \min(nf^4,m)}}} \leq \frac{nm}{2^{(\sqrt{\epsilon}-4/5\sqrt{\epsilon})\sqrt{\log \min(n,m)}}} \in \frac{nm}{2^{\Omega(\sqrt{\log \min(n,m)})}}.$$ ◀

4 Lower Bounds for Pattern Matching

Abboud and Bringmann showed in [1] a lower bound for pattern matching (and membership) in general of $\mathcal{O}(nm/\log^{7+\epsilon} n)$, unless FSH is false. We use this result and the corresponding reduction as a basis to show similar lower bounds for the remaining hard pattern types. But we also do not start our reductions directly from Formula-SAT but from Formula-Pair as defined in Section 2 and use the corresponding Formula-Pair Hypothesis from Hypothesis 2.3.

▶ **Theorem 4.1.** *There are constants $c_{\circ\star} = 76, c_{\circ+\circ} = c_{\circ|+} = 72, c_{\circ|\circ} = 81$, and $c_{\circ+|} = 27$ such that pattern matching with patterns of type $T \in \{\circ\star, \circ+\circ, \circ|\circ, \circ+|, \circ|+\}$ cannot be solved in time $\mathcal{O}(nm/\log^{c_T} n)$ even for constant sized alphabets, unless FPH is false.*

We show the lower bounds by a reduction from Formula-Pair to pattern matching:

▶ **Lemma 4.2.** *Given a Formula-Pair instance with a formula of size s, depth d, and sets A and B with n and $m \leq n$ assignments. (If $m > n$, swap A and B.) We can reduce this to pattern matching with a text t and a pattern p of type $T \in \{\circ\star, \circ+\circ, \circ|\circ, \circ+|, \circ|+\}$ over a constant sized alphabet in time linear in the output size.*

$|t| \in \mathcal{O}(n5^d s \log s)$ except for $\circ+|$, there we have $|t| \in \mathcal{O}(n2^d s \log s)$ Further, $|p| \in \mathcal{O}(mb_T^d s \log s)$ with $b_{\circ\star} = 6, b_{\circ+\circ} = b_{\circ|+} = 5, b_{\circ|\circ} = 8$, and $b_{\circ+|} = 1$.

Proof of Theorem 4.1. We show the result only for patterns of type $\circ+\circ$, the proof for the other types is analogous.

Let F be a formula of size s with two sets of n half-assignments each, and d be the depth of F. Applying the depth-reduction technique of Bonet and Buss [7] gives us an equivalent formula F' with size $s' \leq s^2$ and depth $d' \leq 6 \ln s$. By Lemma 4.2 we get a pattern matching instance with a text t and pattern p. Both of size $\mathcal{O}(n5^{d'} s' \log s') = \mathcal{O}(n5^{6 \ln s} s^2 \log s) = \mathcal{O}(ns^{6 \ln 5+2} \log s)$. Now assume there is an algorithm for pattern matching with the stated running time and run it on t and p:

$$\mathcal{O}\left(\frac{ns^{6 \ln 5+2} \log s \cdot ns^{6 \ln 5+2} \log s}{\log^{72}(ns^{6 \ln 2+2} \log s)}\right) \subseteq \mathcal{O}\left(\frac{n^2 s^{12 \ln 5+4} \log^2 s}{\log^{72} n}\right) \subseteq \mathcal{O}\left(\frac{n^2 s^{23.314}}{\log^{72} n}\right).$$

But this contradicts FPH which was assumed to be true. ◀

4.1 Proof of Lemma 4.2

Again we only give the proof for patterns of type $\circ+\circ$. The reduction for the other pattern types can be found in the full version of the paper. We first encode the evaluation of a formula on two half-assignments, then the encoding for finding such a pair. In the following we define the actual text t_g and pattern p_g. The universal text u_g and universal pattern q_g are needed for technical purposes and do not depend on the assignments.

4.1.1 Encoding the Formula

A formula of size s (i.e. s leaves) has $s-1$ inner gates and thus $2s-1$ gates in total. We assign every gate g a unique integer in $[2s-1]$, its ID, and write $\langle g \rangle$ for the binary encoding of the ID of gate g. We can always see $\langle g \rangle$ as a sequence of $\lfloor \log(2s-1) \rfloor + 1 \leq \lfloor \log s \rfloor + 2 = \Theta(\log s)$ bits padded with zeros if necessary. For a fixed gate g we define a separator gadget $G := 2\langle g \rangle 2$ with 2 as a new symbol.

INPUT Gate The text and the pattern depend on the variable that is read:

For $F_g(a, b) = a_i$ define $t_g := 0a_i1$ as the text and $p_g := 0^+11^+$ as the pattern.

For $F_g(a, b) = b_i$ define $t_g := 011$ as the text and $p_g := 0^+b_i1^+$ as the pattern.

Define $u_g := 0011$ as the universal text and $q_g := 0^+1^+$ as the universal pattern.

AND Gates We define: $t_g := t_1 G t_2$, $p_g := p_1 G p_2$, $u_g := u_1 G u_2$, and $q_g := q_1 G q_2$.

OR Gates The texts and the patterns for gate g are defined as follows where the parentheses are just for grouping and are not part of the text or pattern:

$$\begin{aligned} t_g &:= (u_1GGu_2)G(u_1GGu_2)G(t_1GGt_2)G(u_1GGu_2)G(u_1GGu_2) \\ u_g &:= (u_1GGu_2)G(u_1GGu_2)G(u_1GGu_2)G(u_1GGu_2)G(u_1GGu_2) \\ q_g &:= (u_1GGu_2)G(u_1GGu_2)G(q_1GGq_2)G(u_1GGu_2)G(u_1GGu_2) \\ p_g &:= (u_1GGu_2G)^+(q_1GGp_2)G(p_1GGq_2)(Gu_1GGu_2)^+ \end{aligned}$$

▶ **Lemma 4.3** (Correctness of the Construction). *For all assignments a, b and gates g:*

- $F_g(a,b) = \mathit{true} \iff t_g(a) \in \mathcal{L}(p_g(b))$
- $t_g(a) \in \mathcal{L}(q_g)$
- $u_g \in \mathcal{L}(q_g) \cap \mathcal{L}(p_g(b))$

Proof. The proofs of the second and third claim follow inductively from the encoding of the gates and especially because of the encoding of the INPUT gate. For the first claim we do a structural induction on the output gate of the formula.

INPUT Gate "⇒" Follows directly from the definition.

INPUT Gate "⇐" If the gate is not satisfied, then there are not enough 0s or 1s in the text than the pattern has to match.

AND Gate "⇒" Follows directly from the definition.

AND Gate "⇐" By the uniqueness of the binary encoding, the G in the middle of the text and the pattern have to match. Since the whole text is matched, we get $t_1 \in \mathcal{L}(p_1)$ and $t_2 \in \mathcal{L}(p_2)$ and $F_g(a,b)$ is satisfied by the induction hypothesis.

OR Gate "⇒" $F_g(a,b) = F_{g_1}(a,b) \vee F_{g_2}(a,b) = \mathsf{true}$. Assume w.l.o.g. that $F_{g_1}(a,b) = \mathsf{true}$, the other case is symmetric. Repeat $(u_1GGu_2G)^+$ only once to transform q_1GGp_2 into the second u_1GGu_2 by our third claim of the lemma. Now p_1GGq_2 matches t_1GGt_2 by the second claim and the assumption $t_1 \in \mathcal{L}(p_1)$. Finally, we match $Gu_1GGu_2Gu_1GGu_2$ by two repetitions of $(Gu_1GGu_2)^+$.

OR Gate "⇐" By the uniqueness of the binary encoding there are exactly 14 Gs in the text and the pattern can match 11 Gs when taking both repetitions once. Since each additional repetition increases the number by 3, exactly one repetition is taken twice. If the first repetition is taken once, the following q_1GGp_2 has to match the second u_1GGu_2 in the text. But then p_1 is transformed into t_1 showing that F_g is satisfied by the inductive hypothesis. The case for the second repetition is symmetric. ◀

Length of the text and the pattern. All texts and patterns for a specific gate only depend on the texts and patterns for the two sub-gates. Thus, we can compute the texts and patterns in a bottom-up manner and the encoding can be done in time linear in the size of the output. It remains to analyse the length of the texts and the size of the patterns:

▶ **Lemma 4.4.** $|u_r|, |t_r|, |p_r|, |q_r| \in \mathcal{O}(5^d s \log s)$.

Proof. p_g is obviously smaller than u_g. Since the sizes of u_g, t_g, and q_g are asymptotically equal, it suffices to analyse the length of u_g: $|u_g| \leq 5|u_1| + 5|u_2| + \mathcal{O}(\log s)$. Inductively over the $d(F_g)$ levels of F_g, i.e. the depth of F_g, this yields $|u_g| \leq \mathcal{O}(5^{d(F_g)} s \log s)$. The factor of $s \log s$ is due to the $\mathcal{O}(s)$ inner gates each introducing $\mathcal{O}(\log s)$ additional symbols. ◀

4.1.2 Final Reduction

In the first part of the reduction we have seen how to evaluate a formula on one specific pair of half-assignments. It remains to design a text and a pattern such that such a pair of half-assignments can be chosen. For this let $A = \{a^{(1)}, \dots, a^{(n)}\}$ be the first set and $B = \{b^{(1)}, \dots, b^{(m)}\}$ be the second set of half-assignments. Inspired by the reduction in Section 3.4 in the full version of [4] we define the final text and pattern as follows:

$$t \coloneqq \bigodot_{i=1}^{3n} \Big(33u_r3u_r3u_r3t(a^{(i)})3u_r3u_r3u_r3u_r\Big)$$

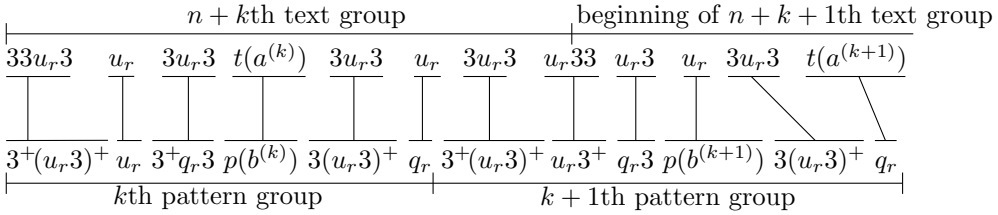
$$p \coloneqq 3u_r3u_r3u_r3u_r \bigodot_{j=1}^{m} \Big(3^+(u_r3)^+u_r3^+q_r3p(b^{(j)})3(u_r3)^+q_r\Big)3u_r3u_r3u_r3u_r.$$

Where we set $a^{(j)} = a^{(j \mod n)}$ for $j \in [n+1, 3n]$. We call the concatenations in t and p for each i and j the ith text group and the jth pattern group, respectively.

▶ **Lemma 4.5.** *If there are $a^{(k)}$ and $b^{(l)}$ such that $F(a^{(k)}, b^{(l)}) = \mathsf{true}$, then $t \in \mathcal{M}(p)$.*

Proof. Assume w.l.o.g. $a^{(k)}$ and $b^{(k)}$ satisfy F. Otherwise we have to shift the indices for the text and the pattern accordingly in the proof. We match the prefix of p to the suffix of the nth text group. Then we match the $n+i$th text group by the ith pattern group for $i = 1, \dots, k-1$: Both $(u_r3)^+$ are repeated twice. Then the remaining parts are matched in a straightforward way by transforming the q_rs into $t(a^{(i)})$ and u_r, and $p(b^{(i)})$ into u_r.

Then, we match the kth and $k+1$th pattern group to the $n+k$th text group and a part of the $n+k+1$th text group:



For the last step we shift the groups in the remaining text t' such that it becomes easier to prove which part of the text the remaining pattern matches:

$$\begin{aligned} t' =& 3u_r3u_r3u_r3u_r \bigodot_{i=n+k+2}^{3n} \Big(33u_r3u_r3u_r3t(a^{(i)})3u_r3u_r3u_r3u_r\Big) \\ =& \bigodot_{i=n+k+2}^{3n} \Big(3u_r3u_r3u_r3u_r33u_r3u_r3u_r3t(a^{(i)})\Big)3u_r3u_r3u_r3u_r. \end{aligned}$$

For each of the remaining pattern groups the first repetition is taken three times. With this the $n+i$th group of t' and the ith pattern group are matched in a straightforward way for $i = k+2, \dots, m$. The suffix of the pattern is matched to the start of the $n+m+1$th text group in the obvious way. ◀

▶ **Lemma 4.6.** *If $t \in \mathcal{M}(p)$, then there are $a^{(k)}$ and $b^{(l)}$ such that $F(a^{(k)}, b^{(l)}) = \mathsf{true}$.*

Proof. By the design of the pattern and the text, there must be a $j \leq n$ such that the prefix of the pattern is matched to the suffix of the $j-1$th text group. Likewise the suffix of the pattern has to match the same sequence in some other text group because nowhere else the

four $3u_r$ could be matched. Thus, not all text groups and pattern groups match each other precisely and there is a text group k and a pattern group l such that the pattern group does not match the whole text group or it matches more than this group. Choose the first of these groups, i.e. the pair with smallest k and l.

Since all prior groups have been matched precisely, the first repetition can be taken at most twice. Otherwise the following u_r could not be transformed into a part of the text. Now assume it is repeated exactly once. Then the following u_r matches the second u_r of the text group. Since 3 is a fresh symbol, q_r has to match the third u_r. But then $p(b^{(k)})$ has to be transformed into $t(a^{(l)})$ and Lemma 4.3 gives us a satisfying assignments.

It remains to check the case when $(u_r 3)^+$ is repeated twice. Then q_r is transformed into $t(a^{(l)})$ and $p(b^{(k)})$ is transformed into the fourth u_r. The second repetition has to be taken exactly twice in this case. Because otherwise the 33 from the beginning of the next text group could not be matched. But if the pattern $(u_r 3)^+$ is repeated twice, this pattern group is completely matched to a text group, contradicting our assumption. ◀

▶ **Lemma 4.7.** *The final text has length $\mathcal{O}(n5^d s \log s)$ and the pattern has size $\mathcal{O}(m5^d s \log s)$.*

By this we conclude the proof of Lemma 4.2 for this pattern type. ⌟

5 Lower Bounds for Membership

Instead of giving all reductions from scratch, we reduce pattern matching to membership and make use of the results in Lemma 4.2. By this we get the same bounds as for pattern matching given in Theorem 4.1. For the remaining pattern type $|{+}|\circ$ we give a new reduction from scratch which is necessary due to the missing concatenation as outer operation.

5.1 Reducing Pattern Matching to Membership

▶ **Lemma 5.1** (Reducing Pattern Matching to Membership). *Given a text t and a pattern p with type in $\{\circ\star, \circ{+}\circ, \circ|\circ, \circ{+}|, \circ|{+}\}$ over a constant sized alphabet.*

We can construct a text t' and a pattern p' of the same type as p in linear time such that $t \in \mathcal{M}(p) \iff t' \in \mathcal{L}(p')$. Further, $|t'| \in \mathcal{O}(|t|)$ and $|p'| \in \mathcal{O}(|t| + |p|)$, except for $\circ{+}|$, there we even have $|p'| \in \mathcal{O}(|p|)$.

Proof. Again we only show the proof for type $\circ{+}\circ$. The reductions for the other types can be found in the full version.

Let $\Sigma = \{1, \dots, s\}$ be the alphabet. We first encode every symbol such that we can simulate a universal pattern (i.e. matching any symbol) by some gadget U of type $\circ{+}$. Let $f : \Sigma \to \Sigma^{s+1}$ be this encoding with $f(x) = 1 \cdots (x-1)xx(x+1) \cdots s$. Since we can extend f in the natural way to texts by applying it to every symbol, we can also modify patterns of type $\circ{+}\circ$ by applying f to every symbol *without* changing the type. After applying f we still have $t \in \mathcal{M}(p) \iff f(t) \in \mathcal{M}(f(p))$.

For the step from pattern matching to membership we set $U \coloneqq 1^+2^+ \cdots s^+$ and $R \coloneqq 12 \cdots s$. Obviously $R \in \mathcal{L}(U)$ and $f(\sigma) \in \mathcal{L}(U)$ for all $\sigma \in \Sigma$. But we also get $R \notin \mathcal{L}(f(\sigma))$ since R does not contain a repetition of σ. Finally, we define $t' \coloneqq R^{|t|+1} f(t) R^{|t|+1}$ and $p' = R^+ U^{|t|} f(p) U^{|t|} R^+$. We claim $t \in \mathcal{M}(p) \iff t' \in \mathcal{L}(p')$.

"$\Rightarrow$" If $t \in \mathcal{M}(p)$, then there is a substring $\hat{t}$ of t matched by p. By the above observations, $f(p)$ matches $f(\hat{t})$ which is a substring of $f(t)$. Then we use $U^{|t|}$ to match the not matched suffix and prefix of $f(t)$ and a part of $R^{|t|+1}$. The remaining repetitions of R are matched by the R^+ in the beginning and the end. "$\Leftarrow$" If $t' \in \mathcal{L}(p')$, then $f(p)$ has to match some substring of $f(t)$ because R cannot be matched by the above observation. ◀

5.2 Patterns of Type $|+|\circ$

Even though the remaining hard pattern type $|+|\circ$ does not have a concatenation as outer operation, we can still show a similar lower bound as for the other types.

▶ **Theorem 5.2.** *$|+|\circ$-membership cannot be solved in time $\mathcal{O}(nm/\log^{17} n)$ even for constant sized alphabets, unless FPH is false.*

To proof the theorem it suffices to show that Formula-Pair can be reduced to membership with a text of length $\mathcal{O}(ns^2 \log s)$ and a pattern of size $\mathcal{O}(ms^3 \log s)$. Then the claim directly follows from the definition of FPH as for the other types.

Idea of the reduction. As for the other lower bounds, we first encode the evaluation of the formula on two fixed half-assignments. We define for each gate g a text t_g and two dictionaries D_g^M and D_g^S of words. The final dictionary for a gate g is defined as $D_g = \bigcup_{g' \in F_g} D_{g'}^S \cup D_{g'}^M$. The final pattern is D_r^+ where r is the root of F.

D_g^M corresponds to p_g and allows us to match the whole text t_g if the formula is satisfied. The texts of the sub-gates are then matched by the corresponding dictionaries. But for the OR gate we have to be able to ignore the evaluation of one sub-formula. For this we define the set D_g^S which corresponds to q_g and allows us to match the text independently from the assignments. As main idea we include the path from the root of the formula to the current gate in the encoding. This trace is appended to the text as a prefix and in reverse as suffix. The words in D_g^M for OR gates g allow us to jump to a gate in such a trace of exactly one sub-formula. Then we use corresponding words from D^S to propagate this jump to the sub-formulas. Because the included trace started at the root, we can proceed to the INPUT gates. There we add words to accept all evaluations of the gate. For the way back up we add the corresponding words in reverse to the dictionaries.

We make sure that these words are just used at one specific position by embedding the encoding of the corresponding gate in the trace. Since the gate number can be made unique these words can only be used at one specific position. This procedure allows us to write down the words as a set and not as a concatenation as for the other reductions.

Encoding the Formula. We identify each gate g with its ID, i.e. an integer in $[2s]$. Let $\langle g \rangle$ be the binary encoding of the gate ID with $\lfloor \log s \rfloor + 2 = \Theta(\log s)$ bits padded with zeros if necessary. Further, let $h_0 h_1 \dots h_d$ be the path from the root $r = h_0$ of F to the gate $g = h_d$ of depth $d \geq 0$. To simplify notation we define $h_i^g = 2\langle h_i \rangle \langle g \rangle 2$, i.e. the encoding of the gate on the path and the gate where the path ends.

INPUT Gates We set $D_g^S \coloneqq \{h_i^g \dots h_d^g 0 h_d^g \dots h_i^g, h_i^g \dots h_d^g 1 h_d^g \dots h_i^g \mid i \in [d]\}$.
For $F_g(a,b) = a_i$, we set $t_g \coloneqq h_0^g \dots h_d^g a_i h_d^g \dots h_0^g$ and $D_g^M \coloneqq \{h_0^g \dots h_d^g 1 h_d^g \dots h_0^g\}$
For $F_g(a,b) = b_i$, we set $t_g \coloneqq h_0^g \dots h_d^g 1 h_d^g \dots h_0^g$ and $D_g^M \coloneqq \{h_0^g \dots h_d^g b_i h_d^g \dots h_0^g\}$

AND Gate We define the text and the corresponding dictionaries as follows:

$$
\begin{aligned}
t_g &\coloneqq h_0^g \dots h_d^g t_1 t_2 h_d^g \dots h_0^g \\
D_g^M &\coloneqq \{h_0^g \dots h_d^g, h_d^g \dots h_0^g\} \\
D_g^S &\coloneqq \{h_i^g \dots h_d^g h_0^{g_1} \dots h_{i-1}^{g_1}, h_{i-1}^{g_1} \dots h_0^{g_1} h_0^{g_2} \dots h_{i-1}^{g_2}, h_{i-1}^{g_2} \dots h_0^{g_2} h_d^g \dots h_i^g \mid i \in [d]\}
\end{aligned}
$$

OR Gate We define the text and the additional dictionaries for g as:

$$\begin{aligned} t_g &:= h_0^g \dots h_d^g t_1 h_d^g t_2 h_d^g \dots h_0^g \\ D_g^M &:= \{h_0^g \dots h_d^g, h_d^g h_0^{g_2} \dots h_d^{g_2}, h_d^{g_2} \dots h_0^{g_2} h_d^g \dots h_0^g\} \\ &\quad \cup \{h_0^g \dots h_d^g h_0^{g_1} \dots h_d^{g_1}, h_d^{g_1} \dots h_0^{g_1} h_d^g, h_d^g \dots h_0^g\} \\ D_g^S &:= \{h_i^g \dots h_d^g h_0^{g_1} \dots h_{i-1}^{g_1}, h_{i-1}^{g_1} \dots h_0^{g_1} h_d^g h_0^{g_2} \dots h_{i-1}^{g_2}, h_{i-1}^{g_2} \dots h_0^{g_2} h_d^g \dots h_i^g \mid i \in [d]\} \end{aligned}$$

▶ **Lemma 5.3.** *For all assignments a, b and gates g:*

- $t_g(a) \in \mathcal{L}(h_0^g \dots h_{i-1}^g (D_g(b))^+ h_{i-1}^g \dots h_0^g)$ *for all* $i \in [d]$.
- $t_g(a) \notin \mathcal{L}(h_0^g \dots h_{i-1}^g (D_g(b))^+ h_{j-1}^g \dots h_0^g)$ *for all* $i \neq j \in [0, d]$, *where* $h_0^g h_{-1}^g$ *and* $h_{-1}^g h_0^g$ *denote the empty string.*

Proof. The first claim follows by a structural induction on the output gate using only words from $D_{g'}^S$ for the current gate g'. Likewise we show the second case by a structural induction on the output gate.

INPUT Gate The statement holds by the definition of the dictionary.

AND Gate Assume the claim is false for g. We can only match the "prefix" $h_i^g \dots h_d^g$ with the word $h_i^g \dots h_d^g h_0^{g_1} \dots h_{i-1}^{g_1}$. And analogously for the "suffix". The joining part of $t_1 t_2$ has to be matched by some $h_{k-1}^{g_1} \dots h_0^{g_1} h_0^{g_2} \dots h_{k-1}^{g_2}$ for $k \in [0, \dots, d]$ (possibly the empty string). Hence, $t_1 \in \mathcal{L}(h_0^{g_1} \dots h_{i-1}^{g_1} (D_{g_1}(b))^+ h_{k-1}^{g_1} \dots h_0^{g_1})$ and $t_2 \in \mathcal{L}(h_0^{g_2} \dots h_{k-1}^{g_2} (D_{g_2}(b))^+ h_{j-1}^{g_2} \dots h_0^{g_2})$. But from $i \neq j$ it follows that $k \neq i$ or $k \neq j$ and we have a contradiction to the induction hypothesis for g_1 and g_2.

OR Gate The "prefix" $h_i^g \dots h_d^g$ has to be matched by $h_i^g \dots h_d^g h_0^{g_1} \dots h_{i-1}^{g_1}$ and analogously for the "suffix". If the joining part of $t_1 h_d^g t_2$ was matched by $h_{k-1}^{g_1} \dots h_0^{g_1} h_d^g h_0^{g_2} \dots h_{k-1}^{g_2}$ for some $k \in [d]$, the same proof as for the AND gate applies. Otherwise, either $h_d^{g_1} \dots h_0^{g_1} h_d^g$ or $h_d^g h_0^{g_2} \dots h_d^{g_2}$ was used. Let it w.l.o.g. be the first one. Since $i \in [0, d]$, we have $i \neq d+1$ and hence a contradiction to the inductive hypothesis for t_1. ◀

▶ **Lemma 5.4** (Correctness of the Construction)**.** *For all assignments a, b and gates g:* $F_g(a, b) = \mathit{true} \iff t_g(a) \in \mathcal{L}((D_g(b))^+)$.

Proof. We proof the claim by an induction on the output gate.

INPUT Gate Follows directly from the construction of the text and the dictionary.

AND Gate "⇒" We can use D_1^+ and D_2^+ to match t_1 and t_2 by the induction hypothesis, respectively. The remaining parts are matched by the words in D_g^M.

AND Gate "⇐" The initial and last h_0^g of the text have to be matched. Since the gate g is part of the encoding, we can only use words from D_g^M for this. It follows directly that t_1 is matched by words from D_1 because the initial $h_0^{g_1}$ has to be matched too and the words in D_g^S are not eligible for this. The same argument shows that t_2 is matched by words from D_2. Hence, the claim follows by the induction hypothesis.

OR Gate "⇒" Assume w.l.o.g. that $F_{g_1}(a, b) = \mathsf{true}$, the other case is symmetric. We match the prefix of t_g in the obvious way by the corresponding word from D_g^M. By assumption we match t_1 with words from D_1. The prefix $h_0^{g_2} \dots h_d^{g_2}$ of t_2 is matched by the corresponding word in D_g^M. By the first claim of the previous lemma, we have $t_2 \in \mathcal{L}(h_0^{g_2} \dots h_d^{g_2} (D_{g_2})^+ h_d^{g_2} \dots h_0^{g_2})$ and the remaining suffix can be matched by the corresponding word from D_g^M.

OR Gate "⇐" By Lemma 5.3 the joining part of $t_1 h_d^g t_2$ has to be matched by either $h_d^g h_0^{g_2} \dots h_d^{g_2}$ or $h_d^{g_1} \dots h_0^{g_1} h_d^g$. Let it w.l.o.g. be the first one. Then t_1 has to be matched by words from D_1 again by the lemma. The inductive hypothesis gives us a satisfying assignment. ◀

▶ **Lemma 5.5.** *We have the following size bounds:*
- $|t_r| \in \mathcal{O}(sd \log s) \subseteq \mathcal{O}(s^2 \log s)$
- $|D_r| \in \mathcal{O}(sd) \subseteq \mathcal{O}(s^2)$
- $\forall x \in D_r : |x| \in \mathcal{O}(d \log s) \subseteq \mathcal{O}(s \log s)$

Proof. The lemma follows directly from the definitions and the observations that $|t_g| \leq |t_1| + |t_2| + \mathcal{O}(d \log s)$, $|D_g^M| \in \mathcal{O}(1)$, and $|D_g^S| \in \mathcal{O}(d)$. ◀

Outer OR. Let $A = \{a^{(1)}, \ldots, a^{(n)}\}$ be the first set and $B = \{b^{(1)}, \ldots, b^{(m)}\}$ be the second set of half-assignments. Again we encode A by the text and B by the pattern. For this we observe that the first step of the reduction produced a pattern of type $+|\circ$. Thus, we can use the outer alternative to encode the outer OR to select a specific $b^{(j)}$. To match the whole text, we blow up the text and the pattern and pad each symbol with three new symbols such that we can distinguish between the following three matching states: (1) ignore the padding and match a part of the original text to the original pattern, i.e. we evaluate the formula on two half-assignments. (2) Match an arbitrary prefix, i.e. the symbols before the actual match in state (1). (3) Match some arbitrary suffix, i.e. the symbols after the actual match from state (1). We allow a change between these states only at the end of a text group and require that we go through all three states if and only if the text can be matched by the pattern.

▶ **Definition 5.6** (Blow-Up of a Text). *Let $t = t_1 \cdots t_n$ be a text of length n and u be some arbitrary string. We define $t\Uparrow^u := ut_1ut_2 \cdots ut_n$ and extend it in the natural way to sets.*

Using this we define the final text and pattern as follows:

$$t := 563 \bigodot_{i=1}^{n} \Big(t(a^{(i)})3 \Uparrow^{456} \Big) 45$$

$$p := p_1^+ \mid p_2^+ \mid \cdots \mid p_m^+$$

$$p_j := 5604 \mid 5614 \mid 5624 \mid 5634 \mid 563 \mid D_r(b^{(j)}) \Uparrow^{456} \mid 456345 \mid 6045 \mid 6145 \mid 6245 \mid 6345$$

▶ **Lemma 5.7.** *If there are $a^{(k)}$ and $b^{(l)}$ such that $F(a^{(k)}, b^{(l)}) = \mathsf{true}$, then $t \in \mathcal{L}(p)$.*

Proof. It suffices to show that we can match t to p_l^+. The prefix of t and the first $k-1$ text groups are matched by repetitions of $56x4$ for values $x \in \{0, 1, 2, 3\}$ while the last three symbols of the $k-1$th group are matched by 563. This is possible by our blow-up with 456. By Lemma 5.4 and the definition of the blow-up we get $t(a^{(k)}) \Uparrow^{456} \in \mathcal{L}((D_r(b^{(l)}) \Uparrow^{456})^+)$. The following 456345 is matched by the corresponding pattern while the remaining symbols of the text are matched in a straight forward way by repetitions of $6x45$. ◀

▶ **Lemma 5.8.** *If $t \in \mathcal{L}(p)$, then there are $a^{(k)}$ and $b^{(l)}$ such that $F(a^{(k)}, b^{(l)}) = \mathsf{true}$.*

Proof. By the structure of the pattern we can already fix l. As there is no way to match the text just with words $56x4$ or $6x45$, the word 563 must have been used at the end of some group to switch to the first state. Hence, let the kth text group be the first group not matched by words of the form $56x4$. Observe that we cannot directly switch to an application of $6x45$ and thus get $t(a^{(k)}) \Uparrow^{456} \in \mathcal{L}((D_r(b^{(l)}) \Uparrow^{456})^+)$. Since the blow-up 456 always matches each other, we can ignore it and get $t(a^{(k)}) \in \mathcal{L}(D_r(b^{(l)})^+)$ proving the claim by Lemma 5.4. ◀

▶ **Corollary 5.9.** *The final text has length $\mathcal{O}(nsd \log s) \subseteq \mathcal{O}(ns^2 \log s)$ and the pattern has size $\mathcal{O}(msd^2 \log s) \subseteq \mathcal{O}(ms^3 \log s)$.*

This finishes the proof of Theorem 5.2. ⌟

References

1. Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. Full version: arXiv:1804.08978. doi:10.4230/LIPIcs.ICALP.2018.8.
2. Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.
3. Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.
4. Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466. IEEE Computer Society, 2016. Full version: arXiv:1511.07070. doi:10.1109/FOCS.2016.56.
5. Philip Bille and Mikkel Thorup. Faster regular expression matching. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2009. doi:10.1007/978-3-642-02927-1_16.
6. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. doi:10.1145/362686.362692.
7. Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoffs for boolean fomulae. *Inf. Process. Lett.*, 49(3):151–155, 1994. doi:10.1016/0020-0190(94)90093-0.
8. Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 307–318. IEEE Computer Society, 2017. Full version: arXiv:1611.00918. doi:10.1109/FOCS.2017.36.
9. Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255. SIAM, 2016. doi:10.1137/1.9781611974331.ch87.
10. Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic #sat algorithm for small de morgan formulas. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25–29, 2014. Proceedings, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2014. doi:10.1007/978-3-662-44465-8_15.
11. Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 592–601. ACM, 2002. doi:10.1145/509907.509992.
12. Theodore Johnson, S. Muthukrishnan, and Irina Rozenbaum. Monitoring regular expressions on out-of-order streams. In Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis, editors, *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 1315–1319. IEEE Computer Society, 2007. doi:10.1109/ICDE.2007.369001.
13. Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. Proton: multitouch gestures as regular expressions. In Joseph A. Konstan, Ed H. Chi, and Kristina Höök, editors, *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA – May 05–10, 2012*, pages 2885–2894. ACM, 2012. doi:10.1145/2207676.2208694.

14 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.

15 Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for demorgan formula size. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 588–597. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.69.

16 David Landsman. RNP-1, an RNA-binding motif is conserved in the DNA-binding cold shock domain. *Nucleic Acids Research*, 20(11):2861–2864, June 1992. doi:10.1093/nar/20.11.2861.

17 Quanzhong Li and Bongki Moon. Indexing and querying XML data for regular path expressions. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 361–370. Morgan Kaufmann, 2001. URL: http://www.vldb.org/conf/2001/P361.pdf.

18 Makoto Murata. Extended path expressions for XML. In Peter Buneman, editor, *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*. ACM, 2001. doi:10.1145/375551.375569.

19 Eugene W. Myers. A four russians algorithm for regular expression pattern matching. *J. ACM*, 39(2):430–448, 1992. doi:10.1145/128749.128755.

20 Gonzalo Navarro and Mathieu Raffinot. Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. *Journal of Computational Biology*, 10(6):903–923, 2003. PMID: 14980017. doi:10.1089/106652703322756140.

21 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 183–192. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.25.

22 Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. doi:10.1145/363347.363387.

23 Richard Ryan Williams. The polynomial method in circuit complexity applied to algorithm design (invited talk). In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 47–60. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.47.

24 Fang Yu, Zhifeng Chen, Yanlei Diao, T. V. Lakshman, and Randy H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In Laxmi N. Bhuyan, Michel Dubois, and Will Eatherton, editors, *Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems, ANCS 2006, San Jose, California, USA, December 3-5, 2006*, pages 93–102. ACM, 2006. doi:10.1145/1185347.1185360.

A FSH implies FPH

We use the following relation between Formula-SAT and Formula-Pair to show that FSH implies FPH:

▶ **Lemma A.1** (Weak version of Lemma B.2 in the full version of [1]). *An instance of Formula-SAT on a De Morgan formula of size s over n variables can be reduced to an instance of Formula-Pair with a monotone De Morgan formula of size $k = \mathcal{O}(s)$ and two sets of size $\mathcal{O}(2^{n/2})$ in linear time.*

Proof Idea. Let F be the formula for Formula-SAT on n variables and size s. We define F' to be the same formula as F but each leaf is labeled with a different variable and we remove the negations from the leaves.

For all half-assignments x to the first half of variables of F we construct a new half-assignment a_x for F' as follows: Let l be a leaf in F with a variable from the first half of inputs and let l' be the corresponding variable/leaf in F'. We set $a_x[l'] = \mathsf{true}$ if and only if l evaluates to true under x. We construct the set B analogous for the second half of inputs of F. Since F has n inputs this results in $2^{n/2}$ assignments for A and B. ◀

▶ **Lemma A.2.** *FSH implies FPH.*

Proof. Assume FSH holds and FPH is false for some fixed $k \geq 1$. Let F be a formula for Formula-SAT on N inputs and size $s = N^{3+1/(4k)} \in N^{3+\Omega(1)}$. By Lemma A.1 we transform F into a monotone De Morgan formula F' of size $s' = \mathcal{O}(s)$ and two sets with $n, m \in \mathcal{O}(2^{N/2})$ assignments. We run the algorithm for Formula-Pair on this instance to contradict FSH:

$$\mathcal{O}\left(\frac{n \cdot m \cdot s'^k}{\log^{3k+2} n} \log^{1+o(1)} 2^N\right) \subseteq \mathcal{O}\left(\frac{2^{N/2} 2^{N/2} s^k N^{1.25}}{\log^{3k+2} 2^{N/2}}\right) = \mathcal{O}\left(2^N \frac{N^{3k+0.25+1.25}}{N^{3k+2}(1/2)^{3k+2}}\right)$$
$$= \mathcal{O}\left(2^N \frac{N^{3k+1.5}}{N^{3k+2}}\right) = \mathcal{O}\left(\frac{2^N}{N^{0.5}}\right).$$

See the following paragraph for the additional factor of $N^{1+o(1)}$. ◀

As Abboud and Bringmann [1] we use the *Word-RAM* model as our computational model. The word size of the machine will be fixed to $\Theta(\log N)$ many bits for input size N. Likewise we assume several operations that can be performed in time $\mathcal{O}(1)$ (e.g. AND, OR, NOT, addition, multiplication, ...).

While this is sufficient for our reductions, we also need that the operations are robust to a change of the word size to state FPH. As in [1] we require that we can simulate the operations on words of size $\Theta(\log N)$ on a machine with word size $\Theta(\log \log N)$ in time $(\log N)^{1+o(1)}$.

In the above proof the input size increased from N to $n = 2^N$. Hence, we have to simulate the algorithm for Formula-Pair with word size $\log n = N$ on a machine with word size $\log N$ to get an algorithm for Formula-SAT. Thus, the running time slows down by a factor of $(\log n)^{1+o(1)} = N^{1+o(1)}$.

B Correctness of the Graph Construction for +|∘+-Membership

We show the correctness of the graph construction given in the proof of Theorem 3.1 Item 2.

▷ Claim B.1. If $t \in \mathcal{L}(p)$, then there is a path from v_0^0 to v_n^0.

Proof. Assume $p = (p_1 \mid \ldots \mid p_k)^+$. Since $t \in \mathcal{L}(p)$, we can decompose t into $t = \tau_1 \cdots \tau_\ell$ such that for all $l \in [\ell]$ $\tau_l \in \mathcal{L}(p_{k_l})$ for some $k_l \in [k]$. Define $\lambda_l = |\tau_1 \cdots \tau_l|$ as the length of the first l parts of t for all $l \in [\ell]$. We claim that if $\tau_1 \cdots \tau_l \in \mathcal{L}(p)$, then there is a path from v_0^0 to $v_{\lambda_l}^0$.

For $l = 0$, the claim is vacuously true as $\varepsilon \notin \mathcal{L}(p)$. Now assume the claim holds for arbitrary but fixed l. We define $i = \lambda_l + 1$ and $j = \lambda_{l+1}$ to simplify notation and get $\tau_{l+1} = t_i \cdots t_j$. From $\tau_{l+1} \in \mathcal{L}(p_{k_{l+1}})$ and Lemma 3.6 we know $(f, i', j', e) \in M'$ for some $i \leq i' \leq j' \leq j$. Further, f, e are set to 1 if and only if the first and last run of $p_{k_{l+1}}$ contains a Kleene Plus, respectively. Hence, $v_{j'}^{2e}$ is reachable from $v_{i'-1}^f$. Now it suffices to show that (1) $v_{i'-1}^f$ is reachable from v_{i-1}^0 and (2) v_j^0 is reachable from $v_{j'}^{2e}$. Then the claim follows inductively as v_{i-1}^0 is reachable from v_0^0.

We first show (1). If $f = 0$, we must have $i = i'$ and the claim holds. Thus assume $f = 1$. We know $\tau_{l+1} = t_i \cdots t_j \in \mathcal{L}(p_{k'})$ and $t_{i'} \cdots t_{j'} \in \mathcal{L}(p_{k'})$ for some $k' \in [k]$. As the first run of $p_{k'}$ contains a Kleene Plus, the symbols, $t_i, t_{i+1}, \ldots, t_{i'}$ are all equal. That is, they form a run from i to i'. By the construction of the graph, there are edges $(v^1_{i-1}, v^1_i), \ldots, (v^1_{i'-2}, v^1_{i'-1})$. But there is also the additional edge (v^0_{i-1}, v^1_{i-1}) proving (1).

By a symmetric argument one can show claim (2). ◁

▷ Claim B.2. If there is a path from v^0_0 to v^0_n, then $t \in \mathcal{L}(p)$.

Proof. First observe that it is not possible to reach v^0_n from v^0_0 without using edges introduced by tuples in M'. Now fix some path P from v^0_0 to v^0_n and let $P_1, \ldots, P_\ell$ be the edges on the path that are introduced by tuples in M'. Let $P_l = (v^{f_l}_{i'_l-1}, v^{2e_l}_{j'_l})$, i.e. $(f_l, i'_l, j'_l, e_l) \in M'$.

Assume $j'_0 = 0$ and $i'_{\ell+1} = n + 1$ in the following to simplify notation. For each tuple there are two indices i_l and j_l such that $j'_{l-1} \leq i_l - 1 \leq i'_l - 1$ and $j'_l \leq j_l \leq i'_{l+1} - 1$ and the path P goes through $v^0_{i_l-1}$ and $v^0_{j_l}$. These nodes exist, as every path from $v^{2e_{l-1}}_{j'_{l-1}}$ to $v^{f_l}_{i'_l-1}$ has to go through some node v^0_r. We have $j_l + 1 = i_{l+1}$ for all $l \in [0, \ell]$ with $j_0 = 0$ and $i_{\ell+1} = n + 1$ and hence, $t = t_{i_1} \cdots t_{j_1} t_{i_2} \cdots t_{j_2} \cdots t_{i_\ell} \cdots t_{j_\ell}$. Thus, it suffices to show that for every $l \in [\ell]$ there is a $k' \in [k]$ such that $t_{i_l} \cdots t_{j_l} \in \mathcal{L}(p_{k'})$.

We fix l in the following and omit it as index to simplify notation. By the construction of the graph we have $(f, i', j', e) \in M'$ and hence by Lemma 3.6 $t_{i'} \cdots t_{j'} \in \mathcal{L}(p_{k'})$ for some $k' \in [k]$. We extend this result and claim $t_i \cdots t_{j'} \in \mathcal{L}(p_{k'})$. Recall, that there is a path from v^0_{i-1} to $v^f_{i'-1}$ in P. If $f = 0$, then $i' = i$ and the claim follows. Otherwise, we know that the first run of $p_{k'}$ contains a Kleene Plus for some symbol α. As no edge resulting from a tuple in M' can be chosen, the edge (v^0_{i-1}, v^1_{i-1}) is contained in the path P. By the construction of the graph, the sequence $t_i \cdots t_{i'}$ is contained in some run β^c. But $\alpha = \beta$ and we get $t_i \cdots t_{i'-1} t_{i'} \cdots t_{j'} \in \mathcal{L}(p_{k'})$.

We can apply the symmetric argument to show that $t_i \cdots t_{j'} t_{j'+1} \cdots t_j \in \mathcal{L}(p_{k'})$ proving the claim. ◁

Space-Efficient, Fast and Exact Routing in Time-Dependent Road Networks

Ben Strasser
Karlsruhe Institute of Technology, Germany
academia@ben-strasser.net

Dorothea Wagner
Karlsruhe Institute of Technology, Germany
dorothea.wagner@kit.edu

Tim Zeitz
Karlsruhe Institute of Technology, Germany
tim.zeitz@kit.edu

Abstract

We study the problem of computing shortest paths in massive road networks with traffic predictions. Incorporating traffic predictions into routing allows, for example, to avoid commuter traffic congestions. Existing techniques follow a two-phase approach: In a preprocessing step, an index is built. The index depends on the road network and the traffic patterns but not on the path start and end. The latter are the input of the query phase, in which shortest paths are computed. All existing techniques have either large index size, slow query running times, or may compute suboptimal paths. In this work, we introduce CATCHUp (Customizable Approximated Time-dependent Contraction Hierarchies through Unpacking), the first algorithm that simultaneously achieves all three objectives. The core idea of CATCHUp is to store paths instead of travel times at shortcuts. Shortcut travel times are derived lazily from the stored paths. We perform an experimental study on a set of real world instances and compare our approach with state-of-the-art techniques. Our approach achieves the fastest preprocessing, competitive query running times and up to 30 times smaller indexes than competing approaches.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases realistic road networks, time-dependent route planning, shortest paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.81

Supplementary Material `https://github.com/kit-algo/catchup`

Acknowledgements We thank Lars Gottesbüren and Michael Hamann for fruitful discussions and feedback. We also thank Marcel Radermacher for his input on approximation algorithms.

1 Introduction

Routing in road networks is a well-studied topic with a plethora of real world applications. The core problem is to compute a fastest route between a source and a target. The idealized problem can be formalized as the classic shortest path problem. Streets are modeled as arcs. Street intersections are modeled as nodes. Travel times are modeled as scalar arc weights. Unfortunately, this idealized view does not model certain important real world effects. An important example are recurring commuter congestions. In this paper, we consider an extended problem in which travel times are time-dependent. The travel time of an arc is a function of the moment where a car enters the arc. Figure 1 depicts an example.

Computing shortest-paths using Dijkstra's [12] algorithm is possible both in the classical and in the time-dependent setting. However, for many applications, its running time is too large. To achieve fast running times, a two-phase approach is used. In the first phase, the

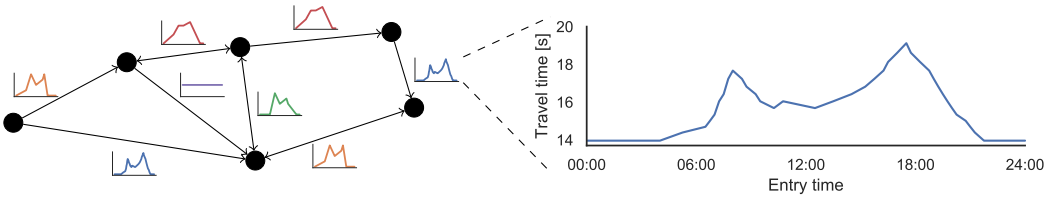


Figure 1 The graph of a small road network with predicted travel times for each road segment.

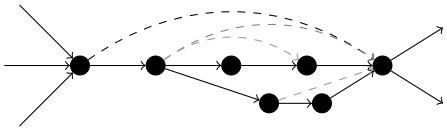
preprocessing phase, an index is constructed. The index only depends on the road networks and the time-dependent arc weights. In the second phase, the *query* phase, shortest paths are computed utilizing this index.

An important ingredient for many such two-phase techniques [1, 4, 9, 11, 15] are *shortcuts*. Shortcuts are additional arcs introduced during preprocessing, which bypass parts of the input graph like in Figure 2a. The weight of a shortcut is set to the length of the shortest path between its endpoints. When computing shortest paths, only few shortcuts are explored instead of many arcs in the input graph. The path represented by a shortcut can be obtained lazily, for example by running local Dijkstra searches [8], or by iterating over possible middle nodes when shortcuts always represent exactly two other (shortcut) arcs [11, 15].

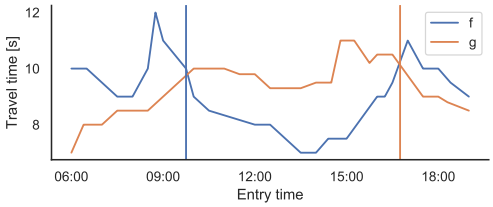
This approach has been extended to the time-dependent setting [2, 5]. Shortcuts are no longer associated with scalar weights. Instead *travel time functions* are used that map the entry time into a shortcut onto the travel time through it. Unfortunately, in practice these functions become very complex. Computing and storing them is expensive. Implementations represent these functions typically as piecewise linear functions. They are stored as a sequence of *breakpoints*. The number of breakpoints in a shortcut's function practically corresponds to the accumulated number of breakpoints of the functions of the arcs it bypasses. Contrary to the classic setting, shortcuts aggregate the complexity of paths they represent, rather than skipping over it. This leads to slow preprocessing and prohibitive memory consumption.

In this paper, we explore an alternative approach to shortcut travel time functions. Rather than explicitly storing them and obtaining paths lazily, we store paths and obtain travel times lazily. We expect that the shortest path between two nodes changes less frequently than the travel time. Intuitively, going via a highway may be slower due to congestion but is usually still the fastest option. Consider the functions f and g in Figure 2b. These functions are travel time functions of two paths between the same endpoints and have many breakpoints. If we want to store the travel time function of a shortcut between these endpoints, we need to store the function $h = \min(f, g)$. Storing h explicitly requires roughly a number of breakpoints proportional to the number of breakpoints in f and g. However, if we only store which path is the fastest, we only need to store the points in time when the faster path switches. We expect significantly fewer switches than breakpoints. In this paper, we employ this alternative approach to adapt an existing speed-up technique to the time-dependent setting, describe engineering techniques employed in our implementation, and present experimental results demonstrating that our approach significantly reduces memory consumption while achieving competitive query times.

Related Work. Routing in road networks has been extensively studied in the past decade. An overview over the field can be found in [1]. Here, we focus on speed-up techniques for time-dependent road networks. Several time-independent speed-up techniques have been generalized to the time-dependent setting. ALT [17], an approach using landmarks to obtain



(a) A shortcut arc (dashed, black) bypassing several nodes. In our implementation, shortcuts always skip over exactly one node and two arcs, which may in turn be shortcut arcs (dashed gray arcs).



(b) Travel time functions for two different paths between the same start and end node.

Figure 2 Shortcuts and their travel time functions.

good A* [19] potentials has been generalized to TD-ALT [24] and successively extended with node contraction to TD-CALT [9]. Even when combined with approximation, TD-CALT queries may take longer than 10 ms on continental sized graphs. SHARC [4], a combination of ARC-Flags [23] with shortcuts which allows unidirectional queries was also extended to the time-dependent scenario [7]. It can additionally be combined with ALT yielding L-SHARC [7]. SHARC can find short paths in less than a millisecond but does not always find a shortest path. MLD/CRP [8, 20] has been extended to TD-CRP [5] which can be used in a time-dependent setting. TD-CRP requires approximation to achieve reasonable memory consumption. It may find suboptimal paths. Another approach is FLAT [21] and its extension CFLAT [22]. CFLAT features sublinear query running time after subquadratic preprocessing and guarantees on the approximation error. Unfortunately, preprocessing takes long in practice and generates a prohibitively large index size.

There are several approaches based on CH [15]. Three were introduced in [2]: Time-dependent CH (TCH), inexact TCH, and Approximated TCH (ATCH). TCH achieve great query performance but at the cost of a huge index size on state-of-the-art continental sized instances. The index size can be reduced at the cost of exactness (inexact TCH) or query performance (ATCH). An open-source reimplementation of [2] named KaTCH[1] exists. A simple heuristic named Time-Dependent Sampling (TD-S) was introduced in [26]. It samples a fixed set of scalar values from the time-dependent functions. It has manageable index sizes and fast query times but does not always find shortest paths.

Contribution and Outline. In this work, we explore a variant of time-dependent CH, where shortcuts store paths instead of travel times. We introduce CATCHUp – Customizable Approximated Time-dependent Contraction Hierarchies trough Unpacking, a time-dependent generalization of Customizable Contraction Hierarchies [11] and a thoroughly engineered implementation. Preprocessing takes only a few minutes even on modern production-grade continental sized instances with tens of millions of nodes. We also present algorithms which allow us to employ approximation to accelerate preprocessing without sacrificing exactness for the queries. Our implementation achieves fast and exact queries with performance competitive to TCH queries while requiring up to 30 times less memory.

The rest of this paper is organized as follows. In Section 2, we introduce some notation and existing algorithms we build on. We describe our shortcut data structure, and the preprocessing and query algorithms in Section 3. In Section 4, we discuss our experimental evaluation. We conclude in Section 5.

[1] `https://github.com/GVeitBatz/KaTCH`

2 Preliminaries

We model road networks as directed graphs $G = (V, A)$. A node $v \in V$ represents an intersection and an arc $a = uv \in A$ with $u, v \in V$ represents a road segment. A path is a sequence of nodes $[v_1, ..., v_k]$ such that $v_i v_{i+1} \in A$. Every arc a has a *travel time function* $f_a : \mathbb{R} \to \mathbb{R}^{>0}$ mapping departure time to travel time. We assume that travel time functions fulfill the *First-In-First-Out* (FIFO) property, that is for any $\sigma, \tau \in \mathbb{R}$ with $\sigma \leq \tau$, $\sigma + f(\sigma) \leq \tau + f(\tau)$ has to hold. Informally, this means that it is not possible to arrive earlier by starting later. If there are arcs that do not fulfill the FIFO property, the shortest path problem becomes $\mathcal{NP}$-hard [25]. In our implementation, travel time functions are periodic piecewise linear functions represented by a sequence of *breakpoints*. We denote the number of breakpoints by $|f|$.

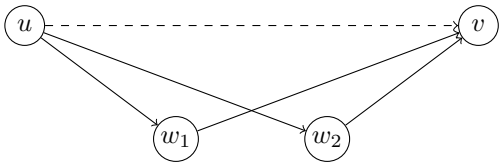
Given two travel time functions f and g for arcs uv and vw, we are often interested in the travel time function of traversing first uv and then vw, that is $(g \circ f)(\tau) = f(\tau) + g(f(\tau) + \tau)$. Computing this function is called *linking*. When combining two travel time functions f and g for different paths $[u, ..., v]$ with the same start and end, we often want to know the travel time of the best path between u und v, that is $\min(f, g)$. Computing this function is called *merging*. Both linking and merging can be implemented with coordinated linear sweeps over the breakpoints of both functions.

Given a departure time τ and nodes s and t, an earliest-arrival query asks for earliest point in time one can arrive at t when starting from s at τ. Such a query can be handled by Dijkstra's algorithm [12] with little modifications [14]. The algorithm keeps track of the earliest known arrival time ea_v at each node v. These labels are initialized with τ for s and ∞ for all other nodes. A priority queue is initialized with (s, τ). In each step, the node u with minimal earliest arrival ea_u is popped from the queue and outgoing arcs are *relaxed*. To relax an arc uv, the algorithm checks if $\text{ea}_u + f_{uv}(\text{ea}_u)$ improves ea_v and updates label and queue position of v accordingly. Once t is extracted from the queue, the earliest arrival at t is known. To retrieve the shortest path, one can use *parent pointers* which for each node store the previous node on the shortest path from s. We refer to this algorithm as *TD-Dijkstra*.

The *A* algorithm* [19] is an extension to Dijkstra's algorithm. It reduces the number of explored nodes by guiding the search towards t. Each node u has a potential $\rho_t(u)$ which is an estimate of the distance to t. The priority queue is then ordered by $\text{ea}_u + \rho_t(u)$.

Contraction Hierarchies (CH) [15] is a speed-up technique exploiting the inherent hierarchy in road networks. Nodes are heuristically ranked by their importance. Nodes with higher rank should cover more shortest paths. During preprocessing, all nodes are *contracted* in order of ascending importance. Contracting a node v means removing it from the network but preserving all shortest distances among the remaining higher ranked nodes. This is achieved by inserting *shortcut arcs* between the neighbors of v if a shortest path goes through v. A shortcut is only necessary if it represents the only shortest path between its endpoints. This can be checked with a local Dijkstra search (called *witness search*) between the endpoints. The result of the preprocessing is called an *augmented graph*. Queries can be answered by performing a bidirectional Dijkstra search on the augmented graph where only arcs to higher ranked nodes are relaxed. The construction guarantees that this algorithm will find a shortest up-down-path which has the same length as shortest paths in the original graph.

Customizable Contraction Hierarchies (CCH) [11] is a CH extension, splitting CH preprocessing into two steps where only the second one uses arc weights. In the first step, a separator decomposition and an associated nested dissection order [3, 16] are computed. This order determines the node ranks. Nodes in the top-level separators have the highest ranks,



00:00	uw_1	w_1v
07:32	uw_2	w_2v
15:42	uw_1	w_1v

Figure 3 A shortcut with associated time-dependent unpacking information.

followed by the nodes of each cell, recursively ordered by the same method. Then, nodes are contracted without running witness searches, so all potential shortcuts are added. In the second step (called *customization*), shortcut arc weights are computed. All arcs are processed in ascending order of their lower ranked endpoint. To process an arc uv, all *lower triangles* $[u, w, v]$, where w has lower rank than u and v are enumerated, checking if the path $[u, w, v]$ can improve the weight of uv. The CH query algorithm can be reused without modifications. Another query algorithm is described in [11] based on the *elimination tree* which does not need priority queues. The parent of each node in the elimination tree is its lowest-ranked upward neighbor in the augmented undirected graph. A nodes ancestors in the elimination tree are exactly the set of nodes that are reachable in a CH search from this node [3]. Thus, instead of exploring the search space through Dijkstra searches, the elimination tree query traverses the same nodes by traversing the path to the root in the elimination tree.

3 Algorithms

In this section, we describe our algorithms, data structures and implementation. Our approach builds upon CCH. However, instead of storing travel time functions at shortcuts, we store unpacking information to efficiently reconstruct the represented paths in the original graph. We start by describing our representation of this unpacking information. Then, we continue by presenting our adapted CCH algorithms for the time-dependent setting.

3.1 Shortcut Data Structure

The key element of our approach is the information we store with each shortcut. We store time-dependent unpacking information, which allows us to efficiently reconstruct the original path represented by a shortcut for a point in time. In (C)CH, shortcuts uv are inserted when a node w is contracted and the arcs uw and wv exist. Thus, a shortcut uv always skips over a triangle $[u, w, v]$. However, there may be several triangles and which one is the fastest may change over time. This is the information our shortcut data structure has to capture.

For each shortcut arc uv, we store a set of time-dependent *expansions* X_{uv} for unpacking. See Figure 3 for an example. For an expansion $x \in X_{uv}$, we denote the time during which x represents the shortest path as the *validity interval* Π_x of x and the lower node of the triangle $[u, w_x, v]$ as w_x. In our implementation, the expansion information is represented as an array of triples (π, uw_x, w_xv). π is the beginning of the validity interval and uw and wv are arc ids. This information can be stored in 16 bytes for each entry – 8 bytes for the timestamp and 4 bytes for each arc id. Beside the unpacking information, we also store a scalar lower bound $\underline{b}_{uv}$ and an upper bound $\overline{b}_{uv}$ to prune unnecessary operations.

To obtain the shortest path represented by a shortcut uv for a time τ, we first need to determine the relevant expansion x such that $\tau \in \Pi_x$. This can be done performing a binary search in X_{uv}. The shortcut can then be expanded to $[u, w_x, v]$. However, uw_x or w_xv may also be shortcuts. Thus, we recursively apply the operation to (u, w_x) at τ and to (w_x, v) at

$\tau + f_{(u,w_x)}(\tau)$. Evaluating the travel time of the first arc is always necessary to determine the time for unpacking for the second arc. The travel time of a shortcut can be obtained by unpacking the path and then evaluating the travel times of the original arcs successively.

3.2 Preprocessing

The first step of CCH preprocessing is performed only on the topology of the graph. Since no travel time functions are involved, we can adapt the algorithms of [11] without modification. We use InertialFlowCutter [18] to obtain the nested dissection order. To generate the shortcut augmented graph, we implement an improved contraction algorithm first presented in [27]. When contracting a node, we insert all upward neighbors of the current node only into the neighborhood of its lowest ranked upward neighbor. This algorithm can be implemented to run in linear time in the size of the output graph.

The goal of the second step of preprocessing for classical CCH is to compute the shortcut weights. So for our approach, we have to compute travel time bounds and unpacking information for all shortcuts[2]. Recall that a shortcut uv always bypasses one or many lower triangles $[u, w_i, v]$ for different nodes w_i, where w_i has lower rank than u and v. For the bounds, we want to find the minimum and maximum travel time of the fastest travel time function between u and v over any w_i. For the unpacking information, we need to determine for each point in time which triangle is the fastest. Assuming we know the final travel time functions of all uw_i and w_iv, we can obtain the travel time function for each triangle by linking f_{uw_i} and f_{w_iv}. By merging these linked functions, we can compute both the final bounds and the times during which each triangle is the fastest. This leads to the following algorithmic schema: Iterate over all arcs in a bottom-up fashion. For each arc enumerate lower triangles. Link and merge their functions to compute the function, bounds, and unpacking information of the current arc. Keep the current arc's travel time function in memory until it is no longer needed.

We implement this schema as follows: We process all arcs uv ordered ascending by their lower ranked endpoint. Since the middle node w of a lower triangle $[u, w, v]$ has always lower rank than u and v, the arcs uw and wv will have been processed already. To process an arc uv we enumerate lower triangles $[u, w, v]$. For each triangle, we obtain the function of $[u, w, v]$ by linking the functions of the arcs uw and wv, and merge it with the current function of the shortcut uv. Once all arcs uv have been processed where u is the lower ranked endpoint, we drop the travel time functions of all arcs wu where u is the higher ranked endpoint.

When enumerating triangles, we order them ascending by $\underline{b}_{uw} + \underline{b}_{wv}$. This way, we process triangles, which are likely faster first. This gives us preliminary bounds on the travel time of uv. Before linking the functions of another triangle f_{uw} and f_{wv}, we check if $\overline{b}_{uv} \leq \underline{b}_{uw} + \underline{b}_{wv}$. If so, the linked path would be dominated by the shortcut, and we can skip linking and merging completely. If not, we link f_{uw} and f_{wv} and obtain $f_{[u,w,v]}$. We still can skip merging if one function is strictly smaller than the other, that is either $\overline{b}_{uv} \leq \min(f_{[u,w,v]})$ or $\max(f_{[u,w,v]}) \leq \underline{b}_{uv}$. Even if the bounds overlap, one function might still dominate the other. To check for this case, we simultaneously sweep over the breakpoints of both functions, determining the value of the respectively other function by linear interpolation. Only when this check fails, we perform the merge operation.

[2] Note that actually all arcs in the augmented graph have to be treated as shortcuts. We represent arcs from the original graph as a special expansion element pointing to the original arc.

Before the time-dependent customization, we first use the basic and perfect customization algorithms from [11] to compute preliminary scalar upper and lower bounds for all shortcuts. With these bounds, we can skip additional linking and merging operations. Also, we can remove some shortcuts completely, when a shorter path through higher ranked nodes exists.

Parallelization. We employ both loop based and task based parallelism. The original CCH publication [11] suggests processing shortcuts with their lower ranked endpoint on the same *level* in parallel. The level of a node is the level of its highest ranked downward neighbor increased by one, or zero if the node does not have downward neighbors. We use this approach to process arcs in the top-level separators.

In [6], a task based parallelism approach utilizing the separator decomposition of the graph is described. Each task is responsible for a subgraph G'. Removing the top-level separator in G' decomposes the subgraph into two or more disconnected components. For each component, a new task is spawned to process the shortcuts the component. After all child tasks are completed, the shortcuts in the separator are processed utilizing the loop based parallelization schema. If the size of subgraph G' is below a certain threshold, the task processes the shortcuts in G' sequentially without spawning subtasks.

Approximation. As we process increasingly higher ranked shortcuts, the associated travel time functions become more and more complex. This leads to two problems. First, linking and merging becomes very time-consuming as running times scale with the complexity of the input functions. Second, storing these functions for later reuse – even though it is only temporary – requires a lot of memory. We employ approximation to mitigate these issues. However, for exact queries, we need exact shortcut unpacking information. We achieve this by lazily reconstructing parts of exact travel time functions during merging.

When approximating, we do not store one approximated function but two – a lower bound function and an upper bound function with maximum error ϵ where ϵ is a configurable parameter. These approximations replace the exact function stored for later merge operations and will also be dropped when no longer needed. To obtain the bound functions, we first compute an approximation using the algorithm of Douglas and Peucker [13]. Then, we add or subtract ϵ to the value of each breakpoint to obtain an upper or lower bound, respectively. This yields valid upper or lower bounds, but they may not be as tight as possible. Therefore, we iterate over all approximated points and move each point back towards the original function. Both adjacent segments in the approximated functions have a minimum absolute error to the original function. We move the breakpoint by the smaller of the two errors. This yields sufficiently good bounds.

When linking approximated functions, we can link both lower and both upper bound functions. Linking two lower bounds yields a valid lower bound function of the linked exact functions because of the FIFO property. The same argument holds for upper bounds.

Merging approximated shortcuts is slightly more involved. Our goal is to determine the exact unpacking information for each shortcut. We use the approximated bounds to narrow down the time ranges where intersections are possible. To identify these parts, we merge the first function's lower bound with the second function's upper bound and vice versa. Where the bounds overlap, an intersection might occur. We then obtain the exact functions in the overlapping parts through unpacking and perform the exact merging. To obtain approximated upper and lower bounds of the merged function, we merge both lower bounds and both upper bounds.

We approximate whenever a function has more than β breakpoints. This includes already approximated functions. Both β and the maximum error ϵ are tuning parameters which influence the performance (but not the correctness).

3.3 Queries

Our query algorithm is based on the CCH elimination tree query algorithm [11]. There are two challenges. First, we can not perform a backwards search, as we do not know the arrival time at the target node. Second, to evaluate the travel time of a shortcut, we need to obtain the path in the original graph. Since unpacking shortcuts is expensive, we try to keep the number of these operations as small as possible.

Our query algorithm works in two phases. In the first phase, we perform an elimination tree interval query using the lower and upper travel time bounds of the arcs. The backward search is possible since these bounds are not time-dependent. This yields a shortest path corridor. This corridor is made up of both original and shortcut arcs and always contains the actual shortest path. In the second phase, we perform a unidirectional Dijkstra search on this corridor, which lazily unpacks shortcuts.

Elimination Tree Interval Query. The elimination tree interval query is a bidirectional search starting from both the source node s and the target node t. Node labels contain an upper $\overline{t}_v$ and a lower bound $\underline{t}_v$ on the travel time to/from the search origin and a parent pointer to the previous node. The bounds $\overline{t}_s$, $\overline{t}_t$, $\underline{t}_s$, $\underline{t}_t$ are all initialized to zero in their respective direction, all other bounds to infinity. We also track tentative travel time bounds for the total travel time from s to t. For both directions, the path from the start node to the root of the elimination tree is traversed. For each node u, all arcs uv to higher ranked neighbors are relaxed, that is checking if $\overline{t}_u + \overline{b}_{uv} < \overline{t}_v$ or $\underline{t}_u + \underline{b}_{uv} < \underline{t}_v$ and improving the bounds of v if possible. When the new travel time bounds from an arc relaxation overlap with the current bounds, more than one label has to be stored. After both directions are finished, we have several meeting nodes in the intersection of the search spaces. Where the sum of the forward and backward distance bounds of a node overlaps with the total travel time bounds, the parent pointers are traversed to the search origin and all arcs on the paths are marked as part of the shortest path corridor.

Lazy Corridor Dijkstra. In the second query phase, we perform Dijkstra's algorithm on the corridor obtained in the first phase. At the beginning, the corridor contains many shortcuts, which need to be unpacked, to determine their travel time. We perform unpacking lazily. The algorithm starts with the same initialization as a regular TD-Dijkstra. All earliest arrivals are set to infinity, except for the start node which is set to the departure time. The start node is inserted into the queue. Then, nodes are popped from the queue until it is empty or the target node is reached. For each node, all outgoing arcs within the shortest path corridor are relaxed. When an arc is from the original graph, the travel time can be evaluated directly. Shortcut arcs, however, need to be unpacked. The unpacking algorithm defers the work as much as possible: Only the first arc of the triangle of each shortcut will be recursively unpacked until an original arc is reached, the second arc will be added to the corridor. See Figure 4 for an example. This way, we unpack only the necessary parts and avoid relaxing arcs multiple times when shortcuts share the same paths.

Corridor A*. The query can be accelerated further, by using the lower bounds obtained during the elimination tree interval query as potentials for an A*-search. For nodes in the CH search space of t, the lower bounds from the backward search can be used. For nodes in the

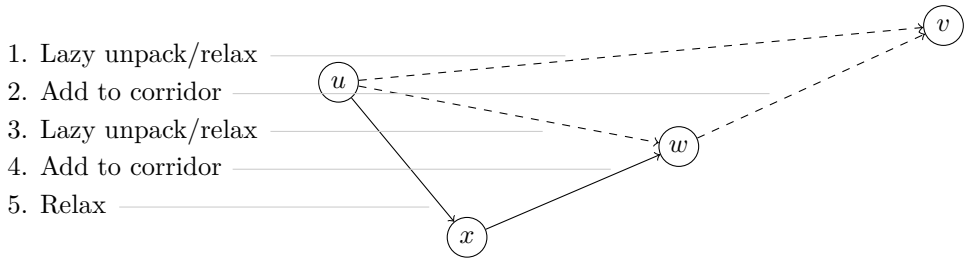


Figure 4 Lazy relaxation of shortcut uv. Since uv is a shortcut, it needs to be unpacked. This causes wv to be added to the corridor and uw to be relaxed. Relaxing uw causes xw to be added to the corridor and ux to be relaxed. In this example, ux is an original arc and the recursion stops. xw will be relaxed (or unpacked) only once x is popped from the queue.

Table 1 Characteristics of test instances used. The third column contains the percentage of arcs with a non-constant travel time. The fourth column the average number of breakpoints among those.

	Nodes [$\cdot 10^3$]	Arcs [$\cdot 10^3$]	TD arcs [%]	Avg. $\lvert f \rvert$ per TD arc	Size [GB]
Ger06	4 688	10 796	7	17.6	0.2
Ger17	7 248	15 752	29	29.6	0.7
Eur17	25 758	55 504	27	27.5	2.3

CH search space of s, we start at the meeting nodes from the corridor search and propagate the bounds backwards down along the parent pointers. This yields potentials for all nodes in the initial corridor. However, we also need potentials for nodes added to the corridor through unpacking. These potentials are computed during the shortcut unpacking. When unpacking a shortcut uv into the arcs uw and wv, then $\rho(w)$ will be set to $\min(\rho(w), \rho(v) + \underline{b}_{wv})$.

4 Experiments

We implement our algorithms in Rust[3] and compile them with rustc 1.36.0-nightly (372be4f36 2019-05-14) in the release profile with the target-cpu=native option[4]. To compile competing implementations written in C++, we use GCC 7.4. All experiments were conducted on a dual 8-core Intel Xeon Gold 6144 CPU with a base frequency of 3.5 GHz with 192 GiB of DDR4 RAM (clocked at 2.6 GHz). Preprocessing utilized all 16 cores (without hyperthreading). The running times are averages over five runs. We generate 100 000 source, target, departure time triples chosen uniformly at random for each graph and report average query times. Queries were performed sequentially.

We use two production-grade instances for Germany and Europe and with traffic predictions from 2017. To compare our algorithms to related work, we also include an old instance of Germany from 2006. The instances were provided by PTV[5] and include traffic predictions as piecewise linear functions. We use traffic predictions for a car on a typical midweek day. Table 1 lists key characteristics of each graph.

Table 2 reports numbers on our preprocessing. On Ger06, the first preprocessing step takes longer than the second. However, for the newer instances with more time-dependent arcs and more breakpoints per function this changes and the second step becomes more

[3] The code is available at `https://github.com/kit-algo/catchup`.
[4] We disabled AVX512 instructions, as they caused misoptimizations.
[5] `https://ptvgroup.com`

Table 2 Preprocessing statistics. Running times are for parallel execution on 16 cores.

	CCH arcs	Expansions per shortcut			Index	Step 1	Step 2
	$[\cdot 10^3]$	Avg.	Max.	$= 1$ [%]	[GB]	[s]	[s]
Ger06	22 521	1.075	44	98.4	1.06	31	18
Ger17	31 517	1.090	107	98.5	1.50	35	92
Eur17	115 023	1.100	115	98.4	5.48	196	479

Table 3 Mean query performance and search space sizes with and without our A* query extension.

	Elimination tree interval query		Lazy corridor Dijkstra					
	Nodes in search space	Relaxed shortcuts	Queue pops		Relaxed arcs		Time [ms]	
			no A*	A*	no A*	A*	no A*	A*
Ger06	735	48 574	3 328	831	3 845	995	1.62	0.54
Ger17	770	58 903	18 499	3 099	19 986	3 502	8.70	1.59
Eur17	1 302	162 295	39 717	6 876	43 654	7 928	19.55	3.76

expensive. Despite that, the size of the final index corresponds to the number of shortcut arcs and does not grow as much for the newer instances. The augmented graphs have about twice as many arcs as the original graphs. On average, only 1.1 expansions per shortcut need to be stored for all graphs. About 98% of all shortcuts have only one expansion. The maximum number of expansions per shortcuts is only 115 even for our largest graph. This is two orders of magnitude less than the number of breakpoints in the travel time function of that shortcut. This clearly shows the superiority of shortcuts with expansion information over explicitly storing travel time functions.

Table 3 depicts the performance of our query algorithms in terms of running time and search space sizes. Without the A* optimization, queries take up to 5.5 times longer and the search space of the corridor Dijkstra grows roughly by the same factor. Both query variants relax only little more arcs than they settle nodes. This is caused by the shortcut unpacking. Large parts of the unpacked search space are just long paths of nodes with degree two.

Table 4 provides an overview over different techniques, their preprocessing and query times, space overhead of the index data structures and average query errors where approximation is used. Where possible, we obtained the code of competing algorithms[6] and evaluated them with same methodology, instances and queries as our algorithms. For other competitors, we report available numbers from the respective publications.

In our comparison, KaTCH, heu SHARC, CFLAT and CATCHUp all achieve query times around 0.6 ms on Ger06. The original research implementation TCH reports slightly slower times than KaTCH. This may be because experiments were run on an older machine, but also because according to the KaTCH documentation, the newer query is somewhat more efficient. TCH pays for this speed with 4.7 GB index data. Reducing the KaTCH memory consumption while keeping exactness (ATCH) brings query times up to 1.24 ms. ATCH also feature a configuration where they only keep upper and lower bounds for each profile (ATCH ∞). This configuration uses even less memory than CATCHUp because the optimized order results in fewer shortcuts. Giving up on exactness allows keeping the query times at 0.7 ms (inex. TCH) but introduces some noticeable errors.

[6] KaTCH: `https://github.com/GVeitBatz/KaTCH`
TD-S: `https://github.com/ben-strasser/td_p`

Table 4 Comparison with related work. We list unscaled numbers as reported in the respective publications for algorithms we could not run ourselves. Values not reported are indicated as n/r. OOM means that the program crashed while trying to allocate more memory than available. A similar overview with scaled numbers can be found in [10].

		Preprocessing		Index	Query		
		Time	Cores	size	Time	Rel. error	
		[s]		[GB]	[ms]	Avg. [%]	Max. [%]
Ger06	TD-Dijkstra	-	-	-	525.48	exact	
	TDCALT [9]	540	1	**0.23**	5.36	exact	
	TDCALT-K1.15 [9]	540	1	**0.23**	1.87	0.050	13.840
	eco L-SHARC [7]	4 680	1	1.03	6.31	exact	
	heu SHARC [7]	12 360	1	0.64	0.69	n/r	0.610
	KaTCH	170	16	4.66	0.63	exact	
	TCH [2]	378	8	4.66	0.75	exact	
	ATCH (1.0) [2]	378	8	1.12	1.24	exact	
	ATCH (∞) [2]	378	8	0.55	1.66	exact	
	inex. TCH (0.1) [2]	378	8	1.34	0.70	0.020	0.100
	inex. TCH (1.0) [2]	378	8	1.00	0.69	0.270	1.010
	TD-CRP (0.1) [5]	289	16	0.78	1.92	0.050	0.250
	TD-CRP (1.0) [5]	281	16	0.36	1.66	0.680	2.850
	FLAT [22]	158 760	6	54.63	1.27	0.015	n/r
	CFLAT [22]	104 220	6	34.63	**0.58**	0.008	0.918
	TD-S+9	547	1	3.61	1.67	0.001	1.523
	CATCHUp	**49**	16	1.06	0.70	exact	
Ger17	TD-Dijkstra	-	-	-	869.79	exact	
	KaTCH	874	16	42.81	**1.38**	exact	
	TD-S+9	617	1	5.28	2.28	0.001	0.963
	CATCHUp	**127**	16	**1.50**	1.86	exact	
Eur17	TD-Dijkstra	-	-	-	2 581.16	exact	
	KaTCH	3 089	16	146.97	OOM	exact	
	TD-S+9	3 368	1	18.84	**4.03**	0.002	1.159
	CATCHUp	**675**	16	**5.48**	4.50	exact	

While achieving competitive query times for acceptable memory consumption, heu SHARC suffers from huge preprocessing times of several hours. The original publication does not report average query errors, only a maximum error of 0.61%. TDCALT has the smallest memory consumption of all approaches but does not achieve competitive query times, even when approximating. FLAT and CFLAT both suffer from extreme preprocessing times and memory consumption despite having no exact queries. CATCHUp offers competitive query times for exact results while keeping memory consumption reasonable. TD-CRP offers even lower memory consumption. But this is only possible through the use of approximation. TD-CRP queries depict a noticeable error and perform somewhat worse than KaTCH or CATCHUp queries. TD-S+9 depicts the smallest average error of all non-exact approaches[7].

[7] [22] report another CFLAT configuration with even smaller errors but significantly slower queries.

On Ger17, KaTCH query times increase by a factor of about two. Memory usage on the other hand, grows by almost an order of magnitude. For TD-S, both the growth in space consumption and query times corresponds roughly to the growth of the graph size, but not the increased number of breakpoints. For the index data of CATCHUp, this growth factor also roughly applies. Query times get about 2.7 times slower.

On Eur17, the memory consumption of KaTCH becomes prohibitive. While KaTCH is still able to finish preprocessing and output 150 GB of data, queries crash since the 192 GB RAM of our machine are not enough. Using ATCH or inexact TCH, the memory consumption could likely be reduced sufficiently to perform queries. But this would either introduce errors or slow down queries significantly. On the other hand, with only 5.5 GB of index data, CATCHUp is still able to perform exact queries in less than 5 ms on average. This is fast enough to enable interactive applications. Total preprocessing for CATCHUp takes less than a quarter of the time KaTCH needs. TD-S+9 is also able to handle this instance with similar query times but only with a small average error.

5 Conclusion and future work

We introduce CATCHUp, a speed-up technique for routing in time-dependent road networks. It features a small index size and fast, exact queries. To the best of our knowledge, our approach is the first to simultaneously achieve all three objectives. We perform an experimental study to evaluate the performance of CATCHUp and compare it to competing approaches. Our approach achieves the fastest preprocessing, competitive query running times and up to 30 times smaller indexes than other approaches.

Revisiting ATCH, TCH, and TD-CRP with the insights gained in this work could be fruitful. Combining ATCH with our A* query extension could reduce ATCH query running times. CATCHUp makes use of travel time independent node orders. Combining CATCHUp with TCH-like node orders could result in even smaller index sizes and query running times. We further expect that some of our optimizations to the preprocessing can also be applied in a TD-CRP context. Another possible direction for future research would be to support partial updates to further accelerate the second step of preprocessing. This could enable the integration of live traffic information.

References

1 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016. URL: `http://www.springer.com/gp/book/9783319494869`.

2 Gernot Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum Time-Dependent Travel Times with Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 18(1.4):1–43, April 2013.

3 Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. *Theoretical Computer Science*, 645:112–127, 2016.

4 Reinhard Bauer and Daniel Delling. SHARC: Fast and Robust Unidirectional Routing. *ACM Journal of Experimental Algorithmics*, 14(2.4):1–29, August 2009. Special Section on Selected Papers from ALENEX 2008. `doi:10.1145/1498698.1537599`.

5 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Dynamic Time-Dependent Route Planning in Road Networks with User Preferences. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA'16)*, volume 9685 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2016. URL: `http://link.springer.com/chapter/10.1007/978-3-319-38851-9_3`.

6 Valentin Buchhold, Peter Sanders, and Dorothea Wagner. Real-time Traffic Assignment Using Engineered Customizable Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 24(2):2.4:1–2.4:28, 2019. URL: https://dl.acm.org/citation.cfm?id=3362693.

7 Daniel Delling. Time-Dependent SHARC-Routing. *Algorithmica*, 60(1):60–94, May 2011. doi:10.1007/s00453-009-9341-0.

8 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable Route Planning in Road Networks. *Transportation Science*, 51(2):566–591, 2017. doi:10.1287/trsc.2014.0579.

9 Daniel Delling and Giacomo Nannicini. Core Routing on Dynamic Time-Dependent Road Networks. *Informs Journal on Computing*, 24(2):187–201, 2012.

10 Julian Dibbelt. *Engineering Algorithms for Route Planning in Multimodal Transportation Networks*. PhD thesis, Karlsruhe Institute of Technology, February 2016. URL: http://nbn-resolving.de/urn:nbn:de:swb:90-530503.

11 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, April 2016. doi:10.1145/2886843.

12 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

13 David H. Douglas and Thomas K. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10:112–122, 1973.

14 Stuart E. Dreyfus. An Appraisal of Some Shortest-Path Algorithms. *Operations Research*, 17(3):395–412, 1969.

15 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

16 Alan George. Nested Dissection of a Regular Finite Element Mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

17 Andrew V. Goldberg and Chris Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 156–165. SIAM, 2005.

18 Lars Gottesbüren, Michael Hamann, Tim Niklas Uhl, and Dorothea Wagner. Faster and Better Nested Dissection Orders for Customizable Contraction Hierarchies. *Algorithms*, 12(9):196, 2019. doi:10.3390/a12090196.

19 Peter E. Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

20 Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering Multilevel Overlay Graphs for Shortest-Path Queries. *ACM Journal of Experimental Algorithmics*, 13(2.5):1–26, December 2008.

21 Spyros Kontogiannis, George Michalopoulos, Georgia Papastavrou, Andreas Paraskevopoulos, Dorothea Wagner, and Christos Zaroliagis. Engineering Oracles for Time-Dependent Road Networks. In *Proceedings of the 18th Meeting on Algorithm Engineering and Experiments (ALENEX'16)*, pages 1–14. SIAM, 2016. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611974317.1.

22 Spyros Kontogiannis, Georgia Papastavrou, Andreas Paraskevopoulos, Dorothea Wagner, and Christos Zaroliagis. Improved Oracles for Time-Dependent Road Networks. In *Proceedings of the 17th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'17)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 4:1–4:17, 2017. doi:10.4230/OASIcs.ATMOS.2017.4.

23 Ulrich Lauther. An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung*, volume 22, pages 219–230. IfGI prints, 2004.

24 Giacomo Nannicini, Daniel Delling, Leo Liberti, and Dominik Schultes. Bidirectional A* Search on Time-Dependent Road Networks. *Networks*, 59:240–251, 2012. Best Paper Award.

25 Ariel Orda and Raphael Rom. Traveling without waiting in time-dependent networks is NP-hard. Technical report, Dept. Electrical Engineering, Technion-Israel Institute of Technology, 1989.

26 Ben Strasser. Dynamic Time-Dependent Routing in Road Networks Through Sampling. In *Proceedings of the 17th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'17)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 3:1–3:17, 2017. doi:10.4230/OASIcs.ATMOS.2017.3.

27 Michael Zündorf. Customizable Contraction Hierarchies with Turn Costs. Bachelor thesis, Karlsruhe Institute of Technology, 2019.

Improved Prophet Inequalities for Combinatorial Welfare Maximization with (Approximately) Subadditive Agents

Hanrui Zhang
Duke University, Durham, NC, USA
hrzhang@cs.duke.edu

Abstract

We give a framework for designing prophet inequalities for combinatorial welfare maximization. Instantiated with different parameters, our framework implies (1) an $O(\log m/\log\log m)$-competitive prophet inequality for subadditive agents, improving over the $O(\log m)$ upper bound via item pricing, (2) an $O(D\log m/\log\log m)$-competitive prophet inequality for D-approximately subadditive agents, where $D\in\{1,\dots,m-1\}$ measures the maximum number of items that complement each other, and (3) as a byproduct, an $O(1)$-competitive prophet inequality for submodular or fractionally subadditive (a.k.a. XOS) agents, matching the optimal ratio asymptotically. Our framework is computationally efficient given sample access to the prior and demand queries.

2012 ACM Subject Classification Theory of computation → Stochastic approximation; Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases Prophet Inequalities, Combinatorial Welfare Maximization, (Approximate) Subadditivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.82

Funding *Hanrui Zhang*: Supported by NSF award IIS-1814056.

Acknowledgements The author thanks Yuan Deng, Kamesh Munagala, and anonymous reviewers for helpful feedback.

1 Introduction

Prophet inequalities are a classical topic in stopping theory. The problem is neat and natural: an agent plays a game, where there are n boxes, each containing a reward (e.g., some amount of cash). The agent cannot see through the boxes to know precisely the amounts of cash inside each box. However, she has the prior knowledge that the amounts are drawn independently for each box, and fortunately, knows the distributions according to which the amounts are drawn. Now nature opens the boxes one by one. Upon seeing the inside of each box, the agent gets to make a choice: she can either (1) take the cash in the box and leave, or (2) let the current box expire (which means she gains nothing from the current box and it disappears), in which case the game proceeds with the remaining boxes. What is the maximum expected amount of cash the agent can get, and how to achieve that?

Quite surprisingly, the agent can guarantee half the amount of reward that a *prophet* is able to get, who sees through the boxes and therefore always picks the box with the largest reward [21, 22]. Moreover, the agent can achieve this by executing a simple threshold-based protocol: accept the first box containing a reward exceeding a pre-calculated amount. The existence of such 2-approximate protocols lead to the name "prophet inequalities."

Prophet inequalities were recently rediscovered in computer science and economics. Since then, they have been drawing increasing interest in both fields. Hajiaghayi et al. [18] observe the connection between prophet inequalities and a pricing problem in auctions. They formulate the problem in the following equivalent way: a seller has an indivisible item to

sell. n buyers arrive one by one, each of which has a value for the item, drawn independently from a distribution known to the seller. When a buyer arrives, the seller learns the value of the buyer (or otherwise negotiates with the buyer), and decides whether to sell the item to the buyer. The buyer then leaves forever (with the item if sold to the buyer). The goal of the seller is to maximize the utility of the buyer who receives the item, or just the total utility (i.e., the *welfare*) of all buyers, since all other buyers have utility 0. Here, each buyer corresponds to a box in the classical formulation, and the value of the buyer represents the reward in the box. A threshold-based protocol can then be translated directly into a take-it-or-leave-it offer – a buyer receives the item (i.e., she *buys*) iff her value exceeds the pre-calculated price, and so buying is preferred to not buying.

Given this connection, various forms of auctions have been considered in the prophet inequality context (see, e.g., the recent survey by Lucier [23]). Examples include (1) the case where the seller has k identical items to sell and each buyer wants only one of them (or equivalently, up to k boxes can be accepted) [18], (2) the setting with a knapsack style constraint, requiring that the total "weight" of sellers who get an item cannot exceed 1 [17], and (3) the setting where m possibly distinct items are available for sale, and each agent has a *combinatorial* (as opposed to additive) valuation function assigning every subset of items a value [16]. The third setting, known as *combinatorial welfare maximization*, appears particularly interesting and general, as it nicely captures the potentially complex interaction between items. For instance, a Coke and a Pepsi substitute each other, in the sense that having a Coke or a Pepsi probably gives one roughly the same utility (say 1), whereas having both likely gives *strictly less* utility than the sum, 2, of the former values, since one can only drink so much at a time.

In this paper, we consider combinatorial welfare maximization in the prophet inequality context. We begin our investigation with *subadditive* (also known as *complement-free*) agents, who regard items only as substitutes but not complements to each other. From there, we generalize our results to accommodate valuations that are *approximately subadditive*, which have remained largely unexplored even in offline environments.

1.1 Our Contributions

1.1.1 Current Landscape of the Problem

Feldman et al. [16] were the first to explicitly study combinatorial welfare maximization with rich valuations in the prophet inequality context. They give an existential 2-approximate protocol and a computationally efficient $(2e)/(e-1)$-approximate protocol when agents are submodular or fractionally subadditive (which are strict subclasses of subadditive valuations). Dütting et al. [10] propose a powerful framework, unifying a number of prophet inequalities, and yielding a computationally efficient 2-approximate protocol for the same class of valuations, which is optimal given a lower bound inherited from the single-item setting. These bounds, through a standard approximation result, extend directly to subadditive agents with a loss of factor $O(\log m)$, where m is the number of items. The best known lower bound for subadditive agents, however, is again 2, leaving a huge gap in between. This gap, as acknowledged by Feldman et al. [16] and Dütting et al. [10], raises a curious question:

Can we do better than $O(\log m)$ for subadditive agents?

1.1.2 The Logarithmic Barrier for Subadditive Agents

The above question did not have an immediate answer. For subadditive agents, all existing results essentially build on the same argument: one first computes prices that $O(\log m)$-approximately "support" the optimal allocation. Such prices have the property, that if one posts the prices on the items, and let agents, one by one in some arbitrary order, purchase their utility-maximizing bundles of items, then the resulting allocation $O(\log m)$-approximately maximizes the expected welfare. With these prices, one can implement an $O(\log m)$-approximate threshold-based protocol, by running a posted-price auction with the supporting prices for each arriving agent, and allocating the set of items purchased to the agent. The bottleneck of this approach is that the $O(\log m)$ factor of approximate supporting prices is tight: there are subadditive valuations for which no $o(\log m)$-approximate supporting prices exist [2]. Therefore, any protocol relying on supporting prices (including all currently existing results) cannot possibly give better ratios than $O(\log m)$. In fact, given the tightness of this $O(\log m)$ factor, one may even suspect that the right ratio for subadditive agents is precisely $\Theta(\log m)$. We show that this is *not* the case.

1.1.3 A Sublogarithmic Prophet Inequality for Subadditive Agents

We give a framework for designing prophet inequalities for combinatorial welfare maximization, which implies an $O(\log m/\log\log m)$-approximate prophet inequality for subadditive agents, breaking the foregoing logarithmic barrier. Our framework is computationally efficient given: (1) sample access to the prior distributions, and (2) demand queries to the sample valuations. Unlike previous results, our protocol is not based on pricing items via approximately supporting prices and running sequential auctions – which enables the protocol to bypass the obstacle discussed above, at the cost of losing incentive compatibility.[1] As a byproduct, we show that our framework, instantiated with different parameters, also gives an $O(1)$-approximately optimal prophet inequality for submodular or fractionally subadditive agents. Our approach provides an alternative view of combinatorial welfare maximization in the prophet inequality context, which may be of independent interest.

Very recently, in independent work, Dütting et al. [11] give an $O(\log\log m)$-approximate prophet inequality for combinatorial welfare maximization with subadditive agents using radically different techniques from ours. Their result is a major breakthrough in the research of prophet inequalities for combinatorial welfare maximization. In particular, the work of Dütting et al. significantly improves over our main result in that (1) they "truely" improve the approximation ratio from $O(\log m)$ all the way to $O(\log\log m)$, and (2) their approach is based on pricing individual items, which shows that it is possible to beat $O(\log m)$ using item-pricing schemes, which are incentive-compatible.

1.1.4 Generalizing to Approximate Subadditivity

Utilizing the same framework, we give a family of parameters, which imply an $O(D\log m/\log\log m)$-approximate prophet inequality when agents have valuations with *superadditive width* at most $D > 0$ (see Definition 1). As a corollary, we obtain an $O(D\log m/\log\log m)$-approximation algorithm for the offline combinatorial welfare maximization problem with the same class of valuations. Here, D roughly measures the maximum

[1] In some scenarios, incentive incompatibility is not an issue – one such example is a government agency allocating resources among projects arriving online.

number of items that may complement each other, and generalizes the notion of subadditivity. In particular, valuations with superadditive width 0 are precisely valuations that are subadditive. The implication of the above bound is twofold: concretely, to our knowledge, this is the first nontrivial prophet inequality for valuations that are *approximately* subadditive; conceptually, the existence of parameters leading to this bound demonstrates the capacity of the parametrized framework we propose. We remark that while Feldman et al. [16] and Dütting et al. [10] present results of a similar flavor, their bounds are based on the *Maximum-over-Positive-Hypergraph (MPH)* hierarchy [15], which does not directly model approximate subadditivity, and thus is incompatible with the goal of this paper.

1.2 Technical Overview

We present a parametrized protocol, which works by rounding online a standard LP relaxation of the welfare maximization problem. The protocol is inspired by the two-stage offline rounding procedure by Dobzinski et al. [9], which works roughly as follows:

1. Compute a distribution over sets of items for each agent. These distributions together satisfy: (1) each item in expectation goes to at most 1 agent, and (2) the total expected value enjoyed by all agents is maximized.
2. For each agent i, draw i's tentative set of items according to i's distribution. Note that after this step, an item can appear in multiple agents' tentative sets.
3. Break ties for items, by independently allocating each item j to one of the agents whose tentative sets contain j, uniformly at random.

Dobzinski et al. [9] show that the above procedure outputs an $O(\log m)$-approximation for the combinatorial welfare maximization problem when agents are subadditive, and Feige [14] further proves a tighter bound of $O(\log m/\log\log m)$.

Intuitively, we wish to carry out Dobzinski's rounding procedure online. One main difficulty, however, is to deal with incomplete information, since when handling one agent, the protocol does not know the actual valuation of any agent yet to arrive. As a result, it is impossible to round online the solution to the LP with respect to all agents' actual valuations. To this end, we create a partially fictitious LP for each agent, by chaining this agent's actual valuation (which becomes available to the protocol upon the agent's arrival) with independently drawn dummy valuations for all other agents. By doing this, each agent is intuitively playing against the average case configuration of all other agents. Moreover, in each agent's fictitious LP, her share of the fractional allocation gives her precisely the expected value she would get in the actual optimal fractional allocation. So if we can round the fictitious LP solutions with mild loss, the resulting welfare will in fact be approximately optimal.

To achieve this, we need to deal with another difficulty, i.e., instead of breaking ties for each item simultaneously with all the tentative sets available, we now have to irrevocably decide which items are being allocated to each agent immediately upon his arrival, before seeing the valuations of all agents yet to arrive. And still, we need to make sure no item is allocated twice. This rules out the possibility of the independent uniform tie-breaking performed in Step 3 of Dobzinski et al.'s rounding procedure, which is crucial in the proofs for the approximation ratios in both papers.[2] To overcome this issue, we break ties in a correlated way, which pairs gracefully with (approximate) subadditivity. Roughly speaking,

[2] With some additional tricks one can simulate online the uniformity and independence conditions required in those papers, but even then the online version of Feige's argument works only for exactly subadditive valuations, and fails for approximately subadditive ones.

we first draw a (not necessarily uniformly) random integer r from $\{1, \ldots, n\}$, where n is the number of agents. We then partition each agent's tentative set into n subsets according to the number of times each item has appeared in some agent's tentative set so far, and give the agent all items that have appeared exactly r times. The intuition is that, if the valuations are subadditive, we then know that the sum of the values of these subsets is at least the value of the tentative set itself, which gives us a way to relate the expected value of the allocated subset to the value of the tentative set. Also, no item is possibly allocated twice, since each item can appear the r-th time only once throughout the procedure. Given the above, by choosing the distribution of r and other parameters in different ways, the parametrized protocol yields the desired bounds for subadditive, approximately subadditive, and fractionally subadditive valuations respectively.

1.3 Additional Related Work

Kleinberg and Weinberg [20] study the setting where possible combinations of boxes that can be accepted satisfy a matroid constraint. Dütting and Kleinberg [12] consider polymatroids, generalizing the above setting. Rubinstein and Singla [25] further consider subadditive reward functions. Their setting, while also being combinatorial, is different from combinatorial welfare maximization considered in this paper.

Another line of research consider revenue maximization in the prophet inequality context. We list a few results here. Blumrosen and Holenstein [3] give a constant factor protocol in the single-item setting. When agents are unit-demand – that is, they only want a single item – Chawla et al. [5] give constant factor posted-price policies. Cai and Zhao [4] study truthful policies for combinatorial auctions with subadditive agents. Their goal, however, is to maximize the revenue of the protocol, instead of the welfare as we consider.

Prior to Feldman et al., Chawla et al. [6] and Alaei [1] consider welfare maximization with unit-demand agents. Cohen-Addad et al. [8] further show that dynamic pricing achieves optimal welfare for unit-demand agents. Ehsani et al. [13] show that the ratio improves to $e/(e-1)$ for combinatorial welfare maximization with submodular or fractionally subadditive agents, if agents arrive in a random order.

2 Preliminaries

Throughout the paper, we use n to denote the number of agents, and m the number of items. In general, we use i as the index of an agent, and j as the index of an item.

2.1 Combinatorial Valuations

A combinatorial valuation function $f : 2^{[m]} \to \mathbb{R}^+$ maps any subset S of the ground set $[m] = \{1, 2, \ldots m\}$ to a nonnegative real number $f(S)$. In this paper, we consider valuation functions that are *monotone*: f is monotone iff for any $S \subseteq T \subseteq [m]$, $f(S) \leq f(T)$. The following subclasses of valuation functions are considered or helpful for our purposes:

- *subadditive*: f is subadditive iff for any $S, T \subseteq [m]$, $f(S) + f(T) \geq f(S \cup T)$.
- *additive* valuations: f is additive iff for any disjoint $S, T \subseteq [m]$, $f(S) + f(T) = f(S \cup T)$.
- *submodular* valuations: f is submodular iff for any $S, T \subseteq [m]$, $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$.
- *fractionally subadditive* (or *XOS*) valuations: f is fractionally subadditive iff there exist additive valuations $c^1, \ldots, c^\ell$, such that for any $S \subseteq [m]$, $f(S) = \max_{k \in [\ell]} c^k(S)$. Each such additive valuation c^k is called a *clause*.

It is known that every additive valuation is submodular, every submodular valuation is fractionally subadditive, and every fractionally subadditive function is subadditive. Beyond subadditive valuations, we also consider valuations of limited superadditivity, parametrized by the superadditive width, defined below.

▶ **Definition 1** (Superadditive Width [7]). *Fix a ground set $M = [m]$. A set $T \subseteq M$ is* superadditive *w.r.t. a valuation function f, if there exists $S \subseteq M$, such that*

$$f(S \mid T) > \max_{T' \subsetneq T} f(S \mid T'),$$

where $f(A \mid B) := f(A \cup B) - f(B)$ is the marginal value of A given B for any two sets $A, B \subseteq M$. The superadditive width of f is defined to be the size of the largest superadditive set, i.e.,

$$\mathrm{SAW}(f) := \max\{|T| \mid T \text{ is superadditive w.r.t. } f\}.$$

In words, the definition says that the superadditive width of a valuation is the size of the largest set, which provides strictly more marginal value for another set, than any of its strict subsets. Intuitively, the smaller this quantity is, the closer a valuation is to being subadditive. In particular, any subadditive valuation has superadditive width 0.

2.2 Problem Formulation

We formulate the problem in the following way: There are n agents and m items, and a prior $\mathcal{F} = \mathcal{F}_1 \times \cdots \times \mathcal{F}_n$ for the valuations of the agents over the items. All agents are monotone and subadditive (resp., fractionally subadditive), i.e., for any agent i, any valuation function f_i in the support of $\mathcal{F}_i$ is monotone and subadditive (resp., fractionally subadditive). Agents arrive one by one in an adversarial order. When agent i arrives, we see the realization $f_i \sim \mathcal{F}_i$ of her valuation (through query oracles discussed below), and must allocate irrevocably some items to the agent. The agent then takes the items and departs, and all items allocated to the agent become unavailable to subsequently arriving agents.

The goal is to maximize the expected (over the realization of the valuations and the randomness of the protocol) welfare of all agents. In particular, we wish to compete against the offline optimal allocation (i.e., the prophet), which has unlimited computational power, and knows beforehand the realization of all agents' valuations $\{f_i\}_i$. The competitive ratio is defined to be the ratio between the expected welfare (denoted OPT) of the offline optimal allocation[3] and the expected welfare produced by the online protocol.

2.3 Oracle Access to Valuation Functions

The representation of a combinatorial valuation function may be exponentially large in the number of items m. Given this complexity, it is standard to assume that the protocol may access valuation functions only through query oracles. In particular, the following two types of queries are commonly allowed (see, e.g., [14, 16, 10]):

- *value* queries: given a valuation function f and a set S, return the value of S, $f(S)$.
- *demand* queries: given a valuation function f and prices $\{p_j\}_{j\in[m]}$, return a utility-maximizing set (i.e., a demand set) with respect to f under the given prices. That is, the query returns a set S that maximizes $f(S) - \sum_{j\in S} p_j$.

In this paper, we assume the protocol has access to both kinds of queries.

[3] Note that the offline optimal allocation may be hard to find, computationally and / or information theoretically.

2.4 The Welfare Maximizing LP

The following LP relaxation of the welfare maximization problem has been considered in [9, 24, 14]:

$$\begin{array}{rll} \text{maximize} & \sum_{i\in[n]}\sum_{S\subseteq[m]} x_{i,S} f_i(S) & \\ \text{s.t.} & \sum_{S\subseteq[m]} x_{i,S} \le 1 & \forall i \in [n] \\ & \sum_{i\in[n],S\subseteq[m]:j\in S} x_{i,S} \le 1 & \forall j \in [m] \\ & x_{i,S} \ge 0 & \forall i \in [n], S \subseteq [m]. \end{array}$$

One may interpret the LP in the following way: $x_{i,S}$ stands for the probability that agent i receives bundle S. The objective is therefore the total expected value of all agents, which is the expected welfare. The first constraint requires that each agent i receives at most 1 bundle, and the second requires that each item goes to at most 1 agent, in expectation.

It is known (see, e.g., [9, 24]) that the above LP can be solved with polynomially many value and demand queries to $f_1, \ldots, f_n$. Let $x_{i,S}(\{f_{i'}\}_{i'})$ (parameters omitted when clear from the context) denote the value of variable $x_{i,S}$ in the optimal solution to the LP with respect to valuation functions $\{f_{i'}\}_{i'}$.[4] Given $\mathcal{F}$, denote the expected value enjoyed by agent i in this optimal solution by

$$\mathsf{LP}_i := \mathbb{E}_{\{f_{i'}\}_{i'}\sim\mathcal{F}}\left[\sum_{S\subseteq[m]} x_{i,S}(\{f_{i'}\}_{i'}) \cdot f_i(S)\right],$$

and the optimal objective value by

$$\mathsf{LP} := \sum_{i\in[n]} \mathsf{LP}_i = \mathbb{E}_{\{f_{i'}\}_{i'}\sim\mathcal{F}}\left[\sum_{i\in[n],S\subseteq[m]} x_{i,S}(\{f_{i'}\}_{i'}) \cdot f_i(S)\right].$$

Note that for any prior $\mathcal{F}$, we always have $\mathsf{LP} \ge \mathsf{OPT}$. We will use the welfare maximizing LP and its solution as a building block of our framework in a blackbox manner.

3 The Framework

In this section, we present our general framework, in the form of a parametrized protocol, for designing prophet inequalities for combinatorial welfare maximization. The framework works for any arrival order of agents, but for ease of presentation, we assume agents arrive according to their indices. That is, agent 1 arrives first, followed by agent 2, etc. We also let $f_{-i} := (f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_n)$ (i.e., f_{-i} denotes the valuations of all agents but i) and use g_{-i} and $\mathcal{F}_{-i}$ similarly. Below is our parametrized protocol:

1. For each item j, initialize counter $c_j \leftarrow 0$.
2. Draw positive integer $r \in [n]$, where $\Pr[r = k] = p_k$ for any $k \in [n]$, and $\{p_k\}_{k\in[n]}$ are parameters of the protocol.
3. Upon agent i's arrival:
   a. Let f_i be agent i's realized valuation; draw dummy valuations $g_{-i} \sim \mathcal{F}_{-i}$ for all other agents.[5]

[4] If there are multiple such solutions, let $\{x_{i,S}(\{f_{i'}\}_{i'})\}_{i,S}$ be the one produced by the efficient LP solving algorithm which we use as a subroutine of the protocol.

[5] Note that we abuse notation here, so for any $i_1 \neq i_2$, g_{-i_1} and g_{-i_2} are independent – they are not different parts of a same group of valuations.

b. Solve the welfare maximizing LP with valuations (f_i, g_{-i}), and let $\{x_{i',S}(f_i, g_{-i})\}_{i',S}$ be the solution.

c. Now view $\{x_{i,S}\}_S$ as a distribution over sets of items.[6] Draw set $S_i \sim \{x_{i,S}\}_S$ from this distribution, where for any S, $\Pr[S_i = S] = x_{i,S}$. We say agent i *demands* set S_i.

d. For each item $j \in S_i$, let $c_j \leftarrow c_j + 1$ (i.e., increase the counter for item j). For any $k \in [n]$, let S_i^k be the set of items in S_i whose counters are exactly k right after demanded by agent i.

e. With probability q, where q is a parameter of the protocol, *serve* agent i by giving i all items in S_i^r (i.e., item j goes to agent i iff agent i is chronologically the r-th agent demanding item j and being served); otherwise, for each item $j \in S_i$, let $c_j \leftarrow c_j - 1$ (i.e., undo the increase for all items demanded by i).

For any agent i, let $\text{ser}_i = \mathbb{I}[i \text{ is served}]$ be the indicator variable that i is served in Step 3(e). Note that the above protocol never allocates an item to more than one agent, since once r is fixed, at most one agent can be the r-th demanding an item. As a result, the protocol always produces a valid allocation.

We now present a meta-analysis for the above protocol, which yields a parameter-dependent welfare guarantee. In later sections, we will show how one can instantiate the protocol by setting different parameters, so that the general guarantee realizes into specific bounds for the respective classes of valuations of interest.

▶ **Theorem 2.** *Let u_i be the value enjoyed by agent i. For each agent i, fixing f_i and S_i, the above protocol guarantees agent i expected value*

$$\mathbb{E}_{R_i}[u_i \mid f_i, S_i] = q \cdot \sum_{k \in [n]} p_k \cdot \mathbb{E}_{R_i}[f_i(S_i^k)],$$

where R_i summarizes all the randomness other than f_i, g_{-i} and S_i, both from the protocol and from the realization of the valuations.

Proof. The theorem is essentially a claim regarding independence of random variables. Recall that for any agent i', $\text{ser}_{i'} = \mathbb{I}[i' \text{ is served}]$. Fixing f_i and S_i, i's expected value can be written as

$$\mathbb{E}_{R_i}[u_i \mid f_i, S_i] = \mathbb{E}_{R_i}\left[\text{ser}_i \cdot \sum_{k \in [n]} \mathbb{I}[r = k] \cdot f_i(S_i^k) \,\middle|\, f_i, S_i\right].$$

R_i here summarizes $\{(f_{i'}, g_{-i'}, S_{i'})\}_{i' \neq i}$, the choice of r, and whether i' is served for all i' (including $i' = i$). To simplify this, first observe that the conditioning can be removed, because the randomness involved in the expectation, R_i, is independent of f_i and S_i. In particular, $\{f_{i'}\}_{i' \neq i}$ are independent of f_i because $\mathcal{F}$ is a product distribution. Moreover, the three factors within the expectation are independent, because whether i is served depends only on the coin flipping in Step 3(e) when agent i arrives, the choice of r depends only on the random draw in Step 2, and fixing S_i, $f_i(S_i^k)$ depends only on the realization of $\{f_{i'}\}_{i' < i}$ and the random bits of the protocol dealing with the agents arriving before i. Given the above, we have

[6] It is possible that $\sum_S x_{i,S} < 1$, in which case with probability $1 - \sum_S x_{i,S}$, $S_i = \emptyset$.

$$\begin{aligned}\mathbb{E}_{R_i}[u_i \mid f_i, S_i] &= \mathbb{E}_{R_i}\left[\mathrm{ser}_i \cdot \sum_{k\in[n]} \mathbb{I}[r=k]\cdot f_i(S_i^k)\right] \\ &= \Pr[\mathrm{ser}_i = 1]\cdot \sum_{k\in[n]} \Pr[r=k]\cdot \mathbb{E}_{\{(f_{i'},g_{-i'},S_{i'},\mathrm{ser}_{i'})\}_{i'<i}}[f_i(S_i^k)] \\ &= q\cdot \sum_{k\in[n]} p_k \cdot \mathbb{E}_{R_i}[f_i(S_i^k)]. \end{aligned}$$

◀

Before proceeding to the specific instantiations of the protocol, we note the following high-level interpretation of the above parameter-dependent bound, which provides important intuition and leads to our choices of parameters for different classes of valuations. For concreteness, suppose agents are subadditive. First observe that the share of agent i in the optimal solution of the welfare maximizing LP is

$$\begin{aligned}\mathsf{LP}_i &= \mathbb{E}_{\{f_{i'}\}_{i'}\sim\mathcal{F}}\left[\sum_S x_{i,S}(\{f_{i'}\}_{i'})\cdot f_i(S)\right] \\ &= \mathbb{E}_{(f_i,g_{-i})\sim\mathcal{F},S_i\sim\{x_{i,S}(f_i,g_{-i})\}_S}[f_i(S_i)] = \mathbb{E}_{f_i,S_i}[f_i(S_i)].\end{aligned}$$

On the other hand, since $\{S_i^k\}_{k\in[n]}$ is a partition of S_i, by the subadditivity of f_i, we always have

$$\sum_{k\in[n]} \mathbb{E}_{R_i}[f_i(S_i^k)] \geq \mathbb{E}_{R_i}[f_i(S_i)] = f_i(S_i).$$

So hypothetically, if somehow we were able to set $q = 1$ and $p_k = 1$ for every $k \in [n]$ simultaneously (which is of course impossible), then fixing any choice of f_i and S_i, the above protocol would yield at least $f_i(S_i)$ as the expected value for agent i. Further taking expectation over f_i and S_i, this would imply a 1-approximate prophet inequality against the welfare maximizing LP.

In reality, however, one cannot set p_k to be large for all k simultaneously. In fact, these probabilities must sum to 1. Still, our goal is to guarantee a decent fraction of $f_i(S_i)$ for any choice of f_i and S_i. This is possible when, for example, the total value of S_i, $f_i(S_i)$, concentrates in relatively few entries among its n parts, $\{\mathbb{E}_{R_i}[f_i(S_i^k)]\}_{k\in[n]}$. In such cases, we can let r be uniformly distributed over the indices of these entries, leading to a competitive ratio proportional to the number of such heavy entries. The key step here is analyzing the distribution of $f_i(S_i)$ into $\{f_i(S_i^k)\}_{k\in[n]}$, over the randomness in R_i. We will further develop the above intuition in later sections.

4 Warmup: the Case of Fractionally Subadditive Agents

We start with the relatively simple case of fractionally subadditive agents, the analysis for which provides tools for the more involved cases to be handled in later sections. Throughout this section, we consider the following choice of parameters: $p_1 = 1$, $p_k = 0$ for any $1 < k \leq n$, and $q = 1/2$. That is, each agent is served with probability $1/2$, and whenever an agent is served, he receives all the items demanded which are not yet taken. We now analyze our protocol with the above parameters for fractionally subadditive agents.

▶ **Theorem 3.** *The protocol in Section 3 with $p_1 = 1$, $p_k = 0$ for any $1 < k \leq n$, and $q = 1/2$ is 4-competitive when agents are fractionally subadditive.*

Proof. In light of Theorem 2, we only need to show that for any agent i, fixing f_i and S_i,

$$\mathbb{E}_{R_i}[f_i(S_i^1)] \geq \frac{1}{2} f_i(S_i). \tag{1}$$

The theorem then follows by plugging the above into Theorem 2 and taking expectation over f_i and S_i, which yields

$$\mathbb{E}_{f_i,S_i}[\mathbb{E}_{R_i}[u_i \mid f_i, S_i]] = \mathbb{E}_{f_i,S_i}\left[\frac{1}{2}\mathbb{E}_{R_i}[f_i(S_i^1)]\right] \geq \frac{1}{4}\mathbb{E}_{f_i,S_i}[f_i(S_i)] = \frac{1}{4}\mathsf{LP}_i.$$

Summing over the agents, we get the following lower bound on the expected welfare:

$$\mathbb{E}\left[\sum_{i\in[n]} u_i\right] \geq \frac{1}{4}\sum_{i\in[n]} \mathsf{LP}_i = \frac{1}{4}\mathsf{LP} \geq \frac{1}{4}\mathsf{OPT},$$

where the expectation is over all the randomness, both from the protocol and from the realization of agents' valuations. This gives precisely the desired competitive ratio of 4. The rest of the proof is dedicated to establishing (1) when i, f_i and S_i are fixed.

Recall the following property of fractionally subadditive functions.

▶ **Lemma 4.** *For fractionally subadditive f and any set of items S, let T be such that for any $j \in S$, $j \in T$ with probability at least p. Then*

$$\mathbb{E}[f(T)] \geq p \cdot f(S).$$

While this is sometimes considered standard, we give below a quick proof for completeness.

Proof. Let c^k be the clause of f such that

$$f(S) = c^k(S) = \sum_{j\in S} c^k(\{j\}).$$

By the fractional subadditivity of f, for any $S' \subseteq S$,

$$f(S') \geq c^k(S').$$

This is in particular true for T, which implies

$$\begin{aligned}
\mathbb{E}[f(T)] &\geq \mathbb{E}[c^k(T)] \\
&= \sum_{j\in S} \mathbb{E}[\mathbb{I}[j \in T] \cdot c^k(\{j\})] \\
&= \sum_{j\in S} \Pr[j \in T] \cdot c^k(\{j\}) \\
&\geq p \cdot \sum_{j\in S} c^k(\{j\}) \\
&= p \cdot f(S).
\end{aligned}$$

◀

Given Lemma 4, the plan is to show that each item j in S_i appears in S_i^1 with probability at least $1/2$. That is, with probability at least $1/2$, no agent $i' < i$ demands item j and gets served simultaneously. To see why this is true, consider the unconditional distribution of the set $S_{i'}$ demanded by any agent i'. Let $y_{i',S} = \Pr[S_{i'} = S]$ be the probability that agent i' demands set S, over $f_{i'}$, $g_{-i'}$, and the random bits of the protocol. Note that $y_{i',S}$ is not random, and depends only on the prior $\mathcal{F}$. We first show that $\{y_{i',S}\}_{i',S}$ form a feasible solution to the welfare maximizing LP, regardless of the actual valuations of the agents. This is well-defined, since fixing n and m, the precise values of sets do not appear in any constraint of the welfare maximizing LP.

▶ **Lemma 5.** *$\{y_{i',S}\}_{i',S}$ satisfy:*

- *for any agent $i' \in [n]$, $\sum_{S\subseteq[m]} y_{i',S} \le 1$,*
- *for any item $j \in [m]$, $\sum_{i'\in[n],S\subseteq[m]:j\in S} y_{i',S} \le 1$, and*
- *for any $i' \in [n]$, $S \subseteq [m]$, $y_{i',S} \ge 0$.*

Proof. Observe that according to the protocol, for any i', S,

$$y_{i',S} = \mathbb{E}_{f_{i'}\sim\mathcal{F}_{i'},g_{-i'}\sim\mathcal{F}_{-i'}}[x_{i',S}(f_{i'},g_{-i'})] = \mathbb{E}_{\{f_{i''}\}_{i''}\sim\mathcal{F}}[x_{i,S}(\{f_{i''}\}_{i''})].$$

In other words, $y_{i',S}$ is the expected value of the variable $x_{i',S}$ in the optimal solution to the welfare maximizing LP, when valuations are distributed according to prior $\mathcal{F}$. Now since for any realization of $\{f_{i''}\}_{i''}$, $\{x_{i',S}(\{f_{i''}\}_{i''})\}_{i',S}$ satisfy the LP constraints, it follows from linearity of expectation that the expected values $\{y_{i',S}\}_{i',S}$ also satisfy the constraints. The lemma follows. ◀

For any agent i' and item j, let $d^j_{i'}$ be the probability that item j is demanded by agent i'. That is,

$$d^j_{i'} := \Pr_{f_{i'},g_{-i'}}[j \in S_{i'}] = \sum_{S:j\in S} y_{i',S}.$$

The feasibility of $y_{i',S}$ (Lemma 5) implies: for any $j \in [m]$,

$$\sum_{i'\in[n]} d^j_{i'} = \sum_{i'\in[n],S:j\in S} y_{i',S} \le 1.$$

Also, whether agent i' demands j is independent of whether i' is served, so for any $j \in [m]$,

$$\begin{aligned}
\Pr[\exists i' < i : (j \in S_{i'} \wedge i' \text{ is served})] &\le \sum_{i'<i} \Pr[j \in S_{i'} \wedge i' \text{ is served}] && \text{(union bound)}\\
&\le \sum_{i'\in[n]} \Pr[j \in S_{i'} \wedge i' \text{ is served}]\\
&= \sum_{i'\in[n]} d^j_{i'} \cdot \Pr[\mathrm{ser}_{i'} = 1]\\
& && \text{(independence of } S_{i'} \text{ and } \mathrm{ser}_{i'})\\
&\le 1 \cdot q = \frac{1}{2}, && \text{(Lemma 5)}
\end{aligned}$$

which concludes the proof of the theorem. ◀

5 The Case of Subadditive Agents

Equipped with tools developed in previous sections, now we proceed to the case of subadditive agents. Here we choose $q = 1$ and

$$p_k = \begin{cases} 1/C, & 1 \le k \le C \\ 0, & C < k \le n, \end{cases}$$

where $C = \min(100 \log m/\log\log m, n) = O(\log m/\log\log m)$. We prove the following bound for subadditive agents.

▶ **Theorem 6.** *The protocol in Section 3 with the above parameters is $O(\log m/\log\log m)$-competitive when agents are subadditive.*

Proof. When $n \le 100 \log m / \log\log m$, the theorem is easy to prove. Below we focus on the case where $C = 100 \log m / \log\log m < n$. Consider any agent i. Fix f_i and S_i. In light of Theorem 2, our goal here is to show

$$\sum_{1 \le k \le C} \mathbb{E}_{R_i}[f_i(S_i^k)] = \Omega(f_i(S_i)). \tag{2}$$

That is, the first C terms in $\{f_i(S_i^k)\}_{k \in [n]}$ contribute a constant fraction of $f_i(S_i)$ in expectation. The theorem then again follows by plugging the above into Theorem 2, yielding

$$\mathbb{E}_{R_i}[u_i \mid f_i, S_i] \ge q \cdot \sum_{1 \le k \le C} p_k \cdot \mathbb{E}_{R_i}[f_i(S_i^k)] = \Omega(f_i(S_i)/C).$$

Taking expectation over f_i and S_i, and summing over the agents, we have

$$\sum_i \mathbb{E}[u_i] \ge \Omega(1/C) \cdot \sum_i \mathbb{E}[f_i(S_i)] = \Omega(1/C) \cdot \mathsf{LP} \ge \Omega(1/C) \cdot \mathsf{OPT}.$$

A competitive ratio of $O(C) = O(\log m / \log\log m)$ follows. The rest of the proof is dedicated to establishing (2) when i, f_i and S_i are fixed.

The plan is to show, with constant probability over R_i, $S_i^k = \emptyset$ for all $k > C$. Denote this event by $\mathcal{E}_i$. Whenever this happens, $\bigcup_{1 \le k \le C} S_i^k = S_i$, and by the subadditivity of f_i,

$$\sum_{1 \le k \le C} f_i(S_i^k) \ge f_i(S_i).$$

As a result,

$$\sum_{1 \le k \le C} \mathbb{E}_{R_i}[f_i(S_i^k)] \ge \sum_{1 \le k \le C} \mathbb{E}_{R_i}[f_i(S_i^k) \mid \mathcal{E}_i] \cdot \Pr[\mathcal{E}_i] \ge \Pr[\mathcal{E}_i] \cdot f_i(S_i).$$

We show below $\Pr[\mathcal{E}_i] = \Omega(1)$.

Fix an item j, and consider the probability that $j \in S_i^k$ for some $k > C$, or equivalently,

$$\Pr\left[\sum_{i' < i} \mathbb{I}[j \in S_{i'}] \ge C\right].$$

To bound the above probability, recall the following fact about independent Bernoulli variables (see, e.g., [14, 19]).

▶ **Lemma 7.** *For any $n \in \mathbb{Z}^+$, independent Bernoulli random variables $X_1, \ldots, X_n$ where $\mathbb{E}\left[\sum_{i \in [n]} X_i\right] \le 1$, and any $k \in \mathbb{Z}^+$,*

$$\Pr\left[\sum_i X_i \ge k\right] = O\left(\frac{1}{k!}\right) = k^{-\Omega(k)}.$$

The lemma says, that if the sum of independent Bernoulli variables in expectation does not exceed 1, then the tail of this sum decays factorially fast. Now observe that $S_1, \ldots, S_{i-1}$, and therefore $\mathbb{I}[j \in S_1], \ldots, \mathbb{I}[j \in S_{i-1}]$ are independent. This is because for any i', $S_{i'}$ depends only on $f_{i'}$ and $g_{-i'}$. Also, since

$$\mathbb{E}\left[\sum_{i' < i} \mathbb{I}[j \in S_{i'}]\right] = \sum_{i' < i} \Pr[j \in S_{i'}] = \sum_{i' < i} d_j^{i'} \le \sum_{i' \in [n]} d_j^{i'} \le 1,$$

random variables $\{\mathbb{I}[j \in S_{i'}]\}_{i'<i}$ satisfy the conditions of Lemma 7. So for large enough m, we can bound the probability that $j \in S_i^k$ for some $k > C$ using Lemma 7. Recall that $C = 100 \log m / \log\log m$. Plugging this in, Lemma 7 then gives

$$\Pr\left[\sum_{i'<i} \mathbb{I}[j \in S_{i'}] \geq C\right] \leq C^{-\Omega(C)} = O\left(\frac{1}{m^2}\right).$$

Now taking a union bound over all items, we get

$$\Pr[\mathcal{E}_i] \geq 1 - \sum_{j\in[m]} \Pr\left[\sum_{i'<i} \mathbb{I}[j \in S_{i'}] \geq C\right] = 1 - \sum_{j\in[m]} O\left(\frac{1}{m^2}\right) = 1 - O\left(\frac{1}{m}\right) = \Omega(1).$$

This concludes the proof of the theorem. ◀

6 Generalizing to Approximately Subadditive Agents

In this section, we consider valuations with superadditive width at most $D > 0$. In order to utilize the boundedness of the superadditive width, we set $q = \frac{1}{4D}$, and

$$p_k = \begin{cases} \frac{1}{2} + \frac{1}{2C}, & k = 1 \\ \frac{1}{2C}, & 1 < k \leq C \\ 0, & C < k \leq n, \end{cases}$$

where again $C = \min(100 \log m / \log\log m, n) = O(\log m / \log\log m)$. One may check that $\{p_k\}_{k\in[n]}$ in fact sum to 1. We prove the following competitive ratio.

▶ **Theorem 8.** *The protocol in Section 3 with the above parameters is $O(D \log m / \log\log m)$-competitive when all valuations have superadditive width at most D.*

Proof. Again, fix an agent i, f_i and S_i. Our goal is to show that one of the following two claims is always true:

- Claim (i):

$$\mathbb{E}_{R_i}[f_i(S_i^1)] = \Omega\left(f_i(S_i)/C\right).$$

- Claim (ii):

$$\sum_{1\leq k\leq C} \mathbb{E}_{R_i}[f_i(S_i^k)] = \Omega(f_i(S_i)).$$

We first show how the above condition implies the theorem. Again, by applying Theorem 2,

$$\begin{aligned} \mathbb{E}_{R_i}[u_i \mid f_i, S_i] &\geq q \cdot \sum_{1\leq k\leq C} p_k \cdot \mathbb{E}_{R_i}[f_i(S_i^k)] \\ &= \Omega(1/D) \cdot \left(\mathbb{E}_{R_i}[f_i(S_i^k)] + \frac{1}{C} \sum_{1\leq k\leq C} \mathbb{E}_{R_i}[f_i(S_i^k)]\right) \\ &= \Omega(1/D) \cdot \Omega(f_i(S_i)/C) \\ &= \Omega\left(\frac{D \log m}{\log\log m} \cdot f_i(S_i)\right). \end{aligned}$$

The theorem then follows by taking expectation over f_i and S_i, and summing over the agents.

The rest of the proof is dedicated to establishing the above condition. We consider the most valuable $2D$ items for agent i in S_i, i.e.,

$$S_i^* := \text{argmax}_{S:S\subseteq S_i,|S|\leq 2D} f_i(S).$$

We show below that when the value of S_i^*, $f_i(S_i^*)$, is small, then Claim (i) holds. Otherwise, Claim (ii) holds. Since f_i and S_i are fixed, one of the two claims is always true.

First suppose

$$f_i(S_i^*) \geq f_i(S_i)/(2C).$$

In such cases, we show that with constant probability, $S_i^* \subseteq S_i^1$. Whenever this happens, monotonicity implies

$$f_i(S_i^1) \geq f_i(S_i^*) \geq f_i(S_i)/(2C).$$

Taking expectation over R_i, this implies Claim (i).

We now bound the probability that $S_i^* \subseteq S_i^1$. Fix $j \in S_i^*$. Consider the probability that $j \notin S_i^1$, or equivalently,

$$\Pr\left[\sum_{i'<i} \mathbb{I}[j \in S_{i'} \wedge i' \text{ is served}] \geq 1\right].$$

We upper bound this probability in the following way.

$$\begin{aligned}
\Pr\left[\sum_{i'<i} \mathbb{I}[j \in S_{i'} \wedge i' \text{ is served}] \geq 1\right] &\leq \sum_{i'<i} \Pr[j \in S_{i'} \wedge i' \text{ is served}] && \text{(union bound)}\\
&= \sum_{i'<i} d_{i'}^j \cdot q \leq \sum_{i'\in[n]} d_{i'}^j \cdot q\\
&\leq q = \frac{1}{4D}. && \text{(Lemma 5)}
\end{aligned}$$

So for any $j \in S_i^*$,

$$\Pr[j \notin S_i^1] \leq \frac{1}{4D}.$$

Since $|S_i^*| = 2d$, taking a union bound over all items in S_i^*, we get

$$\Pr[S_i^* \subseteq S_i^1] \geq 1 - \sum_{j\in S_i^*} \Pr[j \notin S_i^1] \geq 1 - 2D \cdot \frac{1}{4D} = \frac{1}{2}.$$

This implies Claim (i) as argued above.

Now suppose

$$f_i(S_i^*) < f_i(S_i)/(2C).$$

Given this, we show Claim (ii) holds. First we need the following property of valuations with bounded superadditive width.

▶ **Lemma 9** ([7]). *Let $f : 2^{[m]} \to \mathbb{R}_+$ be a valuation function such that $\text{SAW}(f) \leq D$. For any $S, T \subseteq [m]$,*

$$f(S \mid T) \leq \min_{T':T'\subseteq T,|T'|\leq D} f(S \mid T').$$

Consider the prefix unions of $\{S_i^k\}_{k\in[C]}$. To be precise, let $U_i^0 = \emptyset$, and for any $k \in [C]$,

$$U_i^k = U_i^{k-1} \cup S_i^k.$$

Recall that for any $S, T \subseteq [m]$, $f_i(S \mid T) = f_i(S \cup T) - f_i(T)$ is the marginal value of S given T. For any $k \in [C]$, we have

$$\begin{aligned}
f_i(U_i^k) &= f_i(S_i^k \mid U_i^{k-1}) + f_i(U_i^{k-1}) \\
&\le f_i(S_i^k \mid X) + f_i(U_i^{k-1}), \text{ where } X = \operatorname{argmin}_{S:S\subseteq U_i^{k-1}, |S|\le D} f_i(S_i^k \mid S) \\
&\qquad\qquad \text{(Lemma 9)} \\
&\le f_i(S_i^k \cup X) + f_i(U_i^{k-1}) \qquad \text{(monotonicity of } f_i\text{)} \\
&= f_i(X \mid S_i^k) + f_i(S_i^k) + f_i(U_i^{k-1}) \\
&\le f_i(X \mid Y) + f_i(S_i^k) + f_i(U_i^{k-1}), \text{ where } Y = \operatorname{argmin}_{S:S\subseteq S_i^k, |S|\le D} f_i(X \mid S) \\
&\qquad\qquad \text{(Lemma 9)} \\
&\le f_i(X \cup Y) + f_i(S_i^k) + f_i(U_i^{k-1}) \qquad \text{(monotonicity)} \\
&\le f_i(S_i^*) + f_i(S_i^k) + f_i(U_i^{k-1}). \quad \text{(definition of } S_i^* \text{ and } |X \cup Y| \le |X| + |Y| \le 2D\text{)}
\end{aligned}$$

Now recall that $\mathcal{E}_i$ is the event that for all $k > C$, $S_i^k = \emptyset$. In the proof of Theorem 6, we have shown that $\Pr[\mathcal{E}_i] = \Omega(1)$. Also, whenever $\mathcal{E}_i$ happens, we have $U_i^C = S_i$. As a result, when $\mathcal{E}_i$ happens, we can bound $f_i(S_i)$ in the following way.

$$\begin{aligned}
f_i(S_i) = f_i(U_i^C) &\le f_i(S_i^*) + f_i(S_i^C) + f_i(U_i^{C-1}) \\
&\le 2f_i(S_i^*) + f_i(S_i^C) + f_i(S_i^{C-1}) + f_i(U_i^{C-2}) \\
&\le \dots \\
&\le C \cdot f_i(S_i^*) + \sum_{1\le k\le C} f_i(S_i^k).
\end{aligned}$$

Compared to the subadditive case, here we have the additional term $C \cdot f_i(S_i^*)$. However, as we are in the world where $f_i(S_i^*)$ is small, this term does not affect the bound too much. Concretely, whenever $\mathcal{E}_i$ happens, we have

$$\sum_{1\le k\le C} f_i(S_i^k) \ge f_i(S_i) - C \cdot f_i(S_i^*) > f_i(S_i) - C \cdot \frac{f_i(S_i)}{2C} = \frac{f_i(S_i)}{2}.$$

Again, since $\Pr[\mathcal{E}_i] = \Omega(1)$, we have

$$\sum_{1\le k\le C} \mathbb{E}_{R_i}[f_i(S_i^k)] = \Omega(f_i(S_i)),$$

which is precisely Claim (ii), and therefore concludes the proof of the theorem. ◀

References

1. Saeed Alaei. Bayesian combinatorial auctions: Expanding single buyer mechanisms to many buyers. *SIAM Journal on Computing*, 43(2):930–972, 2014.
2. Kshipra Bhawalkar and Tim Roughgarden. Welfare guarantees for combinatorial auctions with item bidding. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 700–709. Society for Industrial and Applied Mathematics, 2011.
3. Liad Blumrosen and Thomas Holenstein. Posted prices vs. negotiations: an asymptotic analysis. In *EC*, page 49. Citeseer, 2008.

4 Yang Cai and Mingfei Zhao. Simple mechanisms for subadditive buyers via duality. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 170–183. ACM, 2017.

5 Shuchi Chawla, Jason D Hartline, and Robert Kleinberg. Algorithmic pricing via virtual valuations. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 243–251. ACM, 2007.

6 Shuchi Chawla, Jason D Hartline, David L Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 311–320. ACM, 2010.

7 Wei Chen, Shang-Hua Teng, and Hanrui Zhang. Capturing complementarity in set functions by going beyond submodularity/subadditivity. *10th Innovations in Theoretical Computer Science*, 2019.

8 Vincent Cohen-Addad, Alon Eden, Michal Feldman, and Amos Fiat. The invisible hand of dynamic market pricing. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 383–400. ACM, 2016.

9 Shahar Dobzinski, Noam Nisan, and Michael Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 610–618. ACM, 2005.

10 Paul Dütting, Michal Feldman, Thomas Kesselheim, and Brendan Lucier. Prophet inequalities made easy: Stochastic optimization by pricing non-stochastic inputs. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 540–551. IEEE, 2017.

11 Paul Dütting, Thomas Kesselheim, and Brendan Lucier. An o (log log m) prophet inequality for subadditive combinatorial auctions. *arXiv preprint*, 2020. `arXiv:2004.09784`.

12 Paul Dütting and Robert Kleinberg. Polymatroid prophet inequalities. In *Algorithms-ESA 2015*, pages 437–449. Springer, 2015.

13 Soheil Ehsani, MohammadTaghi Hajiaghayi, Thomas Kesselheim, and Sahil Singla. Prophet secretary for combinatorial auctions and matroids. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 700–714. SIAM, 2018.

14 Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM Journal on Computing*, 39(1):122–142, 2009.

15 Uriel Feige, Michal Feldman, Nicole Immorlica, Rani Izsak, Brendan Lucier, and Vasilis Syrgkanis. A unifying hierarchy of valuations with complements and substitutes. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

16 Michal Feldman, Nick Gravin, and Brendan Lucier. Combinatorial auctions via posted prices. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 123–135. Society for Industrial and Applied Mathematics, 2015.

17 Moran Feldman, Ola Svensson, and Rico Zenklusen. A simple $O(\log \log(\text{rank}))$-competitive algorithm for the matroid secretary problem. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1189–1201. SIAM, 2014.

18 Mohammad Taghi Hajiaghayi, Robert Kleinberg, and Tuomas Sandholm. Automated online mechanism design and prophet inequalities. In *AAAI*, volume 7, pages 58–65, 2007.

19 Svante Janson. Large deviation inequalities for sums of indicator variables. *arXiv preprint*, 2016. `arXiv:1609.00533`.

20 Robert Kleinberg and Seth Matthew Weinberg. Matroid prophet inequalities. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 123–136. ACM, 2012.

21 Ulrich Krengel and Louis Sucheston. Semiamarts and finite values. *Bulletin of the American Mathematical Society*, 83(4):745–747, 1977.

22 Ulrich Krengel and Louis Sucheston. On semiamarts, amarts, and processes with finite value. *Probability on Banach spaces*, 4:197–266, 1978.

23 Brendan Lucier. An economic view of prophet inequalities. *ACM SIGecom Exchanges*, 16(1):24–47, 2017.

24 Noam Nisan and Ilya Segal. The communication requirements of efficient allocations and supporting prices. *Journal of Economic Theory*, 129(1):192–224, 2006.

25 Aviad Rubinstein and Sahil Singla. Combinatorial prophet inequalities. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1671–1687. SIAM, 2017.

On the Approximation Ratio of the k-Opt and Lin-Kernighan Algorithm for Metric and Graph TSP

Xianghui Zhong
University of Bonn, Germany
zhong@cs.uni-bonn.de

Abstract

The k-Opt and Lin-Kernighan algorithm are two of the most important local search approaches for the Metric TSP. Both start with an arbitrary tour and make local improvements in each step to get a shorter tour. We show that for any fixed $k \geq 3$ the approximation ratio of the k-Opt algorithm for Metric TSP is $O(\sqrt[k]{n})$. Assuming the Erdős girth conjecture, we prove a matching lower bound of $\Omega(\sqrt[k]{n})$. Unconditionally, we obtain matching bounds for $k = 3, 4, 6$ and a lower bound of $\Omega(n^{\frac{2}{3k-3}})$. Our most general bounds depend on the values of a function from extremal graph theory and are tight up to a factor logarithmic in the number of vertices unconditionally. Moreover, all the upper bounds also apply to a parameterized version of the Lin-Kernighan algorithm with appropriate parameter. We also show that the approximation ratio of k-Opt for Graph TSP is $\Omega\left(\frac{\log(n)}{\log\log(n)}\right)$ and $O\left(\left(\frac{\log(n)}{\log\log(n)}\right)^{\log_2(9)+\epsilon}\right)$ for all $\epsilon > 0$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases traveling salesman problem, metric TSP, graph TSP, k-Opt algorithm, Lin-Kernighan algorithm, approximation algorithm, approximation ratio.

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.83

Related Version A full version of the paper is available at `https://arxiv.org/abs/1909.12755`.

Funding The author was supported by the Bonn International Graduate School.

Acknowledgements I want to thank Fabian Henneke, Stefan Hougardy, Yvonne Omlor, Heiko Röglin and Fabian Zaiser for reading this paper and making helpful remarks.

1 Introduction

The traveling salesman problem (TSP) is probably the best-known problem in discrete optimization. An instance consists of the pairwise distances of n vertices and the task is to find a shortest Hamiltonian cycle, i.e. a tour visiting every vertex exactly once. The problem is known to be NP-hard [12]. A special case of the TSP is the Metric TSP. Here the distances satisfy the triangle inequality. This TSP variant is still NP-hard [15].

Since the problem is NP-hard, a polynomial-time algorithm is not expected to exist. In order to speed up the calculation of a good tour in practice, several approximation algorithms are considered. The approximation ratio is one way to compare approximation algorithms. It is the maximal ratio, taken over all instances, of the output of the algorithm divided by the optimum solution. The best currently known approximation algorithm in terms of approximation ratio for Metric TSP was independently developed by Christofides and Serdjukov [6, 24] with an approximation ratio of $\frac{3}{2}$. However, in practice other algorithms are usually easier to implement and have better performance and runtime [3, 14, 22]. One natural approach is the k-Opt algorithm which is based on local search. It starts with an arbitrary tour and replaces at most k edges by new edges such that the resulting tour is

shorter. It stops if the procedure cannot be applied anymore. For the 2-Opt algorithm Plesník showed that there are infinitely many instances with approximation ratio $\sqrt{\frac{n}{8}}$, where n is the number of vertices [21]. Chandra, Karloff and Tovey showed that the approximation ratio of 2-Opt is at most $4\sqrt{n}$ [5]. Levin and Yovel observed that the same proof yields an upper bound of $\sqrt{8n}$ [19]. Recently, Hougardy, Zaiser and Zhong closed the gap and proved that the approximation ratio of the 2-Opt algorithm is at most $\sqrt{\frac{n}{2}}$ and that this bound is tight [13]. For general $k > 2$ Chandra, Karloff and Tovey gave a lower bound of $\frac{1}{4}\sqrt[2k]{n}$ [5], no non-trivial upper bound is known so far. In the case where the instances can be embedded into the normed space $\mathbb{R}^d$ the approximation ratio of 2-Opt is between $\Omega(\frac{\log(n)}{\log\log(n)})$ and $O(\log(n))$ [5].

Beyond the worst-case analysis there are also results about the average case behavior of the algorithm. For example the smoothed analysis of the 2-Opt algorithm by Englert, Röglin and Vöcking [7]. In their model each vertex of the TSP instance is a random variable distributed in the d dimensional unit cube by a given probability density function $f_i : [0,1]^d \to [0,\phi]$ bounded from above by a constant $1 \leq \phi < \infty$ and the distances are given by the L_p norm. They show that in this case the expected approximation ratio is bounded by $O(\sqrt[d]{\phi})$ for all p. In the model where any instance is given in $[0,1]^d$ and perturbed by a Gaussian noise with standard deviation σ the approximation ratio was improved to $O(\log(\frac{1}{\sigma}))$ by Künnemann and Manthey [17].

One of the best practical heuristics by Lin and Kernighan is based on k-Opt [20]. The Lin-Kernighan algorithm, like the k-Opt algorithm, modifies the tour locally to obtain a new tour. Instead of replacing arbitrary k edges with new edges, which results in a high runtime for large k, it searches for specific changes: Changes where the edges to be added and deleted are alternating in a closed walk, a so called closed alternating walk. Since the Lin-Kernighan algorithm uses a super set of the modification rules of the 2-Opt algorithm, the same upper bound as for 2-Opt also applies. Apart from this, no other upper bound was known.

A special case of the Metric TSP is the Graph TSP. In this case an undirected unweighted graph is given and the distance between two vertices is the distance between them in the graph. Apart from the upper bounds of the Metric TSP, which also apply to the special case, only a lower bound of $2(1-\frac{1}{n})$ on the approximation ratio of the k-Opt algorithm is known so far: Rosenkrantz, Stearns and Lewis describe a Metric TSP instance with this ratio that is also a Graph TSP instance [23].

New results. For fixed $k \geq 3$, we show that the approximation ratio of the k-Opt algorithm is related to the extremal graph theoretic problem of maximizing the number of edges in a graph with fixed number of vertices and no short cycles. Let $\mathrm{ex}(n, 2k)$ be the largest number of edges in a graph with n vertices and girth at least $2k$, i.e. it contains no cycles with less than $2k$ edges. For instances with n vertices we show for Metric TSP that:

▶ **Theorem 1.** *If* $\mathrm{ex}(n, 2k) \in O(n^c)$ *for some* $c > 1$*, the approximation ratio of* k*-Opt is* $O(n^{1-\frac{1}{c}})$ *for all fixed* k*.*

▶ **Theorem 2.** *If* $\mathrm{ex}(n, 2k) \in \Omega(n^c)$ *for some* $c > 1$*, the approximation ratio of* k*-Opt is* $\Omega(n^{1-\frac{1}{c}})$ *for all fixed* k*.*

Using known upper bounds on $\mathrm{ex}(n, 2k)$ in [1] we can conclude:

▶ **Corollary 3.** *The approximation ratio of* k*-Opt is in* $O(\sqrt[k]{n})$ *for all fixed* k*.*

If we further assume the Erdős girth conjecture [10], i.e. $\mathrm{ex}(n, 2k) \in \Theta(n^{1+\frac{1}{k-1}})$, we have:

▶ **Corollary 4.** *Assuming the Erdős girth conjecture, the approximation ratio of k-Opt is in $\Omega(\sqrt[k]{n})$ for all fixed k.*

Using known lower bounds on $\mathrm{ex}(n, 2k)$ from [8, 9, 4, 2, 25, 26, 18] we obtain:

▶ **Corollary 5.** *The approximation ratio of k-Opt is in $\Omega(\sqrt[k]{n})$ for $k = 3, 4, 6$ and in $\Omega(n^{\frac{2}{3k-4+\epsilon}})$ for all fixed k where $\epsilon = 0$ if k is even and $\epsilon = 1$ if k is odd.*

Comparing our upper and lower bounds we obtain:

▶ **Theorem 6.** *Our most general upper bound depending on $\mathrm{ex}(n, 2k)$ is tight up to a factor of $O(\log(n))$.*

The upper bounds can be carried over to a parameterized version of the Lin-Kernighan algorithm:

▶ **Theorem 7.** *The same upper bounds from Theorem 1 and 3 hold for a parameterized version of the Lin-Kernighan algorithm with appropriate parameter.*

Although the Lin-Kernighan algorithm only considers special changes, namely changes by augmenting a closed alternating walk, we are able to show the same upper bound as for the general k-Opt algorithm. For the original version of Lin-Kernighan we get an improved upper bound of $O(\sqrt[3]{n})$. Our results solve two of the four open questions in [5], namely:

- Can the upper bounds given in [5] be generalized to the k-Opt algorithm, i.e. for increasing k the performance guarantee improves?
- Can we show better upper bounds for the Lin-Kernighan algorithm than the upper bound obtained from the 2-Opt algorithm?

We also bound the approximation ratio of the k-Opt algorithm for Graph TSP.

▶ **Theorem 8.** *The approximation ratio of k-Opt with $k \geq 2$ for Graph TSP is $\Omega\left(\frac{\log(n)}{\log\log(n)}\right)$.*

▶ **Theorem 9.** *The approximation ratio of 2-Opt for Graph TSP is $O\left(\left(\frac{\log(n)}{\log\log(n)}\right)^{\log_2(9)+\epsilon}\right)$ for all $\epsilon > 0$.*

Note that the same upper bound also applies to the k-Opt algorithm and the Lin-Kernighan algorithm since they produce 2-optimal tours. Hence, up to a constant factor of at most $\log_2(9)$ in the exponent the k-Opt algorithm does not achieve asymptotically better performance than the 2-Opt algorithm in contrast to the metric case.

Outline of the paper. We start with the basic definitions we need for the analysis in the preliminaries. Then, an outline of the analysis roughly describes the main ideas for the lower and upper bounds on the approximation ratio for Metric and Graph TSP. In the main part of the paper we will only focus on the upper bound on the approximation ratio of the k-Opt algorithm for Metric TSP. Note that the same analysis can be carried over to the Lin-Kernighan algorithm by showing that the k-moves we consider can be performed by augmenting appropriate alternating cycles. For more details on this and the analysis of the other bounds we refer to the full version of the paper.

1.1 Preliminaries

1.1.1 TSP

An instance of Metric TSP is given by a complete weighted graph (K_n, c) where the costs are non-negative and satisfy the triangle inequality: $c(\{x, z\}) + c(\{z, y\}) \geq c(\{x, y\})$ for all $x, y, z \in V(K_n)$. A *cycle* is a closed walk that visits every vertex at most once. A *tour* is a cycle that visits every vertex exactly once. For a tour T, let the *length* of the tour be defined as $c(T) := \sum_{e \in T} c(e)$. The task is to find a tour of minimal length. We fix an *orientation* of the tour, i.e. we consider the edges of the tour as directed edges such that the tour is a directed cycle. From now on, let n denote the number of vertices of the instance.

Graph TSP is a special case of the Metric TSP. Each instance arises from an unweighted, undirected connected graph G. To construct a TSP instance (K_n, c), we set $V(K_n) = V(G)$. The cost $c(\{u, v\})$ of the edge connecting any two vertices $u, v \in V(G)$ is given by the length of the shortest u-v-path in G.

An algorithm A for the traveling salesman problem has *approximation ratio* $\alpha(n) \geq 1$ if for every TSP instance with n vertices it finds a tour that is at most $\alpha(n)$ times as long as a shortest tour and this ratio is achieved by an instance for every n. Note that we require here the sharpness of the approximation ratio deviating from the standard definition in the literature to express the approximation ratio in terms of the Landau symbols. Nevertheless, the results also hold for the standard definition with more complicated notation.

1.1.2 k-Opt and Lin-Kernighan Algorithm

A *k-move* replaces at most k edges of a given tour by other edges to obtain a new tour. It is called *improving* if the resulting tour is shorter than the original one. A tour is called *k-optimal* if there is no improving k-move.

For the 2-Opt algorithm recall the following well known fact: Given a tour T with a fixed orientation, it stays connected if we replace two edges of T by the edge connecting their heads and the edge connecting their tails, i.e. if we replace edges $(a, b), (c, d) \in T$ by (a, c) and (b, d).

An *alternating walk* of a tour T is a walk starting with an edge in T where exactly one of two consecutive edges is in T. An edge of the alternating walk is called *tour edge* if it is contained in T, otherwise it is called *non-tour edge*. A *closed alternating walk* and *alternating cycle* are alternating walks whose edges form a closed walk and cycle, respectively.

We consider a parameterized version of the Lin-Kernighan algorithm described in Section 21.3 of [16] for the analysis. In this version two parameters p_1 and p_2 specify the depth the algorithm is searching for an improvement. Since this extended abstract will focus on the k-Opt algorithm, we do not describe the Lin-Kernighan algorithm here and refer to the Section 21.3 of [16] or the full version of the paper.

1.2 Girth and Ex

The *girth* of a graph is the length of the shortest cycle contained in the graph if it contains a cycle and infinity otherwise. Let $\mathrm{ex}(n, 2k)$ be the maximum number of edges in a graph with n vertices and girth at least $2k$. Moreover, define $\mathrm{ex}^{-1}(m, 2k)$ as the minimal number of vertices of a graph with m edges and girth at least $2k$.

2 Outline of the Analysis

In this section we give an outline of the analysis for the lower and upper bounds of k-Opt for the Metric TSP and Graph TSP.

2.1 Outline of Lower Bound for Metric TSP

In this subsection we sketch the lower bound of k-Opt for the Metric TSP given by Theorem 2. We use the following lemma from [5]:

▶ **Theorem 10** (Lemma 3.6 in [5]). *Suppose there exists a Eulerian unweighted graph $G_{k,n,m}$ with n vertices and m edges, having girth at least $2k$. Then, there is a Metric TSP instance with m vertices and a k-optimal tour T such that $\frac{c(T)}{c(T^*)} \geq \frac{m}{2n}$, where T^* is the optimal tour of the instance.*

For the previous lower bound the theorem was applied to regular Eulerian graphs with high girth. Instead, we show that for every graph that there is a Eulerian subgraph with similar edge vertex ratio and apply the theorem to the Eulerian subgraphs of dense graphs with high girth to get the new bound.

2.2 Outline of Upper Bound for Metric TSP

In this subsection we briefly summarize the ideas for the analysis of the upper bound for the Metric TSP given by Theorem 1. For a fixed k assume that an instance is given with a k-optimal tour T. We fix an orientation of T and assume w.l.o.g. that the length of the optimal tour is 1. To bound the approximation ratio it is enough to bound the length of T. Our general strategy is to construct an auxiliary graph depending on T and bound its girth. More precisely, we show that if this graph has a short cycle this would imply the existence of an improving k-move contradicting the k-optimality of T. Moreover, the auxiliary graph contains many long edges of T so the bound on its girth also bounds the number of long edges in the tour and hence the approximation ratio.

Let the graph G consist of the vertices of the instance and the edges of T, i.e. $G := (V(K_n), T)$. We first divide the edges of T in length classes such that the lth length class consists of the edges with length between c^{l+1} and c^l for some constant $c < 1$, we call these edges *l-long*. For each $l \in \mathbb{N}_0$ we want get an upper bound on the number of l-long edges that depends on the number of vertices.

If we performed the complete analysis on G, we would get a bad bound on the number of l-long edges since G contains too many vertices. To strengthen the result we first construct an auxiliary graph containing all l-long edges for some fixed l but fewer vertices and bound the number of l-long edges in that graph: We partition $V(G)$ into classes with help of the optimal tour such that in each class any two vertices have small distance to each other. We contract the vertices in each class to one vertex and delete self loops to get the multigraph G_1^l. We can partition $V(G)$ in such a way that G_1^l contains all the l-long edges. Note we did not delete parallel edges in G_1^l and hence every edge in G_1^l has a unique preimage in G.

Unfortunately, we cannot directly bound the girth of G_1^l since the existence of a short cycle would not necessarily imply an improving k-move for T. For that we need a property of the cycles in the graph: The common vertex of consecutive edges in any cycle has to be head of both or tail of both edges according to the orientation of T. Therefore, we construct the auxiliary graph G_2^l from G_1^l as follows: We start with G_2^l as a copy of G_1^l and color the vertices of G_2^l red and blue. We only consider l-long edges in G_2^l from a red vertex to a blue vertex according to the orientation of T and delete all other edges. We can show that the coloring can be done in such a way that at least $\frac{1}{4}$ of the l-long edges remain in G_2^l.

We claim that the underlying undirected graph of G_2^l has girth at least $2k$. Note that by construction the graph is bipartite and hence all cycles have even length. Assume that there is a cycle C with $2h < 2k$ edges. We call the preimage of the edges of C in G the *C-edges*. Our aim is to construct a tour T' with the assistance of C that arises from T by an improving k-move.

For every common vertex w of two consecutive edges e_1, e_2 of C in G_2^l we consider the preimage e_1^{-1}, e_2^{-1} of e_1, e_2 in G. Then there have to be endpoints $u \in e_1^{-1}$ and $v \in e_2^{-1}$ such that the images of u and v after the contraction in G_2^l are both w. We will call the edge $\{u, v\}$ a *short edge*. In fact since both endpoints of a short edge are mapped to the same vertex in G_1^l after the contraction and we contracted vertices which have small distance to each other, they are indeed short. Furthermore, we can show that the total length of all the short edges is shorter than that of any single C-edge. The number of the short edges is equal to the number of C-edges which is $2h$. Now, observe that the cycle C defines an alternating cycle in G in a natural way: Let the preimages of C in G be the tour edges and the short edges be the non-tour edges.

To construct a new tour T' from T we start by augmenting the alternating cycle. Afterwards, the tour may split into at most $2h$ connected components. A key property is that the coloring of the vertices in G_2^l ensures that every connected component contains at least two short edges. Since there are $2h$ short edges, we know that after the augmentation we actually get at most h connected components. To reconnect and retain the degree condition we add twice a set L of at most $h-1$ different C-edges, i.e. in total at most $2h-2$ edges. In the end we shortcut to the new tour T' in a particular way without decreasing $|T \cap T'|$.

Note that the original tour T contains $2h$ C-edges, thus T' contains at least 2 fewer C-edges than T. The additional short edges T' contains are cheap, therefore T' is cheaper than T. Moreover, T' arises from T by replacing at most $2h - |L|$ C-edges since we deleted the C-edges and added twice the set L consisting of C-edges. Therefore, we know that T' arises from T by a $2h - |L| \leq 2h$-move. By the k-optimality of T, we have $2h > k$ or $2h \geq k + 1$. This already gives us a lower bound of $k + 1$ for the girth of the graph G_2^l as C contains $2h$ edges.

In the next step we use the previous result to show that there is actually a cheaper tour T' that arises by an $h + 1$-move. This implies that $h + 1 > k$ or $2h \geq 2k$, i.e. the girth of G_2^l is at least $2k$. As we have seen above the number of edges we have to replace to obtain T' from T depends on $|L|$, the number of C-edges T' contains. Therefore, we modify T' iteratively such that the number of C-edges in T' increases by 1 after every iteration while still maintaining the property that T' is cheaper than T. We stop when the number of C-edges in T' is $h - 1$ as then T' would arise from T by a $2h - (h - 1) = h + 1$-move.

To achieve this we start with the constructed tour T' and iteratively perform 2-moves that are not necessarily improving but add one more C-edge to T'. In every iteration we consider C-edges e not in the current tour T'. We can show that there is an edge in $T' \backslash T$ incident to each of the endpoints of e. Let the two edges be f_1 and f_2. We want to replace f_1 and f_2 in T' by e_1 and the edge connecting the endpoints of f_1 and f_2 not incident to e. To ensure the connectivity after the 2-move we need to find edges e such that the corresponding edges f_1, f_2 fulfill the following condition: Either both heads or both tails of f_1 and f_2 have to be endpoints of e. It turns out that we can find such edges e in enough iterations to construct T' with the desired properties.

In the end we notice that a lower bound on the girth of G_2^l gives us an upper bound on the number of edges in G_2^l by previous results on extremal graph theory. This implies an upper bound on the number of l-long edges as G_2^l contains at least $\frac{1}{4}$ of the l-long edges in T. That gives us an upper bound on the length of T and thus also an upper bound on the approximation ratio as we assumed that the optimal tour has length 1.

2.3 Outline of Lower Bound for Graph TSP

For the lower bound of Graph TSP given by Theorem 8 let an integer f be given. We construct an instance with approximation ratio $\Theta(f)$ and $(c_1 f)^{c_2 f}$ vertices for some constants $c_1, c_2 > 0$. Thus, the approximation ratio is $\Omega\left(\frac{\log n}{\log\log n}\right)$.

The construction starts with a dense $2f$-regular Eulerian graph G with high girth. Let $W = (v_0, v_1, \ldots, v_{|E(G)|-1})$ be a Eulerian walk of G. Traverse through G according to W starting at v_0 and mark every fth vertex both in G and in W. Whenever we would mark an already marked vertex v in G, we add a new copy v' of v adjacent exactly to the neighbors of v and mark v' instead. Moreover, we replace this occurrence of v in W by v' and mark v'. Let G' be the graph containing G and all the copies of the vertices we made. After the traversal of W, we mark for every unmarked vertex in G' one occurrence of it in W. Since we only need the property that every vertex of G' is marked in W it does not matter which occurrence we mark. The tour T consists of the edges connecting consecutive marked vertices in W.

The proof that T is k-optimal uses the same basic idea as the lower bound of the approximation ratio in the metric case in [5]: If there was an improving k-move, it has to contain an alternating cycle with negative cost. By construction, the length of every edge in T is bounded by f. Thus, such an alternating cycle would imply the existence of a short cycle in G' that can be transformed to a short cycle in G contradicting its high girth.

The constructed instance has approximation ratio $\Theta(f)$ since on the one hand, almost every edge in T has length $\Theta(f)$ leading to a total length of approximately $f|V(G')|$. On the other hand, the length of the optimal tour can be bounded by twice the length of the minimum spanning tree which is at most $2|V(G')|$ in the case of Graph TSP.

2.4 Outline of Upper Bound for Graph TSP

This subsection comprises a sketch of the proof of Theorem 9. Assume that an instance of Graph TSP (K_n, c) is given where c arises from the unweighted graph G. Let a 2-optimal tour T be given for the instance and fix an orientation.

First, note that every edge with length l corresponds to shortest paths with l edges in G between the endpoints of the edges. Now, if the corresponding shortest paths of two edges share a common directed edge, we see that there is an improving 2-move contradicting the assumed 2-optimality of T (Figure 1). Hence, the directed edges of the corresponding shortest paths are disjoint. Note that the optimal tour contains n edges and hence has length at least n. Thus, if the approximation ratio is high, we must have many edges in the union of the shortest paths corresponding to the edges in T and hence also in G. The main challenge now is to exploit this fact in a good way since a simple bound of $n(n-1)$ on the number of directed edges in G would only give an upper bound of $O(n)$ on the approximation ratio, which is worse than the upper bound of $O(\sqrt{n})$ for Metric TSP.

To get a better result we use the same idea from the analysis of the upper bound for Metric TSP: We contract vertices and get a graph with fewer vertices and many edges. Instead of contracting once, we iteratively partition the vertices into sets and contract each set to a single vertex to get a new graph. (We note that we actually just contract the vertices and construct the edges of the new graph in a slightly different way. But let us assume for simplicity that the edges of the new graph are images of the contraction of edges in the old graph.) Starting with G in every iteration we ideally want to partition the vertices of the current graph into sets, contract each set to a vertex and delete self loops such that:

1. The number of vertices decreases much faster than the number of edges.
2. The subgraphs induced by the sets we contract have small diameter.

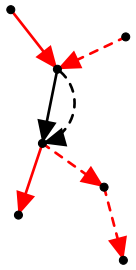


Figure 1 The solid and dashed edges are shortest paths that correspond to two edges in T. If they share a directed edge, there exists an improving 2-move replacing these two edges. The cost of the new edges is bounded by the number of the red edges which is less than the total cost of the two original edges.

The first condition ensures that we get a better bound after every iteration. The second condition builds the connection between the approximation ratio and the number of edges in the contracted graph: It ensures that if the shortest paths corresponding to two edges of T share a directed edge in the contracted graph, then they are also not far away in G, so there is an improving 2-move replacing these two edges. This means that a high approximation ratio would imply a high number of edges in the contracted graph.

Unfortunately, it is not easy to ensure both conditions at the same time even if we know that the graph has many edges as the edges are not equally distributed in the graph. It might happen that there are many vertices with very low degree. If we contract them while still ensuring that the subgraphs have small diameter, the number of vertices cannot decrease fast enough. Therefore, we consider a subset of vertices we call *active* vertices and only require that the number of active vertices decreases fast. If an active vertex has low degree, we will not contract it and consider it as inactive in future iterations. Initially, all vertices are active and we use the following theorem from [11] to find a good partition of the active vertices:

▶ **Theorem 11** (Theorem 6 in [11]). *Given $\epsilon > 0$ every graph G on n vertices can be edge partitioned $E = E_0 \cup E_1 \cup \cdots \cup E_l$ such that $|E_0| \leq \epsilon n^2, l \leq 16\epsilon^{-1}$, and for $1 \leq i \leq l$ the diameter of E_i is at most 4.*

In every iteration we apply the theorem to the subgraph induced by the currently active vertices. The vertices only incident to edges in E_0 become inactive after this iteration. For each of the sets $E_1, \ldots, E_l$ we contract the vertices incident to an edge in the set to a single vertex. These are the active vertices in the next iteration. By choosing ϵ appropriately, we can ensure that the number of vertices decreases significantly and the number of vertices that become inactive in every iteration is small.

After a fixed number of iterations, we have at least one edge and one active vertex remaining. Since the number of active vertices decreased much faster than the edges, we can conclude that G only contains few edges compared to the number of vertices. This implies a bound on the approximation ratio.

3 Upper Bound for Metric TSP

In this section we give an upper bound on the approximation ratio of the k-Opt algorithm.

Fix a $k > 2$ and assume that a worst-case instance with n vertices is given. Let T be a k-optimal tour of this instance. We fix an orientation of the optimal tour and T. Moreover, let w.l.o.g. the length of the optimal tour be 1. We divide the edges of T into length classes.

▶ **Definition 12.** *An edge e is* l-long *if* $(\frac{4k-5}{4k-4})^{l+1} < c(e) \leq (\frac{4k-5}{4k-4})^{l}$.

Note that the shortest path between every pair of vertices has length at most $\frac{1}{2}$ since the optimal tour has length 1. Thus, by the triangle inequality every edge with positive length in T has length at most $\frac{1}{2}$ and is l-long for exactly one l. For every l we want to bound the number of l-long edges. Let us consider from now on a fixed l. In the following we define auxiliary graphs we need for the analysis and show some useful properties of them.

▶ **Definition 13.** *We view the optimal tour as a circle with circumference 1. Let the vertices of the instance lie on that circle in the order of the oriented tour where the arc distance of two consecutive vertices is the length of the edge between them. Divide the optimal tour circle into* $4(k-1)\lceil(\frac{4k-4}{4k-5})^{l}\rceil$ *consecutive arcs of length* $\frac{1}{4(k-1)\lceil(\frac{4k-4}{4k-5})^{l}\rceil}$. *Two vertices are called* near *to each other if they lie on the same arc.*

▶ **Definition 14.** *Let the directed graph* $G := (V(K_n), T)$ *consist of the vertices of the instance and the oriented edges of* T *(an example is shown in Figure 2, the colors of the edges will be explained later). The directed multigraph* G_1^l *arises from* G *by contracting all vertices near to each other to a vertex and deleting self-loops (Figure 3).*

Note that G_1^l may contain parallel edges. Moreover, edges between vertices which are near to each other are not l-long and hence G_1^l contains all l-long edges.

▶ **Lemma 15.** *There exists a coloring of the vertices of* G_1^l *with two colors such that at least* $\frac{1}{4}$ *of the* l*-long edges in* G_1^l *go from a red vertex to a blue vertex according to the fixed orientation of* T.

Proof Sketch. The proof is similar to the standard proof that a maximal cut of a graph contains at least $\frac{1}{2}$ of the edges (see for example Theorem 5.3 in [27]). ◀

▶ **Definition 16.** *We obtain the directed multigraph* G_2^l *by coloring the vertices of* G_1^l *red and blue according to Lemma 15 and deleting all edges that are not* l*-long edges from a red vertex to a blue vertex according to the fixed orientation of* T *(Figure 4, the colors of the edges will be explained later).*

Now, we claim that the underlying undirected graph of G_2^l has girth at least $2k$. In particular, it is a simple graph. Assume the contrary, then there has to be a cycle C with $2h < 2k$ edges since G_2^l is bipartite by construction. We call the preimage of the edges of C in G the C*-edges*. Note that the preimages are unique since we do not delete parallel edges after the contraction.

▶ **Definition 17.** *Let the* connecting paths *be the connected components of* $(V(K_n), T \backslash C)$, *i.e. the paths in* T *between consecutive heads and tails of* C*-edges (the red edges in Figure 2 and 4). Define* head *and* tail *of a path* p *as the head of the last edge and the tail of the first edge of* p *according to the orientation of* T, *respectively. The head and tail of a connecting path are also called the* endpoints *of the connecting path.*

▶ **Definition 18.** *For any two endpoints* v_1, v_2 *of* C*-edges in* G *which are near to each other we call the edge* $\{v_1, v_2\}$ *a* short edge.

The definition of near ensures that the short edges are indeed short. In fact the total length of all short edges is smaller than that of any C-edge. The number of short edges is $2h$ which is equal to the number of C-edges.

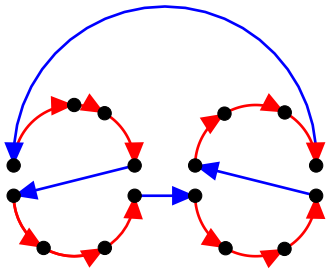


Figure 2 An example instance with a k-optimal tour, i.e. the directed graph G. The blue and red edges are the C-edges and connecting path edges that arise from the chosen cycle in G_2^l in Figure 4, respectively.

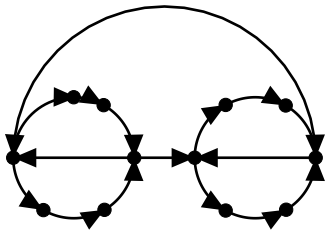


Figure 3 The directed multigraph G_1^l: We contracted vertices that lie near to each other in the optimal tour. Note that the optimal tour is not drawn here, so it is not clear from the figure which vertices to contract.

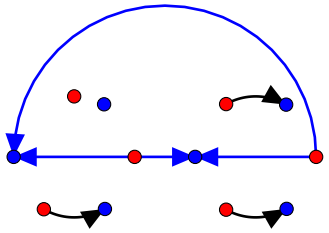


Figure 4 The directed multigraph G_2^l: Coloring the vertices and only considering the l-long edges from red to blue. In this example the upper left edge is not l-long and hence not drawn. The blue edges form the undirected cycle C, the black edges are the remaining edges of the connecting paths corresponding to this cycle.

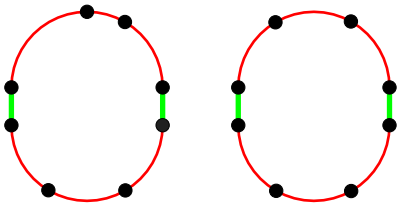


Figure 5 The graph $G_3^{l,C}$: The green edges are the short edges, the red edges are the connecting paths.

▶ **Definition 19.** *We construct the graph $G_3^{l,C}$ as follows: The vertex set of $G_3^{l,C}$ is that of G and the edge set consists of the connecting paths and the short edges (Figure 5).*

▶ **Lemma 20.** *$E(G_3^{l,C})$ is the union of at most h disjoint cycles.*

Proof Sketch. We can show that the degree of every vertex in $G_3^{l,C}$ is two. Moreover, we can see by considering the incident C-edges that the two endpoints of a connecting path are not near to each other since otherwise the common endpoint of the connecting path in G_2^l has to be colored red and blue. Hence, every connected component consists of at least two out of $2h$ connecting paths and we have at most h disjoint cycles. ◀

Now, we show that the existence of C implies that there is an improving k-move contradicting the k-optimality of T.

▶ **Lemma 21.** *There is a tour T' containing the connecting paths and $u-1$ C-edges, where u is the number of connected components of $G_3^{l,C}$.*

Proof Sketch. We construct such a tour T'. Start with a graph G' with the same vertex set and edge set as $G_3^{l,C}$. First, add a set of C-edges to $E(G')$ that makes the graph connected. This is possible since T consists of the C-edges and connecting paths and is connected. We call these C-edges the *fixed C-edges*. Next, add another copy of the fixed C-edges (Figure 6). After shortcutting in a particular way without decreasing $|T \cap T'|$, we get a tour with the desired properties. ◀

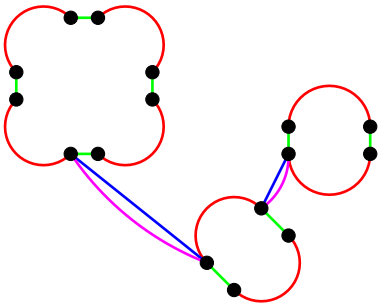


Figure 6 Sketch for Lemma 21. The red curves represent the connecting paths. The green edges are the short edges, the blue edges are the fixed C-edges and the pink edges are the copies of the fixed C-edges. The tour T' results from shortcutting the green and pink edges while leaving the other edges fixed.

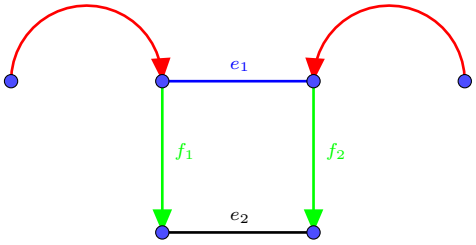


Figure 7 Sketch for Lemma 25. The drawn orientation is that of T'. The red curves represent oppositely oriented connecting paths connected by a C-edge e_1. The green edges f_1 and f_2 are the non-connecting path edges of T' incident to e_1. The edge e_2 connects the other two endpoints of f_1 and f_2 not incident to e_1.

▶ Remark 22. The last lemma already gives us a bound on the girth of G_2^l: By Lemma 20, $G_3^{l,C}$ has at most h connected components. Therefore, we added in the construction above at most $2h-2$ C-edges and some cheap short edges. As T contains $2h$ C-edges, we can show that T' is shorter than T. Moreover, T' arises from T by replacing at most $2h$ C-edges, i.e. by a $2h$-move. If $2h \leq k$, this would contradict the k-optimality of T, hence G_2^l has girth at least $k+1$.

Next, we improve this result and show that G_2^l has girth at least $2k$. We achieve this by starting at T' and iteratively performing 2-moves that are not necessarily improving but include one more C-edge in T'. We stop when the number of C-edges in T' is $h-1$. Then, T' arises from T by a $2h-(h-1) = h+1$ move.

▶ **Definition 23.** *Given a tour T' containing the connecting paths. An* ambivalent 2-move *replaces two non-connecting path edges of T' to obtain a new tour containing at least one more C-edge.*

▶ **Definition 24.** *Fix an orientation of T', we call a connecting path p* wrongly oriented *if the orientation of p in T' is opposite to the orientation in T. Otherwise, it is called* correctly oriented.

▶ **Lemma 25.** *If a tour T' contains a short edge and all connecting paths, then there is an ambivalent 2-move that increases the length of the tour by at most two C-edges.*

Proof Sketch. The coloring of the vertices in G_2^l ensures that every short edge e connects either two heads or two tails of connecting paths. If in addition $e \in T'$, one of them is correctly oriented and the other one is wrongly oriented. Thus, as long as there is a short edge in T', there has to be at least one correctly oriented and one wrongly oriented connecting path. In this case there has to be a C-edge e_1 connecting two oppositely oriented connecting paths since the C-edges connect the connecting paths to the tour T. By definition, every C-edge connects a head and a tail of two connecting paths. If $e_1 \in T'$, the incident connecting paths would be both correctly or both wrongly oriented. Thus, e_1 is not contained in T'. Let e_2, f_1 and f_2 be defined as in Figure 7. Now, we can make a 2-move replacing f_1, f_2 by e_1 and e_2 to obtain a new tour with the additional C-edge e_1. The property that e_1 connects

two oppositely oriented connecting paths ensures that the tour stays connected after the 2-move. By the triangle inequality, we have $c(e_2) \leq c(f_1) + c(e_1) + c(f_2)$ and thus each of the 2-moves increases the length of the tour by at most two C-edges. ◀

▶ **Lemma 26.** *T is not $h+1$-optimal.*

Proof Sketch. By Lemma 21, we can construct a tour T' using the connecting paths and $u-1$ C-edges where u is the number of connected components of $G_3^{l,C}$. We iteratively perform ambivalent 2-moves to increase the number of C-edges in T'. Since after every such 2-move the number of short edges decreases by at most two, we can perform by Lemma 25 a sufficient number of iterations such that we get a tour with $h-1$ C-edges. There are in total $2h$ C-edges, hence the resulting tour arises by an $2h-(h-1) = h+1$-move from T. As every ambivalent 2-move increases the length of the tour by at most two C-edges, we can show that in the end the resulting T' is still shorter than T. ◀

Since $h < k$, this is a contradiction to the assumption that T is k-optimal. Hence, such a cycle C with less than $2k$ edges cannot exist and this gives us a lower bound of $2k$ on the girth of G_2^l. Next, we conclude an upper bound on the length of T:

▶ **Corollary 27.** *For $l^* := \min_j\{j | \sum_{l=0}^{j} 4\,\mathrm{ex}(4(k-1)\lceil(\frac{4k-4}{4k-5})^l\rceil, 2k) \geq n\}$ we have*

$$c(T) \leq \sum_{l=0}^{l^*} \frac{4\,\mathrm{ex}(4(k-1)\lceil(\frac{4k-4}{4k-5})^l\rceil, 2k)}{(\frac{4k-4}{4k-5})^l}.$$

Proof Sketch. Let q_l be the number of l-long edges in T. The definition of near ensures that two vertices which are near to each other have shorter distance than the length of any l-long edge. Hence, G_1^l also has q_l l-long edges. Since we have chosen a coloring according to Lemma 15, G_2^l has at least $\frac{1}{4}q_l$ edges. By the k-optimality and Lemma 26, G_2^l has girth at least $2k$ and thus at most $\mathrm{ex}(|V(G_2^l)|, 2k) \leq \mathrm{ex}(4(k-1)\lceil(\frac{4k-4}{4k-5})^l\rceil, 2k)$ edges. Therefore, $q_l \leq 4\,\mathrm{ex}(4(k-1)\lceil(\frac{4k-4}{4k-5})^l\rceil, 2k)$. This leads to a bound on the length of T. ◀

This is also a bound on the approximation ratio since the length of the optimal tour is 1. With certain assumptions about the growth of $\mathrm{ex}(n, 2k)$, we obtain the main result:

▶ **Theorem 28.** *If $\mathrm{ex}(x, 2k) \in O(x^c)$ for some $c > 1$, the approximation ratio of the k-Opt algorithm is $O(n^{1-\frac{1}{c}})$.*

By a rather technical calculation comparing the upper and lower bound, we get:

▶ **Theorem 29.** *The upper bound from Corollary 27 for the approximation ratio of k-Opt is tight up to a factor of $O(\log(n))$.*

References

1. Noga Alon, Shlomo Hoory, and Nathan Linial. The Moore bound for irregular graphs. *Graphs and Combinatorics*, 18(1):53–57, March 2002. doi:10.1007/s003730200002.
2. Clark T. Benson. Minimal regular graphs of girths eight and twelve. *Canadian Journal of Mathematics*, 18:1091–1094, 1966. doi:10.4153/CJM-1966-109-8.
3. Jon J. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on computing*, 4(4):387–411, 1992. doi:10.1287/ijoc.4.4.387.
4. William G. Brown. On graphs that do not contain a thomsen graph. *Canadian Mathematical Bulletin*, 9(3):281–285, 1966.

5 Barun Chandra, Howard Karloff, and Craig Tovey. New results on the old k-opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*, 28(6):1998–2029, 1999. doi:10.1137/S0097539793251244.

6 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

7 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. *Algorithmica*, 68(1):190–264, January 2014. doi:10.1007/s00453-013-9801-4.

8 Paul Erdős and Alfréd Rényi. On a problem of graph theory. *Magyar Tudományos Akadémia Math. Kuató Int. Közl.*, 7:623–641, 1962.

9 Paul Erdős, Alfréd Rényi, and Vera T. Sós. On a problem of graph theory. *Studia Scientiarum Mathematicarum Hungarica*, 1:215–235, 1966.

10 Paul Erdős. Extremal problems in graph theory. In *Proc. Symp. Theory of Graphs and its Applications*, pages 29–36, 1963.

11 Jacob Fox and Benny Sudakov. Decompositions into subgraphs of small diameter. *Combinatorics, Probability and Computing*, 19(5-6):753–774, 2010. doi:10.1017/S0963548310000040.

12 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

13 Stefan Hougardy, Fabian Zaiser, and Xianghui Zhong. The approximation ratio of the 2-opt heuristic for the metric traveling salesman problem. *Operations Research Letters*, 48(4):401–404, 2020. doi:10.1016/j.orl.2020.05.007.

14 David S. Johnson. Local optimization and the traveling salesman problem. In *International colloquium on automata, languages, and programming*, pages 446–461. Springer, 1990. doi:10.1007/BFb0032050.

15 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. doi:10.1007/978-3-540-68279-0_8.

16 Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 4th edition, 2007. doi:10.1007/978-3-642-24488-9.

17 Marvin Künnemann and Bodo Manthey. Towards understanding the smoothed approximation ratio of the 2-opt heuristic. In *International Colloquium on Automata, Languages, and Programming*, pages 859–871. Springer, 2015. doi:10.1007/978-3-662-47672-7_70.

18 Felix Lazebnik, Vasiliy A. Ustimenko, and Andrew J. Woldar. A new series of dense graphs of high girth. *Bulletin of the American mathematical society*, 32(1):73–79, 1995. doi:10.1090/S0273-0979-1995-00569-0.

19 Asaf Levin and Uri Yovel. Nonoblivious 2-opt heuristics for the traveling salesman problem. *Networks*, 62(3):201–219, 2013. doi:10.1002/net.21512.

20 Shen Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973. doi:10.1287/opre.21.2.498.

21 Ján Plesník. Bad examples of the metric traveling salesman problem for the 2-change heuristic. *Acta Mathematica Universitatis Comenianae*, 55:203–207, 1986.

22 Gerhard Reinelt. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, 1994. doi:10.1007/3-540-48661-5.

23 Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, 1977. doi:10.1007/978-1-4020-9688-4_3.

24 A. I. Serdjukov. Some extremal bypasses in graphs [in Russian]. *Upravlyaemye Sistemy*, 17:76–79, 1978.

25 Robert Singleton. On minimal graphs of maximum even girth. *Journal of Combinatorial Theory*, 1(3):306–332, 1966. doi:10.1016/S0021-9800(66)80054-6.

26 Rephael Wenger. Extremal graphs with no C^4's, C^6's, or C^{10}'s. *J. Comb. Theory, Ser. B*, 52(1):113–116, 1991. doi:10.1016/0095-8956(91)90097-4.

27 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.