

41st International Symposium on Theoretical Aspects of Computer Science

STACS 2024, March 12–14, 2024, Clermont-Ferrand, France

Edited by

Olaf Beyersdorff

Mamadou Moustapha Kanté

Orna Kupferman

Daniel Lokshtanov



Editors

Olaf Beyersdorff 

Friedrich Schiller University Jena, Germany
olaf.beyersdorff@uni-jena.de

Mamadou Moustapha Kanté 

Université Clermont Auvergne, Clermont Auvergne INP, LIMOS, CNRS, Aubière, France
mamadou.kante@uca.fr

Orna Kupferman 

Hebrew University, Jerusalem, Israel
orna@cs.huji.ac.il

Daniel Lokshtanov

University of California Santa Barbara, CA, USA
daniello@ucsb.edu

ACM Classification 2012

Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory; Theory of computation → Formal languages and automata theory; Theory of computation → Logic; Theory of computation → Design and analysis of algorithms; Theory of computation → Computational complexity and cryptography; Theory of computation → Models of computation

ISBN 978-3-95977-311-9

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-311-9>.

Publication date

March, 2024

Bibliographic information published by the Deutsche Nationalbibliothek

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

License

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0):
<https://creativecommons.org/licenses/by/4.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.STACS.2024.0

ISBN 978-3-95977-311-9

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

LIPICs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Luca Aceto (Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University, Brno, CZ)
- Meena Mahajan (*Chair*, Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Pierre Senellart (ENS, Université PSL, Paris, FR)

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

■ Contents

Preface	
<i>Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshтанov</i>	0:ix
Conference Organization	
.....	0:xi
List of Authors	
.....	0:xv

Invited Talks

Polynomial-Time Pseudodeterministic Constructions	
<i>Igor C. Oliveira</i>	1:1–1:1
The Role of Local Algorithms in Privacy	
<i>Sofya Raskhodnikova</i>	2:1–2:1
Structurally Tractable Graph Classes	
<i>Szymon Toruńczyk</i>	3:1–3:1

Regular Papers

Max Weight Independent Set in Sparse Graphs with No Long Claws	
<i>Tara Abrishami, Maria Chudnovsky, Marcin Pilipczuk, and Paweł Rzążewski</i>	4:1–4:15
Satisfiability of Context-Free String Constraints with Subword-Ordering and Transducers	
<i>C. Aiswarya, Soumodev Mal, and Prakash Saivasan</i>	5:1–5:20
On a Hierarchy of Spectral Invariants for Graphs	
<i>V. Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky</i>	6:1–6:18
Computing Twin-Width Parameterized by the Feedback Edge Number	
<i>Jakub Balabán, Robert Ganian, and Mathis Rocton</i>	7:1–7:19
Faster Graph Algorithms Through DAG Compression	
<i>Max Bannach, Florian Andreas Marwitz, and Till Tantau</i>	8:1–8:18
Testing Equivalence to Design Polynomials	
<i>Omkar Baraskar, Agrim Dewan, and Chandan Saha</i>	9:1–9:22
Expressive Quantale-Valued Logics for Coalgebras: An Adjunction-Based Approach	
<i>Harsh Beohar, Sebastian Gurke, Barbara König, Karla Messing, Jonas Forster, Lutz Schröder, and Paul Wild</i>	10:1–10:19
A Characterization of Efficiently Compilable Constraint Languages	
<i>Christoph Berkholz, Stefan Mengel, and Hermann Wilhelm</i>	11:1–11:19



Modal Logic Is More Succinct Iff Bi-Implication Is Available in Some Form <i>Christoph Berkholz, Dietrich Kuske, and Christian Schwarz</i>	12:1–12:17
Temporalizing Digraphs via Linear-Size Balanced Bi-Trees <i>Stéphane Bessy, Stéphan Thomassé, and Laurent Viennot</i>	13:1–13:12
A Subquadratic Bound for Online Bisection <i>Marcin Bienkowski and Stefan Schmid</i>	14:1–14:18
An Improved Approximation Algorithm for Dynamic Minimum Linear Arrangement <i>Marcin Bienkowski and Guy Even</i>	15:1–15:19
Gapped String Indexing in Subquadratic Space and Sublinear Query Time <i>Philip Bille, Inge Li Gørtz, Moshe Lewenstein, Solon P. Pissis, Eva Rotenberg, and Teresa Anna Steiner</i>	16:1–16:21
Contributions to the Domino Problem: Seeding, Recurrence and Satisfiability <i>Nicolás Bitar</i>	17:1–17:18
Removable Online Knapsack and Advice <i>Hans-Joachim Böckenhauer, Fabian Frei, and Peter Rossmanith</i>	18:1–18:17
The Complexity of Homomorphism Reconstructibility <i>Jan Böker, Louis Härtel, Nina Runde, Tim Seppelt, and Christoph Standke</i>	19:1–19:20
Solving Discontinuous Initial Value Problems with Unique Solutions Is Equivalent to Computing over the Transfinite <i>Olivier Bournez and Riccardo Gozzi</i>	20:1–20:19
Local Certification of Local Properties: Tight Bounds, Trade-Offs and New Parameters <i>Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun</i>	21:1–21:18
Spectral Approach to the Communication Complexity of Multi-Party Key Agreement <i>Geoffroy Caillat-Grenier and Andrei Romashchenko</i>	22:1–22:19
Fault-tolerant k -Supplier with Outliers <i>Deeparnab Chakrabarty, Luc Cote, and Ankita Sarkar</i>	23:1–23:19
Approximate Circular Pattern Matching Under Edit Distance <i>Panagiotis Charalampopoulos, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba</i>	24:1–24:22
Depth-3 Circuit Lower Bounds for k -OV <i>Tameem Choudhury and KartEEK Sreenivasaiah</i>	25:1–25:17
A Myhill-Nerode Theorem for Generalized Automata, with Applications to Pattern Matching and Compression <i>Nicola Cotumaccio</i>	26:1–26:19
Nonnegativity Problems for Matrix Semigroups <i>Julian D’Costa, Joël Ouaknine, and James Worrell</i>	27:1–27:16
One n Remains to Settle the Tree Conjecture <i>Jack Dippel and Adrian Vetta</i>	28:1–28:16

Semënov Arithmetic, Affine VASS, and String Constraints <i>Andrei Draghici, Christoph Haase, and Florin Manea</i>	29:1–29:19
Fixed-Parameter Debordering of Waring Rank <i>Pranjal Dutta, Fulvio Gesmundo, Christian Ikenmeyer, Gorav Jindal, and Vladimir Lysikov</i>	30:1–30:15
On the Power of Border Width-2 ABPs over Fields of Characteristic 2 <i>Pranjal Dutta, Christian Ikenmeyer, Balagopal Komarath, Harshil Mittal, Saraswati Girish Nanoti, and Dhara Thakkar</i>	31:1–31:16
$O(1/\varepsilon)$ Is the Answer in Online Weighted Throughput Maximization <i>Franziska Eberle</i>	32:1–32:15
On the Exact Matching Problem in Dense Graphs <i>Nicolas El Maalouly, Sebastian Haslebacher, and Lasse Wulf</i>	33:1–33:17
Hardness of Linearly Ordered 4-Colouring of 3-Colourable 3-Uniform Hypergraphs <i>Marek Filakovský, Tamio-Vesa Nakajima, Jakub Opršal, Gianluca Tasinato, and Uli Wagner</i>	34:1–34:19
The 2-Attractor Problem Is NP-Complete <i>Janosch Fuchs and Philip Whittington</i>	35:1–35:13
Directed Regular and Context-Free Languages <i>Moses Ganardi, Irmak Sağlam, and Georg Zetsche</i>	36:1–36:20
Online Simple Knapsack with Bounded Predictions <i>Matthias Gehnen, Henri Lotze, and Peter Rossmanith</i>	37:1–37:20
The AC^0 -Complexity of Visibly Pushdown Languages <i>Stefan Göller and Nathan Grosshans</i>	38:1–38:18
Quantum and Classical Communication Complexity of Permutation-Invariant Functions <i>Ziyi Guan, Yunqi Huang, Penghui Yao, and Zekun Ye</i>	39:1–39:19
A Faster Algorithm for Vertex Cover Parameterized by Solution Size <i>David G. Harris and N. S. Narayanaswamy</i>	40:1–40:18
Linear Loop Synthesis for Quadratic Invariants <i>S. Hitarth, George Kenison, Laura Kovács, and Anton Varonka</i>	41:1–41:18
Algorithms for Claims Trading <i>Martin Hoefer, Carmine Ventre, and Lisa Wilhelmi</i>	42:1–42:17
A Faster Algorithm for Constructing the Frequency Difference Consensus Tree <i>Jesper Jansson, Wing-Kin Sung, Seyed Ali Tabatabaee, and Yutong Yang</i>	43:1–43:17
Decremental Sensitivity Oracles for Covering and Packing Minors <i>Lawqueen Kanesh, Fahad Panolan, M. S. Ramanujan, and Peter Strulo</i>	44:1–44:19
Circuit Equivalence in 2-Nilpotent Algebras <i>Piotr Kawalek, Michael Kompatscher, and Jacek Krzaczkowski</i>	45:1–45:17
The Subpower Membership Problem of 2-Nilpotent Algebras <i>Michael Kompatscher</i>	46:1–46:17

Parameterized and Approximation Algorithms for Coverings Points with Segments in the Plane <i>Katarzyna Kowalska and Michał Pilipczuk</i>	47:1–47:16
FPT Approximation of Generalised Hypertree Width for Bounded Intersection Hypergraphs <i>Matthias Lanzinger and Igor Razgon</i>	48:1–48:17
Sub-Exponential Time Lower Bounds for #VC and #Matching on 3-Regular Graphs <i>Ying Liu and Shiteng Chen</i>	49:1–49:18
Arena-Independent Memory Bounds for Nash Equilibria in Reachability Games <i>James C. A. Main</i>	50:1–50:18
Weighted HOM-Problem for Nonnegative Integers <i>Andreas Maletti, Andreea-Teodora Nász, and Erik Paul</i>	51:1–51:17
Worst-Case and Smoothed Analysis of the Hartigan–Wong Method for k-Means Clustering <i>Bodo Manthey and Jesse van Rhijn</i>	52:1–52:16
Homomorphism-Distinguishing Closedness for Graphs of Bounded Tree-Width <i>Daniel Neuen</i>	53:1–53:12
Positionality in Σ_2^0 and a Completeness Result <i>Pierre Ohlmann and Michał Skrzypczak</i>	54:1–54:18
Tree-Layout Based Graph Classes: Proper Chordal Graphs <i>Christophe Paul and Evangelos Protopapas</i>	55:1–55:18
Randomized Query Composition and Product Distributions <i>Swagato Sanyal</i>	56:1–56:19
Shortest Two Disjoint Paths in Conservative Graphs <i>Ildikó Schlotter</i>	57:1–57:17
Algorithms for Computing Closest Points for Segments <i>Haitao Wang</i>	58:1–58:17
Lower Bounds for Set-Blocked Clauses Proofs <i>Emre Yolcu</i>	59:1–59:17

■ Preface

The International Symposium on Theoretical Aspects of Computer Science (STACS) conference series is an internationally leading forum for original research on theoretical aspects of computer science.

STACS 2024 consists of two tracks, A and B. Track A is dedicated to algorithms and data structures, complexity and games. Track B covers automata, logic, semantics and theory of programming.

STACS is held alternately in France and in Germany. This year's conference, taking place in Clermont-Ferrand (Université Clermont Auvergne) from March 12 to March 14, is the 41st in the series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), Freiburg (2009), Nancy (2010), Dortmund (2011), Paris (2012), Kiel (2013), Lyon (2014), München (2015), Orléans (2016), Hannover (2017), Caen (2018), Berlin (2019), Montpellier (2020), Saarbrücken (2021, taking place virtually), Marseille (2022, taking place virtually), and Hamburg (2023).

The interest in STACS has remained at a very high level over the past years. The STACS 2024 call for papers led to 203 submissions (153 for Track A and 50 for Track B) with authors from 37 countries. Each paper was assigned to three program committee members who, at their discretion, asked external reviewers for reports. For the tenth time within the STACS conference series, there was also a rebuttal period during which authors could submit remarks to the PC concerning the reviews of their papers. In addition, and for the fourth time, STACS 2024 employed a lightweight double-blind reviewing process: submissions should not reveal the identity of the authors in any way. However, it was still possible for authors to disseminate their ideas or draft versions of their paper as they normally would, for instance by posting drafts on the web or giving talks on their results. The committee selected 56 papers for publication (40 for Track A and 16 for Track B). This means an acceptance rate of around 28%. As co-chairs of the program committee, we would like to sincerely thank all its members and the 313 external reviewers for their valuable work. In particular, there were intense and interesting discussions inside the PC committee. The very high quality of the submissions made the selection an extremely difficult task.

We would like to express our thanks to the three invited speakers: Igor Carboni Oliveira (Warwick University, UK), Sofya Raskhodnikova (Boston University, USA), and Szymon Toruńczyk (Warsaw University, Poland).

We thank Michael Didas from the LIPIcs team for assisting us in the publication process and the final production of the proceedings. These proceedings contain extended abstracts of the accepted contributions and abstracts of the invited talks. The authors retain their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series. We also thank Sourav S Bhowmick who helped at the beginning of the reviewing process and assignment to PC members to identify conflicts with the Closet system. Finally, we would like to thank Université Clermont Auvergne, Clermont Auvergne INP, ISIMA, LIMOS, CNRS and Clermont Auvergne Metropole for their support. Our special



0:x Preface

thanks go to Béatrice Bourdieu, Dipayan Chakraborty, Solène Drouet, Florent Foucaud and Bruno Guillon – the local organising team at Université Clermont Auvergne – for all their help with the webpages, the registration and all social events.

Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman and Daniel Lokshtanov

■ Conference Organisation

Program Committee

Track A

Prabhanjan Ananth	University of California, Santa Barbara, USA
Edouard Bonnet	ENS Lyon, France
Zongchen Chen	MIT, USA
Alex Conway	VMWare Research Lab, USA
Fabien Dufoulon	University of Houston, USA
Christoph Durr	CNRS, Université Paris Sorbonne, France
Sebastian Forster	University of Salzburg, Germany
Elena Grigorescu	Purdue University, USA
Jacob Holm	University of Copenhagen, Denmark
Tony Huynh	Sapienza Università di Roma, Italia
Mamadou Moustapha Kanté	Université Clermont Auvergne, Clermont Auvergne INP, France, co-chair
Nutan Limaye	IT University of Copenhagen, Denmark
Daniel Lokshtanov	University of California, Santa Barbara, USA, co-chair
Or Meir	University of Haifa, Israel
Florin Manea	University of Göttingen, Germany
Benjamin Moseley	Carnegie Mellon University, USA
Alanta Newman	CNRS, Université Grenoble-Alpes, France
Kim Thang Nguyen	Grenoble INP, Université Grenoble-Alpes, France
Pan Peng	University of Science and Technology of China, China
Stephen Piddock	Royal Holloway University of London, UK
Saket Saurabh	Institute of Mathematical Sciences, Chennai, India
Sebastian Siebertz	University of Bremen, Germany
Nodari Sitchinava	University of Hawaii, USA
Sophie Spirkl	University of Waterloo, Canada
Jie Xue	New York University of Shanghai, China
Hang Zhou	Ecole Polytechnique, France

Track B

Paul Bell	Keele University, UK
Olaf Beyersdorff	Friedrich Schiller University of Jena, Germany, co-chair
Ilario Bonacina	UPC Barcelona, Spain
Andrea Cali	University of London, Birkbeck, UK
Joel Day	Loughborough University, UK
Hadar Frenkel	CISPA, Germany
Manfred Kufleitner	University of Stuttgart, Germany
Orna Kupferman	The Hebrew University of Jerusalem, Israel, co-chair
Markus Lohrey	University of Siegen, Germany
Sebastian Maneth	University of Bremen, Germany
Paige Randall North	Utrecht University, The Netherlands
Henning Schnoor	University of Kiel, Germany
Ana Sokolova	University of Salzburg, Germany
Anthony Widjaja Lin	The University of Edinburgh, UK



Steering Committee

Dietmar Berwanger	LMF, CNRS, Université Paris-Saclay
Marthe Bonamy	LaBRI, CNRS, Université de Bordeaux
Cyril Nicaud	LIGM, Université Paris-Est
Sylvain Schmitz	IRIF, Université de Paris
Luc Segoufin	DI ENS, INRIA, ENS Ulm
Ioan Todinca	LIFO, Université d'Orléans (co-chair)
Petra Berenbrink	Hamburg University
Olaf Beyersdorff	Friedrich Schiller University of Jena
Florin Manea	University of Göttingen
Arne Meier	University of Hannover
Heiko Röglin	University of Bonn
Thomas Schwentick	University of Dortmund (co-chair)

Local Organising Committee (Université Clermont Auvergne)

Béatrice Bourdieu
 Dipayan Chakraborty
 Solène Drouet
 Florent Foucaud
 Bruno Guillon
 Mamadou Moustapha Kanté

Subreviewers

313 external subreviewers assisted the PC. We apologize to any subreviewers who do not appear in this list (because their reviews were entered manually into EasyChair).

Abhranil Chatterjee	Austen Z. Fan	Colin Geniet
Achim Blumensath	Aviad Rubinstein	Cornelius Brand
Adele Rescigno	Balagopal Komarath	Da Wei Zheng
Aditya Gulati	Barnaby Martin	Daniel Neuen
Akanksha Agrawal	Benjamin Böhm	Daniel Paul Pena
Akira Suzuki	Benjamin Monmege	Daniel Schmand
Alessio Conte	Benjamin Moore	Daniel Stan
Alexander Lindermayr	Binghui Peng	Daniel Vaz
Alexandra Wesolek	Brigitte Vallee	Daochen Wang
Alexandre Vigny	Bruno Guillon	David Eppstein
Alexis de Colnet	C Ramya	David R. Wood
Amaury Pouly	Calum MacRury	David Roberson
Amer Mouawad	Ce Jin	David Wajc
Amey Bhangale	Chenyang Xu	Davide Bilò
Ana Silva	Chien-Chung Huang	Dawn Nye
Anamay Tengse	Chinmay Sonar	Dietrich Kuske
Anders Schlichtkrull	Christian Wulff-Nilsen	Diptapriyo Majumdar
Anirudh Chandramouli	Christof Löding	Dmitry Shkatov
Anthony Ostuni	Christoph Grunau	Dominik Kempa
Argyrios Deligkas	Chun-Hung Liu	Dr. Rakesh Mohanty
Arindam Khan	Clemens Kupke	Eduard Eiben
Ashwin Jacob	Clément Requilé	Eiji Miyano

Elvira Mayordomo	Karolina Okrasa	Murilo de Lima
Emmanouil Vasileios	Katharina Klost	Neil Lutz
Engel Lefaucheux	Keri D'Angelo	Neil Thapen
Enikő Kevi	Kevin Hendrey	Nicolaos Matsakis
Erhard Aichinger	Kevin Schewior	Nicolas Bonichon
Eric Blais	Kirk Pruhs	Niklas Metzger
Éric Colin de Verdière	Konrad K. Dabrowski	Nils Morawietz
Eric Samperton	Kord Eickmeyer	Ning Xie
Evripidis Bampis	Kuan Cheng	Nithish Kumar Kumar
Falko Hegerfeld	Lars Rohwedder	Noah Fleming
Fatih Kaleoglu	Laure Morelle	Noleen Köhler
Felipe A. Louza	Laurent Doyen	O-Joung Kwon
Florent Capelli	Lauri Hella	Oleg Verbitsky
Florent Foucaud	Lene Favrholt	Olivier Carton
Florent Jacquemard	Liana Khazaliya	Omar Fawzi
Floris Geerts	Linda Cook	Omri Ben-Eliezer
Francesco Scarcello	Luca Prigioniero	Orr Fischer
Georg Anegg	Luisa Herrmann	Oscar Defrain
Georg Zetsche	Lukáš Holík	Pablo Barcelo
Giorgio Lucarelli	Mads Toftrup	Pablo Pérez-Lantero
Giovanni Manzini	Manaswi Paraashar	Pallavi Jain
Giovanni Pighizzini	Manoj Gupta	Panagiotis Kanellopoulos
Giulia Bernardini	Manuel Aprile	Pascal Gollin
Gregory Schwartzman	Maoyuan Song	Pascal Schweitzer
Guillaume Malod	Marc Dufay	Patricia Bouyer
Gwenaël Joret	Marcin Bienkowski	Paul Harrenstein
Hanna Komlos	Marie-Pierre Béal	Paul Wollan
Heather Newman	Marius Zimand	Pavel Semukhin
Heiko Röglin	Mark Jones	Pawel Gawrychowski
Helen Xu	Markus Holzer	Pedro Montealegre
Henry Sinclair-Banks	Martin S. Krejca	Petar Iliev
Heribert Vollmer	Mathieu Hoyrup	Peter Kiss
Hervé Fournier	Mathieu Liedloff	Peter Mayr
Hoang La	Matthew Drescher	Peter Rossmann
Holger Dell	Matthias Englert	Peter Zeman
Honghao Fu	Matthias Pfretzschner	Petr Golovach
Hongxun Wu	Matus Telgarsky	Peyman Afshani
Ilan Doron	Max Hopkins	Pierre Bergé
Irene Muzi	Maxwell Levit	Pierre Fraigniaud
Ismaël Jecker	Mehrdad Ghadiri	Pierre Vandenhove
Itai Leigh	Meike Hatzel	Pierre-Alain Reynier
Jakob Burkhardt	Merlin Carl	Prafullkumar Tale
Jared Saia	Michael Drmota	Pranabendu Misra
Jarkko Kari	Michael Forbes	Prashant Vasudevan
Jason Schoeters	Michael Raskin	Prerona Chatterjee
Jayalal Sharma	Michał Włodarczyk	Qiyi Tang
Jeroen Zuiddam	Michaël Cadilhac	Qizheng He
Jesper Nederlof	Michał Pilipczuk	R Govind
Jianyu Xu	Mikhail Volkov	Radu Curticapean
Josh Alman	Mingyu Xiao	Rafael Oliveira
Joshua Grochow	Mirosław Kowaluk	Ramanujan M. Sridharan
João F. Doriguello	Moritz Lichter	Ramprasad Satharishi
Julian Siber	Morteza Monemizadeh	Raven Beutner
Julien Duron	Moses Ganardi	Rebecca Whitman
Jungho Ahn	Mostafa Milani	Reino Niskanen

René Romen	Stefan Siemer	Viet Anh Martin Vu
Rimma Hamalainen	Stefano Fioravanti	Viktor Zamaraev
Robert Mercas	Stefano Leucci	Viktoriia Korchemna
Rohan Ghuge	Steffen van Bergerem	Vishnu Iyer
Rolf Fagerberg	Sudatta Bhattacharya	Vishwas Bhargava
Roohani Sharma	Suhail Sherif	Vlatakis Gkaragkounis
Ruiwen Dong	Suryajith Chillara	Weiyun Ma
Ryan Williams	Sushmita Gupta	Will Ma
Rémy Belmonte	Tamalika Mukherjee	William Kretschmer
Saladi Rahul	Tamás Róbert Mezei	William Lochet
Sam Thompson	Tanmay Inamdar	Wojciech Czerwiński
Samir Datta	Telikepalli Kavitha	Wolfgang Mulzer
Samson Zhou	Thomas Schibler	Xingjian Li
Satyabrata Jana	Théophile Thiery	Xue Chen
Sayan Bandyapadhyay	Tianyi Zhang	Yanjia Li
Serge Gaspers	Till Miltzow	Yanyi Liu
Shaofeng Jiang	Tom van der Zanden	Yao-Ting Lin
Shreya Nasa	Tomasz Kociumaka	Yaqiao Li
Shubhang Kulkarni	Tomer Grossman	Yassine Hamoudi
Simon Perdrix	Tore Koß	Yota Otachi
Simon Weber	Tsz Chiu Kwok	Young-San Lin
Sophie Huiberts	Ugo Giocanti	Youssouf Oualhadj
Sophie Laplante	Umang Mathur	Yu Chen
Sriram Bhyravarapu	Ursula Hebert-Johnson	Yuichi Yoshida
Sriram P. Chockalingam	Vaishali Surianarayanan	Yusuke Kobayashi
Stanislav Živný	Vasilis Livanos	Yvo Meeres
Stefan Kiefer	Vedat Levi Alev	Zipei Nie
Stefan Neumann	Vera Traub	

■ List of Authors

Tara Abrishami (4)
Department of Mathematics,
University of Hamburg, Germany

C. Aiswarya (5)
Chennai Mathematical Institute, India;
CNRS, ReLaX, IRL 2000, Chennai, India

V. Arvind (6)
The Institute of Mathematical Sciences (HBNI),
Chennai, India;
Chennai Mathematical Institute, India

Jakub Balabán (7)
Faculty of Informatics, Masaryk University,
Brno, Czech Republic

Max Bannach (8)
European Space Agency, Advanced Concepts
Team, Noordwijk, The Netherlands

Omkar Baraskar (9)
Indian Institute of Science, Bengaluru, India

Harsh Beohar (10)
University of Sheffield, UK

Christoph Berkholz (11, 12)
Technische Universität Ilmenau, Germany

Stéphane Bessy (13)
LIRMM, Univ Montpellier, CNRS,
Montpellier, France

Marcin Bienkowski (14, 15)
University of Wrocław, Poland

Philip Bille (16)
Technical University of Denmark,
Lyngby, Denmark

Nicolás Bitar (17)
Université Paris-Saclay, CNRS, LISN,
91190 Gif-sur-Yvette, France

Olivier Bournez (20)
École polytechnique, LIX, Paris, France

Nicolas Bousquet (21)
Univ Lyon, CNRS, INSA Lyon, UCBL, LIRIS,
UMR5205 F-69622 Villeurbanne, France

Hans-Joachim Böckenhauer (18)
Department of Computer Science,
ETH Zürich, Switzerland

Jan Böker (19)
RWTH Aachen University, Germany

Geoffroy Caillat-Grenier (22)
LIRMM, University of Montpellier, CNRS,
Montpellier, France

Deeparnab Chakrabarty (23)
Department of Computer Science,
Dartmouth College, Hanover, NH, USA

Panagiotis Charalampopoulos (24)
Birkbeck, University of London, UK

Shiteng Chen (49)
State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of
Sciences, Beijing, China;
University of Chinese Academy of Sciences,
Beijing, China

Tameem Choudhury (25)
Department of Computer Science and
Engineering, IIT Hyderabad, India

Maria Chudnovsky (4)
Princeton University, NJ, USA

Luc Cote (23)
Thayer School of Engineering,
Dartmouth College, Hanover, NH, USA

Nicola Cotumaccio (26)
Gran Sasso Science Institute, L'Aquila, Italy;
Dalhousie University, Halifax, Canada

Julian D'Costa (27)
Department of Computer Science,
University of Oxford, UK

Agrim Dewan (9)
Indian Institute of Science, Bengaluru, India

Jack Dippel (28)
McGill University, Montreal, Canada

Andrei Draghici (29)
Department of Computer Science,
University of Oxford, UK

Pranjal Dutta (30, 31)
School of Computing, National University of
Singapore (NUS), Singapore

Franziska Eberle (32)
Technische Universität Berlin, Germany

Nicolas El Maalouly (33)
Department of Computer Science,
ETH Zurich, Switzerland

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).
Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and
Daniel Lokshтанov
















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



- Guy Even  (15)
Tel Aviv University, Israel
- Laurent Feuilloley  (21)
Univ Lyon, CNRS, INSA Lyon, UCBL, LIRIS,
UMR5205, F-69622 Villeurbanne, France
- Marek Filakovský  (34)
Masaryk University, Brno, Czech Republic
- Jonas Forster  (10)
Friedrich-Alexander-Universität Erlangen-,
Nürnberg, Germany
- Fabian Frei  (18)
CISPA Helmholtz Center for Information
Security, Saarbrücken, Germany
- Janosch Fuchs  (35)
RWTH Aachen University, Germany
- Frank Fuhlbrück  (6)
Institut für Informatik, Humboldt-Universität
zu Berlin, Germany
- Moses Ganardi  (36)
Max Planck Institute for Software Systems
(MPI-SWS), Kaiserslautern, Germany
- Robert Ganian  (7)
Algorithms and Complexity Group,
TU Wien, Austria
- Matthias Gehnen  (37)
RWTH Aachen University, Germany
- Fulvio Gesmundo  (30)
Institut de Mathématiques de Toulouse,
Université Paul Sabatier, Toulouse, France
- Riccardo Gozzi  (20)
École polytechnique, LIX, Paris, France;
University Paris-Est Créteil Val de Marne,
LACL, Paris, France
- Nathan Grosshans  (38)
Independent Scholar, Paris Region, France
- Ziyi Guan (39)
EPFL, Lausanne, Switzerland
- Sebastian Gurke  (10)
Universität Duisburg-Essen, Germany
- Stefan Göller (38)
School of Electrical Engineering and Computer
Science, Universität Kassel, Germany
- Inge Li Gørtz  (16)
Technical University of Denmark,
Lyngby, Denmark
- Christoph Haase  (29)
Department of Computer Science,
University of Oxford, UK
- David G. Harris  (40)
Department of Computer Science,
University of Maryland, College Park, MD, USA
- Sebastian Haslebacher  (33)
Department of Computer Science,
ETH Zurich, Switzerland
- S. Hitarth  (41)
Hong Kong University of Science and
Technology, Hong Kong
- Martin Hofer  (42)
Goethe University Frankfurt, Germany
- Yunqi Huang (39)
University of Technology Sydney, Australia
- Louis Härtel  (19)
RWTH Aachen University, Germany
- Christian Ikenmeyer  (30, 31)
University of Warwick, UK
- Jesper Jansson (43)
Kyoto University, Japan
- Gorav Jindal  (30)
Max Planck Institute for Software Systems,
Saarbrücken, Germany
- Lawqueen Kanesh  (44)
Indian Institute of Technology Jodhpur, India
- Piotr Kawalek  (45)
Institute of Discrete Mathematics and Geometry,
Vienna University of Technology, Austria;
Institute of Computer Science, University of
Maria Curie-Skłodowska, Lublin, Poland
- George Kenison  (41)
Liverpool John Moores University, UK
- Balagopal Komarath (31)
Indian Institute of Technology Gandhinagar,
India
- Michael Kompatscher  (45, 46)
Department of Algebra, Faculty of Mathematics
and Physics, Charles University, Prague, Czech
Republic
- Laura Kovács  (41)
TU Wien, Austria

- Katarzyna Kowalska (47)
Faculty of Mathematics, Informatics, and
Mechanics, University of Warsaw, Poland
- Jacek Krzaczkowski (45)
Institute of Computer Science, University of
Maria Curie-Skłodowska, Lublin, Poland
- Dietrich Kuske (12)
Technische Universität Ilmenau, Germany
- Johannes Köbler (6)
Institut für Informatik, Humboldt-Universität
zu Berlin, Germany
- Barbara König (10)
Universität Duisburg-Essen, Germany
- Matthias Lanzinger (48)
TU Wien, Austria; University of Oxford, UK
- Moshe Lewenstein (16)
Bar-Ilan University, Ramat-Gan, Israel
- Ying Liu (49)
State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of
Sciences, Beijing, China;
University of Chinese Academy of Sciences,
Beijing, China
- Henri Lotze (37)
RWTH Aachen University, Germany
- Vladimir Lysikov (30)
Ruhr-Universität Bochum, Germany
- James C. A. Main (50)
F.R.S.-FNRS & UMONS – Université de Mons,
Belgium
- Soumodev Mal (5)
Chennai Mathematical Institute, India
- Andreas Maletti (51)
Institute of Computer Science,
Leipzig University, Germany
- Florin Manea (29)
Computer Science Department and
Campus-Institut Data Science,
Göttingen University, Germany
- Bodo Manthey (52)
Faculty of Electrical Engineering, Mathematics,
and Computer Science, University of Twente,
Enschede, The Netherlands
- Florian Andreas Marwitz (8)
Institute of Information Systems,
Universität zu Lübeck, Germany
- Stefan Mengel (11)
Univ. Artois, CNRS, Centre de Recherche en
Informatique de Lens (CRIL), France
- Karla Messing (10)
Universität Duisburg-Essen, Germany
- Harshil Mittal (31)
Indian Institute of Technology Gandhinagar,
India
- Tamio-Vesa Nakajima (34)
University of Oxford, UK
- Saraswati Girish Nanoti (31)
Indian Institute of Technology Gandhinagar,
India
- N. S. Narayanaswamy (40)
Department of Computer Science and
Engineering, Indian Institute of Technology
Madras, India
- Daniel Neuen (53)
University of Bremen, Germany
- Andreea-Teodora Nász (51)
Institute of Computer Science,
Leipzig University, Germany
- Pierre Ohlmann (54)
Institute of Informatics,
University of Warsaw, Poland
- Igor C. Oliveira (1)
University of Warwick, UK
- Jakub Opršal (34)
University of Birmingham, UK
- Joël Ouaknine (27)
Max Planck Institute for Software Systems,
Saarland Informatics Campus, Saarbrücken,
Germany
- Fahad Panolan (44)
School of Computing, University of Leeds, UK
- Christophe Paul (55)
CNRS, Université Montpellier, France
- Erik Paul (51)
Institute of Computer Science,
Leipzig University, Germany
- Marcin Pilipczuk (4)
Institute of Informatics, Faculty of Mathematics,
Informatics and Mechanics, University of
Warsaw, Poland

- Michał Pilipczuk (47)
Institute of Informatics,
University of Warsaw, Poland
- Solon P. Pissis (16, 24)
CWI, Amsterdam, The Netherlands;
Vrije Universiteit, Amsterdam, The Netherlands
- Evangelos Protopapas (55)
CNRS, Université Montpellier, France
- Jakub Radoszewski (24)
University of Warsaw, Poland
- M. S. Ramanujan (44)
University of Warwick, UK
- Sofya Raskhodnikova (2)
Department of Computer Science,
Boston University, MA, USA
- Igor Razgon (48)
Birkbeck, University of London, UK
- Mathis Rocton (7)
Algorithms and Complexity Group,
TU Wien, Austria
- Andrei Romashchenko (22)
LIRMM, University of Montpellier, CNRS,
Montpellier, France
- Peter Rossmanith (18, 37)
Department of Computer Science,
RWTH Aachen, Germany
- Eva Rotenberg (16)
Technical University of Denmark,
Lyngby, Denmark
- Nina Runde (19)
RWTH Aachen University, Germany
- Wojciech Rytter (24)
University of Warsaw, Poland
- Paweł Rzążewski (4)
Faculty of Mathematics and Information Science,
Warsaw University of Technology, Poland;
Institute of Informatics, Faculty of Mathematics,
Informatics and Mechanics, University of
Warsaw, Poland
- Chandan Saha (9)
Indian Institute of Science, Bengaluru, India
- Prakash Saivasan (5)
The Institute of Mathematical Sciences, HBNI,
Chennai, India;
CNRS, ReLaX, IRL 2000, Chennai, India
- Swagato Sanyal (56)
Department of Computer Science and
Engineering, Indian Institute of Technology
Kharagpur, India
- Ankita Sarkar (23)
Department of Computer Science,
Dartmouth College, Hanover, NH, USA
- Irmak Sağlam (36)
Max Planck Institute for Software Systems
(MPI-SWS), Kaiserslautern, Germany
- Ildikó Schlotter (57)
Centre for Economic and Regional Studies,
Budapest, Hungary;
Budapest University of Technology and
Economics, Hungary
- Stefan Schmid (14)
TU Berlin, Germany;
Weizenbaum Institute, Berlin, Germany
- Lutz Schröder (10)
Friedrich-Alexander-Universität
Erlangen-Nürnberg, Germany
- Christian Schwarz (12)
Technische Universität Ilmenau, Germany
- Tim Seppelt (19)
RWTH Aachen University, Germany
- Michał Skrzypczak (54)
Institute of Informatics,
University of Warsaw, Poland
- Kartteek Sreenivasaiiah (25)
Department of Computer Science and
Engineering, IIT Hyderabad, India
- Christoph Standke (19)
RWTH Aachen University, Germany
- Teresa Anna Steiner (16)
Technical University of Denmark,
Lyngby, Denmark
- Peter Strulo (44)
University of Warwick, UK
- Wing-Kin Sung (43)
The Chinese University of Hong Kong, China;
Hong Kong Genome Institute, Hong Kong
Science Park, China
- Syed Ali Tabatabaee (43)
Dept. of Computer Science, University of British
Columbia, Vancouver, Canada

- Till Tantau  (8)
Institute for Theoretical Computer Science,
Universität zu Lübeck, Germany
- Gianluca Tasinato  (34)
ISTA, Klosterneuburg, Austria
- Dhara Thakkar (31)
Indian Institute of Technology Gandhinagar,
India
- Stéphan Thomassé (13)
Univ Lyon, EnsL, UCBL, CNRS, LIP,
F-69342, LYON Cedex 07, France
- Szymon Toruńczyk  (3)
Institute of Informatics,
University of Warsaw, Poland
- Jesse van Rhijn  (52)
Faculty of Electrical Engineering, Mathematics,
and Computer Science, University of Twente,
Enschede, The Netherlands
- Anton Varonka  (41)
TU Wien, Austria
- Carmine Ventre  (42)
King's College London, UK
- Oleg Verbitsky  (6)
Institut für Informatik, Humboldt-Universität
zu Berlin, Germany
- Adrian Vetta  (28)
McGill University, Montreal, Canada
- Laurent Viennot (13)
Inria Paris, DI ENS, Paris, France
- Uli Wagner  (34)
ISTA, Klosterneuburg, Austria
- Tomasz Waleń  (24)
University of Warsaw, Poland
- Haitao Wang  (58)
Kahlert School of Computing,
University of Utah, Salt Lake City, UT, USA
- Philip Whittington  (35)
ETH Zürich, Switzerland
- Paul Wild  (10)
Friedrich-Alexander-Universität
Erlangen-Nürnberg, Germany
- Hermann Wilhelm  (11)
Technische Universität Ilmenau, Germany
- Lisa Wilhelmi  (42)
Goethe University Frankfurt, Germany
- James Worrell  (27)
Department of Computer Science,
University of Oxford, UK
- Lasse Wulf  (33)
Institute of Discrete Mathematics,
TU Graz, Austria
- Yutong Yang (43)
Hong Kong Genome Institute,
Hong Kong Science Park, China
- Penghui Yao (39)
State Key Laboratory for Novel Software
Technology, Nanjing University, China;
Hefei National Laboratory, China
- Zekun Ye (39)
State Key Laboratory for Novel Software
Technology, Nanjing University, China
- Emre Yolcu  (59)
Carnegie Mellon University,
Pittsburgh, PA, USA
- Sébastien Zeitoun  (21)
Univ Lyon, CNRS, INSA Lyon, UCBL, LIRIS,
UMR5205, F-69622 Villeurbanne, France
- Georg Zetsche  (36)
Max Planck Institute for Software Systems
(MPI-SWS), Kaiserslautern, Germany
- Wiktor Zuba  (24)
CWI, Amsterdam, The Netherlands

Polynomial-Time Pseudodeterministic Constructions

Igor C. Oliveira   

University of Warwick, UK

Abstract

A randomised algorithm for a search problem is pseudodeterministic if it produces a fixed canonical solution to the search problem with high probability. In their seminal work on the topic, Gat and Goldwasser (2011) posed as their main open problem whether prime numbers can be pseudodeterministically constructed in polynomial time.

We provide a positive solution to this question in the infinitely-often regime. In more detail, we give an unconditional polynomial-time randomised algorithm B such that, for infinitely many values of n , $B(1^n)$ outputs a canonical n -bit prime p_n with high probability. More generally, we prove that for every dense property Q of strings that can be decided in polynomial time, there is an infinitely-often pseudodeterministic polynomial-time construction of strings satisfying Q . This improves upon a subexponential-time pseudodeterministic construction of Oliveira and Santhanam (2017).

This talk will cover the main ideas behind these constructions and discuss their implications, such as the existence of infinitely many primes with succinct and efficient representations.

2012 ACM Subject Classification Theory of computation \rightarrow Computational complexity and cryptography; Theory of computation \rightarrow Pseudorandomness and derandomization

Keywords and phrases Pseudorandomness, Explicit Constructions, Pseudodeterministic Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.1

Category Invited Talk



© Igor C. Oliveira;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 1; pp. 1:1–1:1



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



The Role of Local Algorithms in Privacy

Sofya Raskhodnikova   

Department of Computer Science, Boston University, MA, USA

Abstract

We will discuss research areas at the intersection of local algorithms and differential privacy. The main focus will be on using local Lipschitz filters to enable black-box differentially private queries to sensitive datasets. We will also cover new sublinear computational tasks arising in private data analysis. Finally, we will touch upon distributed models of privacy.

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

Keywords and phrases Sublinear algorithms, differential privacy, reconstruction of Lipschitz functions, local algorithms

Digital Object Identifier 10.4230/LIPICs.STACS.2024.2

Category Invited Talk

1 Summary

This talk will be mostly based on joint work with Madhav Jha [2], Jane Lange, Ephraim Linder, and Arsen Vasilyan [3], Satchit Sivakumar, Adam Smith, and Marika Swanberg [4], and Talya Eden, Quanquan Liu, and Adam Smith [1].

References

- 1 Talya Eden, Quanquan C. Liu, Sofya Raskhodnikova, and Adam D. Smith. Triangle counting with local edge differential privacy. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 52:1–52:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ICALP.2023.52.
- 2 Madhav Jha and Sofya Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. *SIAM J. Comput.*, 42(2):700–731, 2013. doi:10.1137/110840741.
- 3 Jane Lange, Ephraim Linder, Sofya Raskhodnikova, and Arsen Vasilyan. Local Lipschitz filters for bounded-range functions. *CoRR*, abs/2308.14716, 2023. doi:10.48550/ARXIV.2308.14716.
- 4 Sofya Raskhodnikova, Satchit Sivakumar, Adam D. Smith, and Marika Swanberg. Differentially private sampling from distributions. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 28983–28994, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/f2b5e92f61b6de923b063588ee6e7c48-Abstract.html>.



© Sofya Raskhodnikova;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 2; pp. 2:1–2:1



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Structurally Tractable Graph Classes

Szymon Toruńczyk  

Institute of Informatics, University of Warsaw, Poland

Abstract

Sparsity theory, initiated by Ossona de Mendez and Nešetřil, identifies those classes of sparse graphs that are tractable in various ways – algorithmically, combinatorially, and logically – as exactly the *nowhere dense* classes. An ongoing effort aims at generalizing sparsity theory to classes of graphs that are not necessarily sparse. *Twin-width* theory, developed by Bonnet, Thomassé and co-authors, is a step in that direction. A theory unifying the two is anticipated. It is conjectured that the relevant notion characterising dense graph classes that are tractable, generalising nowhere denseness and bounded twin-width, is the notion of a *monadically dependent class*, introduced by Shelah in model theory. I will survey the recent, rapid progress in the understanding of those classes, and of the related monadically stable classes. This development combines tools from structural graph theory, logic (finite and infinite model theory), and algorithms (parameterised algorithms and range search queries).

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Complexity theory and logic; Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

Keywords and phrases Structural graph theory, Monadic dependence, monadic NIP, twin-width

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.3

Category Invited Talk



© Szymon Toruńczyk;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 3; pp. 3:1–3:1



Leibniz International Proceedings in Informatics


LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany




Max Weight Independent Set in Sparse Graphs with No Long Claws

Tara Abrishami ✉


Department of Mathematics, University of Hamburg, Germany

Maria Chudnovsky ✉ 

Princeton University, NJ, USA

Marcin Pilipczuk ✉ 

Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Paweł Rzażewski ✉ 

Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Abstract

We revisit the recent polynomial-time algorithm for the MAX WEIGHT INDEPENDENT SET (MWIS) problem in bounded-degree graphs that do not contain a fixed graph whose every component is a subdivided claw as an induced subgraph [Abrishami, Chudnovsky, Dibek, Rzażewski, SODA 2022].

First, we show that with an arguably simpler approach we can obtain a faster algorithm with running time $n^{\mathcal{O}(\Delta^2)}$, where n is the number of vertices of the instance and Δ is the maximum degree. Then we combine our technique with known results concerning tree decompositions and provide a polynomial-time algorithm for MWIS in graphs excluding a fixed graph whose every component is a subdivided claw as an induced subgraph, and a fixed biclique as a subgraph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms

Keywords and phrases Max Weight Independent Set, subdivided claw, hereditary classes

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.4

Related Version *Extended Version*: <https://arxiv.org/abs/2309.16995> [2]

Funding *Tara Abrishami*: Supported by the National Science Foundation Award Number DMS-2303251 and the Alexander von Humboldt Foundation.

Maria Chudnovsky: Supported by NSF-EPSRC Grant DMS-2120644 and by AFOSR grant FA9550-22-1-008.

Marcin Pilipczuk: Supported by Polish National Science Centre SONATA BIS-12 grant number 2022/46/E/ST6/00143.

Paweł Rzażewski: This work is a part of the project BOBR that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 948057).

Acknowledgements We acknowledge the welcoming and productive atmosphere at Dagstuhl Seminar 22481 “Vertex Partitioning in Graphs: From Structure to Algorithms,” where a crucial part of the work leading to the results in this paper was done.



© Tara Abrishami, Maria Chudnovsky, Marcin Pilipczuk, and Paweł Rzażewski; licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov; Article No. 4; pp. 4:1–4:15



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

A *vertex-weighted graph* is an undirected graph G equipped with a weight function $\mathbf{w} : V(G) \rightarrow \mathbb{N}$. For $X \subseteq V(G)$, we use $\mathbf{w}(X)$ as a shorthand for $\sum_{x \in X} \mathbf{w}(x)$ and for a subgraph H of G , $\mathbf{w}(H)$ is a shorthand for $\mathbf{w}(V(H))$. By convention we use $\mathbf{w}(\emptyset) = 0$. Throughout the paper we assume that arithmetic operations on weights are performed in unit time.

For a graph G , a set $I \subseteq V(G)$ is *independent* or *stable* if there is no edge of G with both endpoints in I . In the MAX WEIGHT INDEPENDENT SET (MWIS) problem we are given an undirected vertex-weighted graph (G, \mathbf{w}) , and ask for a maximum-weight independent set in (G, \mathbf{w}) . MWIS is one of canonical hard problems – it is NP-hard [23], W[1]-hard [15], hard to approximate [22]. Thus a very natural research direction is to consider restricted instances and try to capture the boundary between “easy” and “hard” cases.

State of the art. The study of complexity of MWIS in restricted graph classes is a central topic in algorithmic graph theory [20, 45, 27, 19, 14, 4]. A particular attention is given to classes that are *hereditary*, i.e., closed under vertex deletion. Among such classes a special role is played by ones defined by forbidding certain substructures. For graphs G and H , we say that G is *H-free* if it does not contain H as an *induced subgraph*.

In this paper we are interested in graph classes defined by forbidding certain induced trees. Let \mathcal{S} be the family of subcubic trees H with at most one vertex of degree 3. In other words, every such H is either a path or a subdivision of the *claw*: the three-leaf star. For integers $a, b, c \geq 1$, by $S_{a,b,c}$ we denote the claw whose edges were subdivided, respectively, $a - 1$, $b - 1$, and $c - 1$ times.

As already observed by Alekseev in the early 1980s [3], if H is connected, MWIS remains NP-hard in H -free graphs unless $H \in \mathcal{S}$. In the past few years we have witnessed rapid progress in development of algorithms for MWIS in these remaining cases [6, 10, 16, 42, 13, 17, 35]. In particular, it is known that for each $H \in \mathcal{S}$, the MWIS problem can be solved in quasipolynomial time in H -free graphs [16, 42, 17]. This is a strong indication that the problem is not NP-hard. However, we are still very far from obtaining polynomial-time algorithms. Such results are known only for very small forbidden paths [27, 19], subdivided claws [36, 43, 5, 28], or their disjoint unions [31, 7]. There is also a long line of research concerning graphs excluding a fixed (but still small) path or a subdivided claw and, simultaneously, some other small graphs, see e.g. [26, 8, 18, 32, 34, 38, 29, 30, 21, 37, 39, 40, 41, 9].

In a somewhat perpendicular direction, Abrishami, Chudnovsky, Dibek, and Rzażewski [1] proved that for every $H \in \mathcal{S}$, the MWIS problem can be solved in polynomial time in H -free graphs of bounded degree (where the degree of the polynomial depends on H , and the degree bound). Let us remark that the algorithm of [1] is very technical, and the dependency of the complexity on the degree bound is involved.

Our results. As a warm-up, we present a polynomial-time algorithm for H -free graphs of bounded degree, where $H \in \mathcal{S}$. It is significantly simpler than the one by Abrishami et al. [1] and has much better dependency on the maximum degree.

► **Theorem 1.** *There exists an algorithm that, given a vertex-weighted graph (G, \mathbf{w}) on n vertices with maximum degree Δ and an integer t , in time $n^{\mathcal{O}(t\Delta^2)}$ either finds an induced $S_{t,t,t}$ or the maximum possible weight of an independent set in (G, \mathbf{w}) .*

Note that by picking appropriate t , Theorem 1 yields a polynomial-time algorithm for MWIS for bounded-degree graphs excluding a fixed graph from \mathcal{S} as an induced subgraph.

Then we proceed to the main result of the paper: we show that MWIS remains polynomial-time solvable in $S_{t,t,t}$ -free graphs, even if instead of bounding the maximum degree, we forbid a fixed biclique as a subgraph.

► **Theorem 2.** *For every fixed integer t , and s there exists a polynomial-time algorithm that, given a vertex-weighted graph (G, \mathbf{w}) that does not contain $S_{t,t,t}$ as an induced subgraph nor $K_{s,s}$ as a subgraph, returns the maximum possible weight of an independent set in (G, \mathbf{w}) .*

Let us remark that by the celebrated Kővári-Sós-Turán theorem [25], classes that exclude $K_{s,s}$ as a subgraph capture all hereditary classes of *sparse graphs*, where by “sparse” we mean “where every graph has subquadratic number of edges.” Furthermore, by a simple Ramsey argument, for every positive integer r there exists an integer s such that if G is K_r -free and $K_{r,r}$ -free then G does not contain $K_{s,s}$ as a subgraph. Hence, equivalently, Theorem 2 yields a polynomial-time algorithm for MWIS for graphs that are simultaneously H -free (for some $H \in \mathcal{S}$), K_r -free, and $K_{r,r}$ -free.

Our techniques. As in the previous works [1, 13, 17], the crucial tool in handling $S_{t,t,t}$ -free graphs is an *extended strip decomposition*. Its technical definition can be found in preliminaries; for now, it suffices to say that it is a wide generalization of the preimage graph of a line graph (recall that line graphs are $S_{1,1,1}$ -free) that allows for recursion for the MWIS problem: An extended strip decomposition of a graph G identifies some induced subgraphs of G as *particles* and, knowing the maximum possible weight of an independent set in each particle, one can compute in polynomial time the maximum possible weight of an independent set in G . (We remark that this computation involves advanced combinatorial techniques as it relies on a reduction to the maximum weight matching problem in an auxiliary graph.) In other words, finding an extended strip decomposition with small particles compared to $|V(G)|$ is equally good for the MWIS problem as splitting the graph into small connected components.

The starting point is the following theorem of [35].

► **Theorem 3** ([35, Corollary 12] in a semi-weighted setting). *There exists an algorithm that, given an n -vertex graph G with a set $U \subseteq V(G)$ and an integer t , in polynomial time outputs either:*

- *an induced copy of $S_{t,t,t}$ in G , or*
- *a set X of size at most $(11 \log n + 6)(t + 1)$ and a rigid extended strip decomposition of $G - N[X]$ with every particle containing at most $|U|/2$ vertices of U .*

(A rigid extended strip decomposition is an extended strip decomposition that does not have some unnecessary empty sets. By $N[X]$ we denote the set consisting of X and all vertices with a neighbor in X .) Let us remark that the result stated in [35, Theorem 2] is for unweighted graphs (i.e., $U = V(G)$ using the notation from Theorem 3), but the statement of Theorem 3 can be easily derived from the proof, see also [17].

Consider the setting of Theorem 1, i.e., the graph G has maximum degree Δ . Apply Theorem 3 to G with $U = V(G)$. If we get the first outcome, i.e., an induced $S_{t,t,t}$ in G , we return it and terminate. So assume that we get the second outcome, i.e., the set X . Note that as $|X| = \mathcal{O}(t \log n)$, we have $|N[X]| = \mathcal{O}(t\Delta \log n)$. It is now tempting to exhaustively branch on $N[X]$ (i.e., guess the intersection of the sought independent set with $N[X]$) and recurse on the particles of the extended strip decomposition of $G - N[X]$. However, implementing this strategy directly gives quasipolynomial (in n) running time bound of $n^{\mathcal{O}(t\Delta \log n)}$, as the branching step yields up to $2^{|N[X]|} = n^{\mathcal{O}(t\Delta)}$ subcases and the depth of the recursion is $\mathcal{O}(\log n)$.

4:4 Max Weight Independent Set in Sparse Graphs with No Long Claws

Our main new idea now is to perform this branching lazily, by considering a more general *border* version of the problem, where the input graph is additionally equipped with a set of *terminals* and we ask for a maximum weight of an independent set for every possible behavior on the terminals.

BORDER MWIS

Input: A vertex-weighted graph (G, \mathbf{w}) with a set $T \subseteq V(G)$ of *terminals*.
Task: Compute $f_{G, \mathbf{w}, T} : 2^T \rightarrow \mathbb{N} \cup \{-\infty\}$ defined for every $I_T \subseteq T$ as
 $f_{G, \mathbf{w}, T}(I_T) = \max\{\mathbf{w}(I) \mid I \subseteq V(G) \text{ and } I \text{ is independent, and } I \cap T = I_T\}$.

A similar application of a border version of the problem to postpone branching in recursion appeared for example in the technique of recursive understanding [24, 11].

Let us return to our setting, where we have a set X of size $\mathcal{O}(t \log n)$ and an extended strip decomposition of $G - N[X]$ with particles of size at most half of the size of $V(G)$. We would like to remove $N[X]$ from the graph, indicate $N(N[X])$ as terminals and solve BORDER MWIS in $(G - N[X], \mathbf{w}, T := N(N[X]))$ using the extended strip decomposition for recursion. Note that, thanks to the bounded degree assumption, the size of $T = N(N[X])$ is bounded by $\mathcal{O}(t\Delta^2 \log n)$.

This approach *almost* works: the only problem is that, as the recursion progresses, the set of terminals accumulates and its size can grow beyond the initial $\mathcal{O}(t\Delta^2 \log n)$ bound. Luckily, this can be remedied in a standard way: we alternate recursive steps where Theorem 3 is invoked with $U = V(G)$ with steps where Theorem 3 is invoked with $U = T$. In this manner, we can maintain a bound of $\mathcal{O}(t\Delta^2 \log n)$ on the number of terminals in every recursive call. Note that this bound also guarantees that the size of the domain of the requested function $f_{G, \mathbf{w}, T}$ is of size $n^{\mathcal{O}(t\Delta^2)}$, which is within the promised time bound.

We remark that this approach is arguably significantly simpler and more direct than the decomposition techniques used in [1] and, furthermore, results in a much better dependency on Δ in the exponent in the final running time bound.

Let us now move to the more general setting of Theorem 2. Here, the starting point are recent results of Weißauer [44] and Lozin and Razgon [33] that show that in the $S_{t,t,t}$ -free case, excluding a biclique as a subgraph is not that much different than bounding the maximum degree.

A k -block in a graph is a set of k vertices, no two of which can be separated by deleting fewer than k vertices. The following result was shown by Weißauer (we refer to preliminaries for standard definitions of tree decompositions and torsos).

► **Theorem 4** (Weißauer [44]). *Let G be a graph and $k \geq 2$ be an integer. If G has no $(k+1)$ -block, then G admits a tree decomposition with adhesion less than k , in which every torso has at most k vertices of degree larger than $2k(k-1)$.*

Even though the statement of the result in [44] is just existential, the proof actually yields a polynomial-time algorithm to compute such a tree decomposition.

It turns out that $S_{t,t,t}$ -free graphs with no large bicliques have no large blocks.

► **Lemma 5** (Lozin and Razgon [33]). *For any t and s there exists k such that the following holds. Every $S_{t,t,t}$ -free graph with no subgraph isomorphic to $K_{s,s}$ has no k -block.*

Combining Theorem 4 and Lemma 5 we immediately obtain the following.

► **Corollary 6.** *For any t , and s there exists k such that the following holds. Given an $S_{t,t,t}$ -free graph with no subgraph isomorphic to $K_{s,s}$, in polynomial time one can compute a tree decomposition with adhesion less than k , in which every torso has at most k vertices of degree larger than $2k(k-1)$.*

To prove Theorem 2 using Corollary 6 we need to carefully combine explicit branching on the (bounded number of) vertices of large degree in a single bag with – as in the bounded degree case – applying Theorem 3 to the remainder of the graph and indicating $N(N[X])$ as the terminal set of the border problem passed to the recursive calls. Finally, one requires some care to combine this with the information passed over adhesions in the tree decomposition.

2 Preliminaries

Our algorithms take a vertex-weighted graph (G, \mathbf{w}) as an input. In the recursion, we will be working on various induced subgraphs of G with vertex weight inherited from \mathbf{w} . Somewhat abusing notation, we will keep \mathbf{w} for the weight function in any induced subgraph of G .

Tree decompositions. Let G be a graph. A *tree decomposition* of G is a pair (\mathcal{T}, β) where \mathcal{T} is a tree and $\beta : V(\mathcal{T}) \rightarrow 2^{V(G)}$ is a function satisfying the following: (i) for every $uv \in E(G)$ there exists $t \in V(\mathcal{T})$ with $u, v \in \beta(t)$, and (ii) for every $v \in V(G)$ the set $\{t \in V(\mathcal{T}) \mid v \in \beta(t)\}$ induces a connected nonempty subtree of \mathcal{T} . For every $t \in V(\mathcal{T})$ and $st \in E(\mathcal{T})$, the set $\beta(t)$ is the *bag* at node t and the set $\sigma(st) := \beta(s) \cap \beta(t)$ is the *adhesion* at edge st . The critical property of a tree decomposition (\mathcal{T}, β) is that if $st \in E(\mathcal{T})$ and V_s and V_t are two connected components of $\mathcal{T} - \{st\}$ that contain s and t , respectively, then $\sigma(st)$ separates $\bigcup_{x \in V_s} \beta(x) \setminus \sigma(st)$ from $\bigcup_{x \in V_t} \beta(x) \setminus \sigma(st)$ in G .

The *torso* of a bag $\beta(t)$ in a tree decomposition (\mathcal{T}, β) is a graph H with $V(H) = \beta(t)$ and $uv \in E(H)$ if $uv \in E(G)$ or there exists a neighbor $s \in N_{\mathcal{T}}(t)$ with $u, v \in \sigma(st)$. That is, the torso of $\beta(t)$ is created from $G[\beta(t)]$ by turning the adhesion $\sigma(st)$ into a clique for every neighbor s of t in \mathcal{T} .

Extended strip decompositions. We follow the notation of [35, 17]. For a graph H , by $T(H)$ we denote the set of triangles in H . An *extended strip decomposition* of a graph G is a pair (H, η) that consists of:

- a simple graph H ,
- a *vertex set* $\eta(x) \subseteq V(G)$ for every $x \in V(H)$,
- an *edge set* $\eta(xy) \subseteq V(G)$ for every $xy \in E(H)$, and its subsets $\eta(xy, x), \eta(xy, y) \subseteq \eta(xy)$,
- a *triangle set* $\eta(xyz) \subseteq V(G)$ for every $xyz \in T(H)$,

which satisfy the following properties:

1. The family $\{\eta(o) \mid o \in V(H) \cup E(H) \cup T(H)\}$ is a partition of $V(G)$.
2. For every $x \in V(H)$ and every distinct $y, z \in N_H(x)$, the set $\eta(xy, x)$ is complete to $\eta(xz, x)$.
3. Every $uv \in E(G)$ is contained in one of the sets $\eta(o)$ for $o \in V(H) \cup E(H) \cup T(H)$, or is as follows:
 - $u \in \eta(xy, x), v \in \eta(xz, x)$ for some $x \in V(H)$ and $y, z \in N_H(x)$, or
 - $u \in \eta(xy, x), v \in \eta(x)$ for some $xy \in E(H)$, or
 - $u \in \eta(xyz)$ and $v \in \eta(xy, x) \cap \eta(xy, y)$ for some $xyz \in T(H)$.

4:6 Max Weight Independent Set in Sparse Graphs with No Long Claws

An extended strip decomposition (H, η) is *rigid* if for every $xy \in E(H)$, the sets $\eta(xy)$, $\eta(xy, x)$, and $\eta(xy, y)$ are nonempty, and for every isolated $x \in V(H)$, the set $\eta(x)$ is nonempty. Note that if (H, η) is a rigid extended strip decomposition of G , then $|V(H)|$ is bounded by $|V(G)|$.

For an extended strip decomposition (H, η) of a graph G , we identify five *types of particles*.

- vertex particle: $A_x := \eta(x)$ for each $x \in V(H)$,
- edge interior particle: $A_{xy}^\perp := \eta(xy) \setminus (\eta(xy, x) \cup \eta(xy, y))$ for each $xy \in E(H)$,
- half-edge particle: $A_{xy}^x := \eta(x) \cup \eta(xy) \setminus \eta(xy, y)$ for each $xy \in E(H)$,
- full edge particle: $A_{xy}^{xy} := \eta(x) \cup \eta(y) \cup \eta(xy) \cup \bigcup_{z : xyz \in T(H)} \eta(xyz)$ for each $xy \in E(H)$,
- triangle particle: $A_{xyz} := \eta(xyz)$ for each $xyz \in T(H)$.

As announced in the introduction, to solve MWIS in G it suffices to know the solution to MWIS in particles. The proof of the following lemma follows closely the lines of proofs of analogous statements of [1, 13] and is included for completeness in Appendix A.

► **Lemma 7.** *Given a BORDER MWIS instance (G, \mathbf{w}, T) , an extended strip decomposition (H, η) of G , and a solution $f_{G[A], \mathbf{w}, T \cap A}$ to the BORDER MWIS instance $(G[A], \mathbf{w}, T \cap A)$ for every particle A of (H, η) , one can in time $2^{|T|}$ times a polynomial in $|V(G)| + |V(H)|$ compute the solution $f_{G, \mathbf{w}, T}$ to the input (G, \mathbf{w}, T) .*

We need the following simple observations.

► **Lemma 8.** *Let G be a K_t -free graph and let (H, η) be a rigid extended strip decomposition of G . Then the maximum degree of H is at most $t - 1$.*

Proof. Let $x \in V(H)$. Observe that the sets $\{\eta(xy, x) \mid y \in N_H(x)\}$ are nonempty and complete to each other in G . Hence, G contains a clique of size equal to the degree of x in H . ◀

► **Lemma 9.** *Let G be a graph and let (H, η) be an extended strip decomposition of G such that the maximum degree of H is at most d . Then, every vertex of G is in at most $\max(4, 2d + 1)$ particles.*

Proof. Pick $v \in V(G)$ and observe that:

- If $v \in \eta(x)$ for some $x \in V(H)$, then v is in the vertex particle of x and in one half-edge and one full-edge particle for every edge of H incident with x . Since there are at most d such edges, v is in at most $2d + 1$ particles.
- If $v \in \eta(xy)$ for some $xy \in E(H)$, then v is in at most four particles for the edge xy .
- If $v \in \eta(xyz)$ for some $xyz \in T(H)$, then v is in the triangle particle for xyz and in three full edge particles, for the three sides of the triangle xyz . ◀

3 Bounded-degree graphs: Proof of Theorem 1

This section is devoted to the proof of Theorem 1.

Let t be a positive integer and let (G, \mathbf{w}) be the input vertex-weighted graph. We denote $n := |V(G)|$ and Δ to be the maximum degree of G . Let

$$\ell := \lceil 11 \log n + 6 \rceil (t + 2) = \mathcal{O}(t \log n)$$

be an upper bound on the size of X for any application of Theorem 3 for any induced subgraph of G .

We describe a recursive algorithm that takes as input an induced subgraph G' of G with weights \mathbf{w} and a set of terminals $T \subseteq V(G')$ of size at most $4\ell\Delta^2$ and solves BORDER MWIS on (G', \mathbf{w}, T) . The root call is for $G' := G$ and $T := \emptyset$; indeed, note that $f_{G, \mathbf{w}, \emptyset}(\emptyset)$ is the maximum possible weight of an independent set in G .

Let (G', \mathbf{w}, T) be an input to a recursive call. First, the algorithm initializes $f_{G', \mathbf{w}, T}(I_T) := -\infty$ for every $I_T \subseteq T$.

If $|V(G')| \leq 4\Delta^2\ell$, the algorithm proceeds by brute-force: it enumerates independent sets $I \subseteq V(G')$ and updates $f_{G', \mathbf{w}, T}(I \cap T)$ with $\mathbf{w}(I)$ whenever the previous value of that cell was smaller. As $\ell = \mathcal{O}(t \log n)$, this step takes $n^{\mathcal{O}(t\Delta^2)}$ time. This completes the description of the leaf step of the recursion.

If $|V(G')| > 4\Delta^2\ell$, the algorithm proceeds as follows. If $|T| \leq 3\Delta^2\ell$, let $U := V(G')$, and otherwise, let $U := T$. The algorithm invokes Theorem 3 on G' and U . If an induced $S_{t,t,t}$ is returned, then it can be returned by the main algorithm as it is in particular an induced subgraph of G . Hence, we can assume that we obtain a set $X \subseteq V(G)$ of size at most ℓ and an extended strip decomposition (H, η) of $G^* := G' - N_{G'}[X]$ whose every particle contains at most $|U|/2$ vertices of U .

Observe that as $|X| \leq \ell$ and the maximum degree of G is Δ , we have $|N_{G'}(N_{G'}[X])| \leq \Delta^2\ell$. Let $T^* := (T \cap V(G^*)) \cup N_{G'}(N_{G'}[X])$. Note that we have $T^* \subseteq V(G^*)$ and $|T^*| \leq 5\Delta^4\ell$. For every particle A of (H, η) , invoke a recursive call on $(G_A^* := G^*[A], \mathbf{w}, T_A^* := T^* \cap A)$, obtaining $f_{G_A^*, \mathbf{w}, T_A^*}$ (or an induced $S_{t,t,t}$, which can be directly returned). Use Lemma 7 to obtain a solution f_{G^*, \mathbf{w}, T^*} to BORDER MWIS instance (G^*, \mathbf{w}, T^*) .

Finally, iterate over every $I_T \subseteq T^* \cup N_{G'}[X]$ (note that $T \subseteq T^* \cup N_{G'}[X]$) and, if I_T is independent, update the cell $f_{G', \mathbf{w}, T}(I_T \cap T)$ with the value $\mathbf{w}(I_T \setminus T^*) + f_{G^*, \mathbf{w}, T^*}(I_T \cap T^*)$, if this value is larger than the previous value of this cell. This completes the description of the algorithm.

The correctness of the algorithm is immediate thanks to Lemma 7 and the fact that $N_{G'}[X]$ is adjacent in G' only to $N_{G'}(N_{G'}[X])$ which is a subset of T^* .

For the complexity analysis, consider a recursive call to $(G_A^*, \mathbf{w}, T_A^*)$ for a particle A . If $|T| \leq 3\Delta^2\ell$, then $|T_A^*| \leq |T^*| \leq 4\Delta^2\ell$. Otherwise, $U = T$ and $|T \cap A| \leq |T|/2 \leq 2\Delta^2\ell$. As $|N_{G'}(N_{G'}[X])| \leq \Delta^2\ell$, we have $|T_A^*| \leq 3\Delta^2\ell$. Hence, in the recursive call the invariant of at most $4\Delta^2\ell$ terminals is maintained and, moreover:

- if $|T| \leq 3\Delta^2\ell$, then $U = |V(G')|$ and $|V(G_A^*)| = |A| \leq |V(G')|/2$;
- otherwise, $V(G_A^*) \subseteq V(G')$ and $|T_A^*| \leq 3\Delta^2\ell$, hence the recursive call will fall under the first bullet.

We infer that the depth of the recursion is at most $2\lceil \log n \rceil$.

At every non-leaf recursive call, we spend $n^{\mathcal{O}(1)}$ time on invoking the algorithm from Theorem 3, $n^{\mathcal{O}(t\Delta^2)}$ time to compute f_{G^*, \mathbf{w}, T^*} using Lemma 7, and $n^{\mathcal{O}(t\Delta^2)}$ time for the final iteration over all subsets $I_T \subseteq T^* \cup N_{G'}[X]$. Hence, the time spent at every recursive call is bounded by $n^{\mathcal{O}(t\Delta^2)}$.

At every non-leaf recursive call, we make subcalls to $(G_A^*, \mathbf{w}, T_A^*)$ for every particle A of (H, η) . Lemmata 8 and 9 ensure that the sum of $|V(G_A^*)|$ over all particles A is bounded by $(2\Delta + 3)|V(G')|$. Hence, the total size of all graphs in the i -th level of the recursion is bounded by $n \cdot (2\Delta + 3)^i$. Since the depth of the recursion is bounded by $2\lceil \log n \rceil$, the total size of all graphs in the recursion tree is bounded by $n^{\mathcal{O}(\log \Delta)}$. Since this also bounds the size of the recursion tree, we infer that the whole algorithm runs in time $n^{\mathcal{O}(t\Delta^2)}$.

This completes the proof of Theorem 1.

4 Graphs with no large bicliques: Proof of Theorem 2

This section is devoted to the proof of Theorem 2.

Let t be a positive integer and let k be the constant depending on t from Corollary 6. Again, let (G, \mathbf{w}) be the input vertex-weighted graph, let $n := |V(G)|$, and let

$$\ell := \lceil 11 \log n + 6 \rceil (t + 2) = \mathcal{O}(t \log n)$$

be an upper bound on the size of X for any application of Theorem 3 for any induced subgraph of G .

The general framework and the leaves of the recursion are almost exactly the same as in the previous section, but with different thresholds. That is, we describe a recursive algorithm that takes as input an induced subgraph G' of G with weights \mathbf{w} and a set of terminals $T \subseteq V(G')$ of size at most $32k^5\ell$ and solves BORDER MWIS on (G', \mathbf{w}, T) . The root call is for $G' := G$ and $T := \emptyset$ and the algorithm returns $f_{G, \mathbf{w}, \emptyset}(\emptyset)$ as the final answer.

Let (G', \mathbf{w}, T) be an input to a recursive call. The algorithm initiates first $f_{G', \mathbf{w}, T}(I_T) = -\infty$ for every $I_T \subseteq T$.

If $|V(G')| \leq 32k^5\ell$, the algorithm proceeds by brute-force: it enumerates independent sets $I \subseteq V(G')$ and updates $f_{G', \mathbf{w}, T}(I \cap T)$ with $\mathbf{w}(I)$ whenever the previous value of that cell was smaller. As $\ell = \mathcal{O}(t \log n)$ and k is a constant depending on t and s , this step takes polynomial time. This completes the description of the leaf step of the recursion.

Otherwise, if $|V(G')| > 32k^5\ell$, we invoke Corollary 6 on G' , obtaining a tree decomposition (\mathcal{T}, β) of G' . If $|T| \leq 24k^5\ell$, let $U := V(G') \setminus T$, and otherwise, let $U := T$.

For every $t_1 t_2 \in E(\mathcal{T})$, proceed as follows. For $i = 1, 2$, let \mathcal{T}_i be the connected component of $\mathcal{T} - \{t_1 t_2\}$ that contains t_i and let $V_i = \bigcup_{x \in \mathcal{T}_i} \beta(x) \setminus \sigma(t_1 t_2)$. Clearly, $\sigma(t_1 t_2)$ separates V_1 from V_2 . Orient the edge $t_1 t_2$ towards t_i with larger $|U \cap V_i|$, breaking ties arbitrarily.

There exists $t \in V(\mathcal{T})$ of outdegree 0. Then, for every connected component C of $G' - \beta(t)$ we have $|C \cap U| \leq |U|/2$. Fix one such node t and let $B := \beta(t)$ and let \mathcal{C} be the set of connected components of $G' - B$. Let G^B be a supergraph of $G'[B]$ obtained from $G'[B]$ by turning, for every $C \in \mathcal{C}$, the neighborhood $N_{G'}(C)$ into a clique. Note that G^B is a subgraph of the torso of $\beta(t)$. Hence, by the properties promised by Corollary 6, for every $C \in \mathcal{C}$ we have $|N_{G'}(C)| < k$ (as this set is contained in a single adhesion of an edge incident with t in \mathcal{T}) and G^B contains at most k vertices of degree larger than $2k(k-1)$. Let Q be the set of vertices of G^B of degree larger than $2k(k-1)$.

We perform exhaustive branching on Q . That is, we iterate over all independent sets $J \subseteq Q$ and denote $G^J := G' - Q - N_{G'}(J)$, $T^J := T \cap V(G^J)$, $U^J := U \cap V(G^J)$. For one J , we proceed as follows.

We invoke Theorem 3 to G^J with set U^J , obtaining a set X^J of size at most ℓ and an extended strip decomposition (H^J, η^J) of $G^J - N_{G^J}[X^J]$ whose every particle has at most $|U^J|/2 \leq |U|/2$ vertices of U . Note that G^J is an induced subgraph of G' , which is an induced subgraph of G , so there is no induced $dS_{t,t,t}$ in G^J .

A component $C \in \mathcal{C}$ is *dirty* if $N_{G^J}[X^J] \cap N_{G'}[C] \neq \emptyset$ and *clean* otherwise. Let

$$Y^J := (N_{G^J}[X^J] \cap B) \cup \bigcup_{C \in \mathcal{C}: C \text{ is dirty}} (N_{G'}(C) \cap V(G^J)).$$

The following bounds will be important for further steps.

$$|N_{G^J}[X^J] \cap B| \leq (2k(k-1) + 1)|X^J|. \quad (1)$$

To see (1) observe that a vertex $v \in X^J \cap B$ has at most $2k(k-1)$ neighbors in B (as every vertex of $B \setminus Q$ has degree at most $2k(k-1)$ in G_B) while every vertex $v \in X^J \setminus B$ has at most k neighbors in B , as every component of $G' - B$ has at most k neighbors in B . This proves (1).

$$|Y^J| \leq (k + (2k(k-1) + 1)^2)|X^J| \leq 4k^4|X^J| \leq 4k^4\ell = \mathcal{O}(k^4 t \log n). \quad (2)$$

To see (2), consider a dirty component $C \in \mathcal{C}$. Observe that either C contains a vertex of X^J or $N_{G'}(C) \cap V(G^J)$ contains a vertex of $N_{G^J}[X^J] \cap B$. There are at most $|X^J|$ dirty components of the first type, contributing in total at most $k|X^J|$ vertices to Y^J . For the dirty components of the second type, although there can be many of them, we observe that if $v \in N_{G'}(C) \cap N_{G^J}[X^J] \cap B$, then $N_{G'}(C) \cap V(G^J) \subseteq N_{G_B}[v]$. Hence, for every dirty component of the second type, it holds that $N_{G'}(C) \cap V(G^J) \subseteq N_{G_B}[N_{G^J}[X^J] \cap B]$. Since the degree of each vertex of G_B is at most $2k(k-1)$, by (1) we have

$$|N_{G_B}[N_{G^J}[X^J] \cap B]| \leq (2k(k-1) + 1)^2|X^J|.$$

The bound (2) follows.

A component $C \in \mathcal{C}$ is *touched* if it is dirty or $N_{G'}(C)$ contains a vertex of Y^J . Let

$$Z^J := (N_{G^J}[Y^J] \cap B) \cup \bigcup_{C \in \mathcal{C}: C \text{ is touched}} N_{G'}(C) \cap V(G^J).$$

Using similar arguments as before, we can prove

$$|Z^J| \leq (2k(k-1) + 1)|Y^J| \leq 8k^5|X^J| \leq 8k^5\ell = \mathcal{O}(k^5 t \log n). \quad (3)$$

Indeed, if C is touched, then $N_{G'}(C)$ contains a vertex $v \in Y^J$ (if C is dirty, $N_{G'}(C) \cap V(G^J)$ is contained in Y^J), and then $N_{G'}(C)$ is contained in $N_{G_B}[v]$. Also, for $v \in Y^J$ we have $N_{G^J}[v] \cap B \subseteq N_{G_B}[v]$. Hence, $Z^J \subseteq N_{G_B}[Y^J]$. Since the maximum degree of a vertex of $B \setminus Q$ is $2k(k-1)$, this proves (3).

For every touched $C \in \mathcal{C}$, denote $G_C := G^J[N_{G'}[C] \cap V(G^J)]$ and $T_C := ((T \cap C) \cup N_{G'}(C)) \cap V(G^J)$. Recurse on (G_C, \mathbf{w}, T_C) , obtaining f_{G_C, \mathbf{w}, T_C} .

Let

$$G^Y := G^J - Y^J - \bigcup_{C \in \mathcal{C}: C \text{ is touched}} C.$$

Note that, by the definition of dirty and touched, G^Y is an induced subgraph of $G^J - N_{G^J}[X^J]$. Hence, (H^J, η^J) can be restricted to a (not necessarily rigid) extended strip decomposition $(H^J, \eta^{J,Y})$ of G^Y .

Let $T^Y := (T \cup Z^J) \cap V(G^Y)$. For every particle A of $(H^J, \eta^{J,Y})$, recurse on $(G^Y[A], \mathbf{w}, T^Y \cap A)$, obtaining $f_{G^Y[A], \mathbf{w}, T^Y \cap A}$. Then, use these values with Lemma 7 to solve a BORDER MWIS instance (G^Y, \mathbf{w}, T^Y) , obtaining f_{G^Y, \mathbf{w}, T^Y} .

Iterate over every independent set $I_T \subseteq (T \cap V(G^J)) \cup T^Y \cup Y^J$. Observe that G' admits an independent set I with $I \cap (Q \cup T \cup T^Y \cup Y^J) = J \cup I_T$ and weight:

$$\begin{aligned} \mathbf{w}(J) + \mathbf{w}(I_T \setminus T^Y) + f_{G^Y, \mathbf{w}, T^Y}(I_T \cap T^Y) + \\ \sum_{C \in \mathcal{C}: C \text{ is touched}} (f_{G_C, \mathbf{w}, T_C}(N_{G'}[C] \cap I_T) - \mathbf{w}(I_T \cap N_{G'}(C))). \end{aligned}$$

Update the cell $f_{G', \mathbf{w}, T}((I_T \cup J) \cap T)$ with this value if it is larger than the previous value of this cell. This finishes the description of the algorithm.

4:10 Max Weight Independent Set in Sparse Graphs with No Long Claws

For correctness, it suffices to note that for every touched component C , the whole $N_{G'}(C) \cap V(G^J)$ is in the terminal set for the recursive call (G_C, \mathbf{w}, T_C) and the whole $N_{G'}(C) \cap V(G^Y)$ is in Z^J and thus in the terminal set for the BORDER MWIS instance (G^Y, \mathbf{w}, T^Y) .

For the sake of analysis, consider a recursive call on (G_C, \mathbf{w}, T_C) for a touched component C . If $|T| \leq 24k^5\ell$ and $U = V(G') \setminus T$, then $|T_C| \leq |T| + k \leq 32k^5\ell$ and $|V(G_C) \setminus T_C| \leq |C \setminus T| \leq |V(G') \setminus T|/2$. Otherwise, if $|T| > 24k^5\ell$ and $U = T$, then $|T_C| \leq |T|/2 + k \leq 16k^5\ell + k \leq 24k^5\ell$. Thus, the recursive call on (G_C, \mathbf{w}, T_C) will fall under the first case of at most $24k^5\ell$ terminals.

Analogously, consider a recursive call on $(G^Y[A], \mathbf{w}, T^Y \cap A)$ for a particle A of $(H^J, \eta^{J,Y})$. If $|T| \leq 24k^5\ell$ and $U = V(G') \setminus T$, then $|T^Y \cap A| \leq |T^Y| \leq |T| + |Z^J| \leq 32k^5\ell$ due to (3). Furthermore, $|V(G^Y[A]) \setminus T^Y| \leq |V(G') \setminus T|/2$. Otherwise, if $|T| > 24k^5\ell$ and $U = T$, then $|T^Y \cap A| \leq |T|/2 + |Z^J| \leq 16k^5\ell + 8k^5\ell \leq 24k^5\ell$ again due to (3). Thus, the recursive call on $(G^Y[A], \mathbf{w}, T^Y \cap A)$ will fall under the first case of at most $24k^5\ell$ terminals.

Finally, note that a recursive call (G', \mathbf{w}, T) without nonterminal vertices (i.e., with $T = V(G')$) is a leaf call.

We infer that all recursive calls satisfy the invariant of at most $32k^5\ell$ terminals and the depth of the recursion tree is bounded by $2\lceil \log n \rceil$ (as every second level the number of nonterminal vertices halves).

At each recursive call, we iterate over at most 2^k subsets $J \subseteq Q$. Lemma 8 ensures that the maximum degree of H^J is at most $2t - 1$, while Lemma 9 ensures that every vertex of G^Y is used in at most $4t$ particles of $(H^J, \eta^{J,Y})$. In a subcall (G_C, \mathbf{w}, T_C) for a touched component C , vertices of C are not used in any other call for the current choice of J , while all vertices of $V(G_C) \setminus C$ are terminals. Consequently, every nonterminal vertex v of G' is passed as a nonterminal vertex to a recursive subcall at most $2^k \cdot 4t$ number of times (and a terminal is always passed to a subcall as a terminal). Furthermore, a recursive call without nonterminal vertices is a leaf call. As the depth of the recursion is $\mathcal{O}(\log n)$, we infer that, summing over all recursive calls in the entire algorithm, the number of nonterminal vertices is bounded by $n^{\mathcal{O}(\log t + k)}$ and the total size of the recursion tree is $n^{\mathcal{O}(\log t + k)}$.

At each recursive call, we iterate over all 2^k subsets $J \subseteq Q$ and then we invoke Theorem 3 and iterate over all independent sets I_T in $(T \cap V(G^J)) \cup T^Y \cup Y^J$. Thanks to the invariant $|T| \leq 32k^5\ell$ and bounds (2), and (3), this set is of size $\mathcal{O}(k^5\ell)$. Hence, every recursive call runs in time $n^{\mathcal{O}(k^5t) + k \cdot c_t}$, where c_t is a constant depending on t . As k is a constant depending on t and s , the final running time bound is polynomial.

This completes the proof of Theorem 2.

5 Conclusion

While it is generally believed that MWIS is polynomial-time-solvable in $S_{t,t,t}$ -free graphs (with no further assumptions), such a result seems currently out of reach. Thus it is interesting to investigate how further can we relax the assumptions on instances, as we did when going from Theorem 1 to Theorem 2. In particular, we used the assumption of K_r -freeness twice: once in Lemma 5 and then to argue that H (the pattern of an extended strip decomposition we obtain) is of bounded degree. On the other hand, the assumption of $K_{r,r}$ -freeness was used just once: in Lemma 5. Thus it seems natural to try to prove the following conjecture.

► **Conjecture 10.** *For every integers t, r there exists a polynomial-time algorithm that, given an $S_{t,t,t}$ -free and K_r -free vertex-weighted graph (G, \mathbf{w}) computes the maximum possible weight of an independent set in (G, \mathbf{w}) .*

References

- 1 Tara Abrishami, Maria Chudnovsky, Cemil Dibek, and Paweł Rzażewski. Polynomial-time algorithm for maximum independent set in bounded-degree graphs with no long induced claws. In Niv Buchbinder Joseph (Seffi) Naor, editor, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference, January 9-12, 2022*, pages 1448–1470. SIAM, 2022. doi:10.1137/1.9781611977073.61.
- 2 Tara Abrishami, Maria Chudnovsky, Marcin Pilipczuk, and Paweł Rzażewski. Max Weight Independent Set in sparse graphs with no long claws. *CoRR*, abs/2309.16995, 2023. doi:10.48550/ARXIV.2309.16995.
- 3 Vladimir E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982.
- 4 Vladimir E. Alekseev. On easy and hard hereditary classes of graphs with respect to the independent set problem. *Discret. Appl. Math.*, 132(1-3):17–26, 2003. doi:10.1016/S0166-218X(03)00387-1.
- 5 Vladimir E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discrete Applied Mathematics*, 135(1):3–16, 2004. Russian Translations II. doi:10.1016/S0166-218X(02)00290-1.
- 6 Gábor Bacsó, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Zsolt Tuza, and Erik Jan van Leeuwen. Subexponential-time algorithms for Maximum Independent Set in P_t -free and broom-free graphs. *Algorithmica*, 81(2):421–438, 2019. doi:10.1007/s00453-018-0479-5.
- 7 Andreas Brandstädt and Raffaele Mosca. Maximum weight independent set for ℓ -claw-free graphs in polynomial time. *Discret. Appl. Math.*, 237:57–64, 2018. doi:10.1016/j.dam.2017.11.029.
- 8 Andreas Brandstädt and Raffaele Mosca. Maximum weight independent sets for $(P_7, \text{triangle})$ -free graphs in polynomial time. *Discret. Appl. Math.*, 236:57–65, 2018. doi:10.1016/j.dam.2017.10.003.
- 9 Andreas Brandstädt and Raffaele Mosca. Maximum weight independent sets for $(S_{1,2,4}, \text{triangle})$ -free graphs in polynomial time. *Theor. Comput. Sci.*, 878-879:11–25, 2021. doi:10.1016/j.tcs.2021.05.027.
- 10 Christoph Brause. A subexponential-time algorithm for the Maximum Independent Set Problem in P_t -free graphs. *Discret. Appl. Math.*, 231:113–118, 2017. doi:10.1016/j.dam.2016.06.016.
- 11 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.
- 12 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the Maximum Weight Independent Set Problem in H -free graphs. *CoRR*, abs/1907.04585, 2019. arXiv:1907.04585.
- 13 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the Maximum Weight Independent Set Problem in H -free graphs. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2260–2278. SIAM, 2020. doi:10.1137/1.9781611975994.139.
- 14 Maria Chudnovsky, Stéphan Thomassé, Nicolas Trotignon, and Kristina Vuskovic. Maximum independent sets in (pyramid, even hole)-free graphs. *CoRR*, abs/1912.11246, 2019. arXiv:1912.11246.
- 15 M. Cygan, F. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.
- 16 Peter Gartland and Daniel Lokshtanov. Independent set on P_k -free graphs in quasi-polynomial time. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 613–624. IEEE, 2020. doi:10.1109/FOCS46700.2020.00063.

- 17 Peter Gartland, Daniel Lokshtanov, Tomáš Masařík, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzażewski. Maximum weight independent set in graphs with no long claws in quasi-polynomial time. *CoRR*, abs/2305.15738, 2023. doi:10.48550/arXiv.2305.15738.
- 18 Michael U. Gerber, Alain Hertz, and Vadim V. Lozin. Stable sets in two subclasses of banner-free graphs. *Discrete Applied Mathematics*, 132(1-3):121–136, 2003. doi:10.1016/S0166-218X(03)00395-0.
- 19 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for Maximum Weight Independent Set on P_6 -free graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1257–1271. SIAM, 2019. doi:10.1137/1.9781611975482.77.
- 20 M. Grötschel, L. Lovász, and A. Schrijver. Polynomial algorithms for perfect graphs. In C. Berge and V. Chvátal, editors, *Topics on Perfect Graphs*, volume 88 of *North-Holland Mathematics Studies*, pages 325–356. North-Holland, 1984. doi:10.1016/S0304-0208(08)72943-8.
- 21 Ararat Harutyunyan, Michael Lampis, Vadim V. Lozin, and Jérôme Monnot. Maximum independent sets in subcubic graphs: New results. *Theor. Comput. Sci.*, 846:14–26, 2020. doi:10.1016/j.tcs.2020.09.010.
- 22 Johan Håstad. Clique is hard to approximate within $n^{(1-\varepsilon)}$. In *Acta Mathematica*, pages 627–636, 1996.
- 23 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 24 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k -way cut of bounded size is fixed-parameter tractable. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 160–169. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.53.
- 25 T. Kövari, V. Sós, and P. Turán. On a problem of K. Zarankiewicz. *Colloquium Mathematicae*, 3(1):50–57, 1954. URL: <http://eudml.org/doc/210011>.
- 26 Ngoc Chi Lê, Christoph Brause, and Ingo Schiermeyer. The Maximum Independent Set Problem in subclasses of $S_{i,j,k}$ -free graphs. *Electron. Notes Discret. Math.*, 49:43–49, 2015. doi:10.1016/j.endm.2015.06.008.
- 27 Daniel Lokshantov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5 -free graphs in polynomial time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 570–581. SIAM, 2014. doi:10.1137/1.9781611973402.43.
- 28 Vadim V. Lozin and Martin Milanič. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *J. Discrete Algorithms*, 6(4):595–604, 2008. doi:10.1016/j.jda.2008.04.001.
- 29 Vadim V. Lozin, Martin Milanič, and Christopher Purcell. Graphs without large apples and the Maximum Weight Independent Set problem. *Graphs Comb.*, 30(2):395–410, 2014. doi:10.1007/s00373-012-1263-y.
- 30 Vadim V. Lozin, Jérôme Monnot, and Bernard Ries. On the maximum independent set problem in subclasses of subcubic graphs. *J. Discrete Algorithms*, 31:104–112, 2015. doi:10.1016/j.jda.2014.08.005.
- 31 Vadim V. Lozin and Raffaele Mosca. Independent sets in extensions of $2K_2$ -free graphs. *Discret. Appl. Math.*, 146(1):74–80, 2005. doi:10.1016/j.dam.2004.07.006.
- 32 Vadim V. Lozin and Dieter Rautenbach. Some results on graphs without long induced paths. *Inf. Process. Lett.*, 88(4):167–171, 2003. doi:10.1016/j.ipl.2003.07.004.
- 33 Vadim V. Lozin and Igor Razgon. Tree-width dichotomy. *Eur. J. Comb.*, 103:103517, 2022. doi:10.1016/j.ejc.2022.103517.
- 34 Frédéric Maffray and Lucas Pastor. Maximum weight stable set in (P_7, bull) -free graphs. *CoRR*, abs/1611.09663, 2016. URL: <http://arxiv.org/abs/1611.09663>, arXiv:1611.09663.

- 35 Konrad Majewski, Tomáš Masařík, Jana Novotná, Karolina Okrasa, Marcin Pilipczuk, Paweł Rzażewski, and Marek Sokolowski. Max Weight Independent Set in Graphs with No Long Claws: An Analog of the Gyárfás' Path Argument. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 93:1–93:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.93.
- 36 George J. Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.
- 37 Raffaele Mosca. Stable sets in certain P_6 -free graphs. *Discret. Appl. Math.*, 92(2-3):177–191, 1999. doi:10.1016/S0166-218X(99)00046-3.
- 38 Raffaele Mosca. Stable sets of maximum weight in (P_7, banner) -free graphs. *Discrete Mathematics*, 308(1):20–33, 2008. doi:10.1016/j.disc.2007.03.044.
- 39 Raffaele Mosca. Independent sets in $(P_6, \text{diamond})$ -free graphs. *Discret. Math. Theor. Comput. Sci.*, 11(1):125–140, 2009. doi:10.46298/dmtcs.473.
- 40 Raffaele Mosca. Maximum weight independent sets in $(P_6, \text{co-banner})$ -free graphs. *Inf. Process. Lett.*, 113(3):89–93, 2013. doi:10.1016/j.ipl.2012.10.004.
- 41 Raffaele Mosca. Independent sets in $(P_{4+4}, \text{triangle})$ -free graphs. *Graphs Comb.*, 37(6):2173–2189, 2021. doi:10.1007/s00373-021-02340-7.
- 42 Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzażewski. Quasi-polynomial-time algorithm for independent set in P_t -free graphs via shrinking the space of induced paths. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 204–209. SIAM, 2021. doi:10.1137/1.9781611976496.23.
- 43 Najiba Sbihi. Algorithmes de recherche d'un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics*, 29(1):53–76, 1980. doi:10.1016/0012-365X(90)90287-R.
- 44 Daniel Weißauer. On the block number of graphs. *SIAM J. Discret. Math.*, 33(1):346–357, 2019. doi:10.1137/17M1145306.
- 45 Nikola Yolov. Minor-matching hypertree width. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 219–233. SIAM, 2018. doi:10.1137/1.9781611975031.16.

A Appendix: Proof of Lemma 7

Iterate over every $I_T \subseteq T$. For fixed I_T , we aim at computing $f_{G, \mathbf{w}, T}(I_T)$. If I_T is not independent, we set $f_{G, \mathbf{w}, T}(I_T) = -\infty$. In the remainder of the proof, we show how to compute in polynomial time the value $f_{G, \mathbf{w}, T}(I_T)$ for fixed independent $I_T \subseteq T$.

For a particle A of (H, η) , let $a(A) := f_{G[A], \mathbf{w}, T \cap A}(I_T \cap A)$ and let $I(A)$ be an independent set witnessing this value, that is, an independent set in $G[A]$ of weight $\mathbf{w}(a(A))$ with $I(A) \cap T \cap A = I_T \cap A$. Note that as I_T is independent, the value $a(A)$ is not equal to $-\infty$ and such an independent set exists.

We say that $x \in V(H)$ is *forced* if $I_T \cap \bigcup_{y \in N_H(x)} \eta(xy, x) \neq \emptyset$. Note that since I_T is independent, if x is forced, then $\eta(xy, x) \cap I_T \neq \emptyset$ for exactly one edge xy incident with x . We call such an edge xy the *enforcer* of x . Note that an edge xy may be the enforcer of both x and y .

The arguments now follow very closely the outline of Section 3.3 of [12].

We construct a set \mathcal{P} of particles and an edge-weighted graph (H', \mathbf{w}') as follows. We start with $\mathcal{P} = \emptyset$, $V(H') = V(H)$, and $E(H') = \emptyset$.

4:14 Max Weight Independent Set in Sparse Graphs with No Long Claws

For every $x \in V(H)$ that is not forced, add A_x to \mathcal{P} . For every $xyz \in T(H)$ such that neither of the edges xy , yz , or xz is the enforcer of *both* its endpoints, add A_{xyz} to \mathcal{P} . For every $e = xy \in E(H)$, proceed as follows.

1. If neither x nor y is forced, we add to H' a new vertex t_e and edges $t_e x$, $t_e y$, xy , and set the edge weights \mathbf{w}' as follows:

$$\begin{aligned}\mathbf{w}'(t_e x) &:= a(A_{xy}^x) - a(A_{xy}^\perp) - a(A_x), \\ \mathbf{w}'(t_e y) &:= a(A_{xy}^y) - a(A_{xy}^\perp) - a(A_y), \\ \mathbf{w}'(xy) &:= a(A_{xy}^{xy}) - a(A_{xy}^\perp) - a(A_x) - a(A_y) - \sum_{z, \text{ s.t. } xyz \in T(H)} a(A_{xyz}).\end{aligned}$$

Furthermore, add A_{xy}^\perp to \mathcal{P} .

2. If exactly one of x and y is forced, say w.l.o.g. x is forced and y is not forced, proceed as follows.
 - a. If xy is the enforcer of x , then add to H' an edge xy with weight

$$\mathbf{w}'(xy) := a(A_{xy}^{xy}) - a(A_{xy}^x) - a(A_y) - \sum_{z, \text{ s.t. } xyz \in T(H)} a(A_{xyz}).$$

Furthermore, add A_{xy}^x to \mathcal{P} .

- b. If xy is not the enforcer of x , then add to H' a new vertex t_e and an edge $t_e y$ with weight

$$\mathbf{w}'(t_e y) := a(A_{xy}^y) - a(A_{xy}^\perp) - a(A_y).$$

Furthermore, add A_{xy}^\perp to \mathcal{P} .

3. If both x and y are forced, proceed as follows.
 - a. If xy is neither the enforcer of x nor of y , add A_{xy}^\perp to \mathcal{P} .
 - b. If xy is the enforcer of x but not of y add A_{xy}^x to \mathcal{P} .
 - c. If xy is the enforcer of y but not of x add A_{xy}^y to \mathcal{P} .
 - d. If xy is the enforcer of both x and y , add A_{xy}^{xy} to \mathcal{P} .

This finishes the description of the construction of \mathcal{P} and (H', \mathbf{w}') . In the next two paragraphs we make two observations that follow by a direct check from the definitions.

Observe that $I_0 := \bigcup_{A \in \mathcal{P}} I(A)$ is independent in G and has weight $a_0 := \sum_{A \in \mathcal{P}} a(A)$. Furthermore, for every $A \in \mathcal{P}$, we have $I_0 \cap A \cap T = I_T \cap A$ and $I_0 \cap T = I_T$. We think of I_0 as the “base” solution for $f_{G, \mathbf{w}, T}(I_T)$.

Observe also that all weights \mathbf{w}' of H' are nonnegative, as A_{xy}^x contains both A_{xy}^\perp and A_x while A_{xy}^{xy} contains A_{xy}^\perp , A_x , A_y , as well as all A_{xyz} for all triangles xyz containing the edge xy .

We will be asking for a maximum weight matching in (H', \mathbf{w}') . Intuitively, taking an edge $t_e x$ to such a matching corresponds to replacing in I_0 the parts $I(A_{xy}^\perp)$ and $I(A_x)$ with the part $I(A_{xy}^x)$ while taking an edge xy to such a matching corresponds to replacing in I_0 the parts $I(A_{xy}^\perp)$, $I(A_x)$, $I(A_y)$ and all parts $I(A_{xyz})$ for triangles xyz containing the edge xy with part $I(A_{xy}^{xy})$.

From another perspective, fix $x \in V(H)$ and recall that the sets $\eta(xy, x)$ for $y \in N_H(x)$ are complete to each other. Hence, any independent set in G can contain vertices in at most one of such sets. For an edge $e = xy \in E(H)$, taking an edge xy or $t_e x$ in a matching in H' corresponds to choosing that, among all neighbors of x in H , the neighbor y is such that the set $\eta(xy, x)$ is allowed to contain vertices of the sought independent set. (Choosing $xy \in E(H')$ to the matching corresponds to allowing both $\eta(xy, x)$ and $\eta(yx, y)$ to contain vertices of the sought independent set.)

However, there is a delicacy if I_T contains a vertex of some interface $\eta(xy, x)$. Then, in some sense I_T already forces some choices in the corresponding matching in H' . This is modeled above by having alternate construction for vertices $x \in V(H)$ that are forced.

The following two claims prove that $f_{G, \mathbf{w}, T}(I_T)$ equals a_0 plus the maximum possible weight of a matching in (H', \mathbf{w}') and thus complete the proof of Lemma 7. Their proofs follow exactly the lines of the proofs of Claims 3.7 and 3.8 of Section 3.3 of [12] and are thus omitted.

▷ **Claim 11.** Let I be an independent set in G with $I \cap T = I_T$. Let M be the set of edges of H' defined as follows: for every $e = xy \in E(H)$, if $\eta(xy, x) \cap I \neq \emptyset$ and $\eta(xy, y) \cap I \neq \emptyset$, then $xy \in M$, if $\eta(xy, x) \cap I \neq \emptyset$ and $\eta(xy, y) \cap I = \emptyset$, then $t_e x \in M$, and if $\eta(xy, x) \cap I = \emptyset$ and $\eta(xy, y) \cap I \neq \emptyset$, then $t_e y \in M$. Then, all the above edges indeed exist in H' and M is a matching. Furthermore, the weight of I is at most $a_0 + \sum_{e \in M} \mathbf{w}'(e)$.

▷ **Claim 12.** Let M be a matching in H' . Let \mathcal{P}_M be the set of particles of (H, η) defined as follows. Start with $\mathcal{P}_M := \mathcal{P}$ and then for every edge $e = xy \in E(H)$,

- if $xy \in M$, insert A_{xy}^{xy} into \mathcal{P}_M and remove from \mathcal{P}_M the following particles if present: $A_{xy}^x, A_{xy}^y, A_{xy}^\perp, A_x, A_y, A_{xyz}$ for any $z \in V(H)$ such that $xyz \in T(H)$.
- if $t_e x \in M$ (resp. $t_e y \in M$), insert A_{xy}^x (resp. A_{xy}^y) into \mathcal{P}_M , and remove from \mathcal{P}_M the following particles if present: A_{xy}^\perp and A_x (resp. A_y).

Then $I_M := \bigcup_{A \in \mathcal{P}_M} A$ is an independent set in G with $I_M \cap T = I_T$ and of weight at least $a_0 + \sum_{e \in M} \mathbf{w}'(e)$.

Satisfiability of Context-Free String Constraints with Subword-Ordering and Transducers

C. Aiswarya  

Chennai Mathematical Institute, India
CNRS, ReLaX, IRL 2000, Chennai, India

Soumodev Mal  

Chennai Mathematical Institute, India

Prakash Saivasan  

The Institute of Mathematical Sciences, HBNI, Chennai, India
CNRS, ReLaX, IRL 2000, Chennai, India

Abstract

We study the satisfiability of string constraints where context-free membership constraints may be imposed on variables. Additionally a variable may be constrained to be a subword of a word obtained by shuffling variables and their transductions. The satisfiability problem is known to be undecidable even without rational transductions. It is known to be NEXPTIME-complete without transductions, if the subword relations between variables do not have a cyclic dependency between them. We show that the satisfiability problem stays decidable in this fragment even when rational transductions are added. It is 2NEXPTIME-complete with context-free membership, and NEXPTIME-complete with only regular membership. For the lower bound we prove a technical lemma that is of independent interest: The length of the shortest word in the intersection of a pushdown automaton (of size $\mathcal{O}(n)$) and n finite-state automata (each of size $\mathcal{O}(n)$) can be double exponential in n .

2012 ACM Subject Classification Theory of computation \rightarrow Logic and verification

Keywords and phrases satisfiability, subword, string constraints, context-free, transducers

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.5

Related Version *Full Version:* <https://arxiv.org/abs/2401.07996>

Funding *Prakash Saivasan:* MATRICS GRANT (MTR/2022/000312).

Acknowledgements We thank Paul Gastin for helpful discussions.

1 Introduction

The theory of strings has always been an important and active area of research for long. In fact, as Hilbert notes, it is the very foundation of mathematical logic itself [45, 25]. The recent successes in employing the theory for practical verification has only re-iterated its importance. The study of the theory of string constraints dates back to Tarski and Hermes [44, 33], who in 1933 provided the axiomatic foundation for it. There have been several other advancements of string theories since then, some of the notable ones include [25, 43, 41, 40, 42, 18]. In 1977, Makanin studied the algorithmic aspect of the word equations (equation involving concatenation and equality) and showed that the satisfiability problem is decidable [40]. The complexity for this problem was improved in [42]. Despite receiving much attention, the theory of strings has long standing unsolved open problems, indicating the intrinsic difficult nature of the theory.



© C. Aiswarya, Soumodev Mal, and Prakash Saivasan;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;
Article No. 5; pp. 5:1–5:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



One important aspect of the study here is the satisfiability of string constraints. The question here asks whether it is possible to assign a word to each variable such that the given set of string constraints is satisfied. The constraints themselves can be either relational, which relate variables or membership, that define the domain for each variable.

In the recent years, the constraint satisfaction problem of strings (CSPS) has received much attention from verification community due to its usefulness in modeling and reasoning about programs. This problem has particularly been useful in verifying web services [32] and database applications from injection attacks [11]. In such attacks, the attacker constructs an input string in such a way that the underlying semantics of the interpretation is changed. The CSPS, and more importantly its implementations in solvers [38, 1, 17, 36, 37, 26, 35] have provided the much needed power to model and verify programs for such vulnerabilities. This in turn has directed the study to explore the boundaries of solvability.

However one impediment for this has been the theoretical limitation. For instance, with respect to word equations, adding a transducer renders the model undecidable. Similarly introducing membership in context free language also renders the model undecidable (see [28], [30] for more details). Despite this, there have been several advancements in this regard [20, 39, 34, 23, 22, 19, 5, 8, 29].

The context-free membership constraints are particularly useful feature to have since checking vulnerabilities include checking for programs, that are inherently context-free, masquerading as string queries. In [9], the authors provided first such model that could handle context-free membership queries and yet has decidability for CSPS, under some restrictions. They showed that if every relational constraint has sub-word relation instead of equality and assuming an *acyclicity* restriction, the satisfaction problem is NEXPTIME COMPLETE. In fact, the authors in their model include a more powerful shuffle operator against the usual concatenation. Further they show that the complexity of the satisfiability problem when only regular membership is involved is also the same i.e, NEXPTIME COMPLETE. They also provide an interesting connection of their model with lossy channel systems that include pushdown automata.

Yet another feature in string solvers that has been much desired is that of transductions. As noted in [34, 22], most modern applications, especially browsers include implicit transductions that mutates the input string. To verify such applications, one also needs the power of transductions. There have been very few successful attempt towards decidability of string constraints that involve transductions, some of them being [34, 8, 22, 20].

We investigate string constraints when sub-word ordering, context-free membership and transducers are involved. Unfortunately, in its full generality this problem is undecidable. However we show that imposing the same acyclicity restriction as in [9] gives decidability under this setting. This extends the decidability result of [9] to include transductions.

In [9] the satisfiability of the acyclic variant of the string constraints without transducers was shown to be inter-reducible with the control-state reachability problem of acyclic networks of pushdown systems communicating over lossy fifo channels. They showed that both these problems are NEXPTIME-complete. In our setting, with the additional feature of transductions, we can enrich the model of communicating pushdown to allow transductions to be sent in the channels. Such transductions naturally model encoders such as error correcting codes or injection of noise.

We show that, when only regular membership is allowed, adding transductions do not alter the complexity. It is still NEXPTIME-complete. Interestingly when context-free membership is involved, it becomes 2NEXPTIME-complete.

Our 2NEXPTIME lower bound argument relies on a new technique that is of independent interest. In fact, we show that we can count exactly 2^{2^n} using one pushdown automaton with a binary stack alphabet and 3 states, and n finite state automata each of size $\mathcal{O}(n)$. Along the way we also show that 1) we can count exactly 2^n using a pushdown automaton with $\mathcal{O}(n)$ states and a binary stack alphabet, and 2) we can count exactly 2^n using n finite state automata each of size $\mathcal{O}(n)$.

As an application of this, we obtain a tight bound on the size of the smallest DFA of the downward closure, upward closure and the Parikh image closure of the intersection language of n finite state automata, each of size $\mathcal{O}(n)$. This size is $\Theta(2^n)$. Likewise, the size of the smallest DFA of the downward, upward and Parikh image closure for the intersection of language of n finite state automata with the language of a pushdown automaton, each of size $\mathcal{O}(n)$ is $\Theta(2^{2^n})$.

Related work

Apart from the work mentioned in the introduction, there are several other work on string constraints. In [19], the authors consider word equations equipped with replace all function and show decidability for the acyclic fragment.

In [3], the authors develop an uniform framework to decide the satisfiability and unsatisfiability of string constraints based on identifying patterns. In [2], the authors consider string constraints extended with negation and show how to solve them. In [23], the authors provide a semantic restriction on string manipulating programs that guarantees decidability for checking path feasibility. In [7], the authors study the problem of regular separability of the language of two word equations. In [27], the authors compare the expressive power of the logical theories built around word equations.

In [34], word equations with equality, transducers and regular membership is considered. This problem in full generality is undecidable. The authors consider a straight line fragment and show that the satisfiability problem is EXPSpace COMPLETE. In [23], the authors investigated the decidability of string constraints in the presence of regular membership constraints, `replaceAll` operator involving regular expressions and straight line restriction. In [21], the authors consider a stronger match and replace operator and show decidability. In [8], word equations with equality, transducers, length constraints and regular membership is considered and a chain free fragment of it was shown to be decidable. The authors show that the chain-free fragment of the satisfiability problem in this setting is decidable.

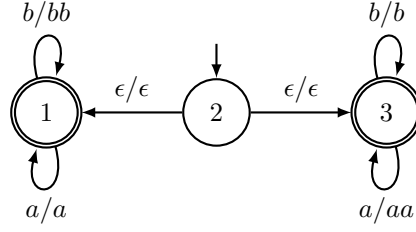
All of these work consider word equation (uses equality for comparison) in the model, our work uses subword ordering as the comparison operator. Further more, none of the work mentioned above considers context free membership constraints. In [9], subword ordering and context free membership is considered, where as it does not include transductions.

Apart from these, there are several approaches which attempts to solve the problem from a practical perspective, some of them being [4, 6, 2, 5, 17, 16, 34].

2 Preliminaries

Sets, Multisets, Functions

We denote the set of natural number $\{1, 2, \dots\}$ by \mathbb{N} . For $n \in \mathbb{N}$, we denote by $[n]$ the set of natural numbers up to n : $\{1, 2, \dots, n\}$. Let \mathbb{N}_0 denote the set $\{0, 1, 2, \dots\}$. That is, $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$.



■ **Figure 1** A transducer. Here the label x/y on a transition t indicates that $\text{label}(t) = x$ and $\text{out}(t) = y$. It nondeterministically chooses to duplicates a s leaving b s as such, or duplicates b s leaving a s as such.

Let S be any set. A *multiset* X of S assigns a multiplicity $X(s) \in \mathbb{N}_0$ to each element $s \in S$. We say that $s \in X$ if $X(s) > 0$. For a usual subset X , the multiplicity $X(s) \in \{0, 1\}$. A multiset X may also be written as $\{\!\{s_1, s_2, \dots\}\!\}$, by listing each element s , $X(s)$ many times. The set of all multisets of S is denoted \mathbb{N}_0^S , and the set of all usual subsets of S is denoted by 2^S . The size of a multiset X , denoted $|X|$ is the sum of the multiplicities of the elements. That is, $|X| = \sum_{s \in X} X(s)$.

Word, Subword, Shuffle, Projection

Let Σ be an alphabet. Σ^* denotes the set of all words over Σ , ϵ denotes the empty word, and $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$. For a word $w = a_1 a_2 \dots a_n \in \Sigma^*$, we denote by $\text{len}(w)$, the *length* of w ($\text{len}(w) = n$) and by $w[i]$ its i th letter a_i . The set of positions of w is denoted $\text{pos}(w)$. That is, $\text{pos}(w) = [\text{len}(w)]$. For $Y \subseteq \text{pos}(w)$, we denote by $w_{\downarrow Y}$ the *projection* of w to the positions in Y . If $Y = \{i_1, i_2, \dots, i_m\}$ with $0 < i_1 < i_2 < \dots < i_m \leq n$, then $w_{\downarrow Y} = a_{i_1} a_{i_2} \dots a_{i_m}$. For $u, v \in \Sigma^*$, we say u is a (*scattered*) *subword* of v , denoted $u \preceq v$, if there is $Y \subseteq \text{pos}(v)$ such that $u = v_{\downarrow Y}$. In this case we say v is a *superword* of u . Let $\Sigma' \subseteq \Sigma$ be a sub-alphabet and let $w \in \Sigma^*$. Projection of w to Σ' , denoted $w_{\downarrow \Sigma'}$, is defined to be $w_{\downarrow Y}$ where $Y = \{i \mid w[i] \in \Sigma'\}$.

Let X be a finite multiset of words from Σ^* given by $X = \{\!\{w_1, \dots, w_n\}\!\}$. We define the *shuffle* of X , denoted $\text{Shuffle}(X)$ to be the set $\{w \mid \text{there are } Y_1, Y_2 \dots Y_n \subseteq \text{pos}(w) \text{ forming a partition of } \text{pos}(w) \text{ and } w_i = w_{\downarrow Y_i} \text{ for all } i \in [n]\}$.

Finite-state automaton, Transducers, Pushdown Automaton

A (nondeterministic) *finite-state automaton* (NFA) over an alphabet Σ is given by a tuple $A = (\text{States}, \text{Trans}, s_{\text{in}}, F)$ where States is the finite set of states, $\text{Trans} \subseteq \text{States} \times \Sigma_\epsilon \times \text{States}$ is the set of transitions, $s_{\text{in}} \in \text{States}$ is the initial state, and $F \subseteq \text{States}$ is the set of final/accepting states. We write $s \xrightarrow{t} s'$ for some $t \in \text{Trans}$ if t is of the form (s, a, s') . Define the homomorphism $\text{label} : \text{Trans}^* \rightarrow \Sigma^*$ given by $\text{label}((s, a, s')) = a$. The *language* of an NFA A , denoted $L(A)$ is given by $L(A) = \{w \mid w = \text{label}(t_1 t_2 \dots t_n) \text{ and } s_{\text{in}} \xrightarrow{t_1} s_1 \xrightarrow{t_2} s_2 \dots s_{n-1} \xrightarrow{t_n} s_n \text{ with } s_n \in F\}$.

A *transducer* from Σ^* to Σ^* is a tuple $T = (\text{States}, \text{Trans}, s_{\text{in}}, F, \text{out})$ where $A = (\text{States}, \text{Trans}, s_{\text{in}}, F)$ is an NFA, and $\text{out} : \text{Trans} \rightarrow \Sigma^*$ defines the outputs on each transition. The function out defines a homomorphism $\text{out} : \text{Trans}^* \rightarrow \Sigma^*$. The *relation* $R \subseteq \Sigma^* \times \Sigma^*$ recognized by T , denoted $R(T)$ is given by $\{(u, v) \mid u = \text{label}(t_1 t_2 \dots t_n), v = \text{out}(t_1 t_2 \dots t_n) \text{ and } s_{\text{in}} \xrightarrow{t_1} s_1 \xrightarrow{t_2} s_2 \dots s_{n-1} \xrightarrow{t_n} s_n \text{ with } s_n \in F\}$. The equality relation is realised by a transducer T_{id} . A transducer is depicted in Figure 1.

A *pushdown automaton* over Σ is given by a tuple $P = (\text{States}, \text{Trans}, s_{\text{in}}, F, \text{op}, \Gamma)$ where $A = (\text{States}, \text{Trans}, s_{\text{in}}, F)$ is an NFA, Γ is the finite set of stack symbols, and $\text{op} : \text{Trans} \rightarrow \text{Ops}$ defines the stack operation of each transition, where $\text{Ops} = \{\text{push}(\gamma) \mid \gamma \in \Gamma\} \cup \{\text{pop}(\gamma) \mid \gamma \in \Gamma\} \cup \{\text{nop}\}$. When depicting the pushdown automaton pictorially, we represent a transition $t = (s, a, s')$ as $s \xrightarrow{a|\text{op}(t)} s'$. When $\text{op}(t) = \text{nop}$, we may simply write $s \xrightarrow{a} s'$. Further if $a = \epsilon$ then we may write it as $s \xrightarrow{\text{op}(t)} s'$. A configuration of a PDA is a pair $(s, w) \in \text{States} \times \Gamma^*$, indicating the current state and the stack contents. For two configurations (s, w) and (s', w') we write $(s, w) \xrightarrow{t} (s', w')$ for some $t \in \text{Trans}$ if t is of the form (s, a, s') and 1) $\text{op}(t) = \text{push}(\gamma)$ and $w' = \gamma \cdot w$, or 2) $\text{op}(t) = \text{pop}(\gamma)$ and $w = \gamma \cdot w'$, or 3) $\text{op}(t) = \text{nop}$ and $w = w'$. The *language* of a PDA P , denoted $L(P)$ is given by $L(P) = \{w \mid w = \text{label}(t_1 t_2 \dots t_n) \text{ and } (s_{\text{in}}, \epsilon) \xrightarrow{t_1} (s_1, w_1) \xrightarrow{t_2} (s_2, w_2) \dots (s_{n-1}, w_{n-1}) \xrightarrow{t_n} (s_n, \epsilon) \text{ with } s_n \in F\}$.

The set of all NFA / transducers / PDA over the alphabet Σ is denoted $\text{NFA}(\Sigma) / \text{TRANSD}(\Sigma) / \text{PDA}(\Sigma)$. A language $L \subseteq \Sigma^*$ is said to be context-free (resp. regular) if there is a PDA (resp. NFA) A such that $L = L(A)$. A relation $R \subseteq \Sigma^* \times \Sigma^*$ is said to be rational if it is recognized by some transducer T .

Given an NFA A (resp. transducer T), its number of states is denoted by $\text{state-size}(A)$ (resp. $\text{state-size}(T)$). Given a PDA P by $\text{state-size}(P)$ we denote the sum of the number of states and number of stack symbols. That is $\text{state-size}(P) = |\text{States}| + |\Gamma|$.

3 String constraints

A string constraint over a set of variables \mathcal{V} and an alphabet Σ is given by a set of membership constraints and a set of subword ordering constraints. The membership constraint is given by associating a pushdown automaton to each variable, indicating that the word assigned to the variable must belong to the language of the pushdown automaton. A subword order constraint is given by a pair (x, Y) where $x \in \mathcal{V}$ and Y is a finite multiset over $\mathcal{V} \times \text{TRANSD}(\Sigma)$.

For example, the constraint $(x, \{(y, T_1), (y, T_2), (z, T_2)\})$ means that the words assigned to x, y and z , say w_x, w_y and w_z respectively, must satisfy $w_x \preceq w$ for some $w \in \text{Shuffle}(\{u_1, u_2, u_3, u_4\})$, where $(w_y, u_1) \in R(T_1)$, $(w_y, u_2) \in R(T_2)$, $(w_y, u_3) \in R(T_2)$, and $(w_z, u_4) \in R(T_2)$. Note that the transducers can be identity in which case the input and the output are the same. For instance, if $T_1 = T_{\text{id}}$ then u_1 must be same as w_y .

We sometimes denote the constraint (x, Y) by $x \preceq \text{Shuffle}(Y)$. Abusing notation, we may write a pair $(x, T) \in \mathcal{V} \times \text{TRANSD}(\Sigma)$ as $T(x)$. If $Y = \{(y, T)\}$ (i.e., a singleton), then we may simply write $x \preceq T(y)$ instead of $x \preceq \text{Shuffle}(\{(y, T)\})$. Further, we may simply write x for (x, T_{id}) . For instance, $(x, \{(y, T_{\text{id}})\})$ may be also written as $x \preceq y$.

► **Definition 1.** A string constraint C is a tuple $(\Sigma, \mathcal{V}, \text{Mem}, \text{Rel})$ where $\text{Mem} : \mathcal{V} \rightarrow \text{PDA}(\Sigma) \cup \text{NFA}(\Sigma)$ assigns a PDA or an NFA to each variable, and $\text{Rel} \subseteq \mathcal{V} \times \mathbb{N}_0^{\mathcal{V} \times \text{TRANSD}(\Sigma)}$ is a finite set of subword-order constraints.

We denote by $\text{TRSET}(C)$ the finite set of transducers occurring in the string constraint C . That is, $\text{TRSET}(C) = \{T \mid \exists (x, Y) \in \text{Rel}, y \in \mathcal{V}, (y, T) \in Y\}$. Similarly, $\text{AUTSET}(C)$ is the finite set of PDA/NFA occurring in C . That is, $\text{AUTSET}(C) = \{\text{Mem}(x) \mid x \in \mathcal{V}\}$. A string constraint is *regular* if for every $v \in \mathcal{V}$, $\text{Mem}(v)$ is an NFA, (equivalently, if $\text{AUTSET}(C) \subseteq \text{NFA}(\Sigma)$). An important parameter for our complexity considerations will be the number of times a variable is used in the right hand side (RHS). We denote it by $\text{multiplicity}_C(x) = \sum_{T \in \text{TRSET}(C), (y, Y) \in \text{Rel}} Y((x, T))$. We omit the subscript and simply write $\text{multiplicity}(x)$ when C is clear from the context.

► **Definition 2.** A string constraint C is satisfiable if there exists an assignment $\sigma : \mathcal{V} \rightarrow \Sigma^*$ that satisfies every membership and relational constraints in C – that is,

1. $\sigma(v) \in L(\text{Mem}(v))$ for all $v \in \mathcal{V}$
2. For every $(x, Y) \in \text{Rel}$, if $Y = \{(y_1, T_1), (y_2, T_2), \dots, (y_n, T_n)\}$, then there are words u_1, u_2, \dots, u_n such that $(\sigma(y_i), u_i) \in R(T_i)$ for each $i \in \{1, 2, \dots, n\}$, and there is a word $w \in \text{Shuffle}(\{u_1, u_2, \dots, u_n\})$ such that $\sigma(x) \preceq w$.

Such an assignment σ is called a satisfying assignment.

► **Example 3.** Consider a string constraint on two variables x and y . The membership constraints are as follows. $\text{Mem}(x)$ is an NFA for $\{ababab\}$, and $\text{Mem}(y)$ is an NFA for $\{ab\}$. There is only one relational constraint: $x \preceq \text{Shuffle}(\{(y, T)(y, T)\})$, where T is the transducer defined in Figure 1. This string constraint is satisfiable.

► **Definition 4 (Satisfiability Problem for String Constraints).**

Input: A string constraint C .

Question: Is C satisfiable?

The satisfiability problem is undecidable already for regular string constraints without transducers (or, equivalently, when only T_{id} is allowed) [9]. To circumvent undecidability, acyclic fragment of string constraints were considered in [9]. Formally, let $x < y$ if $(x, Y) \in \text{Rel}$ with $(y, T) \in Y$ for some transducer T . The string constraint is acyclic if $<$ is acyclic. For the acyclic fragment without transducers, satisfiability was shown in [9] to be NEXPTIME-complete. The lower bound already holds for regular acyclic string constraints without transducers.

We study the satisfiability problem for acyclic string constraints in the presence of transducers. Our main results are:

► **Theorem 5.** *Satisfiability problem for acyclic context-free string constraints with transducers is 2NEXPTIME-complete.*

► **Theorem 6.** *Satisfiability problem for acyclic regular string constraints with transducers is NEXPTIME-complete.*

► **Remark 7.** Our result shows an interesting contrast with string equations (with equality instead of subword order in relational constraints). Satisfiability of string equations (with concatenation, no shuffle) is decidable, when regular membership constraints are allowed. Adding transducers on top however render the satisfiability undecidable. In our setting, where subword order is used instead of equality, adding transducers to the acyclic fragment retains decidability.

► **Remark 8.** Without transducers, regular and context-free string constraints have the same complexity. In the presence of transducers they are in different complexity classes.

► **Remark 9.** It was shown in [9] that concatenation can be expressed by shuffle. This simulation is only linear and furthermore it preserves acyclicity. Thus our complexity upper bounds already hold for string constraints which uses the more popular concatenation operation instead of shuffle. Interestingly, the lower bounds in Theorem 5 and Theorem 6 already hold for the variant without shuffle.

In Section 4 and Section 5 we prove the lower bound and upper bound claimed in Theorem 5 respectively. The proof of Theorem 6, as well as other missing details can be found in the full version of this paper [10]. In Section 6, we discuss some implications of our results and conclude.

4 2NEXPTIME Hardness

We prove the hardness by giving a reduction from a bounded variant of the PCP problem that is 2NEXPTIME-complete.

4.1 (Double-exponentially) Bounded PCP problem

In this decidable variant of the PCP problem, we are also given a parameter ℓ as part of the input in unary, and we ask whether there is a solution of length 2^{2^ℓ} . Formally the problem is stated as follows.

► **Definition 10** ((Double-exponentially) Bounded PCP problem (2eBPCP)).

Input: $(\Sigma_1, \Sigma_2, f, g, \ell)$ where Σ_1 and Σ_2 are two disjoint finite alphabets, $f, g : \Sigma_1 \rightarrow \Sigma_2^*$ are two functions which naturally extend to a homomorphism from $\Sigma_1^* \rightarrow \Sigma_2^*$, and $\ell \in \mathbb{N}$ is a natural number.

Question: Is there a word $w \in \Sigma_1^+$ with $|f(w)| = 2^{2^\ell}$ and $f(w) = g(w)$?

The above problem is 2NEXPTIME-complete. The proof can be found in the full version [10]. If the problem asked for the length of $f(w)$ to be ℓ , it would be NP-complete [31], and if it was 2^ℓ it would be NEXPTIME-complete [9].

► **Theorem 11.** (Double-exponentially) Bounded PCP problem is 2NEXPTIME-complete.

4.2 Towards a reduction

Our idea is to use 4 variables x_1, x_2, x_f, x_g . The membership constraint for x_f is a PDA for the language $L_f = \{w \cdot \#^* \cdot f(w^r) \mid w \in \Sigma_1^*\}$, and that for x_g is a PDA for the language $L_g = \{w \cdot \#^* \cdot g(w^r) \mid w \in \Sigma_1^*\}$. Recall that w^r denotes the reverse of w , and $\#$ is a special symbol not in Σ_1 or Σ_2 . Suppose x_1 and x_2 are constrained to the language $\Sigma_1^m \#^n \Sigma_2^{2^{2^\ell}}$ such that $m + n = 2^{2^\ell}$, by polynomial-sized constraints. Then with the relational constraints 1) $x_1 \preceq x_f$ 2) $x_f \preceq x_g$ and 3) $x_g \preceq x_2$, we will achieve our reduction. Recall that $x \preceq y$ is a short hand for $x \preceq \text{Shuffle}(\{(y, T_{\text{id}})\})$. Indeed these constraints are satisfiable if and only if the 2eBPCP has a solution.

Notice that our constraints for x_1 and x_2 requires *counting* exactly 2^{2^ℓ} . This is not possible with a polynomial-sized PDA. In the above paragraph we did not use transducers either. Without transducers, the satisfiability problem of string constraints is not 2NEXPTIME-hard, it is indeed in NEXPTIME[9].

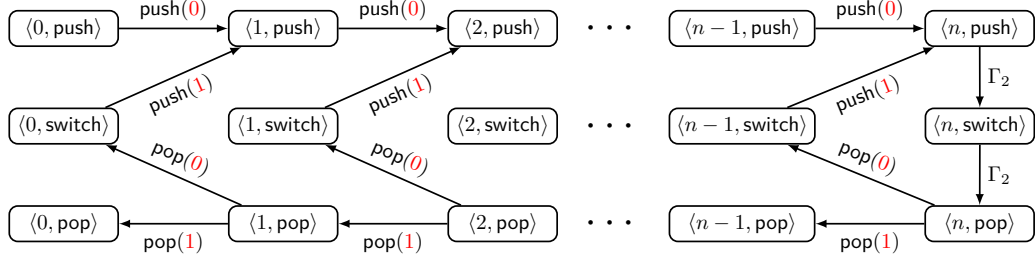
However, with the help of ℓ many transducers (or FSA) of size $\mathcal{O}(\ell)$ we can have a PDA that counts 2^{2^ℓ} . We will describe this technique with PDA and DFA in the next subsection, and in the following subsection using this idea, we complete the reduction.

4.3 Counting 2^{2^ℓ} using one PDA and ℓ DFA

Let $\Gamma_1 = \{0, 1, \text{inc}, \text{dec}\}$, and let Γ_2 be another finite alphabet disjoint from Γ_1 . Our objective is to come up with a PDA A and ℓ DFAs A_1, A_2, \dots, A_ℓ over the alphabet $\Gamma_1 \cup \Gamma_2$, each of size $\mathcal{O}(\ell)$ such that any word accepted by all of them (i.e., in $\cap_i L(A_i) \cap L(A)$) has 2^{2^ℓ} occurrences of letters from Γ_2 .

First we give a PDA with $3n + 3$ states and stack symbols $\{0, 1\}$ that accepts $(\Gamma_2)^{2^{n+1}}$.

▷ **Claim 12.** There is a PDA with $3n + 3$ states and stack symbols $\{0, 1\}$ with stack-height never exceeding n that accepts $(\Gamma_2)^{2^{n+1}}$.



■ **Figure 2** A PDA with $3n + 3$ states that accepts $(\Gamma_2)^{2^{n+1}}$.

Proof. Such a PDA is depicted in Figure 2. In this PDA the stack height never exceeds n . The PDA has three modes - a **push** mode where it keeps pushing **0**s until the stack height is n , a **pop** mode where it keeps popping the symbol **1**, and a **switch** mode that switches from a **pop** mode to **push** mode by replacing a **0** on the top of the stack by a **1**. The states then represent the current stack height and the mode.

When the PDA is in the state $\langle n, \text{push} \rangle$, the stack contents represents an n -bit binary number. At this point it reads two symbols from Γ_2 and goes to the state $\langle n, \text{pop} \rangle$. From there, it does a sequence of transitions such that the next time it reaches $\langle n, \text{push} \rangle$, the binary number in the stack would be incremented. For this it replaces the **01** ^{m} suffix with a **10** ^{m} .

The initial state is $\langle 0, \text{push} \rangle$. The first time it reaches $\langle n, \text{push} \rangle$, the stack content would be **0** ^{n} . The last time it reaches $\langle n, \text{push} \rangle$ (or $\langle n, \text{pop} \rangle$) the stack content would be **1** ^{n} , and from there it reaches $\langle 0, \text{pop} \rangle$ by popping the entire stack. The state $\langle 0, \text{pop} \rangle$ is accepting. Note that this PDA reaches the state $\langle n, \text{push} \rangle$ exactly once for every n -bit binary number, and each time it reaches $\langle n, \text{push} \rangle$ it reads two symbols from Γ_2 . Thus any accepting run will read 2^{n+1} Γ_2 symbols. \triangleleft

If $n = 2^\ell - 1$ we will be able to get 2^{2^ℓ} length words from Γ_2 as we wanted. However, we are allowed to use only $\mathcal{O}(\ell)$ states. To overcome this, we will use ℓ length binary numbers to indicate the current stack height. We then use ℓ DFAs, one for each bit, to update the binary numbers representing the stack height as required. We will next describe the ℓ DFAs that succinctly record the stack height. We then give a PDA that, along with these ℓ DFAs, accepts words with 2^{2^ℓ} occurrences of letters from Γ_2 .

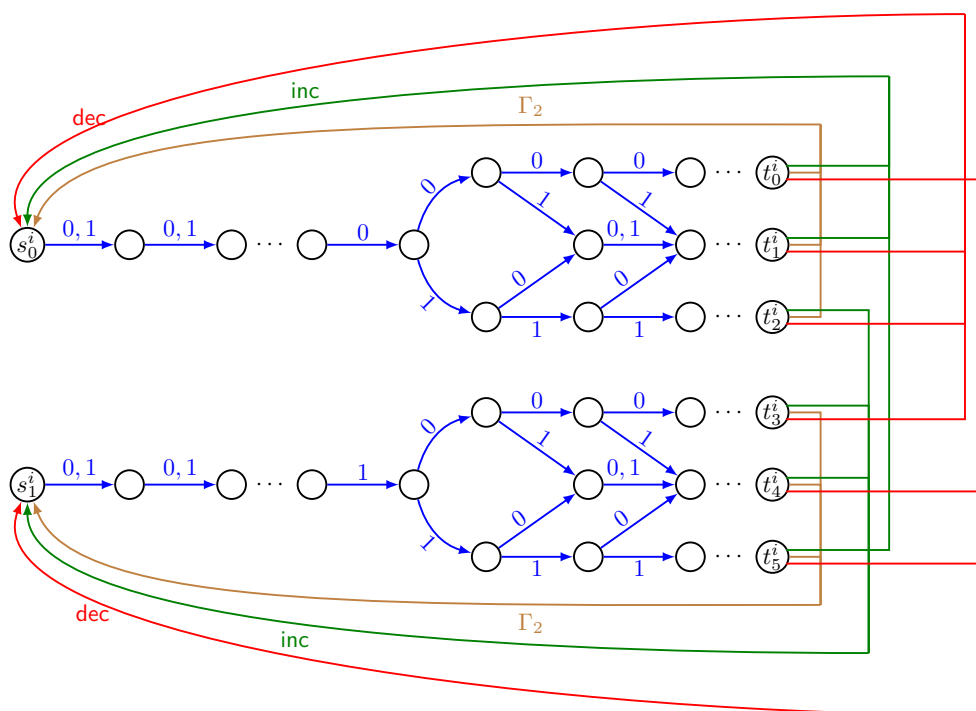
Let $\{\text{inc}, \text{dec}\}$ disjoint from Γ_2 be the increment and the decrement operators on integers. Further, we may treat symbols from Γ_2 as “keep unchanged” operators. That is, $\text{inc}(n) = n + 1$, $\text{dec}(n) = n - 1$ and $a(n) = n$ for all $a \in \Gamma_2$. Consider the following language over the alphabet $\Gamma = \{0, 1, \text{inc}, \text{dec}\} \cup \Gamma_2$ of alternating sequences of ℓ -bit numbers and operators, where each operator when applied on the previous number gives the next number. Here, the binary numbers are written with the most-significant bit on the left. That is $\text{val}(b_1 b_2 \dots b_\ell) = \sum_i b_i \times 2^{\ell-i}$.

$$L_\ell = \{n_0 o_1 n_1 o_2 n_2 \dots o_k n_k \mid \begin{array}{l} k \geq 0, n_i \in (0 + 1)^\ell \text{ for all } i : 0 \leq i \leq k \\ o_i \in \{\text{inc}, \text{dec}\} \cup \Gamma_2 \text{ for all } i : 0 < i \leq k \\ \text{val}(n_i) \equiv o_i(\text{val}(n_{i-1})) \pmod{2^\ell} \text{ for all } i : 0 < i \leq k \end{array}\}$$

▷ **Claim 13.** There are ℓ DFAs B_1, B_2, \dots, B_ℓ , each with $\mathcal{O}(\ell)$ states such that $L_\ell = \bigcap_i L(B_i)$.

Proof. We describe the ℓ DFAs B_1, B_2, \dots, B_ℓ below.

The i th DFA B_i guarantees that the i th bit takes the correct value. This DFA is depicted in the Figure 3. The automaton has two disconnected “forks” (the top one starting at s_0^i and the bottom one starting at s_1^i). In the top fork, the i th bit read is always **0**, and in the



■ **Figure 3** The automaton B_i . The transitions on $\{0, 1\}$ are depicted in blue. The transitions on Γ_2 (resp. inc , dec) are depicted in brown (resp. green, red).

bottom fork the i th bit read is always 1. Consider $n_{j-1}o_jn_j$ occurring in the above sequence. Let $n_{j-1} = b_1b_2 \dots b_\ell$ and let $n_j = b'_1b'_2 \dots b'_\ell$. If o_j is inc , the i th bit b_i is toggled ($b'_i \neq b_i$) iff $b_m = 1$ for all $m : m > i$. If o_j is dec , the i th bit b_i is toggled ($b'_i \neq b_i$) iff $b_m = 0$ for all $m : m > i$. If $o_j \in \Gamma_2$ the i th bit is never toggled. The initial states are s_0^i and s_1^i , and the accepting states are t_0^i, \dots, t_5^i . Clearly $L_\ell = \bigcap_i L(B_i)$. \triangleleft

However, for succinctly simulating the PDA given in Figure 2, we need the ℓ DFAs to faithfully reflect the stack height. For this we consider a slight modification of L_ℓ .

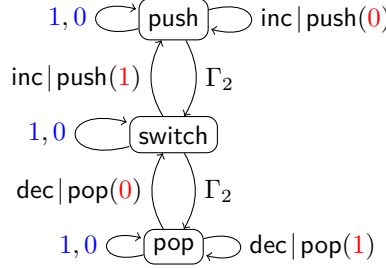
$$L'_\ell = \{n_0o_1n_1o_2n_2 \dots o_kn_k \mid \begin{aligned} &k \geq 0, n_i \in (0+1)^\ell \text{ for all } i : 0 \leq i \leq k \\ &o_i \in \{\text{inc}, \text{dec}\} \cup \Gamma_2 \text{ for all } i : 0 < i \leq k \\ &\text{val}(n_i) \equiv o_i(\text{val}(n_{i-1})) \text{ for all } i : 0 < i \leq k \\ &n_0 = 0^\ell = n_k, \text{ if } o_i = \text{inc} \text{ then } n_{i-1} \neq 1^\ell \\ &\text{if } o_i = \text{dec} \text{ then } n_{i-1} \neq 0^\ell \end{aligned}\}$$

This ensures that the PDA starts and ends with an empty stack. Further inc after 1^ℓ and dec after 0^ℓ are forbidden. Otherwise, the value will not faithfully represent the stack height.

\triangleright **Claim 14.** There are ℓ DFAs A_1, A_2, \dots, A_ℓ , each with $\mathcal{O}(\ell)$ states such that $L'_\ell = \bigcap_i L(A_i)$.

Proof. The states of A_i are exactly those of B_i . For $i \geq 2$, the transitions of A_i is exactly the same as that of B_i . The transitions for A_1 is obtained by removing two transitions from that of B_1 , namely the outgoing inc transition from t_5^1 and the outgoing dec transition from t_0^1 . For all $i \geq 1$, the initial state of A_i is s_0^i and the final state is t_0^i . \triangleleft

▷ Claim 15. There is a PDA A with 3 states and stack symbols $\{0, 1\}$ such that $L(A) \cap L'_\ell$ when projected to Γ_2 is exactly $(\Gamma_2)^{2^{2^\ell}}$.



■ **Figure 4** The PDA A . This PDA and the ℓ DFAs together faithfully encode the accepting runs of the PDA in Figure 2 with $n = 2^\ell - 1$. Thus they accept words with exactly 2^{2^ℓ} occurrences of Γ_2 .

Proof. The PDA A is depicted in Figure 4. The three states represent the three modes of the PDA in Figure 2. The ℓ DFAs will guarantee that we start with the number 0^ℓ . Because of A_1 , the PDA cannot take the inc transition from the state $\langle \text{push} \rangle$ immediately after the number 1^ℓ . It will have to read a Γ_2 symbol and move to the state $\langle \text{switch} \rangle$. The PDA will loop in this state once by reading the same number (1^ℓ) as mandated by the Γ_2 transitions of the DFAs. From this state, again inc is disabled by A_1 , and hence the PDA will read another Γ_2 symbol and go to the state $\langle \text{pop} \rangle$. The PDA will read 1^ℓ staying in the state $\langle \text{pop} \rangle$ after which it can take a dec transition. ◁

4.4 Completing the reduction

Before giving the reduction, let us first define a PDA and the transducers that we use in the reduction. Let P_1 be the PDA in Claim 15 with $\Gamma_2 = \Sigma_1 \cup \{\#\}$. Let P_2 be the PDA for $L(P_1) \cap (\Sigma_1^* \#\Sigma_1^*)$. Let P_3 be the PDA in Claim 15 with $\Gamma_2 = \Sigma_2$. Let P_4 be the PDA for $L(P_2) \cdot L(P_3)$. Let T_1, \dots, T_ℓ be ℓ transducers. The input automaton of T_i is exactly the DFA A_i from Claim 14 with $\Gamma_2 = \Sigma_1 \cup \Sigma_2 \cup \{\#\}$. The output of the transducer T_i on every transition is ϵ .

Now we are ready to give the reduction. Let $\mathcal{P} = (\Sigma_1, \Sigma_2, f, g, \ell)$ be an input to a 2eBPCP problem. We describe how to obtain a string constraint $C_{\mathcal{P}}$ from this. We use the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Gamma_1 \cup \{\#\}$, and the variable set $\mathcal{V} = \{x_0, x_1, x_2, x_f, x_g\}$. Next we define the membership constraints Mem . Let $\text{Mem}(x_0) = A_\epsilon$ where A_ϵ is an NFA for $\{\epsilon\}$. We have $\text{Mem}(x_1) = P_4, \text{Mem}(x_2) = P_4$. Now we need to augment the language of x_f and x_g to also account for the letters from Γ_1 , which can be achieved by adding Γ_1 self-loops in all the states in the PDA for L_f and L_g respectively. Thus the language for x_f is $\mathfrak{W}(L_f, \Gamma_1^*)$, where $\mathfrak{W}(L_f, \Gamma_1^*) = \{w \mid w \in \text{Shuffle}(u, v), u \in L_f, v \in \Gamma_1^*\}$. Similarly the language for x_g is $\mathfrak{W}(L_g, \Gamma_1^*)$. Let P_5 and P_6 be PDAs recognizing $\mathfrak{W}(L_f, \Gamma_1^*)$, and $\mathfrak{W}(L_g, \Gamma_1^*)$ respectively. We have $\text{Mem}(x_f) = P_5$ and $\text{Mem}(x_g) = P_6$. Let Rel be the following relational constraints: 1) $x_0 \preceq T_i(x_1)$, for all $i : 1 < i < \ell$, 2) $x_0 \preceq T_i(x_2)$, for all $i : 1 < i < \ell$, 3) $x_1 \preceq x_f$, 4) $x_f \preceq x_g$ and 5) $x_g \preceq x_2$. We have $2\ell + 3$ relational constraints. Further $\text{multiplicity}(x_1) = \ell = \text{multiplicity}(x_2)$ in our construction. Let $C_{\mathcal{P}} = (\Sigma, \mathcal{V}, \text{Mem}, \text{Rel})$.

▷ Claim 16. The string constraint $C_{\mathcal{P}}$ is satisfiable if and only if the 2eBPCP instance \mathcal{P} has a solution.

Notice that we construct $C_{\mathcal{P}}$ from \mathcal{P} in polynomial time, and it is acyclic. Hence, it follows that the satisfiability checking of acyclic string constraints is 2NEXPTIME-hard, proving the lower bound of Theorem 5.

5 Satisfiability is in 2NEXPTIME

We will show that if an acyclic string constraint is satisfiable, then there is a satisfying assignment of double exponential size.

► **Theorem 17** (Small model property). *Let C be a satisfiable acyclic string constraint. Then C has a satisfying assignment σ such that $\text{len}(\sigma(x)) \leq B$ where B is*

$$2^{2^{\mathcal{O}(m \log t + \log p + \log k)}}$$

where

- $m = \max_{x \in \mathcal{V}} \text{multiplicity}(x)$, the maximum multiplicity of any variable,
- $t = \max_{T \in \text{TRSET}(C)} \text{state-size}(T)$, the maximum number of states of any transducer,
- $p = \max_{P \in \text{AUTSET}(C)} \text{state-size}(P)$, the maximum number of states (and stack symbols) of any automaton.
- $k = |\mathcal{V}|$, the number of variables.

Our aim, towards a 2NEXPTIME procedure, is to non-deterministically guess an assignment of size at most double exponential, and check that it satisfies the Conditions 1 and 2 (see Definition 2). However, Condition 2 uses more existentially quantified variables, and it is not evident that verifying Condition 2 can be done within the complexity limits. Towards this, we define an extended assignment which considers the values given to these existentially quantified variables as well, and show that every word used in this extended assignment is of length at most B . We define the extended assignment and the related notions and notations, and state the small model property for the extended assignment.

Recall that $\text{multiplicity}(x)$ denotes the number of times a variable occurs on the RHS of the constraints. In each such occurrence, the variable occurs in a pair along with a transducer (of the form (x, T)), which belongs to the RHS of a constraint from Rel of the form (y, Y) . Let us fix some enumeration of these occurrences, and define the respective transducer and constraint of the i th occurrence of x by T_i^x and $\text{constraint}(x, i)$. Now, as per Condition 2, there are words (output words of the respective transducers), that witness the transduction. For every $x \in \mathcal{V}$ and $i \in [\text{multiplicity}(x)]$, let o_i^x be a new variable. This variable is intended to take as value a witness word for the output of the transducer T_i^x on the word provided by x , so that the constraint $\text{constraint}(x, i)$ is satisfied. Let $\widehat{\mathcal{V}}(C)$, or simply $\widehat{\mathcal{V}}$ when C is clear from the context, contain the output variables in addition to the original variables. That is, $\widehat{\mathcal{V}} = O \cup \mathcal{V}$, where $O = \{o_i^x \mid x \in \mathcal{V}, i \in [\text{multiplicity}(x)]\}$. An extended assignment $\widehat{\sigma} : \widehat{\mathcal{V}} \rightarrow \Sigma^*$ satisfies a string constraint C if

E1 $\widehat{\sigma}(x) \in \text{Mem}(x)$ for all $x \in \mathcal{V}$

E2 $(\widehat{\sigma}(x), \widehat{\sigma}(o_i^x)) \in R(T_i^x)$, for all $x \in \mathcal{V}, i \in [\text{multiplicity}(x)]$

E3 For every $(y, Y) \in \text{Rel}$, we have $\widehat{\sigma}(y) \preceq \text{Shuffle}(\widehat{\sigma}(Y))$ where $\widehat{\sigma}(Y)$ is an overloaded notation for the multiset

$$\{\{\widehat{\sigma}(o_i^x) \mid x \in \mathcal{V}, i \in [\text{multiplicity}(x)], \text{constraint}(x, i) = (y, Y)\}\}. \quad (1)$$

We will actually prove the small model property for the extended assignments.

► **Lemma 18.** *Let C be a satisfiable acyclic string constraint. Then C has a satisfying extended assignment $\widehat{\sigma}$ such that $\text{len}(\widehat{\sigma}(x)) \leq B$ for all $x \in \mathcal{V}(C)$, and $\text{len}(\widehat{\sigma}(y)) \leq 2ctB$ for all $y \in O$. The parameters B and t are as defined in Theorem 17, and $c = \max\{\text{len}(\text{out}(tr)) \mid tr \text{ is a transition of } T, \text{ and } T \in \text{TRSET}(C)\}$.*

With this, our non-deterministic procedure guesses an extended assignment and verifies that it satisfies the conditions 1, 2 and 3. In fact, checking whether a given word w is a subword of some word in $\text{Shuffle}(W)$ where W is a finite multiset of words is NP-complete [9, 24]. It remains to prove Lemma 18.

Proof of Lemma 18. Consider an acyclic string constraint C . Recall that we write $x < y$ if $(x, Y) \in \text{Rel}$ with $(y, T) \in Y$ for some $T \in \text{TRSET}(C)$. Consider a topological sorting of the variables respecting the relation $<$, say x_1, x_2, \dots, x_k . Note that x_1 does not appear in the RHS of any subword order constraint (in other words, $\text{multiplicity}(x_1) = 0$). If (x_i, T) appears on the RHS of any constraint for some T , then the LHS of that constraint is x_j for some $j < i$.

Suppose C is satisfiable. Let $\widehat{\sigma}_0$ be a satisfying extended assignment. In order to get the $\widehat{\sigma}$ as per Lemma 18, we will construct a sequence of k extended assignments, each progressively modifying the previous one until we reach our goal. That is, we will construct the sequence, $\widehat{\sigma}_0, \widehat{\sigma}_1, \dots, \widehat{\sigma}_k = \widehat{\sigma}$ such that for each $i \in [k]$

- 11 $\widehat{\sigma}_i$ is satisfiable. That is, 1) membership constraints are satisfied (Condition 1), 2) transductions are accepted by the transducers (Condition 2), and 3) the relational constraints are satisfied (Condition 3).
- 12 for all $j : j < i$, $\text{len}(\widehat{\sigma}_i(x_j)) \leq B_j$ and for each $\ell \in \text{multiplicity}(x_j)$, $\text{len}(\widehat{\sigma}_i(o_\ell^{x_j})) \leq 2ctB_j$. We define B_n as follows. $B_1 = 2p^3$, and for $n > 1$, $B_n = 2m \cdot t^{2m} \cdot p^3 \cdot B_{n-1} \cdot 2p^3 t^{2m}$.

Note that, B_n increases with n and B_k is at most B .

Base cases. We consider $\widehat{\sigma}_0$ and $\widehat{\sigma}_1$ as base cases. For $\widehat{\sigma}_0$, it is given to be satisfiable, and Condition 2 above holds vacuously.

Towards constructing $\widehat{\sigma}_1$, consider the variable x_1 , and a context-free grammar G_1 for $\text{Mem}(x_1)$ in Chomsky Normal Form with at most p^3 non-terminals [15]. Note that G_1 can be constructed in polynomial time. Since $\widehat{\sigma}_0$ is satisfying, the word $w_1 = \widehat{\sigma}_0(x_1)$ has a valid parse tree in G_1 . If a non terminal repeats in any leaf to root path in this tree, say at node n_1 and node n_2 with n_2 an ancestor of n_1 , then we can shrink the parse tree (pump down) by replacing the subtree rooted at n_2 by the subtree rooted at n_1 to get a smaller parse tree of a smaller word in the language. Furthermore, this smaller word will be a subword of w_1 . Consider a shrinking of the parse tree of w_1 which cannot be shrunk any further. This tree has size at most $2p^3$, and hence its yield \widehat{w}_1 satisfies Condition 2. Setting x_1 to \widehat{w}_1 will also satisfy Condition 1. Further, note that there are no output variables corresponding to x_1 .

Hence we get $\widehat{\sigma}_1: \widehat{\sigma}_1(x) = \begin{cases} \widehat{w}_1 & \text{if } x = x_1 \\ \widehat{\sigma}_0(x) & \text{otherwise.} \end{cases}$

Inductive Step. Now, for the inductive case, assume we have constructed $\widehat{\sigma}_{i-1}$. We will describe how to obtain $\widehat{\sigma}_i$. Let G_i be the context-free grammar in Chomsky Normal Form for $\text{Mem}(x_i)$ with p^3 non-terminals. We will basically do a ‘‘conservative’’ pumping down of $w_i = \widehat{\sigma}_{i-1}(x_i)$, which ensures that the constraints are still satisfied, which we explain below.

Challenges. In order to bound the length of $\widehat{\sigma}_i(x_i)$ we may consider subwords $w'_i \preceq w_i$, so that the constraints in which x_i appear on the left continue to be satisfied. In addition, such a subword w'_i must not only satisfy the membership constraint ($w'_i \in \text{Mem}(x_i)$) but also admit the specified transductions – that is, for all $\ell \in [\text{multiplicity}(x_i)]$, we must have $(w'_i, u'_\ell) \in R(T_\ell^{x_i})$ for some u'_ℓ . Furthermore, a mere existence of such a u'_ℓ is not sufficient – consider the constraint $(x_j, Y) = \text{constraint}(x_i, \ell)$ and let $\widehat{\sigma}_{i-1}(x_j) = w_j$ and $\widehat{\sigma}_{i-1}(Y) = U$ with $\widehat{\sigma}_{i-1}(o_\ell^{x_i}) = u_\ell$ (recall, $\widehat{\sigma}(Y)$ is defined in Equation 1). Since $\widehat{\sigma}_{i-1}$ is satisfying we know that $w_j \preceq \text{Shuffle}(U)$. However, it need not be the case that $w_j \preceq \text{Shuffle}(U \setminus \{u_\ell\} \cup \{u'_\ell\})$. Hence we need to find a suitable u'_ℓ such that $w_j \preceq U \setminus \{u_\ell\} \cup \{u'_\ell\}$. One way to ensure this, is by insisting that u'_ℓ provides the same “witnessing subword” that u_ℓ provided. We formalise this notion of “witnessing subword” below.

We give two equivalent definitions for $w \preceq \text{Shuffle}(U)$.

▷ **Claim 19.** Let w be a word and $U = \{\{u'_1, u'_2 \dots u'_n\}\}$ be a multiset of words. The following statements are equivalent.

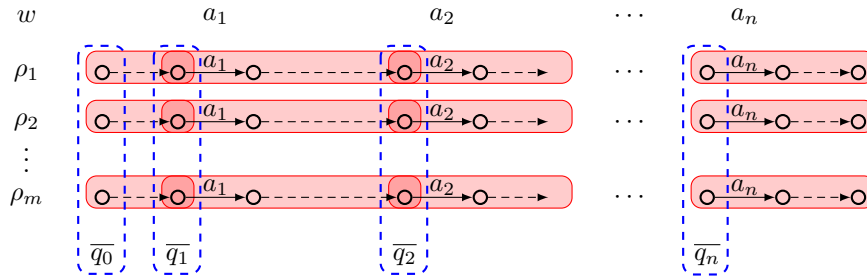
1. There exists w' : 1) $w' \in \text{Shuffle}(U)$ and 2) $w \preceq w'$.
2. There exist $u'_1, u'_2 \dots u'_n$: 1) $w \in \text{Shuffle}(\{\{u'_1, u'_2 \dots u'_n\}\})$ and 2) $u'_i \preceq u_i$.

We refer to v_1, \dots, v_n as the *witnessing subwords* of u_1, \dots, u_n for $w \preceq \text{Shuffle}(U)$. Thus, for the inductive case, we need to find good $w'_i, u'_1, \dots, u'_{\text{multiplicity}(x_i)}$ such that

1. $|w'_i| \leq B_i$
2. for each ℓ , $|u'_\ell| \leq 2 \cdot t \cdot B_i$,
3. $w'_i \in \text{Mem}(x_i)$,
4. $(w'_i, u'_\ell) \in R(T_\ell^{x_i})$ for each ℓ , and
5. $v_\ell \preceq u'_\ell$ where v_ℓ is a witnessing subword of $u_\ell = \widehat{\sigma}_{i-1}(o_\ell^{x_i})$ for the relation $\widehat{\sigma}_{i-1}(y) \preceq \text{Shuffle}(\widehat{\sigma}_{i-1}(Y))$ letting $(y, Y) = \text{constraint}(x_i, \ell)$. (Note that v_ℓ exists because $\widehat{\sigma}_{i-1}$ is satisfiable by induction hypothesis.)

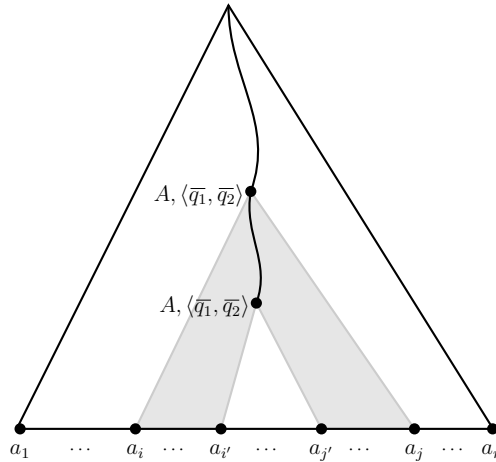
Block decomposition. Towards the above goal, let us consider w_i and $\rho_1 \dots \rho_m$ where $m = \text{multiplicity}(x_i)$ and for $\ell : 1 \leq \ell \leq m$, ρ_ℓ is an accepting run of the transducer $T_\ell^{x_i}$ on the input w_i producing $u_\ell = \widehat{\sigma}_{i-1}(o_\ell^{x_i})$. This is depicted in Figure 5. We factorize each ρ_ℓ into blocks. The number of blocks is exactly $n + 1$, where $n = |w_i|$. For $j > 0$, the j th block contains the transition on the j th letter of w_i , followed by all the trailing ϵ -input transitions. The very first block (block 0), contains the leading ϵ -input transitions if present. Now, we decompose the output of ρ_ℓ according to the blocks. That is, $u_\ell = u_\ell^0 u_\ell^1 \cdot u_\ell^2 \dots u_\ell^n$. Next we want to identify the subword of w_i (and subruns of ρ_ℓ) that needs to be preserved.

Identifying crucial blocks and positions. Consider v_ℓ , the witnessing subword of u_ℓ for $\widehat{\sigma}_{i-1}(y) \preceq \text{Shuffle}(\widehat{\sigma}_{i-1}(Y))$ where $(y, Y) = \text{constraint}(x_i, \ell)$. Fix an embedding of v_ℓ in u_ℓ . If this embedding is incident on the factor u_ℓ^j for $j > 0$, we will mark the the j th block as well as the j th letter of w_i as *crucial*. Since $|\widehat{\sigma}_{i-1}(y)| \leq B_{i-1}$ (by induction hypothesis), the number of crucial blocks in ρ_ℓ is at most B_{i-1} . Hence the number of crucial positions in w_i is at most $m \times B_{i-1}$ where $m = \text{multiplicity}(x_i)$. Notice that if we shrink w_i to a subword that 1) preserves the crucial positions, 2) preserves membership in $\text{Mem}(x_i)$ and 3) yields subruns of ρ_ℓ that preserves the crucial blocks, then the satisfiability would be preserved. Our next aim is to obtain such a shrinking, which is sufficiently small to also satisfy the length requirements.



■ **Figure 5** The figure describes the block decomposition of the runs of transducers. The very first block includes the sequence of ϵ -transitions (if any) before the first letter of the input is read. Every other block includes a transition on input letter followed by a sequence of ϵ transitions. Blocks here are represented as overlapping rectangles coloured in red. The transition on input is represented as a solid arrow and a sequence of ϵ transitions is represented as a dashed arrows.

Annotated parse trees. Consider a grammar G_i for $\text{Mem}(x_i)$ in Chomsky Normal Form and a parse tree of w_i in G_i . Annotate the nodes of this parse-tree by pairs of m -tuple of states. The m -tuple of states \bar{q}_j correspond to the states of the transducers at the boundary between $(j - 1)$ th block and j th block. A node is annotated with $\langle \bar{q}_{j-1}, \bar{q}_j \rangle$ if the yield of the subtree rooted at that node generates the factor of w_i from j th letter to j' th letter (for some $j' \geq j$). Notice that some of the leaves are marked as crucial. We will mark an internal node as crucial if it is the least common ancestor of two crucial nodes.



■ **Figure 6** The figure illustrates the annotations of a nodes and pumping down in a parse tree.

Shrinking the parse tree. Now, if there are two nodes n_1 and n_2 in this tree such that 1) both have the same annotated non-terminal, 2) n_1 is an ancestor of n_2 , 3) there are no crucial nodes in the path from n_1 to n_2 , then we replace the subtree rooted at n_1 with the subtree rooted at n_2 (pumping down). This is illustrated in Figure 6. We repeat this until no more pumping down is possible. The yield of this shrunk parse tree is the required word w'_i . Let us analyse the size of w'_i . Any path without a crucial node is of length at most $p^3 t^{2m}$. Hence the *skeleton* of the parse tree that contains all the crucial nodes and the paths from them to the root will be of size at most $2n_C \times p^3 t^{2m}$, where n_C is the number of crucial positions of w_i .

Any sub tree rooted at any of the nodes of the skeleton is of size at most $2^{p^3 t^{2m}}$. Hence the total size of the tree is at most $2n_C \times p^3 t^{2m} \times 2^{p^3 t^{2m}}$. Since $n_C \leq mB_{i-1}$, we have $\text{len}(w'_i) \leq B_i$.

Shrinking the transducer runs. Note that, since the shrinking preserves the annotations, shrinking the ρ_ℓ s appropriately gives us an accepting subrun ρ'_ℓ that preserves the crucial blocks. The number of blocks in ρ'_ℓ is at most B_i . Now, we need to shrink the size of each block as well, in order to satisfy $\text{len}(u'_\ell) \leq 2ctB_i$. For this, consider the witnessing subword of u_ℓ . Note that it is still embedded in $\text{out}(\rho'_\ell)$. If this embedding is incident on the output of a transition we will mark this transition as crucial. Further all the transitions that read a letter from w'_i are also crucial. Note that the number of crucial transitions is at most $B_i + B_{i-1}$. Now, let us shrink the run ρ'_ℓ without losing crucial transitions to get ρ''_ℓ . The number of transitions in ρ''_ℓ is at most $t \times (B_i + B_{i-1} + 1)$ where t is the number of states. Let $u'_\ell = \text{out}(\rho''_\ell)$. Then it is easy to see that $\text{len}(u'_\ell) \leq 2ctB_i$.

Finally, we can give the required $\hat{\sigma}_i$. Below, x_j comes from \mathcal{V} and o_ℓ^x comes from O .

$$\hat{\sigma}_i(x_j) = \begin{cases} w'_i & \text{if } j = i \\ \hat{\sigma}_{i-1}(x_j) & \text{if } j \neq i \end{cases} \quad \hat{\sigma}_i(o_\ell^x) = \begin{cases} u'_\ell & \text{if } x = x_i \\ \hat{\sigma}_{i-1}(o_\ell^x) & \text{otherwise} \end{cases}$$

This establishes the proof of Lemma 18. \blacktriangleleft

A proof of NEXPTIME-completeness for the setting where only regular membership is allowed (proof of Theorem 6) can be found in the full version of this paper [10].

6 Discussions

6.1 Application: Regular abstractions and DFA sizes

For any language L , we define its Parikh image closure ($\Pi(L)$), downward closure ($L \downarrow$) and upward closure ($L \uparrow$) as follows. Let $\Sigma = \{a_1, \dots, a_n\}$. For $w \in \Sigma^*$, we let $p(w) = \langle \text{len}(w)_{a_1}, \dots, \text{len}(w)_{a_n} \rangle$ denote the Parikh image of w , that is, it counts the occurrences of each letter from Σ . Here, by $\text{len}(w)_a$, we mean the number of times a occurs in w .

$$\begin{aligned} \Pi(L) &= \{v \in \Sigma^* \mid \exists w \in L, p(v) = p(w)\} & L \downarrow &= \{v \in \Sigma^* \mid \exists w \in L, v \preceq w\} \\ L \uparrow &= \{v \in \Sigma^* \mid \exists w \in L, w \preceq v\} \end{aligned}$$

Efficient computability of these regular abstractions of languages of infinite state systems is a relevant question for verification and automata theory [14, 12, 46, 13]. It is interesting to see if small automata representing these abstractions can be computed for succinctly given infinite state systems.

We address here the case where a large pushdown system is presented as a small pushdown system and a certain number of finite state automata (referred to as the smaller components). Here the language of the large pushdown system is same as the intersection of the languages of the smaller components. We argue that the lower bound on the size of the regular abstraction holds even when pushdown system is presented succinctly.

For any $n \in \mathbb{N}$, let $L(n)$ be the language over $\Sigma = \{0, 1, a\}$ that accepts the word $w = n_0 o_1 n_1 o_2 n_2 \dots n_k$, where $n_0 = 0^n$, $n_k = 1^n$, $o_1, o_2, \dots, o_k = \text{inc}$, then $\text{len}(w)_{\text{inc}} = 2^n$. From Section 4 we know that we can construct n DFAs B_1, B_2, \dots, B_n such that $\bigcap_i L(B_i) = \{w\}$. Since any DFA recognising the closure of this language requires at least 2^n states, we have the following claim.

▷ **Claim 20.** Given n regular languages as n finite state automata, let L be the language obtained by intersecting the languages of these automata. Then, the minimal DFA for $\Pi(L)$, $L \downarrow$ and $L \uparrow$ can be of exponential size.

Consider the language given in Claim 15 i.e. $L(A) \cap L'_\ell$, since it can recognize words with exactly 2^{2^n} many symbols from Γ_2 , we have the following claim.

▷ **Claim 21.** Given n regular languages as n finite state automata and a pushdown system, let L be the language got by intersecting the languages of these automata. Then, the minimal DFA for $\Pi(L)$, $L \downarrow$ and $L \uparrow$ can be of double exponential size.

6.2 Concatenation instead of Shuffle

In [9] it is shown that shuffle can express concatenation with a polynomial blow-up, but preserving acyclicity. It is interesting to see if the hardness holds in the presence of concatenation alone. Already, in the setting of [9] (no transductions), if acyclicity is not imposed, it is not known whether satisfiability of the regular string constraints is decidable if only concatenation is allowed instead of shuffle. In our setting (in the presence of transducers), it turns out that satisfiability is undecidable. We show this by modifying the reduction in [9].

Let $\mathcal{P} = (\Sigma_1, \Sigma_2, f, g)$ be a given PCP instance, let $\mathcal{L}_u = (\{i \cdot f(i) \mid i \in \Sigma_1\})^*$, $\mathcal{L}_v = (\{i \cdot g(i) \mid i \in \Sigma_1\})^*$, $\mathcal{L}_i = \Sigma_1^+$ and $\mathcal{L}_s = \Sigma_2^*$. Notice that all these four languages are regular, let $\mathcal{A}_u, \mathcal{A}_v, \mathcal{A}_i$ and \mathcal{A}_s be their corresponding NFA. Then the required string constraint is $(\Sigma, \mathcal{V}, \text{Mem}, \text{Rel})$, where $\mathcal{V} = \{u, v, i, s\}$, for any $x \in \mathcal{V}$, $\text{Mem}(x) = \mathcal{A}_x$. Let $T_\Sigma = \{(a_1 \dots a_n, \Sigma^* a_1 \Sigma^* a_1 \dots \Sigma^* a_n \Sigma^*)\}$ be the transduction that arbitrarily inserts words from Σ . The set Rel is given by

$$\begin{array}{cccc} i \preceq u & s \preceq u & u \preceq T_{\Sigma_1}(s) & u \preceq T_{\Sigma_2}(t) \\ i \preceq v & s \preceq v & v \preceq T_{\Sigma_1}(s) & v \preceq T_{\Sigma_2}(t) \end{array}$$

In the case of acyclic string constraints, one may wonder if the lower bounds hold in a setting where only concatenation is allowed instead of shuffle. This was not discussed in [9]. Infact, in our case, the lower bound holds even when only concatenation is allowed. Notice that, in our 2NEXPTIME-hard reduction, all relational constraints have only one variable in the RHS. Hence, the 2NEXPTIME-hard holds for acyclic pushdown string constraints with transducers, even when shuffle (or even concatenation) is disallowed.

In fact, it is not known whether the lower bounds in [9] hold for the variant with only concatenation instead of shuffle. It is also open whether satisfiability is decidable for the *regular* string constraints without acyclicity restriction when only concatenation is allowed.

7 Conclusions

In this paper, we considered string constraints in the presence of sub-word relation, shuffle operator (which subsumes concatenation [9]) and transducers. We studied this problem for two different kinds of membership constraints, namely regular and context free. We showed that in the case when only regular membership constraints are involved, the problem is NEXPTIME-complete. Whereas, when context-free membership constraints are involved, the problem is 2NEXPTIME-complete. Towards the hardness proof, we showed how to count exactly 2^n using n finite state automata each of size $\mathcal{O}(n)$. As a consequence of this result, we also obtained a lower bound for any regular representation of the upward closure, downward closure and Parikh image closure of the intersection of the language of n finite state automata.

Similarly, we showed that we can count exactly 2^{2^n} using a pushdown automaton and n finite state automata, each of size $\mathcal{O}(n)$. With this, we also obtained a lower bound for any regular representation of the upward closure, downward closure and Parikh image closure of the intersection language of a pushdown and n finite state automata.

References

- 1 A. Parosh Abdulla, F. Mohamed Atig, Yu-Fang Chen, Diep Phi Bui, Lukáš Holík, Ahmed Rezine, and Philipp Rümmer. Trau : SMT solver for string constraints. In *Proceedings of the 18th Conference on Formal Methods in Computer-Aided Design*, pages 165–169. FMCAD Inc., 2019. doi:10.23919/FMCAD.2018.8602997.
- 2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukáš Holík, Denghang Hu, Wei-Lun Tsai, Zhilin Wu, and Di-De Yen. Solving not-substring constraint with flat abstraction. In Hakjoo Oh, editor, *Programming Languages and Systems - 19th Asian Symposium, APLAS 2021, Chicago, IL, USA, October 17-18, 2021, Proceedings*, Lecture Notes in Computer Science. Springer, 2021. doi:10.1007/978-3-030-89051-3_17.
- 3 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukáš Holík, Ahmed Rezine, and Philipp Rümmer. Flatten and conquer: A framework for efficient analysis of string constraints. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 602–617, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3062341.3062384.
- 4 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukáš Holík, Ahmed Rezine, and Philipp Rümmer. Flatten and conquer: A framework for efficient analysis of string constraints. *SIGPLAN Not.*, 52(6), 2017. doi:10.1145/3140587.3062384.
- 5 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukáš Holík, Ahmed Rezine, Philipp Rümmer, and Jari Stenman. String constraints for verification. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 150–166. Springer, 2014. doi:10.1007/978-3-319-08867-9_10.
- 6 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukáš Holík, Ahmed Rezine, Philipp Rümmer, and Jari Stenman. Norn: An SMT solver for string constraints. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 462–469. Springer, 2015. doi:10.1007/978-3-319-21690-4_29.
- 7 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Vrunda Dave, and Shankara Narayanan Krishna. On the separability problem of string constraints. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPICs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CONCUR.2020.16.
- 8 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bui Phi Diep, Lukáš Holík, and Petr Janku. Chain-free string constraints. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*. Springer, 2019. doi:10.1007/978-3-030-31784-3_16.
- 9 C. Aiswarya, Soumodev Mal, and Prakash Saivasan. On the satisfiability of context-free string constraints with subword-ordering. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*. ACM, 2022. doi:10.1145/3531130.3533329.
- 10 C Aiswarya, Soumodev Mal, and Prakash Saivasan. Satisfiability of context-free string constraints with subword-ordering and transducers, 2024. arXiv:2401.07996.

- 11 Roberto Amadini. A survey on string constraint solving. *ACM Comput. Surv.*, 55(2):16:1–16:38, 2023. doi:10.1145/3484198.
- 12 Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. On bounded reachability analysis of shared memory systems. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPICs*, pages 611–623. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICs.FSTTCS.2014.611.
- 13 Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. On the reachability analysis of acyclic networks of pushdown systems. In Franck van Breugel and Marsha Chechik, editors, *Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*. Springer, 2008. doi:10.1007/978-3-540-85361-9_29.
- 14 Mohamed Faouzi Atig, Dmitry Chistikov, Piotr Hofman, K. Narayan Kumar, Prakash Saivasan, and Georg Zetsche. The complexity of regular abstractions of one-counter languages. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 207–216. ACM, 2016. doi:10.1145/2933575.2934561.
- 15 Mohamed Faouzi Atig, K. Narayan Kumar, and Prakash Saivasan. Adjacent ordered multi-pushdown systems. In Marie-Pierre Béal and Olivier Carton, editors, *Developments in Language Theory - 17th International Conference, DLT 2013, Marne-la-Vallée, France, June 18-21, 2013. Proceedings*, volume 7907 of *Lecture Notes in Computer Science*, pages 58–69. Springer, 2013. doi:10.1007/978-3-642-38771-5_7.
- 16 Murphy Berzish, Joel D. Day, Vijay Ganesh, Mitja Kulczynski, Florin Manea, Federico Mora, and Dirk Nowotka. Towards more efficient methods for solving regular-expression heavy string constraints. *Theor. Comput. Sci.*, 2023. doi:10.1016/j.tcs.2022.12.009.
- 17 Murphy Berzish, Vijay Ganesh, and Yunhui Zheng. Z3str3: A string solver with theory-aware heuristics. In *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design, FMCAD '17, Austin, Texas, 2017*. FMCAD Inc. doi:10.23919/FMCAD.2017.8102241.
- 18 J. Richard Büchi and Steven Senger. Definability in the existential theory of concatenation and undecidable extensions of this theory. *Math. Log. Q.*, 34:337–342, 1988.
- 19 Taolue Chen, Yan Chen, Matthew Hague, Anthony W. Lin, and Zhilin Wu. What is decidable about string constraints with the replaceall function. *Proc. ACM Program. Lang.*, 2(POPL):3:1–3:29, 2018. doi:10.1145/3158091.
- 20 Taolue Chen, Alejandro Flores-Lamas, Matthew Hague, Zhilei Han, Denghang Hu, Shuanglong Kan, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Solving string constraints with regex-dependent functions through transducers with priorities and variables. *Proc. ACM Program. Lang.*, 6(POPL), 2022. doi:10.1145/3498707.
- 21 Taolue Chen, Alejandro Flores-Lamas, Matthew Hague, Zhilei Han, Denghang Hu, Shuanglong Kan, Anthony Widjaja Lin, Philipp Rümmer, and Zhilin Wu. Solving string constraints with regex-dependent functions through transducers with priorities and variables. *CoRR*, abs/2111.04298, 2021.
- 22 Taolue Chen, Matthew Hague, Jinlong He, Denghang Hu, Anthony Widjaja Lin, Philipp Rümmer, and Zhilin Wu. A decision procedure for path feasibility of string manipulating programs with integer data type. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 325–342. Springer, 2020. doi:10.1007/978-3-030-59152-6_18.
- 23 Taolue Chen, Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Decision procedures for path feasibility of string-manipulating programs with complex operations. *Proc. ACM Program. Lang.*, 3(POPL), 2019. doi:10.1145/3290362.

- 24 Peter Chini, Jonathan Kolberg, Andreas Krebs, Roland Meyer, and Prakash Saivasan. On the complexity of bounded context switching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPICs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ESA.2017.27.
- 25 John Corcoran, William Frank, and Michael Maloney. String theory. *J. Symb. Log.*, 39(4):625–637, 1974. doi:10.2307/2272846.
- 26 Joel D. Day. Word equations in the context of string solving. In Volker Diekert and Mikhail V. Volkov, editors, *Developments in Language Theory - 26th International Conference, DLT 2022, Tampa, FL, USA, May 9-13, 2022, Proceedings*, Lecture Notes in Computer Science. Springer, 2022. doi:10.1007/978-3-031-05578-2_2.
- 27 Joel D. Day, Vijay Ganesh, Nathan Grewal, and Florin Manea. On the expressive power of string constraints. *Proc. ACM Program. Lang.*, 7(POPL):278–308, 2023. doi:10.1145/3571203.
- 28 Joel D. Day, Vijay Ganesh, Paul He, Florin Manea, and Dirk Nowotka. The satisfiability of word equations: Decidable and undecidable theories. In Igor Potapov and Pierre-Alain Reynier, editors, *Reachability Problems*, pages 15–29, Cham, 2018. Springer International Publishing.
- 29 Diego Figueira, Artur Jez, and Anthony W. Lin. Data path queries over embedded graph databases. In Leonid Libkin and Pablo Barceló, editors, *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 189–201. ACM, 2022. doi:10.1145/3517804.3524159.
- 30 Vijay Ganesh, Mia Minnes, Armando Solar-Lezama, and Martin C. Rinard. Word equations with length constraints: What’s decidable? In Armin Biere, Amir Nahir, and Tanja E. J. Vos, editors, *Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*. Springer, 2012. doi:10.1007/978-3-642-39611-3_21.
- 31 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.
- 32 Matthew Hague, Anthony Widjaja Lin, and C.-H. Luke Ong. Detecting redundant CSS rules in HTML5 applications: a tree rewriting approach. In Jonathan Aldrich and Patrick Eugster, editors, *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, pages 1–19. ACM, 2015. doi:10.1145/2814270.2814288.
- 33 Hermes Hans. Semiotik. eine theorie der zeichengestalten als grundlage für untersuchungen von formalisierten sprachen. *Journal of Philosophy*, 36(13):356–357, 1939. doi:10.2307/2017267.
- 34 Lukás Holík, Petr Janku, Anthony W. Lin, Philipp Rümmer, and Tomáš Vojnar. String constraints with concatenation and transducers solved efficiently. *Proc. ACM Program. Lang.*, 2(POPL), 2018. doi:10.1145/3158092.
- 35 Shuanglong Kan, Anthony Widjaja Lin, Philipp Rümmer, and Micha Schrader. Certistr: a certified string solver. In Andrei Popescu and Steve Zdancewic, editors, *CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, Philadelphia, PA, USA, January 17 - 18, 2022*. ACM, 2022. doi:10.1145/3497775.3503691.
- 36 Adam Kiezun, Vijay Ganesh, Shay Artzi, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. Hampi: A solver for word equations over strings, regular expressions, and context-free grammars. *ACM Trans. Softw. Eng. Methodol.*, 21(4), February 2013.
- 37 Adam Kiezun, Philip J. Guo, Pieter Hooimeijer, Michael D. Ernst, and Vijay Ganesh. Theory and practice of string solvers (invited talk abstract). In Dongmei Zhang and Anders Møller, editors, *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*. ACM, 2019. doi:10.1145/3293882.3338993.

- 38 Tianyi Liang, Andrew Reynolds, Nestan Tsiskaridze, Cesare Tinelli, Clark Barrett, and Morgan Deters. An efficient SMT solver for string constraints. *Form. Methods Syst. Des.*, 48(3), 2016. doi:10.1007/s10703-016-0247-6.
- 39 Anthony Widjaja Lin and Pablo Barceló. String solving with word equations and transducers: towards a logic for analysing mutation XSS. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 123–136. ACM, 2016. doi:10.1145/2837614.2837641.
- 40 G S Makanin. The problem of solvability of equations in a free smigroup. *Mathematics of the USSR-Sbornik*, 32(2), 1977. doi:10.1070/SM1977v032n02ABEH002376.
- 41 Yu. V. Matiyasevich. A connection between systems of words-and-lengths equations and Hilbert’s tenth problem. *Studies in constructive mathematics and mathematical logic. Part II, Zap. Nauchn. Sem. LOMI*, 8, 1968.
- 42 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. In *40th Annual Symposium on Foundations of Computer Science, FOCS ’99, 17-18 October, 1999, New York, NY, USA*, pages 495–500. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814622.
- 43 W. V. Quine. Concatenation as a basis for arithmetic. *The Journal of Symbolic Logic*, 11, 1946. doi:10.2307/2268308.
- 44 Alfred Tarski. Der wahrheitsbegriff in den formalisierten sprachen. *Studia Philosophica*, 1:261–405, 1935.
- 45 Jean van Heijenoort. *From Frege to Gödel : A Source Book in Mathematical Logic*. Harvard University Press, January 2002.
- 46 Georg Zetsche. The complexity of downward closure comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 123:1–123:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.123.

On a Hierarchy of Spectral Invariants for Graphs

V. Arvind 

The Institute of Mathematical Sciences (HBNI), Chennai, India
Chennai Mathematical Institute, India

Frank Fuhlbrück 

Institut für Informatik, Humboldt-Universität zu Berlin, Germany

Johannes Köbler 

Institut für Informatik, Humboldt-Universität zu Berlin, Germany

Oleg Verbitsky 

Institut für Informatik, Humboldt-Universität zu Berlin, Germany

Abstract

We consider a hierarchy of graph invariants that naturally extends the spectral invariants defined by Fürer (Lin. Alg. Appl. 2010) based on the angles formed by the set of standard basis vectors and their projections onto eigenspaces of the adjacency matrix. We provide a purely combinatorial characterization of this hierarchy in terms of the walk counts. This allows us to give a complete answer to Fürer’s question about the strength of his invariants in distinguishing non-isomorphic graphs in comparison to the 2-dimensional Weisfeiler-Leman algorithm, extending the recent work of Rattan and Seppelt (SODA 2023). As another application of the characterization, we prove that almost all graphs are determined up to isomorphism in terms of the spectrum and the angles, which is of interest in view of the long-standing open problem whether almost all graphs are determined by their eigenvalues alone. Finally, we describe the exact relationship between the hierarchy and the Weisfeiler-Leman algorithms for small dimensions, as also some other important spectral characteristics of a graph such as the generalized and the main spectra.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms

Keywords and phrases Graph Isomorphism, spectra of graphs, combinatorial refinement, strongly regular graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.6

Related Version *Full Version:* <https://arxiv.org/abs/2310.04391> [1]

Supplementary Material *Software:* <https://gitlab.com/tcslib/tcsliblua/>
archived at `swh:1:rev:7e429da7fcf9bf8d93dfc50b54e823a47364cb4b`

Funding *Oleg Verbitsky:* Supported by DFG grant KO 1053/8–2. On leave from the IAPMM, Lviv, Ukraine.

1 Introduction

The spectrum of a graph is a remarkable graph invariant that has found numerous applications in computer science; e.g., [17, 20]. These applications are based on analyzing relevant information contained in the eigenvalues of a given graph. The maximum information possible is evidently obtained for graphs that are determined by their spectra up to isomorphism. This graph class, which is of direct relevance to the graph isomorphism problem, is often abbreviated as DS. Thus, a graph G is DS if every graph cospectral to G , i.e., with the same spectrum as G , is actually isomorphic to G . Though the problem of characterizing DS graphs has been intensively studied since the beginning of spectral graph theory (see [10] and



© V. Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 6; pp. 6:1–6:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



references therein), we are still far from a satisfactory solution. In particular, a long-standing open question [16, 25] is whether or not almost all graphs are DS. Here and in the rest of the paper, we say that *almost all* graphs have some property if a uniformly distributed random n -vertex graph¹ has this property with probability approaching 1 as n goes to infinity.

Somewhat surprisingly at first sight, the area is connected to a purely combinatorial approach to the graph isomorphism problem. In their seminal work, Weisfeiler and Leman [28] proposed and studied a method for distinguishing a graph G from another non-isomorphic graph by computing a sequence of canonical partitions of $V(G)^2$ into color classes. The final partition of $V(G)^2$ results in a *coherent configuration*, a concept which is studied in algebraic combinatorics [7] and plays an important role in isomorphism testing [2]. The method of [28] is now called the 2-dimensional Weisfeiler-Leman algorithm (2-WL). A similar approach based on partitioning $V(G)$ is known as *color refinement* and is often called 1-WL. Even this one-dimensional method is quite powerful as it suffices for identification of almost all graphs [3]. On the other hand, construction of graphs not identifiable by 2-WL is rather tricky. Particular examples are based on rare combinatorial objects. General constructions, like the far-reaching one in [6], give rather sporadic families of “hard” graphs. It turns out (see, e.g., [4, 15]) that if two graphs are indistinguishable by 2-WL, then they are cospectral. As a consequence, graphs not identifiable by 2-WL are examples of non-DS graphs.

Our overall goal in this paper is a systematic exploration of connections between spectral and combinatorial approaches to finding expressive graph invariants. A *graph invariant* \mathcal{I} is a function of a graph such that $\mathcal{I}(G) = \mathcal{I}(H)$ whenever $G \cong H$. An invariant \mathcal{I} is *stronger* than invariant \mathcal{I}' if $\mathcal{I}(G) = \mathcal{I}(H)$ implies $\mathcal{I}'(G) = \mathcal{I}'(H)$. Equivalently, we sometimes say that \mathcal{I}' is *weaker* than \mathcal{I} and write $\mathcal{I}' \preceq \mathcal{I}$. A stronger invariant can be more effective in distinguishing non-isomorphic graphs: If $\mathcal{I}' \preceq \mathcal{I}$ and \mathcal{I}' can distinguish non-isomorphic graphs G and H , i.e., $\mathcal{I}'(G) \neq \mathcal{I}'(H)$, then these graphs are distinguishable by \mathcal{I} as well.

Let $\text{Spec}(G)$ denote the spectrum of a graph G , and $\text{WL}_2(G)$ denote the output of 2-WL on G (a formal definition is given in Section 2.2). The discussion above shows that $\text{Spec} \preceq \text{WL}_2$. This can be seen as evidence of limitations of Spec , as well as evidence of the power of WL_2 .

A reasonable question is what can be achieved if $\text{Spec}(G)$ is enhanced by other spectral characteristics of the adjacency matrix of G . One such line of research in spectral graph theory considers $\text{Spec}(G)$ augmented with the multiset of all angles between the standard basis vectors and the eigenspaces of G . The parameters and properties of a graph G which are determined by its eigenvalues and angles are called *EA-reconstructible* and are thoroughly studied by Cvetković and co-authors; see [11, Ch. 4] and [10, Ch. 3].

A further natural step is to take into consideration the multisets of angles between the projections of the standard basis vectors onto eigenspaces. Fürer [15] uses this additional data to define two new graph invariants, namely, the *weak* and *strong spectral invariants*. We denote these spectral invariants by *weak-FSI* and *strong-FSI* respectively; formal definitions are in Section 3. Fürer shows that both *weak-FSI* and *strong-FSI* remain weaker than WL_2 . That is,

$$\text{weak-FSI} \preceq \text{strong-FSI} \preceq \text{WL}_2 \tag{1}$$

(note that $\text{Spec} \preceq \text{weak-FSI}$ by definition). An open problem posed in [15] is to determine which of the relations in (1) are strict. Rattan and Seppelt, in their recent paper [23], show that this small hierarchy does not entirely collapse by separating *weak-FSI* and WL_2 . Hence,

¹ In the Erdős-Rényi $\mathcal{G}(n, 1/2)$ random graph model to be precise.

at least one of the two relations in (1) is strict. Fürer [15] conjectures that the first relation in (1) is strict and does not exclude that the last two invariants in (1) are equivalent, and our aim is to give precise answers to these questions.

In [23] the invariants **weak-FSI** and WL_2 are separated by introducing a new natural graph invariant $WL_{3/2}$, whose strength lies between WL_1 and WL_2 . The authors give an elegant algebraic characterization of $WL_{3/2}$ using which they show that $\text{weak-FSI} \preceq WL_{3/2}$. The final step in their analysis is an example of graph pair that separates $WL_{3/2}$ and WL_2 .

Our approach is different. First, we observe that the invariants **weak-FSI** and **strong-FSI** are part of a broader scheme, presented in Section 2.1, that leads to a potentially infinite hierarchy of graph invariants. We define the corresponding spectral hierarchy containing **weak-FSI** and **strong-FSI** on its lower levels in Section 3. Another level is taken by the aforementioned invariant **EA**. In Section 4 we characterize this hierarchy in terms of walk counts. A connection between spectral parameters and walk counts is actually well known (see an overview in Subsection 4.1). With a little extra effort we are able to show that this connection is tight; see Theorem 4. This yields a purely combinatorial characterization of the invariants **EA**, **weak-FSI**, and **strong-FSI** (Corollary 5), which also reveals some new relations. For example, we notice that **weak-FSI** determines the *generalized spectrum* of a graph (Theorem 11).

As another application of our combinatorial characterization, we prove that almost all graphs are determined up to isomorphism by **weak-FSI**, that is, by the eigenvalues and the angles formed by the standard basis vectors and their projections onto eigenspaces (Corollary 8). We find this interesting in the context of the open problem mentioned above: whether or not almost all graphs are DS.

We present the relations between the spectral and combinatorial invariants under consideration in Section 5; see the diagram in Fig. 1. In Section 6 we prove that this diagram is complete, that is, it shows all existent relations, and all of these relations are strict (perhaps up to higher levels of the hierarchy whose separation remains open). In particular, both relations in (1) are strict, which gives a complete answer to Fürer's questions. Another noteworthy separation is $\text{strong-FSI} \not\preceq WL_{3/2}$ (Theorem 12). Curiously, the separating pair of graphs is the same that was used in [23], which yields more information now because we also use our characterization in Theorem 4.

The more involved separations are shown in Theorems 13, 16, and 18. The corresponding separating examples are not ad hoc. They are obtained by a quite general construction (Lemma 14). The construction is based on a considerable extension of the approach taken in [27] to separate various concepts related to 1-WL and the walk matrix of a graph (an important notion discussed in Section 4.2). Implementation of the construction requires vertex-colored strongly regular graphs with certain properties. The required colorings were found by a computer assisted search among members of the family of strongly regular graphs on 25 vertices.

Some proofs are missing due to the space constraints and can be found in the full version of the paper [1].

2 Preliminaries

2.1 From isomorphism-invariant colorings to isomorphism invariants

Let \mathcal{C} be a set of colors and $\chi : V(G)^2 \rightarrow \mathcal{C}$ be a coloring of vertex pairs in a graph G . It is natural to see $\chi(x, x)$ as the color of a vertex x . We suppose that $\chi = \chi_G$ is defined for every graph G . That is, speaking of a coloring χ , we actually mean a map $G \mapsto \chi_G$. Such a

6:4 On a Hierarchy of Spectral Invariants for Graphs

coloring χ is *isomorphism invariant* if for every isomorphism f from a graph G to a graph H (the equality $G = H$ is not excluded) we have $\chi_G(x, y) = \chi_H(f(x), f(y))$ for all vertices x and y in G .

The simplest isomorphism-invariant colorings are the adjacency relation A and the identity relation I . That is, $A(x, y) = 1$ if x and y are adjacent and $A(x, y) = 0$ otherwise. For the identity relation, $I(x, y) = 1$ if $x = y$ and $I(x, y) = 0$ if $x \neq y$. Below, we will consider A and I also as the adjacency and the identity matrices. Other examples of isomorphism-invariant colorings are the distance $d(x, y)$ between two vertices x and y and the number of all such triangles in the graph that contain the vertices x and y . One can consider also more complex definitions like the triple $\chi(x, y) = (\deg x, d(x, y), \deg y)$, where $\deg x$ denotes the degree of x .

Given an isomorphism-invariant coloring χ , we can build on it to define various graph invariants. The simplest such examples are

$$\begin{aligned} \mathcal{I}_1(G) &= \{\{\chi(x, y)\}_{x, y \in V(G)}\}, \\ \mathcal{I}_2(G) &= \left(\{\{\chi(x, x)\}_{x \in V(G)}\}, \{\{\chi(x, y)\}_{x, y \in V(G)}\} \right), \\ \mathcal{I}_3(G) &= \{\{\{\chi(x, x), \chi(x, y), \chi(y, y)\}_{x, y \in V(G)}\}\}, \\ \mathcal{I}_4(G) &= \left\{ \left\{ \left(\chi(x, x), \{\{\chi(x, y), \chi(y, y)\}_{y \in V(G)}\} \right) \right\}_{x \in V(G)} \right\}, \end{aligned}$$

where $\{\dots\}$ denotes a multiset. Note that $\mathcal{I}_1(G) \preceq \mathcal{I}_2(G) \preceq \mathcal{I}_3(G) \preceq \mathcal{I}_4(G)$.

Given an isomorphism-invariant coloring χ , we can define a hierarchy of ever more complex graph invariants. We first inductively define a sequence of colorings $\chi_0, \chi_1, \chi_2, \dots$ of single vertices by

$$\chi_0(x) = \chi(x, x) \text{ and } \chi_{r+1}(x) = \left(\chi_r(x), \{\{\chi(x, y), \chi_r(y)\}_{y \in V(G)}\} \right). \quad (2)$$

This definition is a natural extension of the well-known concept of *color refinement (CR)* to edge- and vertex-colored graphs. In broad outline, CR computes an isomorphism-invariant color of each vertex in an input graph and recognizes two graphs as non-isomorphic if one of the colors occurs in one graph more frequently than in the other. In the case of an uncolored undirected graph G , CR starts with a uniform coloring χ_0 of $V(G)$, that is, $\chi_0(x) = \chi_0(x')$ for all $x, x' \in V(G)$. In the $(r+1)$ -th round, the preceding coloring χ_r is refined to a new coloring χ_{r+1} . For each vertex x , its new color $\chi_{r+1}(x)$ consists of $\chi_r(x)$ and the multiset $\{\{\chi_r(y)\}_{y \in N(x)}\}$ of all colors occurring in the neighborhood $N(x)$ of x . In other words, CR counts how frequently each χ_r -color occurs among the vertices adjacent to x (or, equivalently, among the vertices non-adjacent to x). In an edge- and vertex-colored graph G , each vertex pair (x, y) is assigned a color, which we denote by $\chi(x, y)$. The edge colors must be taken into account while computing the refined color $\chi_{r+1}(x)$. In the colored case, CR first splits all vertices y into classes depending on $\chi(x, y)$ and then computes the frequencies of $\chi_r(y)$ within each class. This is exactly what (2) does.

Note that

$$\chi_1(x) = \left(\chi(x, x), \{\{\chi(x, y), \chi(y, y)\}_{y \in V(G)}\} \right). \quad (3)$$

In addition, we set

$$\chi_{1/2}(x) = \left(\chi(x, x), \{\{\chi(x, y)\}_{y \in V(G)}\} \right) \quad (4)$$

Now, we define

$$\chi^{(r)}(G) = \{\{\chi_r(x)\}_{x \in V(G)}\}. \quad (5)$$

For each r , the coloring χ_r is isomorphism invariant in the sense that $\chi_r(f(x)) = \chi_r(x)$ for any isomorphism of graphs f . This readily implies that $\chi^{(r)}$ is a graph invariant. As easily seen, $\chi^{(r)} \preceq \chi^{(s)}$ if $r \leq s$.

2.2 First two dimensions of combinatorial refinement

We now give formal descriptions of the isomorphism tests 1-WL and 2-WL already introduced in Section 1. Note that 1-WL is an alternative name for CR. In what follows we will apply the 1-WL procedure to *vertex-colored* graphs. Given a vertex-colored graph G , we consider a coloring χ of $V(G)^2$ defined by $\chi(x, y) = A(x, y)$, i.e., according to the adjacency relation, for $x \neq y$ and by setting $\chi(x, x)$ to be the color of a vertex x . On an input G , 1-WL iteratively computes the vertex colorings χ_r according to (2). After performing n iterations, where n is the number of vertices in G , 1-WL outputs $\text{WL}_1(G) = \chi^{(n)}(G)$ as defined by (5). Two graphs G and H are recognized as non-isomorphic if $\text{WL}_1(G) \neq \text{WL}_1(H)$.

2-WL can be similarly formulated, except that it computes colorings of vertex pairs. If an input graph G is uncolored, then 2-WL begins with an initial coloring χ_0 of $V(G)^2$ defined by $\chi_0(x, y) = A(x, y)$ if $x \neq y$ and by $\chi_0(x, x) = 2$ for every vertex x of G . If G is a vertex-colored graph, then $\chi_0(x, y)$ must include also the colors of x and y . Furthermore,

$$\chi_{r+1}(x, y) = \left(\chi_r(x, y), \left\{ \left(\chi_r(x, z), \chi_r(z, y) \right) \right\}_{z \in V(G)} \right).$$

Thus, the new color of a pair (x, y) can be seen as a kind of “superposition” of the old color pairs observable along all extensions of (x, y) to a triangle xzy . Finally, 2-WL outputs the multiset $\text{WL}_2(G) = \left\{ \left\{ \chi_{n^2}(x, y) \right\}_{x, y \in V(G)} \right\}$.

3 A hierarchy of spectral invariants

Speaking of an n -vertex graph G , we will assume that $V(G) = \{1, 2, \dots, n\}$. Let

$$\mu_1 < \mu_2 < \dots < \mu_m \tag{6}$$

be all pairwise distinct eigenvalues of the adjacency matrix A of G . Let $\text{Spec}(G)$ denote the spectrum of G , i.e., the multiset of all eigenvalues where each μ_i occurs with its multiplicity. As mentioned before, $\text{Spec}(G)$ is a well-studied graph invariant with numerous applications in computer science.

Let E_i be the eigenspace of μ_i . Recall that E_i consists of all eigenvectors of μ_i , i.e., $E_i = \{v \in \mathbb{R}^n : Av = \mu_i v\}$. Let P_i be the matrix of the orthogonal projection of \mathbb{R}^n onto E_i . Note that $P_i^2 = P_i = P_i^\top$. For $1 \leq x, y \leq n$, the matrix entry $P_i(x, y)$ can be seen a color of the vertex pair (x, y) . This coloring is isomorphism invariant.

Throughout the paper, we use the following notational convention for compact representations of sequences.

► **Notation 1.** For an indexed set $\{a_i\}$ with index i ranging through the interval of integers $s, s + 1, \dots, t - 1, t$ we set $a_* = (a_s, \dots, a_t)$.

In particular,

$$P_*(x, y) = (P_1(x, y), \dots, P_m(x, y)).$$

Since the order on the index set is determined by (6), P_* is also an isomorphism-invariant coloring. Following the general framework in Section 2.1, the coloring P_* determines the sequence of graph invariants $P_*^{(0)}, P_*^{(1/2)}, P_*^{(1)}, P_*^{(2)}, \dots$. In particular, by (2)–(5) we have

6:6 On a Hierarchy of Spectral Invariants for Graphs

$$\begin{aligned}
P_*^{(0)}(G) &= \left\{ \left\{ P_*(x, x) \right\}_{1 \leq x \leq n} \right\}, & (7) \\
P_*^{(1/2)}(G) &= \left\{ \left(P_*(x, x), \left\{ P_*(x, y) \right\}_{1 \leq y \leq n} \right) \right\}_{1 \leq x \leq n}, \\
P_*^{(1)}(G) &= \left\{ \left(P_*(x, x), \left\{ (P_*(x, y), P_*(y, y)) \right\}_{1 \leq y \leq n} \right) \right\}_{1 \leq x \leq n}.
\end{aligned}$$

Fürer [15] introduces the *weak* and *strong spectral invariants*. Using our notation, Fürer's spectral invariants (FSI) can be defined as follows:

$$\text{weak-FSI}(G) = \left(\text{Spec}(G), P_*^{(1/2)}(G) \right) \text{ and} \quad (8)$$

$$\text{strong-FSI}(G) = \left(\text{Spec}(G), P_*^{(1)}(G) \right). \quad (9)$$

The entries of the projection matrices P_i have a well-known geometric meaning [10, 12]. For $1 \leq x \leq n$, the standard basis vector e_x of \mathbb{R}^n has 1 in the position x and 0 elsewhere. The *angle* $\alpha_{i,x}$ of a graph G is defined to be the cosine of the angle between e_x and the eigenspace E_i , i.e., the angle between e_x and its projection $P_i e_x$ onto E_i . We have the equality

$$P_i(x, x) = \alpha_{i,x}^2. \quad (10)$$

Indeed, let $\langle u, v \rangle$ denote the scalar product of two vectors $u, v \in \mathbb{R}^n$. Then

$$P_i(x, x) = \langle e_x, P_i e_x \rangle = \|e_x\| \|P_i e_x\| \alpha_{i,x} = \alpha_{i,x}^2.$$

Furthermore, let $\alpha_{i,xy}$ be the cosine of the angle between the projections $P_i e_x$ and $P_i e_y$ of the standard basis vector e_x and e_y onto E_i . If e_x or e_y is orthogonal to E_i , i.e., $\alpha_{i,x} = 0$ or $\alpha_{i,y} = 0$, then the angle is undefined and we set $\alpha_{i,xy} = 0$ in this case. In particular, $\alpha_{i,xx} = 0$ if $\alpha_{i,x} = 0$ while $\alpha_{i,xx} = 1$ if $\alpha_{i,x} \neq 0$. Equality (10) generalizes to

$$P_i(x, y) = \alpha_{i,x} \alpha_{i,y} \alpha_{i,xy}. \quad (11)$$

Indeed,

$$P_i(x, y) = \langle e_x, P_i e_y \rangle = \langle e_x, P_i^2 e_y \rangle = \langle P_i e_x, P_i e_y \rangle = \|P_i e_x\| \|P_i e_y\| \alpha_{i,xy} = \alpha_{i,x} \alpha_{i,y} \alpha_{i,xy}.$$

Now, define a coloring α_i by $\alpha_i(x, x) = \alpha_{i,x}$ and $\alpha_i(x, y) = \alpha_{i,xy}$ for $x \neq y$. This coloring is isomorphism invariant basically because an isomorphism is represented by a permutation matrix, which is the transformation matrix of an isometry of \mathbb{R}^n . Using Notation 1,

$$\alpha_*(x, y) = (\alpha_1(x, y), \dots, \alpha_m(x, y)),$$

where α_* is an isomorphism-invariant coloring as well. The corresponding graph invariants $\alpha_*^{(r)}$ are closely related to the invariants $P_*^{(r)}$. More precisely, we say that two graph invariants \mathcal{I} and \mathcal{I}' are *equivalent* and write $\mathcal{I} \equiv \mathcal{I}'$ if $\mathcal{I}' \preceq \mathcal{I}$ and $\mathcal{I} \preceq \mathcal{I}'$.

► **Lemma 2.** $P_*^{(r)} \equiv \alpha_*^{(r)}$ for every integer $r \geq 0$.

Motivated by the equivalence $P_*^{(0)} \equiv \alpha_*^{(0)}$, we define the graph invariant EA similar to (8)–(9) as

$$\text{EA}(G) = \left(\text{Spec}(G), P_*^{(0)}(G) \right), \quad (12)$$

where the abbreviation EA stands for *Eigenvalues and Angles* and corresponds to the known concept [11, 10] mentioned in the introduction.

4 Characterization of the spectral invariants by walk counts

A walk of length k (or k -walk) from a vertex x to a vertex y is a sequence of vertices $x = x_0, x_1, \dots, x_k = y$ such that every two successive vertices x_i, x_{i+1} are adjacent. Let $w_k(x, y)$ denote the number of walks of length k from x to y in a graph. Obviously, w_k is an isomorphism-invariant coloring in the sense of Section 2.1. In accordance with Notation 1, we also consider the isomorphism-invariant coloring w_* defined by

$$w_*(x, y) = (w_0(x, y), w_1(x, y), \dots, w_{n-1}(x, y)),$$

where n , as usually, denotes the number of vertices in a graph. Note that for each y , the matrix $(w_k(x, y))_{1 \leq x \leq n, 0 \leq k \leq n-1}$ determines the value of $w_k(x, y)$ for every x and for every arbitrarily large k .

It is well known that the walk counts are expressible in terms of spectral parameters of a graph; see, e.g., [10]. On the other hand, it is also well known that the numbers of closed k -walks in a graph determine the graph spectrum. We give a brief overview of these facts in Subsection 4.1. In Subsection 4.2 we make further use of this connection between walks and spectra. We are able to characterize the spectral invariants defined in Section 3 using solely the walk numbers, that is, in purely combinatorial terms without involving any linear algebra.

4.1 Linear-algebraic background and known relations

Since the adjacency matrix A of a graph G is symmetric, the eigenspaces E_i are pairwise orthogonal, and hence $P_i P_j = O$ for $i \neq j$, where O denotes the zero matrix. The spectral theorem for symmetric matrices says in essence that

$$\mathbb{R}^n = E_1 \oplus \dots \oplus E_m,$$

that is, \mathbb{R}^n has an orthonormal basis consisting of eigenvectors of A . This decomposition implies that

$$I = P_1 + \dots + P_m. \tag{13}$$

From Equality (13) it is easy to derive the spectral decomposition

$$A = \mu_1 P_1 + \dots + \mu_m P_m.$$

Raising both sides of this equality to the k -th power and taking into account that $P_i^2 = P_i$ and $P_i P_j = O$ for $i \neq j$, we conclude that

$$A^k = \mu_1^k P_1 + \dots + \mu_m^k P_m.$$

Since $w_k(x, y) = A^k(x, y)$, we get

$$w_k(x, y) = \mu_1^k P_1(x, y) + \dots + \mu_m^k P_m(x, y). \tag{14}$$

Let $c_k(G) = \sum_{x \in V(G)} w_k(x, x)$ denote the total number of closed k -walks in a graph G . In particular, $c_0(G) = n$ and $c_1(G) = 0$.

► **Lemma 3** (folklore). *Spec $G = \text{Spec } H$ if and only if $c_k(G) = c_k(H)$ for $k = 0, 1, \dots, n$.*

4.2 General characterization and its consequences

► **Theorem 4.** $(\text{Spec}, P_*^{(r)}) \equiv w_*^{(r)}$ for every $r = 0, 1/2, 1, 2, \dots$

Before proving Theorem 4, we describe some of its consequences.

► **Corollary 5.**

1. $\text{EA} \equiv w_*^{(0)}$.
2. $\text{weak-FSI} \equiv w_*^{(1/2)}$.
3. $\text{strong-FSI} \equiv w_*^{(1)}$.

Parts 2 and 3 of Corollary 5, which are particular cases of Theorem 4 for $r = 1/2$ and $r = 1$ respectively, provide a characterization of both Fürer’s spectral invariants. We now comment on Part 1. By Definition (12), this part is the special case of Theorem 4 for $r = 0$. Note that $w_*^{(0)}(G) = w_*^{(0)}(H)$ if and only if the graphs G and H are *closed-walk-equivalent* in the sense that there is a bijection $f : V(G) \rightarrow V(H)$ such that $w_k(x, x) = w_k(f(x), f(x))$ for all $x \in V(G)$ and all k . On the other hand, let us say that G and H are *EA-equivalent* if these graphs have the same eigenvalues and angles, i.e., $\text{EA}(G) = \text{EA}(H)$. As seen from the summary in Subsection 4.1, there are well-known connections between the closed walk numbers and the eigenvalues and angles. Part 1 of Corollary 5 pinpoints the fact that the two equivalence concepts actually coincide.

Corollary 5 reveals connections of spectral invariants to other graph invariants studied in the literature, which we introduce now.

The *walk matrix* W of a graph G is indexed by vertices $1 \leq x \leq n$ and the length parameter $0 \leq k \leq n - 1$ and defined by

$$W(x, k) = \sum_{y=1}^n w_k(x, y). \tag{15}$$

That is, $W(x, k)$ is the total number of k -walks starting from the vertex x . Let $WL_1^k(G, x)$ denote the color assigned by 1-WL to a vertex x of G after the k -th refinement round. For a vertex x of G , let G_x denote the version of G with x *individualized*. This means that G_x is a vertex-colored graph where x has a special unique color while the other vertices are colored uniformly. We now state two well-known facts.

► **Lemma 6.**

1. $W(x, k)$ is determined by $WL_1^k(G, x)$;
2. $w_k(x, y)$ is determined by $WL_1^k(G_x, y)$;

Part 1 of Lemma 6 is proved in algebraic terms in [22, Theorem 2]. Another proof, involving logical concepts, is provided in [13, Lemma 4] and a direct combinatorial proof can be found in [27, Lemma 8]. Part 2 is a straightforward extension of Part 1.

In addition to the graph invariants WL_1 and WL_2 introduced in Section 2.2, we define

$$WL_{3/2}(G) = \{\{WL_1(G_x)\}_{x \in V(G)}\}.$$

This yields a chain of graph invariants WL_d for $d \in \{1, 3/2, 2\}$, where $WL_c \preceq WL_d$ if $c \leq d$. The walk matrix naturally gives us a graph invariant, which we denote by WM and define as $WM(G) = \{\{W(x, *)\}_{x \in V(G)}\}$ where $W(x, *) = (W(x, 0), W(x, 1), \dots, W(x, n - 1))$. In other words, $WM(G)$ is the multiset of the rows of the walk matrix of G . Part 1 of Lemma 6 readily implies that $WM \preceq WL_1$. Thus,

$$WM \preceq WL_1 \preceq WL_{3/2} \preceq WL_2.$$

The second relation in the following corollary is the recent result in [23] already described in Section 1.

► **Corollary 7.** $WM \preceq \text{weak-FSI} \preceq WL_{3/2}$.

Proof. By Part 2 of Corollary 5, it is enough to show that

$$WM \preceq w_*^{(1/2)} \preceq WL_{3/2}.$$

The former relation follows directly from the definitions of WM and $w_*^{(1/2)}$. Indeed, $w_*^{(1/2)}(G)$ comprises the multiset $\{\{w_*(x, y)\}_{y \in V(G)}\}$ for each vertex x of G ; see (4). This multiset allows us to calculate the sum (15) for each k . The latter relation follows from the definitions of $w_*^{(1/2)}$ and $WL_{3/2}$ by Part 2 of Lemma 6. ◀

The relationship between the graph invariants discussed above is summarized in Figure 1 below, which also puts these invariants in a somewhat broader context.

The next consequence of Theorem 4 is interesting in view of the long-standing open question whether almost all graphs are determined up to isomorphism by their spectrum [16, 25].

► **Corollary 8.** *Almost all graphs are determined up to isomorphism by weak-FSI.*

Proof. It is known that if the walk matrix is non-singular, then it determines the adjacency matrix [19]. Moreover, the walk matrix of a random graph is non-singular with high probability [21]. As a consequence, almost all graphs are determined by the graph invariant WM; see [19, Th. 7.2]. The same is true also for weak-FSI because weak-FSI is stronger than WM by Corollary 7. ◀

Since $\text{Spec} \preceq \text{EA} \preceq \text{weak-FSI}$, a natural further question is whether Corollary 8 can be improved to the identifiability of almost all graphs by the graph invariant EA, that is, by using the eigenvalues and the angles between each standard basis vector and its projections onto eigenspaces but not between the projections themselves. If true, this would be yet closer to the aforementioned open problem.

4.3 Proof of Theorem 4

The case of $r = 0$. We have to prove that

$$(\text{Spec}, \{\{P_*(x, x)\}_x\}) \equiv \{\{w_*(x, x)\}_x\}. \quad (16)$$

The “ \succeq ” part immediately follows from Equality (14). In the other direction, the relation $\text{Spec} \preceq \{\{w_*(x, x)\}_x\}$ is a direct consequence of Lemma 3. To complete the proof of (16), we show that for each vertex x , the sequence $P_*(x, x)$ can be obtained from the sequences $w_*(x, x)$ and μ_* . To this end, put $y = x$ in Equality (14), obtaining

$$\mu_1^k P_1(x, x) + \cdots + \mu_m^k P_m(x, x) = w_k(x, x). \quad (17)$$

This equality makes sense also for $k = 0$. In this case, it reads

$$P_1(x, x) + \cdots + P_m(x, x) = 1, \quad (18)$$

which is true by Equality (13) (if $\mu_i = 0$, we need to use the convention $0^0 = 1$). Consider Equalities (17) for $k = 0, 1, \dots, m - 1$ as a system of m linear equations for m unknowns $P_1(x, x), \dots, P_m(x, x)$. The coefficients of this system are powers of the m pairwise distinct eigenvalues. They form a Vandermonde matrix. Therefore, the system is uniquely solvable, and the sequence $P_*(x, x)$ is determined.

6:10 On a Hierarchy of Spectral Invariants for Graphs

The case of $r = 1/2$. Now we have to prove that

$$\left(\text{Spec}, \left\{ \left(P_*(x, x), \{ \{ P_*(x, y) \}_y \} \right) \right\}_x \right) \equiv \left\{ \left(w_*(x, x), \{ \{ w_*(x, y) \}_y \} \right) \right\}_x. \quad (19)$$

The “ \succeq ” part is again an immediate consequence of Equality (14).

Let us prove the part “ \preceq ”. That is, we have to show that for a given graph, the left hand side of (19) can be obtained from the right hand side. The spectrum is, as already observed in the case of $r = 0$, determined by the multiset $\{ \{ w_*(x, x) \}_x \}$, which is easy to obtain from the right hand side of (19). Thus, in what follows we can assume that the sequence μ_1, \dots, μ_m of distinct eigenvalues is known. For each vertex x , we have to compute the sequence $P_*(x, x)$ and the multiset $\{ \{ P_*(x, y) \}_y \}$. The former task is solvable exactly as in the case of $r = 0$, and we focus on the latter task. In addition to x , we now fix also y and consider Equalities (14) for $k = 0, 1, \dots, m - 1$. Here, the equality for $k = 0$ is actually a particular instance of Equality (13), that is, this is (18) if $y = x$ and

$$P_1(x, y) + \dots + P_m(x, y) = 0$$

if $y \neq x$. As in the case of $r = 0$, for m unknowns $P_1(x, y), \dots, P_m(x, y)$ we obtain a system of m linear equation whose coefficients form a non-singular Vandermonde matrix. Hence, we can determine the sequence $P_*(x, y)$, completing the proof of (19).

The case of $r \geq 1$. We proceed as above using induction. To facilitate the notation, we set $\pi(x, y) = P_*(x, y)$ and $\omega(x, y) = w_*(x, y)$. We write $a \leftarrow b$ to say that a is obtainable from b . As we have already seen,

$$\omega(x, y) \leftarrow \text{Spec}, \pi(x, y) \quad (20)$$

by Equality (14) and

$$\pi(x, y) \leftarrow \text{Spec}, \omega(x, y) \quad (21)$$

by the Vandermonde matrix argument. The vertex colorings π_r and ω_r are defined as in (2). For every r ,

$$\text{Spec} \leftarrow \{ \{ \omega_0(x) \}_x \} \leftarrow \{ \{ \omega_r(x) \}_x \}. \quad (22)$$

The former relation is, as already observed above, a consequence of Lemma 3, while the latter relation follows directly from the definition of ω_r . Therefore, in order to prove that

$$(\text{Spec}, \{ \{ \pi_r(x) \}_x \}) \preceq \{ \{ \omega_r(x) \}_x \},$$

it suffices to prove for each x that

$$\pi_r(x) \leftarrow \text{Spec}, \omega_r(x). \quad (23)$$

In order to prove that

$$\{ \{ \omega_r(x) \}_x \} \preceq (\text{Spec}, \{ \{ \pi_r(x) \}_x \}),$$

it suffices to prove for each x that

$$\omega_r(x) \leftarrow \text{Spec}, \pi_r(x). \quad (24)$$

We prove (23) and (24) by induction on r .

In the base case we have $r = 0$. The relation (24) for $r = 0$ follows from the relation (20) for $y = x$. Similarly, the relation (23) for $r = 0$ follows from the relation (21) for $y = x$.

For the induction step, suppose that $r \geq 1$. Consider first (23). Recall that

$$\pi_r(x) = \left(\pi_{r-1}(x), \left\{ \left\{ \pi(x, y), \pi_{r-1}(y) \right\} \right\}_y \right).$$

By the induction hypothesis,

$$\pi_{r-1}(x) \leftarrow \text{Spec}, \omega_{r-1}(x) \leftarrow \text{Spec}, \omega_r(x).$$

The latter relation follows from the fact that $\omega_{r-1}(x)$ is a part of $\omega_r(x)$. Another part of $\omega_r(x)$ gives us the multiset $\left\{ \left\{ \omega(x, y), \omega_{r-1}(y) \right\} \right\}_y$. Therefore, it suffices to argue that, for each y ,

$$(\pi(x, y), \pi_{r-1}(y)) \leftarrow \text{Spec}, (\omega(x, y), \omega_{r-1}(y)).$$

Indeed, $\pi(x, y)$ is determined by (21), and $\pi_{r-1}(y)$ is determined by the induction hypothesis.

The argument for (24) is virtually the same, with the roles of π and ω interchanged. In place of (21), we have to refer to (20). The proof is complete.

5 The Hierarchy

Figure 1 shows the invariants from Section 4.2 and the relations between them as a part of a more general picture involving also some other spectral invariants studied in the literature, which we introduce in the next two subsections.

Moreover, we define a new invariant $w_*^{(\bullet)}$ which is, in a sense, the limit of the sequence of invariants $w_*^{(r)}$ for $r = 1, 2, \dots$. The definition is rather general. As we already mentioned, the formal framework of Section 2.1 is analogous to the concept of color refinement. Given an isomorphism-invariant coloring χ , we therefore can along with graph invariants $\chi^{(r)}$ define the stable version $\chi^{(\bullet)}$. One possibility to do this is to set $\chi^{(\bullet)}(G) = \chi^{(n)}(G)$, where n is the number of vertices in G . Note that $\chi^{(r)} \preceq \chi^{(\bullet)}$ for every r . For $\chi = \omega_*$, we obtain

$$w_*^{(r)} \preceq w_*^{(r+1)} \preceq w_*^{(\bullet)} \preceq \text{WL}_2.$$

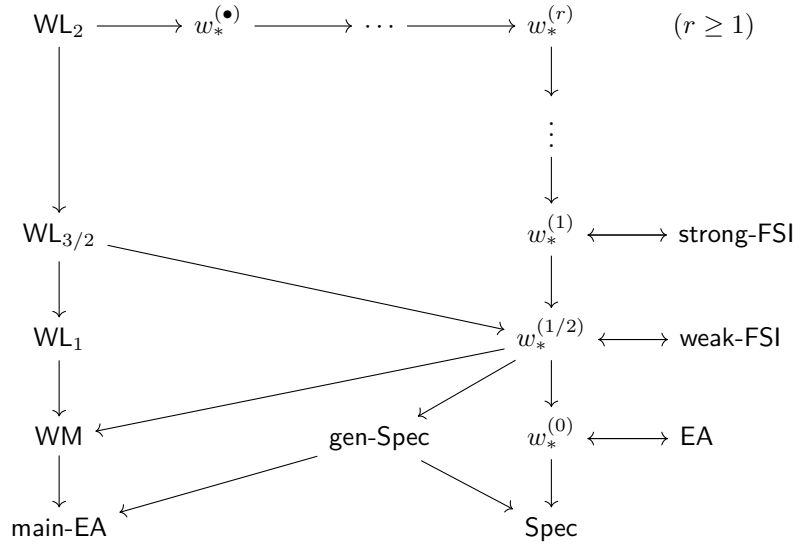
To see the last relation above, we first recall the well-known fact that $w_*(x, y)$ is determined by the color assigned to the vertex pair (x, y) by 2-WL. Furthermore, since $w_*^{(\bullet)}(G)$ is obtained from G endowed with the coloring w_* by running the version of 1-WL for edge-colored graphs, the outcome can be simulated by 2-WL.

5.1 Main eigenvalues and angles

Let \mathbf{j} denote the all-ones vector (the dimension should be clear from the context). Using our usual notation, suppose that G has m distinct eigenvalues μ_1, \dots, μ_m , and let E_1, \dots, E_m be the corresponding eigenspaces of G . Consider the angle between E_i and \mathbf{j} and denote its cosine by β_i . If $\beta_i \neq 0$, then the corresponding eigenvalue μ_i is called a *main eigenvalue*, and then the positive number β_i is called a *main angle*. Let $\nu_1, \dots, \nu_{m'}$ be the sequence of all main eigenvalues in the ascending order and $\theta_1, \dots, \theta_{m'}$ be the sequence of the main angles in the corresponding order. We define a graph invariant main-EA by

$$\text{main-EA}(G) = (\nu_*, \theta_*).$$

6:12 On a Hierarchy of Spectral Invariants for Graphs



■ **Figure 1** Relations between graph invariants. An arrow $\mathcal{I} \rightarrow \mathcal{I}'$ means $\mathcal{I}' \preceq \mathcal{I}$.

A characterization of **main-EA** in terms of walk numbers is known. Let

$$w_k(G) = \sum_{x \in V(G)} w_k(x)$$

be the total number of k -walks in G . By W_G we will denote the corresponding generating function, that is, the formal series

$$W_G(z) = \sum_{k=0}^{\infty} w_k(G) z^k.$$

► **Proposition 9** (folklore, e.g. [24]). *Let G and H be graphs with n vertices. Then $\text{main-EA}(G) = \text{main-EA}(H)$ if and only if $w_k(G) = w_k(H)$ for $k = 1, \dots, n - 1$.*

As a direct consequence of Proposition 9, we get the relation $\text{main-EA} \preceq \text{WM}$.

5.2 The generalized spectrum

Another important spectral invariant of G is the spectrum of the complement graph \overline{G} . The equalities $\text{Spec } G = \text{Spec } H$ and, simultaneously, $\text{Spec } \overline{G} = \text{Spec } \overline{H}$ are equivalent to the condition that the graphs G and H have the same *generalized spectrum*. For the definition of this concept and its various characterizations we refer the reader to [18] and [26, Th. 3]. We define the graph invariant **gen-Spec** by

$$\text{gen-Spec}(G) = (\text{Spec } G, \text{Spec } \overline{G}).$$

We note that

$$\text{main-EA} \preceq \text{gen-Spec}. \tag{25}$$

This follows from Proposition 9 and the following result in [8]. Let P_G denote the characteristic polynomial of a graph G .

► **Proposition 10** (Cvetković [8]). $W_G(z) = \frac{1}{z} \left((-1)^n \frac{P_{\overline{G}}(-1/z-1)}{P_G(1/z)} - 1 \right)$.

Alternatively, (25) can be obtained by using [26, Th. 3].

To complete the diagram in Figure 1, it remains to prove the following relation.

► **Theorem 11.** $\text{gen-Spec} \preceq \text{weak-FSI}$.

Proof. The spectrum of G , hence also the characteristic polynomial P_G , is determined by $\text{weak-FSI}(G)$ just by definition. We have to show that $\text{Spec } \overline{G}$ or, equivalently, $P_{\overline{G}}$ is also determined. By Proposition 10, $P_{\overline{G}}$ is obtainable from P_G and W_G . Using Part 2 of Corollary 5, it remains to notice that W_G is determined by $w_*^{(1/2)}(G)$. Indeed,

$$w_k(G) = \sum_x w_k(x) = \sum_x \sum_y w_k(x, y),$$

where the right hand side is obtainable from the multiset $\left\{ \left\{ w_k(x, y) \right\}_y \right\}_x$, which is a part of $w_*^{(1/2)}(G)$. ◀

6 Separations

Fürer [15] poses the open problem of determining which of the relations in the chain

$$\text{weak-FSI} \preceq \text{strong-FSI} \preceq \text{WL}_2 \tag{26}$$

are strict. As mentioned before, Rattan and Seppelt [23] show that this chain does not entirely collapse. They separate weak-FSI and WL_2 by proving that $\text{weak-FSI} \preceq \text{WL}_{3/2}$ and separating $\text{WL}_{3/2}$ and WL_2 . Hence, at least one of the two relations in (26) is strict. Fürer [15] conjectures that the first relation in (26) is strict and does not exclude the possibility that the last two invariants in (26) are equivalent. We settle this by showing that, in fact, both relations in (26) are strict. We actually prove much more: up to one question remaining open, the diagram shown in Figure 1 is exact in the sense that all present arrows are non-reversible and that any two invariants not connected by arrows are provably incomparable. The only remaining question concerns the chain $w_*^{(\bullet)} \rightarrow \dots \rightarrow w_*^{(r)} \rightarrow \dots \rightarrow w_*^{(1)}$; see Problem 17 stated below and an approach to its solution in Theorem 18.

We now present a minimal set of separations from which all other separations follow. For compatibility with Figure 1, we write $\mathcal{I} \nrightarrow \mathcal{I}'$ to negate $\mathcal{I}' \preceq \mathcal{I}$.

WL₁ \nrightarrow Spec. To show this, we have to present two WL_1 -equivalent but not cospectral graphs. The simplest pair of WL_1 -equivalent graphs, $2C_3$ and C_6 , works. Indeed, the eigenvalues of C_n are $2 \cos \frac{2\pi k}{n}$ for $k = 0, 1, \dots, n-1$, and the spectrum of the disjoint union of graphs is the union of their spectra; see, e.g., [10, Example 1.1.4 and Theorem 2.1.1].

EA \nrightarrow main-EA. It is known [9] that among trees with up to 20 vertices there is a single pair of non-isomorphic trees, with 19 vertices, with the same eigenvalues and angles. Using Proposition 9, a direct computation shows that these two trees are not main-EA -equivalent.

gen-Spec \nrightarrow WM. The smallest, with respect to the number of vertices and the number of edges, pair of generalized cospectral graphs consists of 7-vertex graphs G and H where $G = C_6 \cup K_1$ and H is obtained from the 3-star $K_{1,3}$ by subdividing each edge; see [16, Fig. 4]. Since G has an isolated vertex and H does not, these graphs are not WM -equivalent.

gen-Spec \rightarrow **EA**. The same pair of graphs G and H works. They are not EA-equivalent because connectedness of a graph is determined by its spectrum and angles [10, Th. 3.3.3]. Another reason for this is that every graph with at most 9 vertices is determined by its spectrum and angles up to isomorphism [9].

strong-FSI \rightarrow **WL₁**. An even stronger fact, namely $w_*^{(2)} \rightarrow$ **WL₁** is proved as Theorem 13 below. The separation implies that **strong-FSI** \rightarrow **WL₂**, answering one part of Fürer's question. Another consequence is also the separation **WM** \rightarrow **WL₁**, which follows as well from [27, Theorem 3].

WL₁ \rightarrow **WL_{3/2}**. Consider C_6 and $2C_3$.

WL_{3/2} \rightarrow **strong-FSI**. This is Theorem 12 below. As a consequence, **weak-FSI** \rightarrow **strong-FSI**, answering the other part of Fürer's question. Another consequence is the separation **WL_{3/2}** \rightarrow **WL₂** shown in [23].

$w_*^{(\bullet)}$ \rightarrow **WL₂**. This is Theorem 16 below. It considerably strengthens the negative answer to Fürer's question by showing that the whole hierarchy of the invariants $w_*^{(r)}$ for all r is strictly weaker than **WL₂**.

We state and prove the three results announced above in the rest of this section.

6.1 WL_{3/2} is not stronger than strong-FSI

► **Theorem 12.** **strong-FSI** $\not\rightarrow$ **WL_{3/2}**.

Proof. The separation **WL₂** $\not\rightarrow$ **WL_{3/2}** in [23] is shown by constructing a pair of **WL_{3/2}**-equivalent graphs G and H as follows. Consider two copies of $C_6 * K_1$, where $*$ is the join of graphs. Denote the vertices of degree 6 by u_1 and u_2 . Consider also two copies of $(2C_3) * K_1$, denoting their vertices of degree 6 by v_1 and v_2 . The graph G is obtained by adding four edges forming the cycle $u_1 u_2 v_1 v_2$, and the graph H is obtained by adding the cycle $u_1 v_1 u_2 v_2$. In [23] it is observed that G and H are distinguishable by 2-WL. We now strengthen this observation to show that **strong-FSI**(G) \neq **strong-FSI**(H).

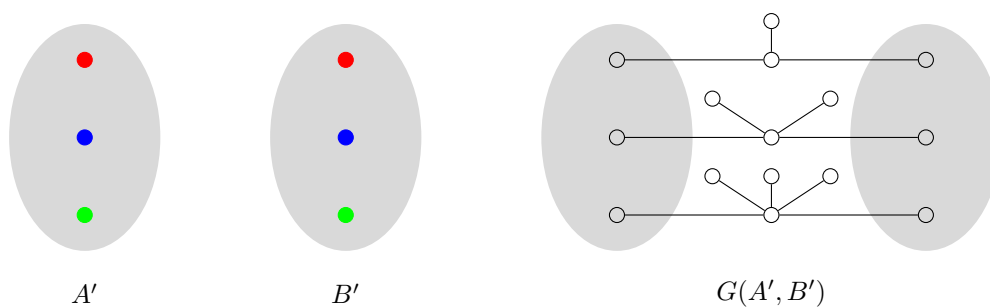
The vertices u_1, u_2, v_1, v_2 will be referred to as Q -vertices. The other vertices are split into two classes, H -vertices and T -vertices, depending on whether they belong to a hexagonal or a triangular part. A vertex x is a Q -vertex exactly when $w_2(x, x) = \deg x = 8$. The H - and the T -vertices are distinguishable by the condition $w_3(x, x) = 6$ for a T -vertex and $w_3(x, x) = 4$ for an H -vertex. Consider an arbitrary T -vertex x in G . Note that from x there is at least one 3-walk to each of the twelve T -vertices y . If we consider a T -vertex x in H , then from x there are 3-walks only to six T -vertices y . This implies that $w_*^{(1)}(G) \neq w_*^{(1)}(H)$. We conclude by Part 3 of Corollary 5 that G and H are not **strong-FSI**-equivalent. ◀

6.2 $w_*^{(2)}$ is not stronger than WL₁

► **Theorem 13.** **WL₁** $\not\rightarrow$ $w_*^{(2)}$.

The proof requires a substantial extension of the approach in [27] to separate various **WL₁**- and **WM**-based concepts.

Construction. Suppose that we have a graph A with m designated vertices a_1, \dots, a_m and a graph B with m designated vertices b_1, \dots, b_m , which will be referred to as *port vertices*. In each of the graphs, the port vertices are colored by different colors. Specifically, a_i and b_i are colored by the same color i . The resulting partially colored graphs are denoted by A' and B' . We construct a graph $G(A', B')$ with no colored vertices as follows. $G(A', B')$



■ **Figure 2** Construction of $G(A', B')$.

consists of the vertex-disjoint union of A and B and a number of new vertices of two sorts, namely *connecting* and *pendant vertices*. For each i , there is a connecting vertex c_i adjacent to a_i and b_i . Moreover, for each i there are i pendant vertices $p_{i,1}, \dots, p_{i,i}$ of degree 1 all adjacent to c_i . An example of the construction for $m = 3$ is shown in Figure 2.

The main lemma. The crux of the proof is the following lemma. Recall that a *strongly regular graph* with parameters (n, d, λ, μ) is an n -vertex d -regular graph where every two adjacent vertices have λ common neighbors, and every two non-adjacent vertices have μ common neighbors. Extending the notation used in Section 4.2, for a graph G we set $WL_1^r(G) = \{ \{ WL_1^r(G, x) \}_{x \in V(G)} \}$.

► **Lemma 14.** *Let A and B be strongly regular graphs with the same parameters (in particular, A and B can be isomorphic). Let A' and B' be their partially colored versions such that each color occurs in A' , as well as in B' , at most once. Assume that $WL_1^0(A') = WL_1^0(B')$, which means the sets of the colors occurring in A' and B' are equal and, therefore, we can construct the uncolored graph $G = G(A', B')$. Consider also $H = G(A', A')$ constructed from two vertex-disjoint copies of A' .*

1. *If $WL_1(A') \neq WL_1(B')$, then $WL_1(G) \neq WL_1(H)$. In words: if color refinement distinguishes A' and B' , then it distinguishes also G and H .*
2. *If $WL_1^r(A') = WL_1^r(B')$ for some $r \geq 1$ (i.e., r rounds of color refinement do not suffice for distinguishing A' and B'), then $w_*^{(r-1)}(G) = w_*^{(r-1)}(H)$.*

The rest of the proof. We can separate WL_1 and $w_*^{(2)}$ by finding partially colored strongly regular graphs A' and B' as in Lemma 14 with $r = 3$. Let $SRG(n, d, \lambda, \mu)$ denote the set of strongly regular graphs with parameters (n, d, λ, μ) . Two suitable colorings A' and B' exist for a graph in the set $SRG(25, 12, 5, 6)$ of *Paulus graphs*, namely for the graph $P_{25,12}$ in Brouwer's collection [5]. These colorings are described in the full version of the paper [1]. They were found by computer search using the Lua package TCSLibLua [14].

Note that $P_{25,12}$ is one of the two Latin square graphs in $SRG(25, 12, 5, 6)$. The other Latin square graph in $SRG(25, 12, 5, 6)$ is $P_{25,15}$, which is the Paley graph on 25 vertices. This is the only vertex-transitive graph in this family. Curiously, it is not suitable for our purposes. Moreover, it seems that strongly regular graphs with less than 25 vertices do not admit appropriate colorings even for $r = 2$.

6.3 Separation of the $w_*^{(r)}$ hierarchy from WL_2

► **Lemma 15.** *Let A and B be (possibly isomorphic) strongly regular graphs with the same parameters. Let A' and B' be their versions, each containing a single individualized vertex. Let $G = G(A', B')$ and $H = G(A', A')$.*

1. $w_*^{(r)}(G) = w_*^{(r)}(H)$ for all r and, therefore, $w_*^{(\bullet)}(G) = w_*^{(\bullet)}(H)$.
2. If $WL_2(A') \neq WL_2(B')$, then $WL_2(G) \neq WL_2(H)$.

Proof. For Part 1, note that $WL_1(A') = WL_1(B')$. Indeed, the distinguishability of A' and B' by 1-WL would imply the distinguishability of A and B by 2-WL, contradicting the assumption that these strongly regular graphs have the same parameters. Thus, $WL_1^r(A') = WL_1^r(B')$ for all $r \geq 1$, and we can apply the second part of Lemma 14. Part 2 is easy. ◀

► **Theorem 16.** $WL_2 \not\preceq w_*^{(\bullet)}$.

Proof. We apply Lemma 15, where A is the Shrikhande graph and B the 4×4 rook's graph, both strongly regular graphs with parameters $(16, 6, 2, 2)$. Since both graphs are vertex-transitive, A' and B' are uniquely defined. The neighborhood of the individualized vertex induces C_6 in the Shrikhande graph and $2C_3$ in the 4×4 rook's graph. Therefore, $WL_2(A') \neq WL_2(B')$. ◀

► **Open Problem 17.** *We leave open the question whether the hierarchy*

$$w_*^{(1)} \preceq w_*^{(2)} \preceq w_*^{(3)} \preceq w_*^{(4)} \preceq \dots \preceq w_*^{(\bullet)} \quad (27)$$

is strict or at least does not collapse to some level. While we know that each $w_^{(r)}$ is strictly weaker than WL_2 , it remains open whether $w_*^{(r)}$ can be stronger than WL_1 for some large r . A negative answer will follow from Lemma 14 if there is an infinite sequence of partially colored strongly regular graphs A'_r and B'_r for $r = 1, 2, 3, \dots$, where the underlying graphs are equal or have the same parameters, such that A'_r and B'_r are distinguished by 1-WL, but requiring at least r refinement rounds.*

Suppose that strongly regular graphs A and B have the same parameters and their partially colored versions A' and B' are distinguished by 1-WL exactly in the $(r + 1)$ -th round. By Lemma 14, $G(A', B')$ and $G(A', A')$ are $w_*^{(r-1)}$ -equivalent and, therefore, this pair of graphs is a good candidate for separation of $w_*^{(r-1)}$ from $w_*^{(r)}$. This approach works indeed pretty well.

► **Theorem 18.** *The hierarchy (27) is strict up to the 4-th level, that is, the first three relations in (27) are strict.*

References

- 1 V. Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On a hierarchy of spectral invariants for graphs. Technical report, arxiv.org/abs/2310.04391, 2023.
- 2 László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC'16)*, pages 684–697, 2016. doi:10.1145/2897518.2897542.
- 3 László Babai, Paul Erdős, and Stanley M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, 1980. doi:10.1137/0209047.
- 4 Amir Rahnamai Barghi and Ilia Ponomarenko. Non-isomorphic graphs with cospectral symmetric powers. *Electron. J. Comb.*, 16(1), 2009. doi:10.37236/209.
- 5 Andries E. Brouwer. Paulus-Rozenfeld graphs. URL: <https://www.win.tue.nl/~aeb/drg/graphs/Paulus.html>.

- 6 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.
- 7 Gang Chen and Ilia Ponomarenko. *Coherent configurations*. Wuhan: Central China Normal University Press, 2019.
- 8 Dragoš Cvetković. The main part of the spectrum, divisors and switching of graphs. *Publ. Inst. Math., Nouv. Sér.*, 23:31–38, 1978. URL: <http://elib.mi.sanu.ac.rs/files/journals/publ/43/6.pdf>.
- 9 Dragoš Cvetković and Mirko Lepović. Cospectral graphs with the same angles and with a minimal number of vertices. *Publ. Elektroteh. Fak., Univ. Beogr., Ser. Mat.*, 8:88–102, 1997. URL: <http://www.jstor.org/stable/43666387>.
- 10 Dragoš Cvetković, Peter Rowlinson, and Slobodan Simić. *An introduction to the theory of graph spectra*, volume 75 of *Lond. Math. Soc. Stud. Texts*. Cambridge: Cambridge University Press, 2010. doi:10.1017/CB09780511801518.
- 11 Dragoš Cvetković, Peter Rowlinson, and Slobodan Simic. *Eigenspaces of Graphs*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1997. doi:10.1017/CB09781139086547.
- 12 Dragoš Cvetković, Peter Rowlinson, and Slobodan Simić. A study of eigenspaces of graphs. *Linear Algebra and its Applications*, 182:45–66, 1993. doi:10.1016/0024-3795(93)90491-6.
- 13 Z. Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, 2010. doi:10.1002/jgt.20461.
- 14 Frank Fuhlbrück. TCSLibLua. GitHub repository, 2023. URL: <https://gitlab.com/tcslib/tcsliblua/-/tree/7e429da7fcf9bf8d93dfc50b54e823a47364cb4b>.
- 15 Martin Fürer. On the power of combinatorial and spectral invariants. *Linear Algebra and its Applications*, 432(9):2373–2380, 2010. doi:10.1016/j.laa.2009.07.019.
- 16 Willem H. Haemers and Edward Spence. Enumeration of cospectral graphs. *European Journal of Combinatorics*, 25(2):199–211, 2004. doi:10.1016/S0195-6698(03)00100-8.
- 17 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Am. Math. Soc., New Ser.*, 43(4):439–561, 2006. doi:10.1090/S0273-0979-06-01126-8.
- 18 Charles R. Johnson and Morris Newman. A note on cospectral graphs. *Journal of Combinatorial Theory, Series B*, 28(1):96–103, 1980. doi:10.1016/0095-8956(80)90058-1.
- 19 Fenjin Liu and Johannes Siemons. Unlocking the walk matrix of a graph. *J. Algebr. Comb.*, 55(3):663–690, 2022. doi:10.1007/s10801-021-01065-3.
- 20 Bojan Mohar and Svatopluk Poljak. Eigenvalues in combinatorial optimization. In *Combinatorial and graph-theoretical problems in linear algebra*, pages 107–151. New York: Springer-Verlag, 1993. doi:10.1007/978-1-4613-8354-3.
- 21 Sean O’Rourke and Behrouz Touri. On a conjecture of Godsil concerning controllable random graphs. *SIAM J. Control. Optim.*, 54(6):3347–3378, 2016. doi:10.1137/15M1049622.
- 22 David L. Powers and Mohammad M. Sulaiman. The walk partition and colorations of a graph. *Linear Algebra Appl.*, 48:145–159, 1982. doi:10.1016/0024-3795(82)90104-5.
- 23 Gaurav Rattan and Tim Seppelt. Weisfeiler-Leman and graph spectra. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA’23)*, pages 2268–2285. SIAM, 2023. doi:10.1137/1.9781611977554.ch87.
- 24 Sirolf. Characterization of k -walk-equivalent graphs, 2018. URL: <https://mathoverflow.net/questions/304106/characterization-of-k-walk-equivalent-graphs>.
- 25 Edwin R. van Dam and Willem H. Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its Applications*, 373:241–272, 2003. doi:10.1016/S0024-3795(03)00483-X.
- 26 E.R. van Dam, W.H. Haemers, and J.H. Koolen. Cospectral graphs and the generalized adjacency matrix. *Linear Algebra and its Applications*, 423(1):33–41, 2007. doi:10.1016/j.laa.2006.07.017.

6:18 On a Hierarchy of Spectral Invariants for Graphs

- 27 Oleg Verbitsky and Maksim Zhukovskii. Canonization of a random graph by two matrix-vector multiplications. In *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 100:1–100:13, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.100.
- 28 B.Yu. Weisfeiler and A.A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Ser. 2*, 9:12–16, 1968. English translation is available at https://www.itl.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

Computing Twin-Width Parameterized by the Feedback Edge Number

Jakub Balabán  

Faculty of Informatics, Masaryk University, Brno, Czech Republic

Robert Ganian  

Algorithms and Complexity Group, TU Wien, Austria

Mathis Rocton  

Algorithms and Complexity Group, TU Wien, Austria

Abstract

The problem of whether and how one can compute the twin-width of a graph – along with an accompanying contraction sequence – lies at the forefront of the area of algorithmic model theory. While significant effort has been aimed at obtaining a fixed-parameter approximation for the problem when parameterized by twin-width, here we approach the question from a different perspective and consider whether one can obtain (near-)optimal contraction sequences under a larger parameterization, notably the feedback edge number k . As our main contributions, under this parameterization we obtain (1) a linear bikernel for the problem of either computing a 2-contraction sequence or determining that none exists and (2) an approximate fixed-parameter algorithm which computes an ℓ -contraction sequence (for an arbitrary specified ℓ) or determines that the twin-width of the input graph is at least ℓ . These algorithmic results rely on newly obtained insights into the structure of optimal contraction sequences, and as a byproduct of these we also slightly tighten the bound on the twin-width of graphs with small feedback edge number.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases twin-width, parameterized complexity, kernelization, feedback edge number

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.7

Related Version *Full Version:* <https://arxiv.org/abs/2310.08243>

Funding *Robert Ganian:* Robert Ganian acknowledges support by the FWF and WWTF Science Funds (FWF project Y1329 and WWTF project ICT22-029).

Mathis Rocton: Mathis Rocton acknowledges support by the European Union’s Horizon 2020 research and innovation COFUND programme (LogiCS@TUWien, grant agreement No 101034440), and the FWF Science Fund (FWF project Y1329).



1 Introduction

Since its introduction by Bonnet, Kim, Thomassé and Watrigant in 2020 [16], the notion of *twin-width* has made an astounding impact on the field of algorithmic model-checking [10, 11, 12, 13, 14]. Indeed, it promises a unified explanation of why model-checking first order logic is fixed-parameter tractable on a number of graph classes which were, up to then, considered to be separate islands of tractability for the model-checking problem, including proper minor-closed graphs, graphs of bounded rank-width, posets of bounded width and map graphs [16]; see also the recent works on other graph classes of bounded twin-width [1, 2, 21]. Beyond this, twin-width was shown to have fundamental connections to rank-width and path-width [14] as well as to matrix theory [13], and has by now been studied even in areas such as graph drawing [21] and SAT Solving [27, 34].



© Jakub Balabán, Robert Ganian, and Mathis Rocton;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 7; pp. 7:1–7:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



And yet, essentially all twin-width based algorithmic results known to date require a corresponding decomposition – a so-called *contraction sequence* – to be provided as part of the input. The fact that the inner workings of these algorithms rely on a contraction sequence is not surprising; after all, the same reliance on a suitable decomposition is present in essentially all graph algorithms parameterized by classical width measures such as treewidth [33] or rank-width [32, 24]. But while optimal decompositions for treewidth and rank-width can be computed in fixed-parameter time when parameterized by the respective width measure [6, 28] and even more efficient algorithms are known when aiming for decompositions that are only a constant-factor worse than optimal [30, 23], the situation is entirely different in the case of twin-width. In particular, it is known that already deciding whether a graph has twin-width at most 4, i.e., admits a 4-contraction sequence, is NP-hard [4] (ruling out fixed-parameter as well as XP algorithms for computing optimal contraction sequences). Moreover, whether one can at least compute approximately-optimal contraction sequences in fixed-parameter time is arguably the most prominent open question in contemporary research of twin-width.

Contribution. Given the difficulty of computing (near-)optimal contraction sequences when parameterized by twin-width itself, in this article we ask whether one can at least compute such contraction sequences under a larger parameterization, i.e., when using an auxiliary parameter which yields stronger restrictions on the input graph¹. Algorithms obtained under such stronger restrictions are not intended to be used as a pre-computation step prior to using twin-width for model checking, but rather aim to further our understanding of the fundamental problem of computing (near-)optimal contraction sequences. In this sense, our work follows in the footsteps of previous work on, e.g., treedepth parameterized by the vertex cover number [29], MIM-width parameterized by the feedback edge number and other parameters [20], treewidth parameterized by the feedback vertex number [9] and the directed feedback vertex number parameterized by the (undirected) feedback vertex number [5].

As our two main contributions, we obtain the first non-trivial fixed-parameter algorithms for computing (near-)optimal contraction sequences.

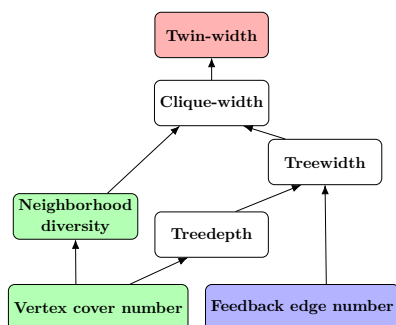
► **Theorem 1.** *The problem of deciding whether the twin-width of an input graph is at most 2 admits a linear bikernel when parameterized by the feedback edge number k . Moreover, a 2-contraction sequence for G (if one exists) can be computed in time $2^{\mathcal{O}(k \cdot \log k)} + n^{\mathcal{O}(1)}$.*

We remark that Theorem 1 providing a *bikernel* [19] (instead of a kernel) is merely due to the output being a *trigraph* [16]. Our second result targets graphs of higher twin-width:

► **Theorem 2.** *There is an algorithm which takes as input an n -vertex graph G with feedback edge number k , runs in time $f(k) \cdot n^{\mathcal{O}(1)}$ for a computable function f , and outputs a contraction sequence for G of width at most $\text{tw}(G) + 1$.*

We note that the graph parameter used in our results – the feedback edge number or equivalently the edge deletion distance to acyclicity – is highly restrictive and provides stronger structural guarantees on the input graph than not only twin-width itself, but also rank-width and treewidth. In a sense, it is one of the two most “restrictive” structural parameters used in the design of fixed-parameter algorithms [35, 3, 26, 25, 22], with the other being the vertex cover number, i.e., the minimum size of a vertex cover (see also Figure 1). But while there is a trivial fixed-parameter algorithm for computing optimal contraction

¹ When approximating width parameters, it is desirable to aim for approximation errors which depend only on the targeted width parameter.



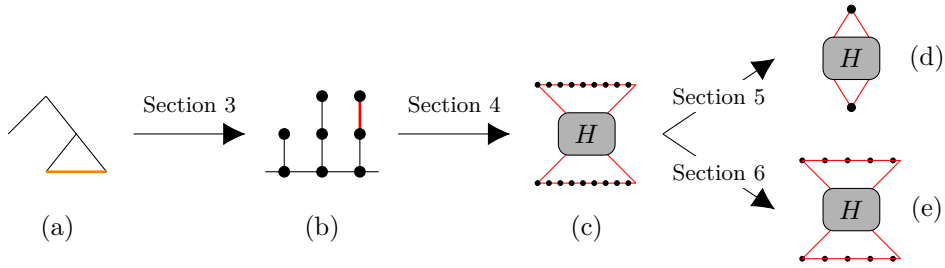
■ **Figure 1** Complexity of computing twin-width with respect to notable structural parameters. A directed path from parameter x to parameter y indicates that y is upper-bounded by a function of x , i.e., that x is more restrictive than y . Green marks parameters where the problem is trivially fixed-parameter tractable; red and white signify para-NP-hardness and cases where the complexity is unknown, respectively; the parameter considered in this paper is highlighted in blue.

sequences w.r.t. the vertex cover number² and also a polynomial-time algorithm for solving the same problem on trees [16], lifting the latter to our more general setting parameterized by the feedback edge number is far from a trivial undertaking and goes hand in hand with the development of new insights into optimal contraction sequences of highly structured graphs.

Proof Overview and Techniques. At its core, both of our results are kernelization routines in the sense that they apply polynomial-time reduction rules in order to transform the input instance into an “equivalent” instance whose size is upper-bounded by a function of the parameter alone. The required reduction rules are in fact very simple – the difficulty lies in proving that they are safe. The used parameterization then allows us to argue that after exhaustive application of these results, we are guaranteed to obtain an instance whose size is upper-bounded by a function of the feedback edge number. Below, we provide a high-level overview of the proof; for brevity, we assume here that readers are familiar with the basic terminology associated with twin-width (as also introduced in Section 2).

The first step towards both desired kernelization algorithms consists of a set of tree-pruning rules which allow us to “cut off” subtrees in the graph that are connected to the rest of the graph via a bridge, i.e., a single edge. Already this step (detailed in Section 3) requires some effort in the context of twin-width, and replaces the cut-off subtrees by one of two kinds of *stumps* which depend on the properties of the replaced subtree. After the exhaustive application of these general rules, in Section 4 we apply a further cleanup step which uses the structural properties guaranteed by our parameter to deal with the resulting stumps. This reduces the instance to an equivalent trigraph consisting of $\mathcal{O}(k)$ -many vertices plus a set of “*dangling*” paths connecting these vertices – paths whose internal vertices have degree 2. We note that some of the reduction rules developed here, and in particular those which allow us to safely remove trees connected via a bridge, provide techniques that could be lifted to remove other kinds of dangling subgraphs and hence may be of general interest. In fact, we

² In particular, this follows by repeatedly deleting vertices which are twins (an operation which is known to preserve twin-width [16]) until one obtains a problem kernel.



■ **Figure 2** (a) The input graph – a tree with k extra edges. (b) Three kinds of stumps, which are obtained by cutting down dangling trees. (c) A small subtrigraph H and long dangling red paths. (d) If the target twin-width is 2, the paths can be shortened to single vertices. (e) Paths can be shortened to bounded length while retaining a guarantee on the twin-width. A detailed example depicting the first two steps is provided in Figure 4.

use our reduction rules to improve the previously known twin-width 2 upper bound from trees to graphs of feedback edge number 1 (Theorem 21), which additionally yields a slightly tighter relationship between twin-width and the feedback edge number (Corollary 22).

The structure of the trigraph at this point seems rather simple: it consists of a bounded-size part plus a small set of arbitrarily long dangling paths. Intuitively, one would like to obtain a bikernel by showing that each sufficiently long dangling path can be replaced by a path of length bounded by some constant without altering the twin-width. In Section 5, we implement this approach by guaranteeing the existence of “well-structured” 2-contraction sequences for graphs of twin-width 2, in turn allowing us to complete the proof of Theorem 1.

The situation becomes significantly more complicated when aiming for contraction sequences for graphs of higher twin-width. In particular, not only does the approach used in Section 5 not generalize, but we prove that there can exist no safe twin-width preserving rule to shorten dangling paths to a constant length, for any constant (see Proposition 9). Circumventing this issue – even when allowing for an additive error of one – in Section 6 forms the most challenging part of our results. The core idea used in the proof here is to partition a hypothetical optimal contraction sequence into a bounded number of stages (defined via so-called *blueprints*). Crucially, we show that the original contraction sequence can be transformed into a “nice” sequence where we retain control over the operations carried out in each stage, at the cost of allowing for a slightly higher width of the sequence. The structure in these nice sequences is defined by aggregating all the descendants of the dangling paths into so-called *centipedes*. Afterward, we use an iterative argument to show that such a well-behaved sequence can also be used to deal with a kernelized trigraph where all the long dangling paths are replaced by paths whose length is not constant, but depends on a function of k .

A mind map of our techniques and algorithmic results is provided in Figure 2.

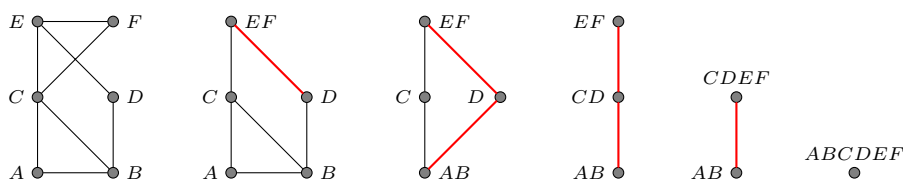
2 Preliminaries

For integers i and j , we let $[i, j] := \{n \in \mathbb{N} \mid i \leq n \leq j\}$ and $[i] := [1, i]$. We assume familiarity with basic concepts in graph theory [18] and parameterized algorithmics [19, 17].

The *length* of a path is the number of edges it contains. An edge set F in an n -vertex graph G is called a *feedback edge set* if $G - F$ is acyclic, and the *feedback edge number* of G is the size of a minimum feedback edge set in G . A *dangling path* in G is a path of vertices which all have degree 2 in G , and a *dangling tree* in G is an induced subtree in G which can be separated from the rest of G by a bridge (see, e.g., Figure 4 later).

Twin-Width. A *trigraph* G is a graph whose edge set is partitioned into a set of *black* and *red* edges. The set of red edges is denoted $R(G)$, and the set of black edges $E(G)$. The *black* (resp. *red*) *degree* of $u \in V(G)$ is the number of black (resp. red) edges incident to u in G . We extend graph-theoretic terminology to trigraphs by ignoring the colors of edges; for example, the degree of u in G is the sum of its black and red degrees. We say a (sub)graph is *black* (resp. *red*) if all of its edges are black (resp. red); for example, P is a red path in G if it is a path containing only red edges. Without a color adjective, the path (or a different kind of subgraph) may contain edges of both colors. We use $G[Q]$ to denote the subtrigraph of G induced on $Q \subseteq V(G)$.

Given a trigraph G , a *contraction* of two distinct vertices $u, v \in V(G)$ is the operation which produces a new trigraph by (1) removing u, v and adding a new vertex w , (2) adding a black edge wx for each $x \in V(G)$ such that $xu, xv \in E(G)$, and (3) adding a red edge wy for each $y \in V(G)$ such that $yu \in R(G)$, or $yv \in R(G)$, or y contains only a single black edge to either v or u . A sequence $C = (G = G_1, \dots, G_n)$ is a *partial contraction sequence* of G if it is a sequence of trigraphs such that for all $i \in [n - 1]$, G_{i+1} is obtained from G_i by contracting two vertices. A *contraction sequence* is a partial contraction sequence which ends with a single-vertex graph. The *width* of a (partial) contraction sequence C , denoted $w(C)$, is the maximum red degree over all vertices in all trigraphs in C ; we also use α -*contraction sequence* as a shorthand for a contraction sequence of width at most α . The *twin-width* of G , denoted $\text{tw}(G)$, is the minimum width of any contraction sequence of G , and a contraction sequence of width $\text{tw}(G)$ is called *optimal*. An example of a contraction sequence is provided in Figure 3.



■ **Figure 3** A 2-contraction sequence of the leftmost graph, consisting of 6 trigraphs.

Let us now fix a contraction sequence $C = (G = G_1, \dots, G_n)$. For each $i \in [n]$, we associate each vertex $u \in V(G_i)$ with a set $\beta(u, i) \subseteq V(G)$, called the *bag* of u , which contains all vertices contracted into u .

Note that if a vertex u appears in multiple trigraphs in C , then its bag is the same in all of them, and so we may denote the bag of u simply by $\beta(u)$. Let us fix $i, j \in [n]$, $i \leq j$. If $u \in V(G_i)$, $v \in V(G_j)$, and $\beta(u) \subseteq \beta(v)$, then we say that u is an *ancestor* of v in G_i and v is the *descendant* of u in G_j (clearly, this descendant is unique). If H is an induced subtrigraph of G_i , then $u \in V(G_j)$ is a *descendant* of H if it is a descendant of at least one vertex of H , and we say that $u \in V(G_i)$ is *contracted to H in G_j* if u is an ancestor of a descendant of H in G_j . A contraction of $u, v \in V(G_j)$ into $w \in V(G_{j+1})$ *involves* $w \in V(G_i)$ if w is an ancestor of uv .

The following definition provides terminology that allows us to partition a contraction sequence into “steps” based on contractions between certain vertices in the original graph.

► **Definition 3.** Let C be a contraction sequence of a trigraph G , and let H be an induced subgraph of G with $|V(H)| = m$. For $i \in [m - 1]$, let $C\langle i \rangle_H$ be the trigraph in C obtained by the i -th contraction between two descendants of H , and let $C\langle 0 \rangle_H = G$. For $i \in [m - 1]$, let U_i and W_i be the bags of the vertices which are contracted into the new vertex of $C\langle i \rangle_H$.

A contraction sequence $C[H] = (H = H_1, \dots, H_m)$ is the restriction of C to H if for each $i \in [m - 1]$, H_{i+1} is obtained from H_i by contracting the two vertices $u, w \in V(H_i)$ such that $\beta(u) = U_i \cap V(H)$ and $\beta(w) = W_i \cap V(H)$.

Next, we introduce a notion that will be useful when dealing with reduction rules in the context of computing contraction sequences.

► **Definition 4.** Let G, G' be trigraphs. We say that the twin-width of G' is effectively at most the twin-width of G , denoted $\text{tww}(G') \leq_e \text{tww}(G)$, if (1) $\text{tww}(G') \leq \text{tww}(G)$ and (2) given a contraction sequence C of G , a contraction sequence C' of G' of width at most $w(C)$ can be constructed in polynomial time. If $\text{tww}(G') \leq_e \text{tww}(G)$ and $\text{tww}(G) \leq_e \text{tww}(G')$, then we say that the two graphs have effectively the same twin-width, $\text{tww}(G') =_e \text{tww}(G)$.

We say that G' is a *pseudoinduced* subtrigraph of G if G' is obtained from an induced subtrigraph of G by the removal of red edges or their replacement with black edges.

► **Observation 5.** If G' is a pseudoinduced subtrigraph of G , then $\text{tww}(G') \leq_e \text{tww}(G)$.

Preliminary Observations and Remarks. We begin by stating a simple brute-force algorithm for computing twin-width.

► **Observation 6.** An optimal contraction sequence of an n -vertex graph can be computed in time $2^{\mathcal{O}(n \cdot \log n)}$.

The following observation not only establishes the twin-width of trees – which form a baseline case for our algorithms – but also notes that the necessary contractions are almost entirely independent of the choice of the root.

► **Observation 7** ([16, Section 3]). For any rooted tree T with root r , there is a contraction sequence C of T of width at most 2 such that the only contraction involving r is the very last contraction in C .

Next, we recall that a 1-contraction sequence can be computed in polynomial time not only on graphs, but also on trigraphs with at most one red edge.

► **Theorem 8** ([15, Section 7]). If G is a trigraph with at most one red edge, then it can be decided in polynomial time whether the twin-width of G is at most 1. In the positive case, the algorithm also returns an optimal contraction sequence of G .

Finally, we formalize the claim made in Section 1 that there can be no “simple” twin-width preserving reduction rule for handling long dangling paths; in particular, any such rule for simplifying dangling paths cannot depend purely on the length of the path itself.

► **Proposition 9.** For every integer $c \geq 1$, there exists a graph G^c with the following properties: (1) G^c contains a dangling path P of length c , (2) $\text{tww}(G^c) \geq 5$, and (3) the graph obtained by subdividing one edge in P (i.e., replacing P with a path P' whose length is $c + 1$) has twin-width at most 4.

Finally, we remark that throughout the paper, we assume the input graph G to be connected; this is without loss of generality, since otherwise one can handle each of the graph’s connected components separately.

3 Cutting Down a Forest

After computing a minimum feedback edge set of an input graph G , the first task on our route towards Theorems 1 and 2 is to devise reduction rules which can safely deal with dangling trees. While it is not possible to delete such trees entirely while preserving twin-width, we show that they can be safely “cut down”; in particular, depending on the structure of the tree it can be replaced by one of the two kinds of stumps defined below.

► **Definition 10.** *Let G be a trigraph, and let $u, v, w \in V(G)$. We say that u has a half stump v if $uv \in E(G)$, and the degree of v in G is 1. We say that u has a red (resp. black) stump vw if $uv \in E(G)$ and $vw \in R(G)$ (resp. $vw \in E(G)$), and the degrees of v and w are 2 and 1, respectively. Half, red, and black stumps are collectively called stumps. The stump vw (or v) then belongs to u .*

We begin by observing that special kinds of subtrees – specifically, stars – can be safely replaced with just a black stump.

► **Observation 11.** *Let G be a trigraph with a dangling tree T connected to the rest of the graph via the bridge $e = uv$ where $v \in V(T)$. Assume that T is a black star consisting of at least one vertex other than v . Then G has effectively the same twin-width as the trigraph G' obtained from $G - T$ by adding a black stump to u .*

Next, we show that all dangling trees not covered by Observation 11 can be safely cut down to a red stump, as long as the trigraph obtained by the cutting has twin-width at least 2 (a condition that will be handled later on). The proof of this case is significantly more difficult than the previous one.

► **Lemma 12.** *Let G be a trigraph with a bridge $e = uv$ such that the connected component T of $G - \{e\}$ containing v is a black dangling tree which contains a vertex at distance 2 from v and let G' be the trigraph obtained from $G - T$ by adding a red stump to u . If $\text{tw}(G) \geq 2$ and $\text{tw}(G') \geq 2$, then G and G' have effectively the same twin-width.*

Proof Sketch. We begin by establishing $\text{tw}(G) \leq_e \text{tw}(G')$. Let C' be a contraction sequence of G' , and we will show how to construct a contraction sequence $C = (G = G_0, G_1, \dots)$ of G of width at most $w(C')$. C starts with contracting T , following the contraction sequence C_T given by Observation 7 (with v as the root), but stops before the first contraction involving v (which is the very last contraction in C_T). Let G_i be the obtained trigraph. By definition of C_T , no vertex of G_j , $j \in [i]$, exceeds the bound on its red degree since $w(C_T) \leq 2 \leq \text{tw}(G')$. G_i is isomorphic to G' and so from here on, C follows C' .

Our task in the remainder of the proof will be to establish $\text{tw}(G') \leq_e \text{tw}(G)$. Let $C = (G, G_1, \dots)$ be a contraction sequence of G , and let $w, x \in V(T) \setminus \{v\}$ be such that $vw, wx \in E(T)$. We need to construct a contraction sequence of G' of width at most $w(C)$; note that there can be vertices with red degree up to 2 in this desired sequence since $w(C) \geq \text{tw}(G) \geq 2$. Let $G^- := G[V(G - T) \cup \{v, w\}]$. Observe that the only difference between G' and G^- is the color of the edge vw (it is red in G' , but black in G^-) and let $C^- = (G^- = G_0^-, G_1^-, \dots, G_m^-)$ be the restriction of C to G^- ; hence $w(C^-) \leq w(C)$. Let us consider the contraction sequence $C' = (G' = G'_0, G'_1, \dots, G'_m)$ of G' which follows C^- in each step (i.e., $V(G_i^-) = V(G'_i)$ for all $i \in [m]$). To avoid any confusion, we explicitly note that it is not possible to rule out $w(C') > w(C^-)$. We complete the proof by performing a case distinction that will allow us to either guarantee $w(C') \leq w(C^-)$, or – in the most difficult case – construct a new contraction sequence C'' of G' such that $w(C'') \leq w(C)$. ◀

Let us now provide some intuition on how we aim to apply Lemma 12 (a formalization is provided in the proof of Theorem 17 at the end of this section). Assume without loss of generality that the input graph G has twin-width at least 2. The first time we want to apply Lemma 12 to go from G to G' , we can verify that $\text{tw}(G') \geq 2$ by Theorem 8; if this check fails then we show how to construct a 2-contraction sequence of G , and otherwise we replace T with a red stump as per the lemma statement. For every subsequent application of Lemma 12, G' will have two red edges and hence we cannot rely on Theorem 8 anymore – but instead, we can guarantee that the condition on G' holds by establishing the following lemma.

► **Lemma 13.** *Let G be a connected trigraph with two red stumps. Then $\text{tw}(G) \geq 2$.*

With Observation 11 and Lemmas 12-13, we can effectively preprocess (tri)graphs by “cutting down” all dangling trees (i.e., replacing them with stumps). However, we will also need to deal with the fact that a vertex could now be connected to many distinct stumps. For half stumps this is not an issue, as multiple half stumps can be assumed to be contracted into a single half stump due to the vertices in them being twins. For all pairs of other kinds of stumps (except for a pair consisting of a half and a black stump), we show that it is sufficient to replace these with a single stump instead.

► **Observation 14.** *Let G' be a trigraph such that $\text{tw}(G') \geq 2$, let $u \in V(G')$ be a vertex with a red stump S in G' , and let G be a trigraph obtained from G' by adding an additional stump to u . Then G' and G have effectively the same twin-width.*

► **Lemma 15.** *Let G' be a trigraph, let $u \in V(G')$ be a vertex with a red stump S in G' , and let G be a trigraph obtained from G' by removing S and adding two black stumps to u . If $\text{tw}(G) \geq 2$ and $\text{tw}(G') \geq 2$, then G' and G have effectively the same twin-width.*

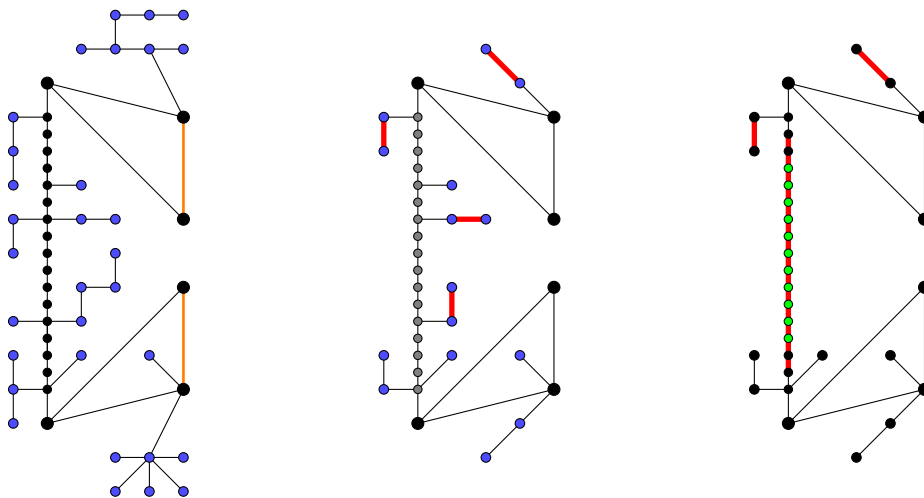
We conclude this section by formalizing the trigraph that can be obtained through the exhaustive application of the reduction rules arising from Observations 11, 14 and Lemmas 12, 15. We say that an induced subtrigraph in G is a *dangling pseudo-path* if it can be obtained from a dangling path P in G by adding, to each of the vertices in P , either (a) one red stump or (b) at most one black stump and at most one half stump.

► **Definition 16.** *A connected trigraph G with $\text{tw}(G) \geq 2$ is an (H, \mathcal{P}) -graph if \mathcal{P} is a set of dangling pseudo-paths in G , and there are two disjoint induced subtrigraphs of G , H and $\sqcup \mathcal{P}$ (the disjoint union of all paths in \mathcal{P}), such that each vertex of G belongs to one of them.*

We proceed with some related terminology that will be used extensively in the subsequent sections. A vertex $u \in V(H)$ is a *connector in G* if u is adjacent to a vertex of $\sqcup \mathcal{P}$ in G . We say that $P \in \mathcal{P}$ is *original* if all edges in P that do not belong to a stump are black and the edges connecting the endpoints of P to H are also black. Later, we will also deal with *tidy* paths in \mathcal{P} , where $P \in \mathcal{P}$ is tidy if P is a dangling red path (i.e., contains no stumps) which additionally satisfies the following three technical conditions for each connector $u \in V(H)$ adjacent to an endpoint v of P :

1. u has black degree 0;
2. v is the only neighbor of u in $\sqcup \mathcal{P}$; and
3. u has a unique neighbor u' in $V(H)$ and u' has positive black degree.

We say that an (H, \mathcal{P}) -graph is *original (tidy)* if all paths $P \in \mathcal{P}$ are original (tidy, respectively). An illustration of these notions is provided in Figure 4, which also showcases the outcome of applying the culmination of this section – Theorem 17 – on an input graph.



■ **Figure 4 Left:** A graph G with feedback edge number two. Feedback edges are orange, vertices in dangling trees are blue. **Middle:** The original (H, \mathcal{P}) -graph obtained from G after all dangling trees have been cut down (i.e., the outcome of Theorem 17). A dangling pseudo-path is depicted via the grey vertices. **Right:** The tidy (H, \mathcal{P}) -graph that will later be obtained from the original (H, \mathcal{P}) -graph by applying Corollary 20 at the end of Section 4. Here, \mathcal{P} contains a single tidy dangling path which is colored green, and all other vertices lie in H .

► **Theorem 17.** *There is a polynomial-time procedure which takes as input a graph G with feedback edge number k and either outputs an optimal contraction sequence of G of width at most 2, or an original (H, \mathcal{P}) -graph G' with effectively the same twin-width as G such that $|V(H)| \leq 16k$ and $|\mathcal{P}| \leq 4k$.*

4 Cleaning the Paths

In the second phase of our proof, our aim is to simplify the instance even further after Theorem 17 in order to reach a trigraph which is “clean” enough to support the path reduction rules developed in the next sections. In particular, we will show that all the dangling pseudo-paths arising from Theorem 17 can be safely transformed into red dangling paths, resulting in an (H, \mathcal{P}) -graph that is tidy as per the definition in the previous section.

We first show how to deal with stumps belonging to a single vertex.

► **Observation 18.** *Let G be a trigraph, let $u \in V(G)$ be a vertex with a single stump or a black and a half stump, and let G' be the trigraph obtained from G by deleting the stumps belonging to u and making all edges incident to u red. There is a partial contraction sequence from G to G' of width $\max\{d_1 + 1, d_2\}$, where d_1 is the red degree of u in G and d_2 is the maximum red degree of any vertex in G' .*

Next, we establish that if a dangling pseudo-path has two consecutive vertices without stumps, the edge between them can be turned red. We remark that this lemma will be applied to subtrigraphs of the considered trigraph, and hence we cannot assume that the trigraph has twin-width at least 2.

► **Lemma 19.** *Let G be a trigraph, let (u_1, u_2, u_3, u_4) be an induced path in G in this order, and assume that the degree of u_2 and u_3 is 2 in G and that the degree of $u \in \{u_1, u_4\}$ in G is 3 if u has a single stump, 4 if u has a half stump and a black stump, and 2 otherwise. Let G' be the trigraph obtained from G by changing the color of the edge u_2u_3 to red. Given a contraction sequence C of G , a contraction sequence C' of G' such that $w(C') = \max\{2, w(C)\}$ can be constructed in polynomial time.*

With Lemma 19, we show that a dangling pseudo-path can either be safely transformed into a “real” dangling path, or (if the path is too short) absorbed into H , which in turn allows us to prove:

► **Corollary 20.** *There is a polynomial-time algorithm which transforms an original (H, \mathcal{P}) -graph into a tidy (H', \mathcal{P}') -graph with effectively the same twin-width such that $|V(H')| \leq |V(H)| + 24 \cdot |\mathcal{P}|$ and $|\mathcal{P}'| \leq |\mathcal{P}|$.*

Before we proceed towards establishing our main algorithmic theorems, we remark that Theorem 17 and Corollary 20 allow us to bound the twin-width of graphs with feedback edge number 1, generalizing the earlier result of Bonnet et al. [16, Section 3] for trees.

► **Theorem 21.** *Every graph with feedback edge number 1 has twin-width at most 2.*

As an immediate corollary, we can obtain an even more general statement:

► **Corollary 22.** *Every graph with feedback edge number $\ell \geq 1$ has twin-width at most $1 + \ell$.*

5 Establishing Theorem 1: Recognizing Twin-width 2

Our aim now is to make the step from Corollary 20 towards a proof of Theorem 1. Towards this, let us fix a tidy n -vertex (H, \mathcal{P}) -graph G . When dealing with a contraction sequence, we will use G_i to denote the i -th trigraph obtained from G , and let H_i be the subtrigraph of G_i induced by the descendants of H . We say that $u \in V(G_i)$ is an *outer vertex* in G_i if $u \notin V(H_i)$, and we lift the previous definition of connectors by saying that u is a *connector* in G_i if $u \in V(H_i)$ and u is adjacent to an outer vertex in G_i .

We begin with a simple observation which will be useful throughout the rest of the section.

► **Observation 23.** *If G_i is a trigraph obtained from a tidy (H, \mathcal{P}) -graph G by a sequence of contractions, then all outer vertices and all connectors in G_i have black degree 0 in G_i .*

Our proof of Theorem 1 relies on establishing that if a tidy (H, \mathcal{P}) -graph G has twin-width 2, then it also admits a contraction sequence which is, in a sense, “well-behaved”. The proof of this fact is based on induction, and hence being “well-behaved” (formalized under the notion of *regularity* below) is defined not only for entire sequences but also for prefixes.

► **Definition 24.** *Let $C = (G_1 = G, G_2, \dots, G_n)$ be a contraction sequence. For $P \in \mathcal{P}$, let us denote by P_i the subtrigraph of G_i induced by the descendants of P which are not in H_i . We say that a prefix (G_1, \dots, G_i) of C is regular if:*

- for all $j \in [i]$, $\{P_j \mid P \in \mathcal{P}\}$ is a set of disjoint red paths, and the endpoints of these paths are adjacent to connectors; and
- for all $j \in [i - 1]$:
 1. G_{j+1} is obtained by a contraction inside H_j , or
 2. there is $P \in \mathcal{P}$ such that if you shorten P_j by one vertex in G_j , you obtain G_{j+1} , or
 3. there is $P \in \mathcal{P}$ such that $|V(P_j)| = 1$ and $|V(P_{j+1})| = 0$.

Let $\text{reg}(C)$ denote the length of the longest regular prefix of C . Below, we show that unless we reach a degenerate trigraph (a case which is handled in the proof of Proposition 26 later), every contraction sequence of width 2 can be made “more regular” until it is entirely regular.

► **Lemma 25.** *Let $C = (G_1 = G, G_2, \dots, G_n)$ be an optimal contraction sequence of G of width 2, and let $i := \text{reg}(C)$. If $i < n$ and G_i is not a red cycle of length 4, then there is an optimal contraction sequence C' of G such that $\text{reg}(C') > i$.*

We can now use Lemma 25 to show that contracting all the tidy dangling paths in G into singletons cannot increase the twin-width of G .

► **Proposition 26.** *Let G' be the trigraph obtained from a tidy (H, \mathcal{P}) -graph G of twin-width 2 by shortening each path in \mathcal{P} to a single vertex. Then $\text{tw}(G') = 2$.*

With Proposition 26 in hand, we can complete the proof of Theorem 1.

► **Theorem 1.** *The problem of deciding whether the twin-width of an input graph is at most 2 admits a linear bikernel when parameterized by the feedback edge number k . Moreover, a 2-contraction sequence for G (if one exists) can be computed in time $2^{\mathcal{O}(k \cdot \log k)} + n^{\mathcal{O}(1)}$.*

Proof. First we use Theorem 17: if it returns an optimal contraction sequence of the input graph G_0 , we immediately know its twin-width. Otherwise, we obtain an original (H, \mathcal{P}) -graph G with effectively the same twin-width as G_0 such that $|V(H)| \leq 16k$ and $|\mathcal{P}| \leq 4k$. Now we use Corollary 20 to transform G into a tidy (H', \mathcal{P}') -graph G' that has effectively the same twin-width as G and that satisfies $|V(H')| \leq 112k$ and $|\mathcal{P}'| \leq 4k$. By transitivity of $=_e$, we obtain $\text{tw}(G_0) =_e \text{tw}(G')$. Finally, let G'' be the trigraph obtained from G' by shortening each path in \mathcal{P}' to a single vertex.

By Proposition 26, $\text{tw}(G') = 2$ implies $\text{tw}(G'') = 2$. Conversely, given a contraction sequence C'' of G'' , we can construct a contraction sequence C' of G' of width at most $w(C'')$ by first shortening each path in \mathcal{P}' to a single vertex via progressive contractions of consecutive vertices, and then following C'' ; thus, $\text{tw}(G') \leq_e \text{tw}(G'')$. Since $\text{tw}(G') \geq 2$, we obtain that $\text{tw}(G'') = 2$ implies $\text{tw}(G') = 2$. By combining these two implications with $\text{tw}(G_0) = \text{tw}(G')$, we obtain that $\text{tw}(G_0) = 2$ if and only if $\text{tw}(G'') = 2$. Since all operations required to construct G'' from G_0 can be performed in polynomial time and $|V(G'')| \leq 116k$, G'' is indeed a linear bikernel for the considered problem.

Finally, if $\text{tw}(G_0) \leq 2$, an optimal contraction sequence of G_0 can be computed in the desired time: either it is given by Theorem 17, or we construct G'' in polynomial time and compute an optimal contraction sequence of G'' in time $2^{\mathcal{O}(k \cdot \log k)}$ as per Observation 6, and then the result follows by the effectiveness in $\text{tw}(G_0) =_e \text{tw}(G') \leq_e \text{tw}(G'')$. ◀

6 Establishing Theorem 2: Almost-Optimal Contraction Sequence

We now move on to the most involved part of the paper: the final step towards proving Theorem 2, which we will outline in the next few paragraphs. Recall that after applying Corollary 20, we obtain a tidy (H, \mathcal{P}) -graph G with effectively the same twin-width as the input graph. Now we “only” need to show that the dangling paths in \mathcal{P} can be shortened to length bounded by the input parameter without increasing the twin-width too much. As we noted earlier, there is no “local” way of shortening a dangling path (see Proposition 9).

Instead, our approach is based on establishing the existence of a $(\text{tw}(G) + 1)$ -contraction sequence C^* for the trigraph G^* obtained from G by shortening its long paths; C^* is obtained by non-trivially repurposing a hypothetical optimal contraction sequence $C = (G_1 =$

G, G_2, \dots, G_n) of G . Once we do that, we will have proven that our algorithm can produce a near-optimal contraction sequence for G by first shortening the paths (by contracting neighbors) and then, when the paths are as short as in G^* , by applying Observation 6. In other words, the complex machinery devised in this section is required for the correctness proof, while the algorithm itself is fairly simple.

Intuitively, the reason why it is difficult to go from C to C^* is that C has too much “freedom”: it can perform arbitrary contractions between vertices of the dangling paths, whereas the only limitation is that the red degrees cannot grow too high (this was not an issue in Section 5, since having twin-width 2 places strong restrictions on how the dangling paths may interact). We circumvent this issue by not following C too closely when constructing C^* . Instead, we only look at a bounded number of special trigraphs in C , called *checkpoints* – forming the *big-step* contraction sequence defined later – and show that we can completely ignore what happened in C between two checkpoints when constructing C^* . Moreover, while checkpoints may be large and complicated, we identify for each checkpoint a small set of characteristics which will be sufficient to carry out our construction; we call this set the *blueprint* of the checkpoint, and it also includes the induced subtrigraph H_i of all descendants of H in a checkpoint G_i (the so-called *core*).

Our aim is to simulate the transition from one checkpoint to another via a “controlled” contraction sequence. For this purpose, we define *representatives* – trigraphs in C^* which match the blueprints of checkpoints in C – and show how to construct a partial contraction sequence from one representative to another in Subsection 6.3. In the end, these partial contraction sequences will be concatenated to create C^* .

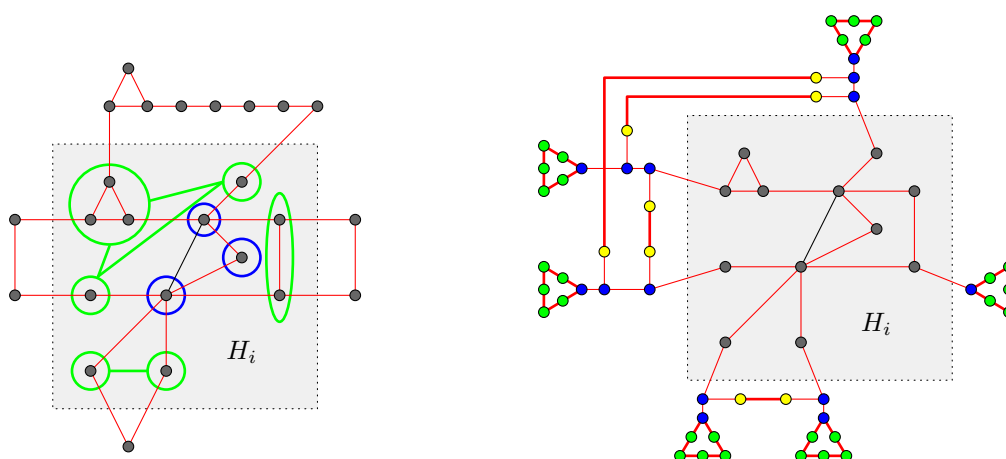
A crucial gadget needed to define these representatives are *centipedes*; these are well-defined and precisely structured objects which simulate the (possibly highly opaque) connections between red components in the core, and are illustrated in Figure 5 later. Subsection 6.2 is dedicated to establishing the operations required to alter the structure and placement of centipedes between individual representatives. These operations rely on the fact that a vertex is allowed to have red degree 4; this is one reason why Theorem 2 produces sequences whose width may be one larger than the optimum (the other reason is that the possibility of supporting an additional red edge provides more flexibility when moving and altering the centipedes between checkpoints).

One final issue we need to deal with is that the paths in \mathcal{P} must be sufficiently long in order to support the creation of the centipedes at the beginning of C^* . Fortunately, there is a simple way of resolving this: paths in \mathcal{P} which are not long enough can be moved into H . However, the cost of this is that each time we add such a path into H , the size of H – and hence also the bound on the length of the paths in G^* – can increase by an exponential factor. For this reason, unlike in the previous section, the bikernel we obtain is not polynomial and not even elementary; its size will be bounded by a tower of exponents whose height is linear in the parameter.

6.1 Initial Setup

Recall that at this point, we are dealing with a tidy (H, \mathcal{P}) -graph G . Let $C = (G_1 = G, G_2, \dots, G_n)$ be a contraction sequence of G , and recall that H_i denotes the subtrigraph of G_i induced on the descendants of H and that we call vertices not in H_i *outer*.

We say that G_i is *decisive* if $H_i \neq H_{i-1}$ or $i = 1$. We define the *big-step contraction sequence* C_{BS} as a subsequence of C which contains G_i if and only if G_i or G_{i+1} is decisive. We call the trigraphs in C_{BS} *checkpoints*. Furthermore, we define $f_H : \mathbb{N} \rightarrow \mathbb{N}$ as follows:



■ **Figure 5 Left:** A possible trigraph G_i . The components of H_i^R (i.e., vertices of B_i) are marked by green or blue circles: blue means isolated and green means non-isolated. The edges of B_i are depicted as green lines. **Right:** a representative for G_i . Blue vertices lie in the body of a centipede, yellow vertices form legs of centipedes, and green vertices lie in the tail of a centipede. Note that the tails and the leg paths, which are drawn as thick red lines, contain many vertices.

$f_H(\ell) = (3^{|V(H)|+4} \cdot |V(H)|^2)^\ell$. Informally, this function describes how big the centipedes will need to be in the ℓ -th trigraph in C_{BS} , counted from the end (the centipedes need to be the largest at the beginning, as they shrink during each transition between checkpoints).

Now we define the blueprints; these capture the information we need about the checkpoints.

► **Definition 27.** The blueprint of a trigraph G_i in C , denoted \mathcal{B}_i , is the tuple (H_i, B_i) where B_i is the vertex-labeled graph constructed in the following way:

- let H_i^R be the subgraph obtained from H_i by removing every edge that is black or incident to a black edge;
- $V(B_i)$ is the set of connected components of H_i^R (the fully-red components);
- $U \in V(B_i)$ is labeled isolated if for all $u \in U$, we have $N_{G_i}(u) \subseteq V(H_i)$, and otherwise it is non-isolated;
- $UU' \in E$ if there is a (red) path between some vertices $u \in U$ and $u' \in U'$ in G_i which contains no vertex of $V(H_i)$ except for u and u' . $E(B_i)$ is then the transitive closure of E . Observe that isolated vertices have degree 0 in B_i but a non-isolated vertex may have degree 0, too; see Figure 5 for an illustration.

A reader may wonder why we define the edge relation of B_i to be transitive. The reason is that with this definition, a connection can disappear only when H_i changes, see Lemma 32 below. We are now ready to define centipedes – the technical gadget underlying our entire construction.

► **Definition 28.** Let $d \geq 0$ and $\ell \geq 1$. The centipede $cen(d, \ell)$ is the following graph:

- the vertex set consists of three disjoint sets: the body $\{u_i \mid i \in [d + 1]\}$, the legs $\{v_i \mid i \in [d]\}$, and the tail $\{w_i \mid i \in [\ell]\}$;
- the edge set is $\{u_i u_{i+1}, u_i v_i \mid i \in [d]\} \cup \{w_i w_{i+1} \mid i \in [\ell - 1]\} \cup \{u_{d+1} w_1, u_{d+1} w_\ell\}$.

We say that u_1 is the head of $cen(d, \ell)$. Let G' be a supergraph of $cen(d, \ell)$. We say that a leg v of $cen(d, \ell)$ is free in G' if its degree is 1 in G' . For $x \in V(G')$, we say that $cen(d, \ell)$ is attached to x if x is adjacent to the head of $cen(d, \ell)$.

Now we define the representatives; trigraphs representing the blueprints in C' . Note that ℓ specifies how big the centipedes in the representative need to be.

► **Definition 29.** Let $\mathcal{B}_i = (H_i, B_i)$ be a blueprint and ℓ be an integer. Now a representative for \mathcal{B}_i of order ℓ , denoted R_i^ℓ , is a trigraph that can be built as follows:

1. start with H_i ; let $\mathcal{U} \subseteq V(B_i)$ be the set of non-isolated vertices of B_i ;
2. for each $U \in \mathcal{U}$, add a red centipede $\Psi_U \cong \text{cen}(\deg_{B_i}(U), f_H(\ell))$;
3. for all $U \in \mathcal{U}$, add a red edge between the head of Ψ_U and some vertex of U which has at least one neighbor outside of H_i in G_i (there must be at least one such vertex because U is non-isolated);
4. for each edge $UU' \in E(B_i)$, do the following:
 - let w (resp. w') be a leg of Ψ_U (resp. $\Psi_{U'}$) free in the current trigraph. Add a red path of length $f_H(\ell)$ connecting w and w' . We call this path, including the two legs, the leg path connecting Ψ_U and $\Psi_{U'}$.

Notice that the construction in Definition 29 works by a simple inductive argument because the number of legs of Ψ_U is $\deg_{B_i}(U)$ by construction; an illustration is provided in Figure 5. Moreover, observe that a representative is not uniquely determined by i and ℓ ; the order of leg paths, as well as the vertices which the centipedes are attached to, can differ.

6.2 Moving Centipedes Around

In this subsection, we describe several operations, i.e., partial contraction sequences, which will later be used to obtain a partial contraction sequence which transitions from a representative of one checkpoint in C_{BS} to a representative of the next checkpoint. These operations will focus on the centipedes introduced in the previous subsection, and may result in trigraphs which are not necessarily a representative for any graph in C ; we refer to these obtained trigraphs as *intermediate graphs* and note that their structure can be precisely formalized.

We start with a few simple operations on generalized intermediate graphs:

► **Observation 30** (Shortening a centipede). *We can shorten the tail of a centipede Ψ , by contracting two neighboring vertices belonging to the tail, to any length. Similarly, we can shorten a leg path to any non-zero length.*

► **Observation 31** (Destroying a centipede). *Let G' be an intermediate graph, let $u \in V(H')$, and let $\Psi = \text{cen}(0, \ell)$ be a centipede attached to u . We can destroy Ψ , i.e., there is a partial contraction sequence of width at most $\text{tw}(G) + 1$ from G' to $G' - V(\Psi)$.*

The remaining operations allow us to

1. reorder the legs of a centipede,
2. move a centipede to a different vertex of the core,
3. connect two centipedes by a new leg path,
4. merge two centipedes into a single centipede, and
5. split a centipede into two new centipedes.

Each of these operations can be formalized by carefully prescribing the initial and final intermediate graph, the impact on the length of the involved centipedes, and a proof ensuring that the contraction subsequence between the initial and final intermediate graph has width at most $\text{tw}(G) + 1$.

6.3 Contraction Sequences for Representatives

In this subsection, we will construct partial contraction sequences between the representatives of consecutive checkpoints by making use of the operations with the centipedes defined in Subsection 6.2. We distinguish between two cases depending on whether the core changes between the checkpoints or not, see Lemmas 33 and 34 below.

We start by observing that if the core does not change between two consecutive checkpoints, then the blueprint of the latter checkpoint contains all edges present in the blueprint of the former checkpoint.

► **Lemma 32.** *Let G_i and G_j be two consecutive trigraphs in C_{BS} such that $H_i = H_j$. This means that $V(B_i) = V(B_j)$, see Definition 27. It holds that $E(B_i) \subseteq E(B_j)$.*

We are now ready to define the partial contraction sequence between two representatives in the case when the core does not change.

► **Lemma 33.** *Let G_i, G_j ($i < j$) be two consecutive trigraphs in C_{BS} , such that $H_i = H_j$. For any $\ell \in \mathbb{N}$, there is a partial contraction sequence from $R_i^{\ell+1}$ to R_j^ℓ whose width is at most $\text{tw}(G) + 1$.*

Proof Sketch. Even though the number of contraction happening between G_i and G_j in C can be huge, the effect on the blueprints (and thus the representatives) is somewhat limited. After checking what could differ between the two representatives, we present a sequence of operations on the centipedes – namely moving, creating, connecting, shortening and destroying centipedes – which is sufficient to obtain R_j^ℓ from $R_i^{\ell+1}$.

To prove that this sequence of operation on centipedes is feasible, we verify that the tails and leg paths are sufficiently longer in $R_i^{\ell+1}$ than in R_j^ℓ to sustain the operations without becoming too short. Moreover, the control we have over the operations enables us to make sure that no vertex has a red degree higher than $\text{tw}(G) + 1$ at any point in the created contraction sequence. ◀

Next, we define the partial contraction sequence between two representatives in the second and final case, namely the case when the core does change. Note that in this case, we allow the sequence to terminate in a slightly different trigraph (which will be handled by Lemma 35).

► **Lemma 34.** *Let G_i, G_j ($i < j$) be two consecutive trigraphs in C_{BS} such that $H_i \neq H_j$. For any $\ell \in \mathbb{N}$, there is a partial contraction sequence C^p from $R_i^{\ell+1}$ to a trigraph that is a pseudoinduced subtrigraph of R_j^ℓ such that $w(C^p) \leq \text{tw}(G) + 1$.*

We now combine the previous two lemmas to obtain a contraction sequence of the representative for G_1 – the first trigraph in C_{BS} as well as in C . More generally:

► **Lemma 35.** *Let G_i be a trigraph in C_{BS} such that there are ℓ trigraphs after G_i in C_{BS} . There is a contraction sequence of R_i^ℓ whose width is at most $\text{tw}(G) + 1$.*

One more thing we need to do is initialization: the trigraph we are interested in, i.e., the trigraph obtained from G by shortening all dangling paths to bounded length, does not contain any centipedes. This is handled by (the proof of) Theorem 36 below, which also summarizes the outcome of this subsection.

► **Theorem 36.** *Let G be a tidy (H, \mathcal{P}) -graph such that $\text{tw}(G) \geq 3$ and all paths in \mathcal{P} have length at least $3 \cdot f_H(|2V(H)|^2) + 9$ and let $G' = (H, \mathcal{P}')$ be any trigraph obtained from G by shortening paths in \mathcal{P} to arbitrary lengths no shorter than $3 \cdot f_H(2|V(H)|^2) + 9$. Then $\text{tw}(G') \leq \text{tw}(G) + 1$.*

6.4 Putting Everything Together

We are now ready to prove Theorem 2.

► **Theorem 2.** *There is an algorithm which takes as input an n -vertex graph G with feedback edge number k , runs in time $f(k) \cdot n^{\mathcal{O}(1)}$ for a computable function f , and outputs a contraction sequence for G of width at most $\text{tw}(G) + 1$.*

Proof Sketch. We begin by handling the case where $\text{tw}(G) \leq 2$ by invoking Theorems 8 and 1. For the rest of the proof, we assume $\text{tw}(G) \geq 3$. Here, we first use Theorem 17 to get in $n^{\mathcal{O}(1)}$ time an original (H, \mathcal{P}) -graph such that $|V(H)| \leq 16k$ and $|\mathcal{P}| \leq 4k$. Recall that this (H, \mathcal{P}) -graph has effectively the same twin-width as G , so any optimal contraction sequence for it can be lifted to an optimal one for G . Using Corollary 20, we obtain – also in $n^{\mathcal{O}(1)}$ time – a tidy (H', \mathcal{P}') -graph G' such that $|V(H')| \leq 112k$, $|\mathcal{P}'| \leq 4k$, and G' still has effectively the same twin-width as G .

At this point, we check the length of each path in \mathcal{P}' , whereas if we identify a path $P \in \mathcal{P}'$ whose length is below the bound required by Theorem 36 (w.r.t. the current size of H'), we add P into H' and update our choices of H' and \mathcal{P}' accordingly. After exhaustively completing the above check, we are guaranteed to have satisfied the conditions of Theorem 36. We now begin constructing our contraction sequence for G' as follows. First, we iteratively contract the paths which remain in \mathcal{P}' until they have length precisely $3 \cdot f_{H'}(2|V(H')|^2) + 9$; recall that by Theorem 36, we are guaranteed that the resulting graph G^* has twin-width at most one larger than G (and also G'). Moreover, the number of vertices in G^* can be upper-bounded by a non-elementary function of our parameter, specifically $2^{2^{\dots} 2^{\mathcal{O}(\log(k))}}$ where the height of the tower of exponents is upper-bounded by $4k + 3$. At this point, we apply Observation 6 to construct an optimal contraction sequence of G^* and append it after the trivial sequence of contractions which produced G^* . The proof now follows by the fact that G' has effectively the same twin-width as G . ◀

7 Concluding Remarks

While the feedback edge number parameterization employed by our algorithms is highly restrictive, we believe Theorems 1 and 2 represent a tangible and important first step towards more general algorithms for computing near-optimal contraction sequences, with the “holy grail” being a fixed-parameter algorithm for computing near-optimal contraction sequences parameterized by twin-width itself. The natural next goals in this line of research would be to obtain fixed-parameter algorithms for the problem when parameterized by treedepth [31] and then by treewidth [33].

Towards this direction, we note that it is not at all obvious how one could apply classical tools such as *typical sequences* [7, 20, 8] in the context of computing contraction sequences. At least for treedepth, it may be possible to employ the general approach developed in Section 6 – in particular, establishing the existence of a near-optimal but “well-structured” contraction sequence and using that to identify safe reduction rules – but the details and challenges arising there seem to differ significantly from the ones handled in this article.

Last but not least, we remark that the algorithms developed here rely on reduction rules which are provably safe, simple to implement, and run in polynomial time; we believe these may potentially be of interest for heuristic and empirical purposes. We also believe that the additive error of 1 incurred by Theorem 2 is avoidable, albeit this may perhaps be seen as a less pressing question than settling the approximability of twin-width under the parameterizations outlined in the previous paragraph.

References

- 1 Jakub Balabán and Petr Hlinený. Twin-width is linear in the poset width. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPICs*, pages 6:1–6:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.6.
- 2 Jakub Balabán, Petr Hlinený, and Jan Jedelský. Twin-width and transductions of proper k -mixed-thin graphs. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2022. doi:10.1007/978-3-031-15914-5_4.
- 3 Michael J. Bannister, Sergio Cabello, and David Eppstein. Parameterized complexity of 1-planarity. *J. Graph Algorithms Appl.*, 22(1):23–49, 2018. doi:10.7155/jgaa.00457.
- 4 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is np-complete. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.18.
- 5 Benjamin Bergougnoux, Eduard Eiben, Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Towards a polynomial kernel for directed feedback vertex set. *Algorithmica*, 83(5):1201–1221, 2021. doi:10.1007/s00453-020-00777-5.
- 6 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 7 Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995. doi:10.1006/jagm.1995.1009.
- 8 Hans L. Bodlaender, Lars Jaffke, and Jan Arne Telle. Typical sequences revisited - computing width parameters of graphs. *Theory Comput. Syst.*, 67(1):52–88, 2023. doi:10.1007/s00224-021-10030-3.
- 9 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. *SIAM J. Discret. Math.*, 27(4):2108–2142, 2013. doi:10.1137/120903518.
- 10 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1977–1996. SIAM, 2021. doi:10.1137/1.9781611976465.118.
- 11 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPICs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.35.
- 12 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Torunczyk. Twin-width IV: ordered graphs and matrices. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 924–937. ACM, 2022. doi:10.1145/3519935.3520037.
- 13 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé. Twin-width V: linear minors, modular counting, and matrix multiplication. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPICs*, pages 15:1–15:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.STACS.2023.15.

- 14 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1036–1056. SIAM, 2022. doi:10.1137/1.9781611977073.45.
- 15 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *Algorithmica*, 84(11):3300–3337, 2022. doi:10.1007/s00453-022-00965-5.
- 16 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.
- 17 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 18 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 19 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 20 Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. A unifying framework for characterizing and computing width measures. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICs*, pages 63:1–63:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ITCS.2022.63.
- 21 David Eppstein. The widths of strict outerconfluent graphs. *CoRR*, abs/2308.03967, 2023. arXiv:2308.03967.
- 22 Johannes Klaus Fichte, Robert Ganian, Markus Hecher, Friedrich Slivovsky, and Sebastian Ordyniak. Structure-aware lower bounds and broadening the horizon of tractability for QBF. In *LICS*, pages 1–14, 2023. doi:10.1109/LICS56636.2023.10175675.
- 23 Fedor V. Fomin and Tuukka Korhonen. Fast fpt-approximation of branchwidth. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 886–899. ACM, 2022. doi:10.1145/3519935.3519996.
- 24 Robert Ganian and Petr Hliněný. On parse trees and myhill-nerode-type tools for handling graphs of bounded rank-width. *Discret. Appl. Math.*, 158(7):851–867, 2010. doi:10.1016/j.dam.2009.10.018.
- 25 Robert Ganian and Viktoriia Korchemna. The complexity of bayesian network learning: Revisiting the superstructure. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 430–442, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/040a99f23e8960763e680041c601acab-Abstract.html>.
- 26 Robert Ganian and Sebastian Ordyniak. The power of cut-based parameters for computing edge-disjoint paths. *Algorithmica*, 83(2):726–752, 2021. doi:10.1007/s00453-020-00772-w.
- 27 Robert Ganian, Filip Pokrývka, André Schidler, Kirill Simonov, and Stefan Szeider. Weighted model counting with twin-width. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPICs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SAT.2022.15.
- 28 Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.
- 29 Yasuaki Kobayashi and Hisao Tamaki. Treedepth parameterized by vertex cover number. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPICs*, pages 18:1–18:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.18.

- 30 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 184–192. IEEE, 2021. doi:10.1109/FOCS52979.2021.00026.
- 31 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.
- 32 Sang-il Oum. Rank-width and vertex-minors. *J. Comb. Theory, Ser. B*, 95(1):79–100, 2005. doi:10.1016/j.jctb.2005.03.003.
- 33 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.
- 34 André Schidler and Stefan Szeider. Computing twin-width with SAT and branch & bound. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 2013–2021. ijcai.org, 2023. doi:10.24963/ijcai.2023/224.
- 35 Johannes Uhlmann and Mathias Weller. Two-layer planarization parameterized by feedback edge set. *Theor. Comput. Sci.*, 494:99–111, 2013. doi:10.1016/j.tcs.2013.01.029.



Faster Graph Algorithms Through DAG Compression

Max Bannach  

European Space Agency, Advanced Concepts Team, Noordwijk, The Netherlands

Florian Andreas Marwitz  

Institute of Information Systems, Universität zu Lübeck, Germany

Till Tantau  

Institute for Theoretical Computer Science, Universität zu Lübeck, Germany

Abstract

The runtime of graph algorithms such as depth-first search or Dijkstra’s algorithm is dominated by the fact that all edges of the graph need to be processed at least once, leading to prohibitive runtimes for large, dense graphs. We introduce a simple data structure for storing graphs (and more general structures) in a compressed manner using directed acyclic graphs (DAGs). We then show that numerous standard graph problems can be solved in time linear in the size of the DAG compression of a graph, rather than in the number of edges of the graph. Crucially, many dense graphs, including but not limited to graphs of bounded twinwidth, have a DAG compression of size linear in the number of *vertices* rather than *edges*. This insight allows us to improve the previous best results for the runtime of standard algorithms from quasi-linear to linear for the large class of graphs of bounded twinwidth, which includes all cographs, graphs of bounded treewidth, or graphs of bounded cliquewidth.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Data structures design and analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases graph compression, graph traversal, twinwidth, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.8

Funding *Florian Andreas Marwitz*: The research for this paper was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy – EXC 2176 “Understanding Written Artefacts: Material, Interaction and Transmission in Manuscript Cultures”, project no. 390893796. The research was conducted within the scope of the Centre for the Study of Manuscript Cultures (CSMC) at Universität Hamburg.

1 Introduction

Graph traversal or graph searching is a fundamental subroutine in algorithmic graph theory. Given a directed graph (digraph) and a source vertex, the task is to explore the graph following a predefined strategy. Two famous incarnations of such algorithms are *depth-first search* (DFS) and *breadth-first search* (BFS), which, as the names suggest, explore the graph by following long paths first or by unraveling the graph layer by layer. Both algorithms have a broad range of applications, including the computation of connected components or a topological ordering of the input, identifying separators, testing whether the input is planar, finding shortest paths, computing maximum flows, and many more [10]. They are also central in more applied fields and, for instance, are a crucial building block in garbage collection [8], artificial intelligence [15, 20], and web crawling [7, 9]. It is well-known that both algorithms can be implemented in time $O(m)$, where m is the number of edges of the input graph [12, Chapter 5.5]. In particular, for the class of *sparse* graphs, where $m = O(n)$, both algorithms



© Max Bannach, Florian Andreas Marwitz, and Till Tantau;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

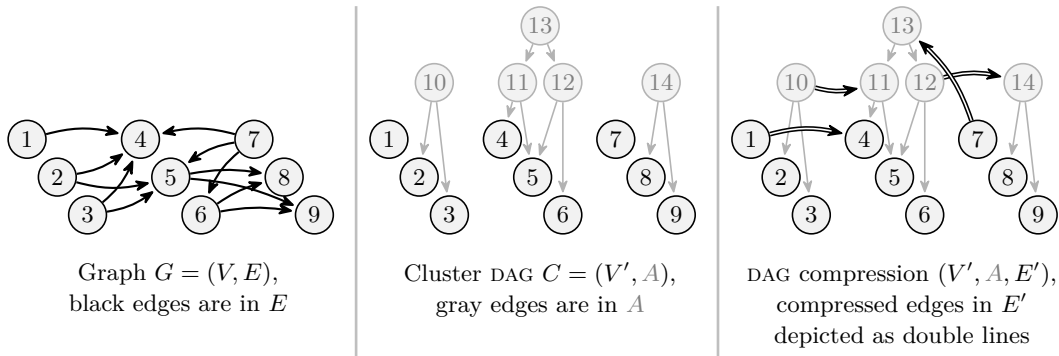
Article No. 8; pp. 8:1–8:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany





■ **Figure 1** Example of how a DAG compression works: For the graph G with vertex set $V = \{1, \dots, 9\}$, one possible cluster DAG is the shown C with vertex set $V' = \{1, \dots, 14\}$. Note that the sinks of C are exactly the vertices in V . The cluster edges of C , shown as straight gray lines, determine subsets of V (“clusters”) via reachability: The cluster $C(10)$ of vertex 10 is the set $\{2, 3\}$; and $C(11) = \{4, 5\}$, $C(12) = \{5, 6\}$, $C(13) = \{4, 5, 6\}$, $C(14) = \{8, 9\}$ and note that, for instance, $C(1) = \{1\}$. Pairing C with a relation $E' \subseteq V' \times V'$, shown as double lines, yields a DAG compression of G : Each edge $(u, v) \in E'$ adds $C(u) \times C(v)$ to E . For instance, the edge $(12, 14) \in E'$ implies that in E there is a complete bipartite graph with shores $C(12) = \{5, 6\}$ and $C(14) = \{8, 9\}$.

run in time linear in the number of vertices. Many (but by far not all) natural graph classes are sparse, including planar graphs, d -degenerate graphs, series-parallel graphs, or graphs of bounded treewidth [17]. In contrast, for very *dense* graphs, where $m = \Omega(n^2)$, until recently, only relatively trivial examples (such as cliques) were known for which these problems could be solved in time $o(n^2)$.

In this paper, we propose a simple data structure, dubbed *DAG compression*, that will prove useful in computing a BFS or DFS on dense graphs $G = (V, E)$. The idea is to represent complete bipartite subgraphs of G by storing *compressed edges*, which are just pairs of vertices of a DAG. Formally, a *cluster DAG* for G is a directed acyclic graph $C = (V', A)$ whose *sinks* are *exactly the vertices in V* and each vertex $v' \in V'$ represents a *cluster* $C(v')$, which is *the set of sinks reachable from v' in C* . A *compressed edge* is a pair $(u', v') \in V' \times V'$ that encodes that there are edges in G from each vertex in $C(u')$ to each vertex in $C(v')$; and to encode all edges in G , we use a *compressed (edge) relation* $E' \subseteq V' \times V'$ such that $E = C(E') := \bigcup_{(u', v') \in E'} C(u') \times C(v')$, see Figure 1 for an example. In case G contains multiple edge relations E_1, E_2, \dots, E_k (like red edges and blue edges), we compress each E_i separately using a compressed relation E'_i (but using the same cluster DAG).

Crucially, we will show that BFS and DFS can be implemented in a way such that their time complexity is linear in the *total number* $|A| + |E'|$ of edges in the DAG compression (called the *size* of the DAG compression in the following) and *no longer necessarily linear in the number* $|E|$ of edges of the original graph. Thus, whenever we can find a DAG compression of a graph whose size is linearly bounded by the number n of vertices in the original graph, we can lower the runtime of BFS and DFS from $O(m)$ to $O(n)$.

A powerful motivation for studying DAG compressions comes from its relation to the prominent *class of graphs of bounded twinwidth*. Twinwidth is a structural graph parameter introduced in 2020 by Bonnet et al. [5] to measure the distance of a graph from being a cograph (detailed definitions will be given later). The importance of this parameter lies in the fact that for many graph classes commonly studied in the literature this parameter is bounded (so the class of graphs of bounded twinwidth is large), but the model checking problem for first-order logic on structures of bounded twinwidth is still fixed-parameter

tractable (so many problems are still in FPT on this class). However, graphs of bounded twinwidth are not only interesting in the context of powerful logical characterizations and algorithmic meta-theorems: It was recently shown [4] that in unweighted graphs of bounded twinwidth, the *single-source shortest path* problem (SSSP) can be solved in time $O(n \log n)$, despite the fact that such graphs can easily have $\Omega(n^2)$ edges. Consequently, the *diameter* of a graph of bounded twinwidth, i.e., the maximum length of a shortest path, can be computed in time $O(n^2 \log n)$, while it is also known [4] that it *cannot* be computed in time $O(n^{2-\epsilon})$ unless the strong exponential time hypothesis fails. Hence, there is a $\log(n)$ -gap between the known lower and upper bounds for determining the diameter of graphs of bounded twinwidth.

One of our main results will be that *graphs of bounded twinwidth admit a linear-size DAG compression*. Combining this with our BFS implementation that runs in time linear in the size of the DAG compression, we see that on graphs of bounded twinwidth, one can actually solve SSSP in time $O(n)$ and, thus, can solve the diameter problem in time $O(n^2)$. In particular, we close the gaps in the runtime left open by previous work.

However, one has to be careful regarding the exact claims of the just-mentioned results: All known algorithms working on graphs of twinwidth d , including our algorithm for computing a linear-size DAG compression of a graph of bounded twinwidth, need access to a so-called *contraction sequence*. Such a sequence is a linear-size witness that a graph has twinwidth d and, indeed, they can be used to show that deciding whether a graph has twinwidth d lies in NP (and it is known [3] that at least for $d = 4$ the problem is NP-complete). For instance, the algorithm from [4] for the diameter problem needs access to a contraction sequence witnessing a twinwidth of d , in order to run in time $O(d \cdot n^2 \log n)$; but the lower bound of $O(n^{2-\epsilon})$ also still holds when a contraction sequence is given.

Our Contributions. Our first main contribution is conceptual: We propose the already mentioned *DAG compression* data structure and study their basic properties. The *size* of these compressions (the number $|A|$ of cluster edges plus the number $|E'|$ of compressed edges) will be of particular interest since we will show that important problems can be solved in a time that is linear with respect to this size.

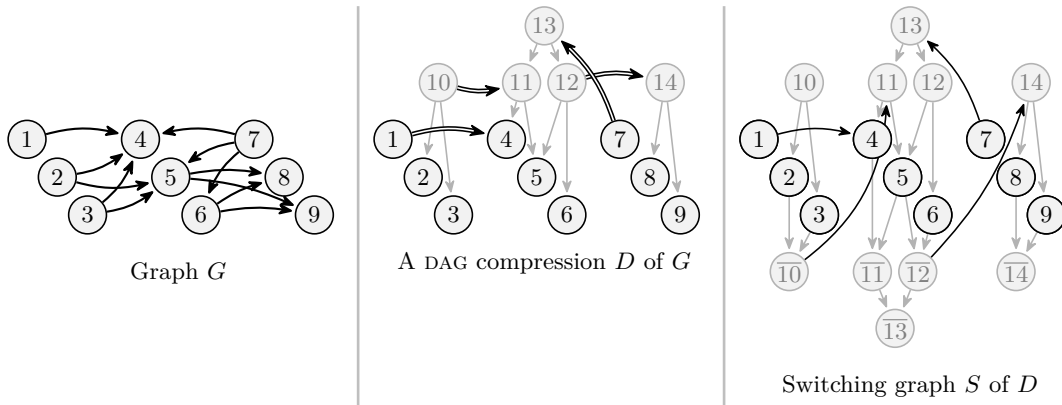
The central tool underlying our algorithms is a construction that uses a DAG compression $D = (V', A, E')$ of a graph $G = (V, E)$ to build an edge-weighted graph $S = (V'', E'', w'')$ with $V \subseteq V''$ and $w'' : E'' \rightarrow \{0, 1\}$, called the *switching graph* of D (since paths in this graph repeatedly switch between two parts of it, see Figure 2 for an example).

► **Theorem 1.1.** *Let $G = (V, E)$ be a directed graph, let $D = (V', A, E')$ be a DAG compression of G , and let $S = (V'', E'', w'')$ be the switching graph of D . Then for every pair $(u, v) \in V \times V$ we have $d_G(u, v) = d_S(u, v)$, that is, the distance from u to v is the same in G and in S .*

Since it will be immediate from the construction that the number $|E''|$ of edges in the switching graph is $2|A| + |E'|$ and thus at most double the size of the DAG compression D , we easily get fast DFS and BFS algorithms for graphs that admit a DAG compression of linear size:

► **Theorem 1.2.** *On input of a DAG compression $D = (V', A, E')$ of a graph $G = (V, E)$ we can visit the vertices in V both in BFS and DFS order in time $O(|V'| + |A| + |E'|)$.*

As graphs with bounded twinwidth have a linearly bounded dag compression, a direct consequence of Theorem 1.2 is that we close the gap between the lower and upper bound for computing the diameter of graphs of bounded twinwidth:



■ **Figure 2** The example graph $G = (V, E)$, DAG compression $D = (V', A, E')$, and cluster DAG $C = (V', A)$ from Figure 1. The switching graph $S = (V'', E'', w'')$ results from first taking the disjoint union of C and the copy \bar{C} , where all edges are reversed, and unifying the vertices in V . This results in $2|A|$ many edges (shown in gray in S above) and we set their weight to 0. In addition, for each edge $(u', v') \in E'$ there is a *switching edge* (\bar{u}', v') in E'' , shown in black, that leads from the lower part to the upper part and has weight 1. A path in G of length 2, like the path $3 \rightarrow 5 \rightarrow 9$, corresponds to a path $3 \rightarrow \bar{10} \rightarrow 11 \rightarrow 5 \rightarrow \bar{12} \rightarrow 14 \rightarrow 9$ in S of weight 2 as it contains two switching edges (black edges of weight 1).

► **Corollary 1.3.** *On input of a contraction sequences that witnesses that a graph G has twinwidth at most d , we can compute the diameter of G in time $O(d \cdot n^2)$.*

Access to a fast depth-first search allows us to implement other operations from algorithmic graph theory in time $O(n)$. For instance, the strongly connected components of a digraph can be computed by two consecutive depth-first searches using Kosaraju’s algorithm [22]:

► **Corollary 1.4.** *On input of a DAG compression $D = (V', A, E')$ of a graph $G = (V, E)$, the strongly connected components of G can be computed in time $O(|V'| + |A| + |E'|)$.*

Another traditional application of the depth-first search is the detection of cycles in directed graphs as well as the computation of a topological sort of the input [23]:

► **Corollary 1.5.** *On input of a DAG compression $D = (V', A, E')$ of a graph $G = (V, E)$, we can test in time $O(|V'| + |A| + |E'|)$ whether G contains a cycle and, if not, compute a topological sorting of G .*

It is well-known that a BFS can compute the shortest path between two vertices in *unweighted* graphs. However, this is different in *weighted* graphs, in which a more refined algorithm must be used. Generalizing the DAG compression to weighted graphs, we obtain:

► **Theorem 1.6.** *On input of a DAG compression $D = (V', A, E', w')$ of a weighted graph $G = (V, E, w)$ with $w: E \rightarrow \mathbb{N}$, the single-source shortest path (SSSP) problem can be solved in time $O((|V'| + |A| + |E'|) \log(|V'| + |A| + |E'|))$.*

Related Work. Many graph compression methods are known in the literature; the one most similar to ours is by Toivonen et al. [26]. They also introduce supernodes and superedges with the idea that an edge between two supernodes represents all edges between vertices within these supernodes. However, they partition the vertex set into a set of supernodes, whereas our compression allows for nested vertex combinations. The representation of Navlakha

et al. is similar to the one of Toivonen et al. with an additional set of edge corrections, i.e., edges that must be deleted or added to retrieve the original graph [18]. Tian et al. provide two operations: One for creating a graph compression based on user-given attributes and another to further control the compression [25]. Zhang et al. further refine this compression to include numerical attributes and add more automation [27].

Using distance-equivalent graphs to speed up routing algorithms is commonly done in theory and practice [24]. However, the objective is usually to replace a large input graph with a smaller graph in which distances are approximately the same as in the input. In contrast, our use of distance-equivalent graphs does not involve any approximations: The distances in the switching graphs are precisely as in the original graph.

Twinwidth was introduced in 2020 for graphs and digraphs by Bonnet et al. [5] and interest quickly increased as witnessed by dozens of new research papers each year since then. One of the earliest and most remarkable results is an fpt-algorithm for the model-checking problem of first-order logic [5]. Graphs of twinwidth 0 and 1 can be recognized in polynomial time [28, 14], but deciding whether a graph has twinwidth at most 4 is NP-complete [3]. Besides the aforementioned meta-theorem, dedicated dynamic programs are known to compute maximum cliques, independent sets, and minimal dominating sets on graphs of bounded twinwidth [4]. It is also known that all triangles of a graph of twinwidth at most d can be counted in time $O(d^2n + m)$ if a corresponding contraction sequence is given [16]. Ahn et al. [2] study the twinwidth of random graphs. Schidler and Szeider provide the first practical strategies to compute contraction sequences using a SAT-solver [21] and Ganian et al. show that weighted model counting can be done efficiently on formulas of small twinwidth [13]. Bonnet et al. introduced twin-models [6], which can also compress graphs and is similar to our result in Theorem 4.4. However, one main thrust of defining DAG compressions is their usefulness independently of twinwidth, which is also one of the reasons we consider DAG compressions rather than tree compressions.

Structure of this Paper. We define DAG compressions and have a look at some basic properties and operations in the next section. In Section 3 on algorithms, we define the switching graph and show how it can be used to implement fast versions of BFS and DFS, and related algorithms. In Section 4, we show how we can build linear-size DAG compressions. Our particular focus will be on graphs of bounded twinwidth, where we turn a given contraction sequence into a DAG compression of linear size.

2 DAG Compressions: Definition, Examples, and Basic Constructions

The idea behind DAG compressions is – as already pointed out in the introduction – to compress complete bipartite subgraphs of a given graph by single “compressed edges” that link vertices of the cluster DAG. The cluster DAG has the job of encoding sets of vertices via the reachability relation: Each vertex of the cluster graph encodes all sinks that are reachable from it. In the following, we formalize these ideas and give examples. We also show how basic update and construction operations on DAG compressions can be implemented.

Basic Terminology. Before we proceed, let us fix some terminology and notation: To simplify the presentation, a *graph* is always a pair (V, E) consisting of a non-empty finite set V of vertices together with a relation $E \subseteq V \times V$. In other words, by “graph” we always refer to a simple, non-empty, directed graph; undirected graphs are just directed graphs with a symmetric edge relation. Throughout this paper, n will refer to the size $|V|$ of the graph G currently under consideration and m will refer to $|E|$.

An (*edge-*)*weighted graph* is a triple (V, E, w) , where $w: E \rightarrow \mathbb{N}$ maps edges to nonnegative integers. The weights are *binary* if $w(e) \in \{0, 1\}$ holds for all $e \in E$. An unweighted graph can also be seen as a weighted graph in which all weights are 1. A *walk of length l* in a graph G is a sequence (v_0, \dots, v_l) of vertices such that $(v_{i-1}, v_i) \in E$ holds for all $i \in \{1, \dots, l\}$. For $s = v_0$ and $t = v_l$, the walk is also called an *s - t -walk* and we say that *t is reachable from s* . The *weight* of a walk is the sum $\sum_{i=1}^l w(v_{i-1}, v_i)$. Note that for unweighted graphs the length and the weight of a walk are the same.

A walk is called a *path* if all vertices are distinct. A walk is called a *cycle* if $l \geq 3$, $v_0 = v_l$, and (v_0, \dots, v_{l-1}) is a path. The *distance function for G* is the function $d_G: V \times V \rightarrow \mathbb{N} \cup \{\infty\}$ that maps each pair (u, v) of vertices to the minimum weight of any u - v -walk in G (or to ∞ , if no such walk exists).

A graph is a *directed acyclic graph* (a *DAG*) if there is no walk in G of length at least 1 with $v_0 = v_l$. A *sink* in a DAG is a vertex $s \in V$ of out-degree 0, that is, without edges leaving s . Note that a DAG must always have at least one sink.

2.1 Definition of DAG Compressions and Examples

In order to formalize the notion of DAG compressions, we start with cluster DAGs:

► **Definition 2.1** (Cluster DAGs and Compressed Edges). *A cluster DAG for a set V is a DAG $C = (V', A)$ such that V is exactly the set of sinks of C . Given a vertex $v' \in V'$, the cluster $C(v')$ of v' is the subset of V of all sinks that are reachable from v' in C . A pair $(u, v) \in V' \times V'$, not necessarily an element of A , is called a compressed edge.*

► **Definition 2.2** (DAG Compression). *Let $G = (V, E)$ be a graph. Let $C = (V', A)$ be a cluster DAG for V . A DAG compression of G is a triple $D = (V', A, E')$, where $E' \subseteq V' \times V'$ is a compressed (edge) relation, such that $E = \bigcup_{(u', v') \in E'} C(u') \times C(v')$. The size of D is the number $|A| + |E'|$.*

We already gave an example of a DAG compression of a graph in Figure 1. In the following we consider three more examples in order to explain the concept.

► **Example 2.3** (No Compression). A trivial way of compressing any graph $G = (V, E)$ is to do no compression at all, that is, to use (V', A, E') with $V' = V$, $A = \emptyset$, and $E' = E$. Note that, indeed, if there are no edges in the cluster DAG, each vertex is a sink.

This trivial example shows that we can always come up with a DAG compression of size m for any graph G . In particular, for any class \mathcal{C} of graphs that has only a linear number of edges (that is, for which there is a constant c such that for all $(V, E) \in \mathcal{C}$ we have $|E| \leq c \cdot |V|$), all graphs in \mathcal{C} admit linear-size DAG compressions. A prominent example of such classes are classes of graphs of bounded treewidth.

A slightly more interesting example are complete graphs, which have a superlinear number of edges, but a linear-size DAG compression:

► **Example 2.4** (Cliques). Let $C_n := (V, E)$ with $E = V \times V$ be the complete graph on n vertices. Note that $m = |E| = n^2$. A linear-size ($n + 1$ to be precise) DAG compression for it is (V', A, E') with $V' = V \cup \{c\}$, where c is a fresh vertex, $A = \{(c, v) \mid v \in V\}$ contains an edge from c to every vertex of V , making all of them sinks, and $E' = \{(c, c)\}$ contains a single loop. Indeed, we then have $E = \bigcup_{(u', v') \in E'} C(u') \times C(v') = C(c) \times C(c) = V \times V$.

A more involved and interesting example are cographs, which are the natural “base class” to define twinwidth (which we will discuss in more detail later):

► **Example 2.5 (Cographs).** The class of *cographs* is defined inductively as follows: First, any single vertex is a cograph. Second, if G and H are cographs, so are their disjoint union and also their disjoint union with all edges between vertices in G and vertices in H added. This inductive definition can be used to obtain a linear-size ($5n - 4$ to be precise) DAG compression of any cograph: Compressing single vertex graphs is trivial, so let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be cographs and let D_G and D_H be DAG compressions of sizes $5n_G - 4$ and $5n_H - 4$, respectively. We will ensure (and assume) that the cluster DAGs C_G and C_H are actually trees with roots r_G and r_H .

A DAG compression of the disjoint union is then obtained by taking the disjoint union of the two compressions, adding a new root r and adding the edges (r, r_G) and (r, r_H) , that is, by considering $(V'_G \cup V'_H \cup \{r\}, A_G \cup A_H \cup \{(r, r_G), (r, r_H)\}, E'_G \cup E'_H)$ and always assuming that all vertex names are distinct. Note that the resulting size is $5n_G - 4 + 5n_H - 4 + 2 = 5n - 6 \leq 5n - 4$.

The interesting case is to obtain a DAG compression of the disjoint union with all edges between G and H added. However, this is easy to achieve by taking the same construction and just adding the edges (r_G, r_H) and (r_H, r_G) to E' as this will cause $C(r_G) \times C(r_H) = V_G \times V_H$ and also the reversed edges $C(r_H) \times C(r_G) = V_H \times V_G$ to be added to E , exactly as needed. This adds two more edges to the size of the DAG compression, meaning that the size is $5n_G - 4 + 5n_H - 4 + 2 + 2 = 5n - 4$ as claimed.

Compressing Weighted Graphs and Arbitrary Structures. It is straightforward to extend the definition of a DAG compression to weighted graphs: Simply add a weight function $w': E' \rightarrow \mathbb{N}$ that assigns weights to compressed edges. The obvious semantics is then that for $(u', v') \in E'$ all edges in $C(u') \times C(v')$ should have weight $w'(u', v')$. However, we then run into the problem that different weights may now be assigned to the same edge (u, v) , namely when $(u, v) \in C(u'_1) \times C(v'_1)$ and also $(u, v) \in C(u'_2) \times C(v'_2)$ for some compressed edges (u'_1, v'_1) and (u'_2, v'_2) with $w'(u'_1, v'_1) \neq w'(u'_2, v'_2)$. We resolve this case by assigning the *minimum* weight to (u, v) of the weights of all compressed edge that uncompress to (u, v) . Formally, we require that for a weighted graph (V, E, w) a weighted DAG compression is a tuple (V', A, E', w') such that (V', A, E') is a DAG compression of (V, E) and for all $e \in E$ we have $w(e) = \min_{(u', v') \in E', e \in C(u') \times C(v')} w'((u', v'))$.

As mentioned in the introduction, it is straightforward to define DAG compressions of graphs with multiple edge relations E_i by using multiple compression relations E'_i (but still using a single cluster DAG). It makes also sense to use DAGs to compress not only binary edge relations, but also unary relations (subsets of V , that is, colors): We can compress a color $X \subseteq V$ using a set $X' \subseteq V'$ such that $X = \bigcup_{x' \in X'} C(x')$, that is, by representing X as the union of some clusters described by the cluster graph. In the other direction, it is also possible to compress *ternary* relations $R \subseteq V \times V \times V$ using a relation $R' \subseteq V' \times V' \times V'$ such that $R = \bigcup_{(u', v', w') \in R'} C(u') \times C(v') \times C(w')$; and note that this potentially allows one to compress relations with $|R| = O(n^3)$ using DAG compressions of size $O(n)$. All told, DAG compressions can be used to compress arbitrary logical structures as well, but for simplicity, we restrict our attention to (weighted) graphs in the following.

Cluster Trees Versus Cluster DAGs. In all of the above examples, the cluster DAG was actually a tree. The following is an important example of a graph for which we appear to need a DAG to compress it to linear size (we believe that one can prove that a linear-size compression using trees is not possible, but are not aware of any simple proof for this claim):

► **Example 2.6 (Rook Graph).** The *rook graph on n vertices*, where $n = s^2$ is the square of some integer $s = \sqrt{n}$, is a graph G with $V = \{1, \dots, s\}^2$ and with $((i, j), (k, l)) \in E$ iff $i = k$ or $j = l$, that is, if a rook could be moved from position (i, j) to position (k, l) in a chess

game in a single move. Another way of viewing a rook graph is as an intertwined union of cliques: Every row is a clique and every column is a clique, but there are not other edges. Note that the rook graph has $(2s)s^2 = \Theta(n^{3/2})$ edges.

We can easily construct a linear-size DAG compression for the rook graph: Consider (V', A, E') with $V' = V \cup \{r_1, \dots, r_s\} \cup \{c_1, \dots, c_s\}$, so we add a *row vertex* r_i for each row and similarly a *column vertex* c_i for each column; we set $A = \{(r_i, (i, j)) \mid i, j \in \{1, \dots, s\}\} \cup \{(c_j, (i, j)) \mid i, j \in \{1, \dots, s\}\}$, that is, each row vertex and each column vertex is directly connected to all vertices of their row or column, respectively; and we set $E' = \{(c_i, c_i) \mid i \in \{1, \dots, s\}\} \cup \{(r_i, r_i) \mid i \in \{1, \dots, s\}\}$, that is, we add self-loops at all row and column vertices, resulting in cliques in E for each row and each column – exactly, what we are looking for. The total size of the described DAG compression is $|A| + |E'| = 2s^2 + 2s = 2n + 2\sqrt{n} = O(n)$.

There is a deeper reason why we can compress the rook graph so well using DAGs rather than trees: Using DAGs in compressions allows us to *implement the union operation on edge sets by uniting the DAG compressions*. The formal statement is the following:

► **Lemma 2.7.** *Let $G = (V, E_1 \cup E_2)$ and let $D_1 = (V'_1, A_1, E'_1)$ and $D_2 = (V'_2, A_2, E'_2)$ be DAG compressions of (V, E_1) and (V, E_2) , respectively, that use distinct vertex sets for non-sink vertices, that is, $V'_1 \cap V'_2 \subseteq V$. Then $(V'_1 \cup V'_2, A_1 \cup A_2, E'_1 \cup E'_2)$ is a DAG compression of G whose size is at most the sum of the sizes of D_1 and D_2 .*

Proof. Since $V'_1 \cap V'_2 \subseteq V$, the edges of the cluster DAGs A_1 and A_2 do not “interfere,” that is, in the new compression DAG for any $v' \in V'_1$ the set $C(v')$ with respect to reachability in $A_1 \cup A_2$ is the same as $C(v')$ with respect to just A_1 ; and symmetrically for $v' \in V'_2$. This implies that for any compressed edge $(u', v') \in E'_1 \cup E'_2$, the set $C(u') \times C(v')$ is the same as before. In particular, the union of all of these sets is exactly $E_1 \cup E_2$. The claim concerning the sizes follows directly from the construction. ◀

By the lemma, the rook graph can be compressed simply because it is the union of $2\sqrt{n}$ cliques, each having \sqrt{n} vertices and, hence, allowing a DAG compression of size $1 + \sqrt{n}$ by Example 2.4; so the lemma tells us that a size of $2\sqrt{n}(1 + \sqrt{n}) = O(n)$ suffices for the union of all these cliques – no matter how they are intertwined.

2.2 Updating DAG Compressions

When defining a new data structure, a natural question is how difficult it is to update it. That is, suppose we have already constructed a DAG compression D of a graph G , with D being stored in memory while G is not stored directly, and we now wish to modify G by adding or deleting edges or vertices. How difficult is it to update D instead (without decompressing it)? In other words, given D , we wish to compute a DAG compression \tilde{D} of \tilde{G} , where \tilde{G} results from G by some small change.

Let us start with simple modifications that are easy to implement. First, we may wish to add an edge, meaning that $\tilde{G} = (V, E \cup \{(u, v)\})$. It is then fairly simple to compute \tilde{D} in this case: We can add the edge as a compressed edge, that is, let $\tilde{E}' = E' \cup \{(u, v)\}$. Note that the size increases only by one. Second, we may wish to add a new vertex. This turns out to be even simpler: Just add it to V' , where it will become an isolated sink. This does not even change the size of the compression. Third, we may wish to delete an existing vertex v from V along with all adjacent edges in E . This is also simple to achieve: Simply delete v from V' and all its occurrences in A and in E' .

One operation is suspiciously missing: Deleting an edge (u, v) from E . It turns out that this can be a difficult operation to implement: If $(u, v) \in C(u') \times C(v')$ for several compressed edges $(u', v') \in E'$, we need to “break up” these compressed edges, meaning that we need to remove the compressed edge (u', v') and to then add new compressed edges that cover exactly the set $(C(u') \times C(v')) \setminus \{(u', v')\}$. It is currently unclear to us what the exact complexity of this operation is.

Another suspiciously missing aspect is the question of what happens when we have multiple edge additions in a row. Clearly, it is not optimal to simply add each edge as a compressed edge that only compresses itself: If we add all edges of, say, a cluster $C(v')$ and thereby making it a clique, we would like to end up with a DAG compression in which there is a single compressed edge (v', v') in E' to represent this clique. Undoubtedly, greedy heuristics exist for locally compressing sets of newly inserted edges, but finding a minimum-size DAG compression (V', A, E') for a given graph (V, E) appears to be a difficult problem. The following theorem shows that minimizing $|E'|$ is NP-complete and we conjecture that minimizing $|A| + |E'|$ (which is the more important question from a practical point of view) is also NP-complete:

► **Theorem 2.8.** *It is NP-complete to decide on input $G = (V, E)$ and a number k whether there is a DAG compression (V', A, E') of G with $|E'| \leq k$.*

Proof. Reduce from the NP-complete problem of covering a bipartite graph with at most k complete bipartite graphs [19]. By definition, a DAG compression (V', A, E') of G with $|E'| \leq k$ immediately yields a cover of E by at most k complete bipartite graphs; and given such a cover of size k , we can easily construct a cluster DAG such that for each complete bipartite graph $X \times Y$ in this cover there are vertices x' and y' with $C(x') = X$ and $C(y') = Y$, allowing us to put the edge (x', y') into E' . ◀

3 DAG Compression: Algorithms

Given a DAG compression D of some graph G , we wish to solve typical algorithmic problems on G , for instance, we would like to compute a topological ordering of G . The objective is, of course, to do so *without* “decompressing” the graph, that is, without storing the large graph G in memory. Rather, we would like to directly work on D and would like to have linear or quasi-linear runtimes in terms of the size of D .

At first sight, DAG compressions seem rather ill-suited for this purpose: Even deciding whether there is an edge between two given vertices $u, v \in V$ is not straightforward. Indeed, to answer this simple question using only $D = (V', A, E')$, we have to determine whether there is a compressed edge $(u', v') \in E'$ such that u is reachable from u' in A and v is reachable from v' in A . If A is a complex graph containing long paths, this is a nontrivial problem. Indeed, even very simple problems like determining the degree of a vertex are difficult if only D is given, as we may need to consider all vertices $v' \in V$ from which v is reachable – and this set may have linear size.

Nevertheless, it turns out that many problems involving *the whole graph* G can be solved in linear-time with respect to D . The core idea behind these algorithms is the construction of the *switching graph*, whose core property is that it is distance-equivalent to G .

Distance Equivalence and the Switching Graph. In order to solve BFS in G using only D , we first construct a new graph S that is *distance equivalent* to G , but has few edges.

► **Definition 3.1** (Distance Equivalence). Let $G_1 = (V_1, E_1, w_1)$ and $G_2 = (V_2, E_2, w_2)$ be two weighted graphs. They are distance equivalent (on $V_1 \cap V_2$) if for all $u, v \in V_1 \cap V_2$ we have $d_{G_1}(u, v) = d_{G_2}(u, v)$.

The key observation, to be formalized later, is that computing, say, a BFS ordering of the vertices in G_1 will also yield a BFS ordering of the vertices in V_2 in G_2 because the ordering in which vertices need to be visited depends on the distances.

Let us now define the switching graph of a DAG compression D and prove that it is distance equivalent to the uncompressed graph G .

► **Definition 3.2** (Switching Graph). Let $D = (V', A, E', w')$ be a DAG compression of a weighted graph $G = (V, E, w)$. For each $v' \in V'$ let \bar{v}' be a new vertex, except when $v' \in V$, in which case $\bar{v}' = v'$. The switching graph S of a DAG compression $D = (V', A, E', w')$ of a weighted graph $G = (V, E, w)$ is a weighted graph $S = (V'', E'', w'')$ such that

1. the vertex set V'' is the union of the three sets
 - upper part $V_{\text{upper}} = \{v' \mid v' \in V' \setminus V\}$,
 - middle part $V_{\text{middle}} = \{v \mid v \in V\} = \{\bar{v} \mid v \in V\}$, and
 - lower part $V_{\text{lower}} = \{\bar{v}' \mid v' \in V' \setminus V\}$,
2. the edge set E'' is the union of the three sets
 - upper cluster edges $\{(u', v') \mid (u', v') \in A\}$ in the upper part,
 - lower cluster edge $\{(\bar{v}', \bar{u}') \mid (u', v') \in A\}$ in the lower part, and
 - switching edges $\{(\bar{u}', v') \mid (u', v') \in E'\}$,
3. and the weight function $w'' : E'' \rightarrow \mathbb{N}$ with $w''((u', v')) = w''((\bar{u}', \bar{v}')) = 0$ for the cluster edges resulting from $(u', v') \in A$ and with $w''((\bar{u}', v')) = w'((u', v'))$ for the switching edges resulting from $(u', v') \in E'$.

An example of a switching graph is depicted in Figure 2, where the weights in G are all 1 and, hence, the weights in S are either 0 (for cluster edges, depicted in gray) or 1 (for switching edges, shown in black). For further reference, we note a trivial observation:

► **Lemma 3.3.** For every switching graph we have $|V''| \leq 2|V'|$ and $|E''| = 2|A| + |E'|$.

Let us now prove the main property of switching graphs:

► **Theorem 3.4.** Let S be the switching graph of a DAG compression D of G . Then S and G are distance equivalent.

Proof. Let u and v be a pair of vertices in V .

First, consider a minimum-weight u - v -walk (v_0, \dots, v_l) in G and let k be its weight. We will construct a u - v -walk in S of the same weight, starting at $v_0 = u$ and extending it for each $i \in \{1, \dots, l\}$ each time to v_i . For a given i , we must have $(v_{i-1}, v_i) \in E$ as we have a walk in G . Since D is a DAG compression of G , there must exist $(v'_i, v'_{i+1}) \in E'$ such that $v_{i-1} \in C(v'_{i-1})$ and $v_i \in C(v'_i)$ and $w((v_{i-1}, v_i)) = w'((v'_{i-1}, v'_i))$. Extend the new walk as follows: From $v_{i-1} = \bar{v}_{i-1}$ use (reversed) cluster edges to reach \bar{v}'_{i-1} in the lower part (which must exist since $v_{i-1} \in C(v'_{i-1})$ means that v_{i-1} is reachable from v'_{i-1} using non-reversed cluster edges, so \bar{v}'_{i-1} is reachable from \bar{v}_{i-1} using reversed cluster edges), use the switching edge (\bar{v}'_{i-1}, v'_i) to get to the upper part, and use cluster edges in the upper part to get to v_i . We only use exactly one switching edge during this extension of the new walk, meaning that the weight of the walk increases exactly by the weight of this edge. This immediately yields the claim concerning the total walk weight.

Second, consider a minimum-weight u - v -walk (v_0'', \dots, v_l'') in S and let k be its weight. Since $u = v_0''$ and $v = v_l''$, we start and end in the middle part of V'' . We cut the walk into subwalk P_1, \dots, P_p of minimal lengths (but at least 1) such that each P_i starts and ends with a vertex in the middle part (that is, in V), while all other vertices are in the lower or in the upper part. Each subwalk (except for P_1) begins with the last vertex of the previous subwalk. As an example, the example 3-9-walk $(3, \overline{10}, 11, 5, \overline{12}, 14, 9)$ in Figure 2 would be cut into the subwalks $P_1 = (3, \overline{10}, 11, 5)$ and $P_2 = (5, \overline{12}, 14, 9)$ since V contains only the single digit numbers. Observe that the number of subwalks is exactly the number of positions $j \in \{1, \dots, l\}$ for which $v_j'' \in V$ holds, that is, how often the walk crosses the middle part.

We claim that each P_i contains exactly one switching edge and all other edges are cluster edges. To see this, let $P_i = (p_1, \dots, p_z)$ and observe that only p_1 and p_z lie in the middle part by construction. From p_1 , all non-switching edges point to a vertex in the lower part – and this is true also for all vertices in the lower part. Thus, up to the first switching edge on P_i , all edges are (reversed) lower cluster edges. Then, at some point, a switching edge $(\bar{p}', q') \in E''$ must be used for some $(p', q') \in E'$ since, otherwise, we could not exit the lower part (p_z is not in the lower part – and we also not allowed to just “rest” at p_1 since we must make at least one step as the length of all P_i is at least 1). Note that \bar{p}' is reachable from p_1 , meaning $p_1 \in C(p')$. By construction, the switching edge brings us to the upper part (or to the middle part, but then we stop and are done). In the upper part, we can only follow upper cluster edges until we reach the middle part; but then we stop one more, having reached p_z . This implies that $p_z \in C(q')$.

We see that each P_i consists of cluster edges (having weight 0) and a single switching edge $(\bar{p}', q') \in E''$ of some weight $w''((\bar{p}', q')) = w'((p', q'))$ for $(p', q') \in E'$. Furthermore, $(p_1, p_z) \in E$ must hold since $p_1 \in C(p')$ and $p_z \in C(q')$. All told, for each subwalk P_i starting at some vertex $p \in V$ and ending at a vertex $q \in V$, we see that there is an edge $(p, q) \in E$. Furthermore, the weight of this edge is at most the weight of the switching edge used in P_i . However, it also cannot be smaller than this weight: Otherwise, we could replace the walk by another walk from p to q in S of lesser weight (simple walk from p to the switching edge having this smaller weight and then walk to q). This shows that the minimal weight of a u - v -walk in G is k . ◀

Our first main theorem from the introduction, Theorem 1.1, is just a restatement of the above theorem.

► **Corollary 3.5.** *Given a DAG compression $D = (V', A, E')$ of a graph $G = (V, E)$, we can run a BFS in time $O(|V'| + |A| + |E''|)$.*

Proof. Run Dijkstra’s algorithm [11] on the switching graph S of D , which has size at most $2|V'|$ vertices and $2|A| + |E'|$ edges by Lemma 3.3. Since all weights are 0 or 1, we can run the Dijkstra algorithm in time $O(|V'| + |A| + |E'|)$. ◀

► **Corollary 3.6.** *Given a DAG compression $D = (V', A, E')$ of a graph G , we can run a DFS in time $O(|V'| + |A| + |E''|)$.*

Proof. Modify Dijkstra’s algorithm in Corollary 3.5 to extract the maximum instead of the minimum. ◀

Theorem 1.2 directly follows from Corollaries 3.5 and 3.6. When the graphs we study are weighted, as in Theorem 1.6, we can still run the Dijkstra algorithm, but the runtime is no longer linear in the size of the DAG compression, but of the order $O(s \log s)$, where $s = |V'| + |A| + |E'|$. We obtain Theorem 1.6 from the introduction.

4 DAG Compressions: The Link to Twinwidth

We have shown that several standard algorithms can be implemented in time linear in the size of the DAG compression of a graph, and we also saw examples of graphs such as cographs or the rook graph that allow us to compress graphs with $n^{1+\delta}$ edges for $\delta > 0$ to size $O(n)$. In the following we show that there is a large class of graphs for which a linear-size DAG compression is always possible: Graphs of bounded twinwidth. Linear-sized DAG compressions are tightly linked with bounded twinwidth, however, they do not capture the same class of graphs (as the rook graph shows). We first define twinwidth and afterwards show how we can build the DAG compression from a so-called contraction sequence. Readers familiar with twin-models [6] will note that the tree compressions developed in the following are very closely related to twin-models (the difference is mainly in the terminology). For simplicity, we only consider undirected graphs, that is, graphs with a symmetric edge relation.

Twinwidth and Contraction Sequences. Following Bonnet et al. it will be useful to consider *trigraphs* [4, 5], which are triples (V, B, W, R) such that $R, B,$ and W partition the set of possible (non-loop) edges $V \times V \setminus \{(v, v) \mid v \in V\}$ into *red edges*, *black edges*, and *white edges*. The *red*, *black*, or *white degree* of a vertex is the number of its incoming (or, equivalently, outgoing) red, black, or white edges, respectively.

A *contraction* in a trigraph G merges two arbitrary vertices u and v into a single fresh vertex z , forming a new trigraph G' by removing u and v and coloring for each $x \in V \setminus \{u, v\}$ the edge between x and z (and also the backward edge as the graph is symmetric) as follows: If the edges between x and u and between x and v were both black, so is the x - z -edge; if the edges were both white, so is the x - z -edge; and in all other cases the x - z -edge becomes red.

A *contraction sequence* of a graph $G = (V, E)$ is a sequence $(G_n, G_{n-1}, \dots, G_1)$ of trigraphs G_i such that the first $G_n = (V, E, V \times V \setminus (E \cup \{(v, v) \mid v \in V\}), \emptyset)$ is the trigraph in which the edges in E are black and everything else is white and there are no red edges; the last graph G_1 is just a single vertex; and each G_i results from G_{i+1} through the contraction of two vertices u_{i+1} and v_{i+1} of G_{i+1} into a fresh vertex z_i . The *red degree* of a contraction sequence is the maximum red degree any vertex in any of the G_i has. The sequence is called a *d-contraction sequence* if its maximum red degree is d . Finally, the *twinwidth* $\text{tw}(G)$ of G is the minimal d for which there is a d -contraction sequence of G .

An example of 1-contraction sequence of the cycle C_4 is show in Figure 3.

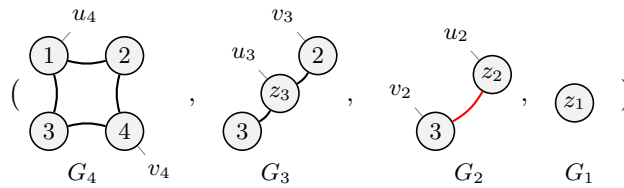


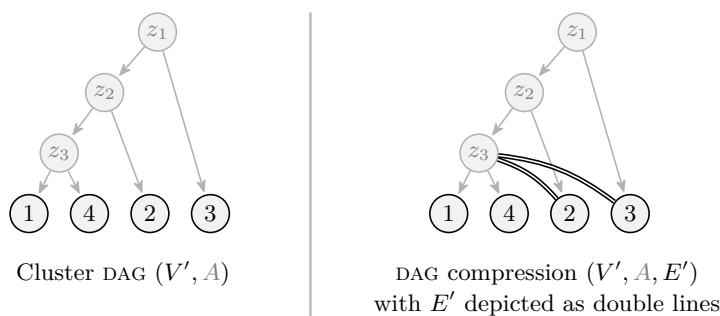
Figure 3 Example of a contraction sequence of the cycle C_4 . In the first step, $u_4 = 1$ and $v_4 = 4$ are contracted to form the new vertex z_3 in G_3 . The edge from z_3 to 2 is black since both the edges from u_4 to 2 and from v_4 to 2 were black (that is, present). Similarly, there is a black edge from z_3 to 3. In contrast, when $v_3 = 2$ and $u_3 = z_3$ are contracted to z_2 in G_2 , the edge from z_2 to 3 is red since there was a (black) edge from u_3 to 3 in G_3 , but a white (not present) edge from v_3 to 3. The sequence is a 1-contraction sequence since the maximum red degree of any vertex in the sequence is 1, proving that the twinwidth of C_4 is at most 1 (it is actually 0 since contracting 2 and 3 in G_3 rather than 2 and z_3 yields a 0-contraction sequence).

From Contraction Sequence to DAG Compression. Intuitively, contraction sequences “form clusters of vertices in a tree-like manner through contractions” and we can use this idea to build cluster DAGs from contraction sequences. Crucially, we can then insert compressed edges whenever a black edge is “about to finally disappear” and this will allow us to keep their number small (that is, linear in the number of vertices).

In detail, let us now assume that a fixed d -contraction sequence (G_n, \dots, G_1) for a graph $G = (V, E)$ is given in which for each $i \in \{n, \dots, 2\}$ we contract u_i and v_i in the trigraph G_i into z_{i-1} in order to form the trigraph G_{i-1} . Our objective is to construct a DAG compression $D = (V', A, E')$ for G using the sequence such that $|A| + |E'| \leq (3d + 4) \cdot n$. In particular, for every fixed d , the size of D is in $O(n)$.

The first step is to define the cluster DAG $C = (V', A)$. This is done as follows, see Figure 4 for an example:

► **Definition 4.1.** *The cluster tree $C = (V', A)$ of a contract sequence (G_n, \dots, G_1) has $V' = V \cup \{z_{n-1}, \dots, z_1\}$ and for each $i \in \{2, \dots, n\}$ there are edges from z_{i-1} to both u_i and v_i to A , that is, $A = \{(z_{i-1}, u_i) \mid i \in \{2, \dots, n\}\} \cup \{(z_{i-1}, v_i) \mid i \in \{2, \dots, n\}\}$.*



■ **Figure 4** The cluster DAG resulting from the contraction sequence from Figure 3 and the DAG compression of C_4 resulting from it. The two compressed edges $(z_3, 2)$ and $(z_3, 3)$ are both added when G_3 is contracted to G_2 , but for different reasons: In G_3 there is a black edge $(z_3, 2)$, but $(\alpha_3(z_3), \alpha_3(2)) = (z_2, z_2)$ is not a black edge in G_2 (there are no self-loops). In G_3 there is also a black edge $(z_3, 3)$, but while in G_2 there is an edge $(\alpha_3(z_3), \alpha_3(3)) = (z_2, 3)$, it is red.

A simple lemma will be useful in the following:

► **Lemma 4.2.** *Let (V', A) be the cluster tree of a contraction sequence (G_n, \dots, G_1) of $G = (V, E)$ and let $x', y' \in V'$ be two different vertices in some G_i . Then there is a black edge between x' and y' in G_i if, and only if, $C(x') \times C(y') \subseteq E$.*

Proof. By induction. The start G_n is trivial. Suppose the claim holds for G_i , we need to show that it also holds for G_{i-1} , where u and v have been contracted to z . Consider two different vertices x' and y' in G_{i-1} . If neither of them is z , then the contraction does not change whether there is a black edge between them, so the induction hypothesis yields the claim. Since the vertices must be different, the only remaining case we need to consider is $x' \neq y' = z$. First, suppose that there is black edge (x', z) in G_{i-1} . By definition of contractions, the existence of this black edge implies that there are black edges (x', u) and (x', v) in G_i . By the induction hypothesis this yields that $C(x') \times C(u) \subseteq E$ and also $C(x') \times C(v) \subseteq E$. But, then, by construction we have $C(y') = C(z) = C(u) \cup C(v)$ and hence $C(x') \times C(y') \subseteq E$. Second, there is no black edge (x', z) in G_{i-1} . Then either (x', u) or (x', v) was not a black edge in G_i . By the induction hypothesis, either $C(x') \times C(u) \not\subseteq E$ or $C(x') \times C(v) \not\subseteq E$. Since we still have $C(y') = C(z) = C(u) \cup C(v)$, we get $C(x') \times C(y') \not\subseteq E$. ◀

The second step is to add as few compression edges as possible, that is, to keep E' small (while, of course, D is still a compression of G), by adding compression edges “as late as possible”. In detail, for $i \in \{n, \dots, 2\}$ let $\alpha_i: V' \rightarrow V'$ be the function that maps both u_i and v_i to z_{i-1} and is the identity otherwise, so $\alpha_i(u_i) = \alpha_i(v_i) = z_{i-1}$ and $\alpha_i(v) = v$ for $v \in V' \setminus \{u_i, v_i\}$. In other words, α_i tells us “what became of a vertex v from G_i in G_{i-1} .” We now add a *compression edge* (u, v) to E' whenever (u, v) is a black edge in G_i but $(\alpha_i(u), \alpha_i(v))$ is no longer black or no longer present in G_{i-1} . Formally:

► **Definition 4.3.** *The set of compressed edges E' of a contraction sequence (G_n, \dots, G_1) is $E' = \{(u, v) \in V' \times V' \mid \text{there is an } i \in \{2, \dots, n\} \text{ with } (u, v) \in B_{G_i}, \text{ but } (\alpha_i(u), \alpha_i(v)) \notin B_{G_{i-1}}\}$.*

► **Theorem 4.4.** *For each (undirected, loop-free) graph G of twinwidth at most d there is a DAG compression $D = (V', A, E')$ with $|V'| = 2n - 1$, $|A| = 2n - 2$, and $|E'| \leq 2 \cdot (2d + 1) \cdot (n - 1)$.*

Proof. Since G has twinwidth at most d , there is a d -contraction sequence (G_n, \dots, G_1) for it. Consider the DAG compression $D = (V', A, E')$ where (V', A) is the cluster tree from Definition 4.1 for the contraction sequences and E' is the set of compressed edges from Definition 4.3 for the sequence.

The claim concerning the size of $|V'|$ follows trivially from Definition 4.1. For the size of $|A|$, just note that a binary tree on k vertices has $k - 1$ edges. For the size of E' , we must count “how many black edges can disappear when G_i is contracted into G_{i-1} .” First, if there is a black edge (u_i, v_i) in G_i , it will disappear, resulting in $(u_i, v_i) \in E'$. Second, if there is a vertex $v \in V' \setminus \{u_i, v_i\}$ such that (v, u_i) is black, but $(\alpha_i(v), \alpha_i(u_i)) = (v, z_{i-1})$ is red, we will add (v, u_i) to E' (and also (u_i, v) , the symmetric edge, which is accounted for by the factor of 2 in the bound of the theorem). Likewise, if (v, v_i) is black, but (v, z_{i-1}) is red, we add (v, v_i) to E' . Crucially, for any v , when at least one of the at most two black edges (v, u_i) or (v, v_i) is added to E' , the edge (v, z_{i-1}) is red. Since the maximum red degree of any vertex, including z_{i-1} , is d , there can be at most d red edges and hence at most $2d$ black edges may have disappeared. All told, from G_i to G_{i-1} we add at most $2 \cdot (2d + 1)$ compressed edges to E' .

It remains to argue that D is, indeed, a DAG compression of G . However, Lemma 4.2 immediately implies that all compressed edges we add to E' are “correct”, that is, for every compressed edge $(u', v') \in E'$ we have $C(u') \times C(v') \subseteq E$. Furthermore, the lemma also implies that we do not “miss” any edges from E : Every edge of E is present in G' as a black edge and remains present as an element of some $C(u') \times C(v')$ until the black edge disappears – which is exactly the moment we add (u', v') to E' . Finally, G_1 is a single vertex and contains no edges, so all edges in E will be accounted for in E' at some point. ◀

By the above theorem, all graphs of twinwidth d admit a DAG compression (indeed, even a tree compression) of size $O(d \cdot n)$. However, *computing* the DAG compression of a graph of bounded twinwidth is a potentially difficult problem since it is already NP-hard to decide whether the twinwidth of a graph is 4 and, hence, it is also impossible to compute optimal contraction sequences in polynomial time unless $P = NP$. For these reasons, we can only hope to be able to compute the DAG compression of G when we are given a d -contraction sequence already as part of the input – and it is standard practice in the literature for algorithms working on graphs of twinwidth d to assume that this is the case.

However, one needs to be careful *how* the contraction sequence is represented in the input. Clearly, it makes little sense to assume that a string encoding the actual sequence $(G_n, G_{n-1}, \dots, G_1)$ is given as input – when G is dense, we wish to avoid explicitly keeping

all edges of G_n in memory, let alone storing the whole sequence. Also, statements like “BFS can be solved in time $O(dn)$ for a graph G when a d -contraction sequence of G is given” are much less impressive if this presupposes that the input may take up $O(n^3)$ cells of memory. On the other hand, it is also not sufficient to just be given, say, for each $i \in \{n, \dots, 2\}$ the vertices u_i and v_i that get contracted to z_{i-1} : We then miss the information which black edges got removed.

What we really need is, in addition to the contraction pairs, for each $i \in \{n, \dots, 2\}$ for each red edge (v, z_{i-1}) the color of the edges (v, u_i) and (v, v_i) : We can then reconstruct which black edges were lost from G_i to G_{i-1} . The following definition and theorem make these observations explicit:

► **Definition 4.5.** *Let (G_n, \dots, G_1) be a contraction sequence such that in G_i we contract u_i and v_i into z_{i-1} to get G_{i-1} . Define the color recording sequence of the contraction sequence as a sequence of tuples $(t_n, t_{n-1}, \dots, t_2)$ such that each t_i contains the following:*

1. *The vertices u_i, v_i, z_{i-1} .*
2. *The color of the edge (u_i, v_i) in G_i .*
3. *For each vertex v such that (v, z_{i-1}) is a red edge in G_{i-1} , the colors of the edges (v, u_i) and of (v, v_i) in G_i .*

Observe that for a d -contraction sequence each tuple t_i in the color recording sequence contains at most $2d + 4$ vertices and, hence, the whole sequence can be stored using $O(d \cdot n)$ words of memory.

► **Theorem 4.6.** *There is an algorithm that gets a color recording sequence of a d -contraction sequences of a graph G as input and outputs in time $O(d \cdot n)$ a DAG compression D of G of size $O(d \cdot n)$.*

Proof. Having a look at the definitions of cluster trees (Definition 4.1) and of how the compressed edges E' are derived from a contraction sequence (Definition 4.3), we immediately see that the color recording sequence contains exactly the information needed to output (V', A, E') in time linear in the size of this output. ◀

Of course, the above theorem and definition beg the question of where the “color recording sequences” might come from. Firstly, it may be the case that we *do* have access (perhaps only during a preprocessing phase) to the graphs G_i . In this case, assuming that they are stored using standard data structures that combine the advantages of an adjacency matrix and list [1] and also assuming that we are told which vertices get contracted in each step, we can easily compute the color recording sequence by iterating over the red edges incident to each z_{i-1} . Secondly, the graph G and the contraction sequence may be the output of some algorithmic process. In this case, one needs to adapt the output or the process so that the color recording sequence gets output.

Graphs with Large Twinwidth and Linear-Size DAG Compressions. The results in this section suggest a tight link between twinwidth and linear-size DAG compressions: When G has twinwidth d , then G also has a size- $O(dn)$ DAG compression. Indeed, it even has a size- $O(dn)$ tree compression, that is, a DAG compression where $C = (V', A)$ is a tree. This raises the question of whether, perhaps, the reverse is also true: It is true that all graphs having a, say, linear-size tree compression, also have low twinwidth? The answer to this is negative:

► **Theorem 4.7.** *There is a sequence of graphs (G_1, G_2, \dots) such that G_d has twinwidth at least d , but each $G_d = (V_d, E_d)$ admits a tree compression of size at most $|V_d|$.*

Proof. For each d , we can construct a graph H_d that has twinwidth at least d : For instance, the rook graph on n vertices from Example 2.6 has twinwidth at least \sqrt{n} since contracting any two different vertices immediately yields a vertex with \sqrt{n} incident red edges.

Let G_d be the graph H_d to which we add $n_d^2 - n_d$ isolated vertices, where n_d is the number of vertices of H_d . Then G_d has n_d^2 vertices and also twinwidth at least d since adding isolated vertices does not change the twinwidth. The number of edges in G_d equals that of H_d , so it can be at most n_d^2 . This shows that $|E_d| \leq n_d^2 = |V_d|$; in other words, the graph has a linear number of edges. As shown in Example 2.3 we can “compress” it by simply doing nothing to get a size- $|V_d|$ compression. ◀

5 Conclusion

In this paper, we presented a new data structure, the *DAG compression*, that represents graphs by storing complete bipartite subgraphs as single compression edges between vertices that represent clusters of vertices. We showed that some update operations are possible on these compressions, but further research is needed to better understand them. A crucial property was that the DAG compression of a graph gives rise to another graph, which we called the *switching graph*, that is distance-equivalent to the original graph and is only twice the size of the compressed graph.

When the size of a DAG compression is linearly bounded by the number of vertices in the original graphs, the compression gives rise to a new framework for graph algorithms with running times that are independent of the number of edges in the input: We can run breadth- and depth-first search in time $O(n)$ where n is the number of vertices in the input, if we have access to a linear-size DAG compression. Moreover, extending the definition to weighted graphs, we can run Dijkstra’s algorithm in time $O(n \log n)$ on such graphs. We believe that further algorithms that work directly on DAG compressions rather than on the original graphs are possible.

We also showed that all graphs of bounded twinwidth admit a linear-size DAG compression. The reverse, however, is not true. A natural research avenue would be to extend this result to further graph classes: Is it true that all graphs of, say, bounded flip-width admit a linear-size DAG compression?

References

- 1 Faisal N. Abu-Khzam, Michael A. Langston, Amer E. Mouawad, and Clinton P. Nolan. A hybrid graph representation for recursive backtracking algorithms. In *Proceedings of the 4th International Workshop on Frontiers in Algorithmics (FAW 2010)*, volume 6213 of *Lecture Notes in Computer Science*, pages 136–147. Springer, 2010. doi:10.1007/978-3-642-14553-7_15.
- 2 Jungho Ahn, Debsoumya Chakraborti, Kevin Hendrey, Donggyu Kim, and Sang il Oum. Twin-width of random graphs. Technical Report arXiv:2212.07880, arXiv, 2022. doi:10.48550/arXiv.2212.07880.
- 3 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics*, pages 18:1–18:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.18.
- 4 Édouard Bonnet, Colin Geniet, Eun J. Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: Max independent set and coloring. In *Proceedings of the 48th International Colloquium on Automata, Languages and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.35.

- 5 Édouard Bonnet, Eun J. Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: Tractable FO model checking. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS 2020)*, pages 601–612. IEEE, 2020. doi:10.1109/FOCS46700.2020.00062.
- 6 Édouard Bonnet, Jaroslav Nesetril, Patrice O. de Mendez, Sebastian Siebertz, and Stephan Thomassé. Twin-width and permutations. In *Proceedings of the European Conference on Combinatorics, Graph Theory and Applications 2023 (EUROCOMB 2023)*. Masaryk University, Brno, Czech Republic, 2023. doi:10.5817/CZ.MUNI.EUROCOMB23-022.
- 7 Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific web resource discovery. *International Journal of Computer and Telecommunications Networking*, 31(11-16):1623–1640, 1999. doi:10.1016/S1389-1286(99)00052-3.
- 8 Chris J. Cheney. A nonrecursive list compacting algorithm. *Communications of the ACM*, 13(11):677–678, 1970. doi:10.1145/362790.362798.
- 9 Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through URL ordering. *International Journal of Computer and Telecommunications Networking*, 30(1-7):161–172, 1998. doi:10.1016/S0169-7552(98)00108-1.
- 10 Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1989.
- 11 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.
- 12 Jeff Erickson. *Algorithms*. Erickson, 2019.
- 13 Robert Ganian, Filip Pokrývka, André Schidler, Kirill Simonov, and Stefan Szeider. Weighted model counting with twin-width. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, volume 236 of *Leibniz International Proceedings in Informatics*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAT.2022.15.
- 14 Michel Habib and Christophe Paul. A simple linear time algorithm for cograph recognition. *Discrete Applied Mathematics*, 145(2):183–197, 2005. doi:10.1016/J.DAM.2004.01.011.
- 15 Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Journal of Artificial Intelligence*, 27(1):97–109, 1985. doi:10.1016/0004-3702(85)90084-0.
- 16 Stefan Kratsch, Florian Nelles, and Alexandre Simon. On triangle counting parameterized by twin-width. Technical Report arXiv:2202.06708, arXiv, 2022. doi:10.48550/arXiv.2202.06708.
- 17 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983. doi:10.1145/2402.322385.
- 18 Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD 2008)*, pages 419–432. Association for Computing Machinery, 2008. doi:10.1145/1376616.1376661.
- 19 James Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977. doi:10.1016/1385-7258(77)90055-5.
- 20 Judea Pearl. *Heuristics – Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.
- 21 André Schidler and Stefan Szeider. A SAT approach to twin-width. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, (ALENEX 2022)*, pages 67–77. Society for Industrial and Applied Mathematics, 2022. doi:10.1137/1.9781611977042.6.
- 22 Micha Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67–72, 1981. doi:10.1016/0898-1221(81)90008-0.
- 23 Robert E. Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6:171–185, 1976. doi:10.1007/BF00268499.

- 24 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.
- 25 Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD 2008)*, pages 567–580. Association for Computing Machinery, 2008. doi:10.1145/1376616.1376675.
- 26 Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2011)*, pages 965–973. Association for Computing Machinery, 2011. doi:10.1145/2020408.2020566.
- 27 Ning Zhang, Yuanyuan Tian, and Jignesh M. Patel. Discovery-driven graph summarization. In *Proceedings of the 26th International Conference on Data Engineering (ICDE 2010)*, pages 880–891. IEEE, 2010. doi:10.1109/ICDE.2010.5447830.
- 28 Édouard Bonnet, Eun J. Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. In Petr A. Golovach and Meirav Zehavi, editors, *Proceedings of the 16th International Symposium on Parameterized and Exact Computation, (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics*, pages 10:1–10:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.IPEC.2021.10.

Testing Equivalence to Design Polynomials

Omkar Baraskar ✉

Indian Institute of Science, Bengaluru, India

Agrim Dewan ✉

Indian Institute of Science, Bengaluru, India

Chandan Saha ✉

Indian Institute of Science, Bengaluru, India

Abstract

An n -variate polynomial g of degree d is a (n, d, t) *design polynomial* if the degree of the gcd of every pair of monomials of g is at most $t - 1$. The power symmetric polynomial $\text{PSym}_{n,d} := \sum_{i=1}^n x_i^d$ and the sum-product polynomial $\text{SP}_{s,d} := \sum_{i=1}^s \prod_{j=1}^d x_{i,j}$ are instances of design polynomials for $t = 1$. Another example is the Nisan-Wigderson design polynomial NW, which has been used extensively to prove various arithmetic circuit lower bounds. Given black-box access to an n -variate, degree- d polynomial $f(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$, how fast can we check if there exist an $A \in \text{GL}(n, \mathbb{F})$ and a $\mathbf{b} \in \mathbb{F}^n$ such that $f(A\mathbf{x} + \mathbf{b})$ is a (n, d, t) design polynomial? We call this problem “testing equivalence to design polynomials”, or alternatively, “equivalence testing for design polynomials”.

In this work, we present a randomized algorithm that finds (A, \mathbf{b}) such that $f(A\mathbf{x} + \mathbf{b})$ is a (n, d, t) design polynomial, if such A and \mathbf{b} exist, provided $t \leq d/3$. The algorithm runs in $(nd)^{O(t)}$ time and works over any sufficiently large \mathbb{F} of characteristic 0 or $> d$. As applications of this test, we show two results – one is structural and the other is algorithmic. The structural result establishes a polynomial-time equivalence between the graph isomorphism problem and the polynomial equivalence problem for design polynomials. The algorithmic result implies that Patarin’s scheme (EUROCRYPT 1996) can be broken in quasi-polynomial time if a random sparse polynomial is used in the key generation phase.

We also give an efficient learning algorithm for n -variate random affine projections of multilinear degree- d design polynomials, provided $n \geq d^4$. If one obtains an analogous result under the weaker assumption “ $n \geq d^\epsilon$, for any $\epsilon > 0$ ”, then the NW family is *not* VNP-complete unless there is a VNP-complete family whose random affine projections are learnable. It is not known if random affine projections of the permanent are learnable.

The above algorithms are obtained by using the vector space decomposition framework, introduced by Kayal and Saha (STOC 2019) and Garg, Kayal and Saha (FOCS 2020), for learning non-degenerate arithmetic circuits. A key technical difference between the analysis in the papers by Garg, Kayal and Saha (FOCS 2020) and Bhargava, Garg, Kayal and Saha (RANDOM 2022) and the analysis here is that a certain adjoint algebra, which turned out to be trivial (i.e., diagonalizable) in prior works, is non-trivial in our case. However, we show that the adjoint arising here is triangularizable which then helps in carrying out the vector space decomposition step.

2012 ACM Subject Classification Theory of computation \rightarrow Algebraic complexity theory

Keywords and phrases Polynomial equivalence, design polynomials, graph isomorphism, vector space decomposition

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.9

Related Version *Full Version:* <https://ecc.weizmann.ac.il/report/2024/004/> [7]

Funding *Chandan Saha:* Partially supported by a MATRICS grant of the Science and Engineering Research Board, DST, India.

Acknowledgements We thank the reviewers for their valuable feedback, which has helped us improve the presentation of this work. In particular, we thank one of the reviewers who pointed us to appropriate citations for the adjoint algebra.



© Omkar Baraskar, Agrim Dewan, and Chandan Saha;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 9; pp. 9:1–9:22



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

The polynomial equivalence problem (PE) is fundamental in algebraic complexity theory. Given two polynomials, f and g , how fast can we check if one is in the orbit of the other? *Orbit* of a polynomial $f \in \mathbb{F}[\mathbf{x}]$ is the set $\{f(A\mathbf{x}) : A \in \text{GL}(|\mathbf{x}|, \mathbb{F})\}$. In other words, PE is the problem of checking if f and g are the same function up to a change of the coordinate system. It can be regarded as the algebraic analogue of the graph isomorphism (GI) problem.

Much is unknown about the exact complexity of PE. Over finite fields, PE is unlikely to be NP-complete [37, 35], but no polynomial-time algorithm is known unless f and g are quadratic forms [29, 4]. PE for cubic forms over \mathbb{Q} is not even known to be decidable. Cubic form equivalence (CFE) is polynomial-time equivalent to several other fundamental problems in algebra and linear algebra [18, 19, 3]. GI reduces to CFE in polynomial time [2], but the converse is not known to be true. A natural question emerges at this point:

Is GI polynomial-time equivalent to PE for some natural class of polynomials?

We provide an affirmative answer to this question in Theorem 11 by studying the problem of testing equivalence to design polynomials which we also refer to as the equivalence testing problem for the family of design polynomials (see Definitions 1 and 2).

Equivalence testing (ET) is closely related to the PE problem. ET for a polynomial family or a circuit class \mathcal{F} is a problem wherein we are given a polynomial f , and we wish to check if f is in the orbit of some polynomial or circuit in \mathcal{F} .¹ Efficient ET algorithms are known for a variety of polynomial families and a few circuit classes, namely the permanent [24], the determinant [24, 15], the iterated matrix multiplication polynomial family [26, 33], the elementary and power symmetric polynomials [23], the sum-product polynomial family [31], the continuant [31], and read-once formulas [21]. One important family that is missing from the above list is the Nisan-Wigderson design polynomial family NW (see Equation 1). The NW family has been used in many results on arithmetic circuit lower bounds in the last decade. But, unlike the other families, NW had no known ET. In fact, ET for NW over \mathbb{Q} was not known to be decidable. We ask a more general question:

Is there an ET algorithm for the family of (general) design polynomials?

Our main result, given in Theorem 3, is an ET algorithm for design polynomials over *any* field of zero or sufficiently large characteristic. The algorithm reveals a structural property of invertible transformations between design polynomials that enables us to prove Theorem 11. The running time of the algorithm also helps us point out a vulnerability of Patarin’s authentication scheme if a random sparse polynomial is chosen in the key generation phase.

Patarin [34] proposed a zero-knowledge authentication scheme based on the presumed hardness of PE for random cubic forms (more generally, constant-degree forms). A random n -variate cubic form $f(\mathbf{x})$ is chosen in the key generation phase along with two random transforms $A_1, A_2 \in \text{GL}(n, \mathbb{F})$. The polynomials $g_1 := f(A_1\mathbf{x})$ and $g_2 := f(A_2\mathbf{x})$ are made public; the secret is the transform $A_1^{-1}A_2$, which maps g_1 to g_2 . A random cubic form has sparsity (i.e., number of monomials) $O(n^3)$. It is natural to ask: What if we choose a *random* $O(n^3)$ -sparse polynomial f of a *higher degree* (see Definition 12) in the key generation step?

Can Patarin’s scheme be broken if a random $n^{O(1)}$ -sparse polynomial is chosen as the key?

¹ Note that the ET problem for \mathcal{F} is not the same as the PE problem for \mathcal{F} . In the latter case, we are given two polynomials $f, g \in \mathcal{F}$, and we wish to check if one is in the orbit of the other. One may alternatively call the ET problem for \mathcal{F} as “testing equivalence to \mathcal{F} ” (as in the title of this article).

It turns out that a random sparse polynomial is a design polynomial and has no nontrivial permutation symmetry with high probability (see Lemma 36 and Proposition 37). These features let us invoke Theorem 3 and answer the above question in Theorem 13 via a reduction to GI.

Our final result is a learning algorithm for random affine projections of design polynomials. An equivalence test for NW is a special case of learning affine projections of NW. Consider the (qd, d, t) design polynomial $\text{NW}_{q,d,t}$ as defined in Equation 1. A polynomial $f = \text{NW}_{q,d,t}(A\mathbf{x} + \mathbf{b})$, where $|\mathbf{x}| = n \leq qd$, $A \in \mathbb{F}^{qd \times n}$ and $\mathbf{b} \in \mathbb{F}^{qd}$, is an n -variate affine projection of $\text{NW}_{q,d,t}$. Given access to f , can we learn the unknown A and \mathbf{b} ? If $n = qd$ and $A \in \text{GL}(n, \mathbb{F})$, then the problem is the same as ET for NW. However, for arbitrary $n < qd$ and $A \in \mathbb{F}^{qd \times n}$, the problem can be rather difficult, even for $t = 1$, as every depth-3 circuit is an affine projection of $\text{NW}_{q,d,1}$ for some q and d . Learning depth-3 circuits in the worst-case is a challenging problem due to known depth reduction results (see the discussion in [27]). But does the task of discovering A become easier if A is randomly chosen? In other words:

Can we learn random affine projections of NW efficiently?

Random affine projections of special design polynomials, such as the power symmetric polynomial and the sum-product polynomial, have been studied, and efficient learning algorithms have been provided in [27]. But it is unclear what to expect for NW. The reason is that, unlike the determinant and the permanent, we do not have a good understanding of the expressive power of affine projections of NW. The permanent is VNP-complete under p-projections² [38], but NW is not known to be so. For $d = n^{O(1)}$, no learning algorithm is known for n -variate random affine projections of the $d \times d$ permanent which has time complexity polynomial in $\binom{n+d}{n}$ – the maximum sparsity of any n -variate, degree- d polynomial. In fact, it is conjectured in [1] that n -variate random affine projections of the $d \times d$ *determinant* form a pseudorandom function family when $d = n^{O(1)}$. If true, then there is no $\binom{n+d}{n}^{O(1)}$ -time learning algorithm for random affine projections of the determinant. If such a conclusion holds for the determinant, which is VBP-complete, then we expect the same to hold for the permanent or any other VNP-complete family, as $\text{VBP} \subseteq \text{VNP}$. So, an efficient learning algorithm for random affine projections of NW may indicate NW is *not* VNP-complete.

In Theorem 18, we give an efficient learning algorithm for n -variate random affine projections of multilinear degree- d design polynomials, provided $n \geq d^4$. If we obtain an analogous result under the weaker assumption “ $n \geq d^\epsilon$, for any $\epsilon > 0$ ”, then the NW family is not VNP-complete assuming that there is no VNP-complete family whose random affine projections are efficiently learnable. On the other hand, if NW happens to be VNP-complete, then it is unlikely that we will be able to weaken the $n \geq d^4$ condition significantly without compromising the other parameters of the theorem considerably.

1.1 Our results

We now state our results formally. Assume that an efficient univariate polynomial factoring algorithm over \mathbb{F} is available; this assumption is well justified over \mathbb{Q} and \mathbb{F}_q [30, 8].

► **Definition 1** (Design polynomial). *An n -variate, degree- d polynomial $g = \sum_{i \in [s]} c_i m_i$, where m_i is a monomial and $c_i \in \mathbb{F}$, is a (n, d, s, t) design polynomial if $\forall i \neq j$, $\deg \gcd(m_i, m_j) < t$.*

² If every row of A has at most one nonzero entry, then it is a p-projection.

9:4 Testing Equivalence to Design Polynomials

Some well-known polynomials that are also design polynomials are the sum-product polynomial $\text{SP}_{s,d} := \sum_{i=1}^s \prod_{j=1}^d x_{i,j}$, which is a $(sd, d, s, 1)$ design polynomial, and the power symmetric polynomial $\text{PSym}_{n,d} := \sum_{i=1}^n x_i^d$, which is a $(n, d, n, 1)$ design polynomial. The most relevant example is the Nisan-Wigderson design polynomial $\text{NW}_{q,d,t}$, defined as:

$$\text{NW}_{q,d,t} := \sum_{h \in \mathbb{F}_q[y], \deg h < t} \prod_{i=0}^{d-1} x_{i,h(i)}, \quad \text{for } t \leq d \leq q, \text{ where } q \text{ is a prime.} \quad (1)$$

As two univariate polynomials of degree $< t$ agree at $\leq t - 1$ points, $\text{NW}_{q,d,t}$ is a (qd, d, q^t, t) design polynomial. Whenever the parameter s is not required, we will write (n, d, t) design polynomial by omitting s . Also, for simplicity, we assume that design polynomials are *homogeneous*. Our results hold for non-homogeneous design polynomials as well.

► **Definition 2** (The ET problem for design polynomials). *Given black-box access to an n -variate, degree- d polynomial $f \in \mathbb{F}[\mathbf{x}]$, check if there exist an $A \in \text{GL}(n, \mathbb{F})$ and a $\mathbf{b} \in \mathbb{F}^n$ such that $f(A\mathbf{x} + \mathbf{b})$ is a (n, d, t) design polynomial, and if so, recover A and \mathbf{b} .*

► **Theorem 3** (ET for design polynomials). *Let $n, d, s, t \in \mathbb{N}$, $d \geq 3t$, $\text{char}(\mathbb{F}) = 0$ or $> d$ and $|\mathbb{F}| > \max(s^3, d^7)$. There is a randomized, $\text{poly}((nd)^t)$ -time³ algorithm that takes input black-box access to an n -variate, degree- d polynomial $f \in \mathbb{F}[\mathbf{x}]$, with the promise that there exist some (n, d, s, t) design polynomial g and some $A \in \text{GL}(n, \mathbb{F})$ such that $f = g(A\mathbf{x})$, and outputs, with high probability, a $B \in \text{GL}(n, \mathbb{F})$ and a (n, d, s, t) design polynomial h such that $B = PSA$ and $f = h(B\mathbf{x})$, where P, S are permutation and scaling matrices, respectively.*

► **Remark 4.** If g is multilinear, then the condition $d \geq 3t$ can be improved to $d > 2t$.

► **Remark 5.** The condition $|\mathbb{F}| > \max(s^3, d^7)$ arises due to the use of the Schwartz–Zippel lemma⁴ in our analysis and in the factorization algorithm of [22]. If f is given as a circuit, the algorithm can work with an extension field, irrespective of the size of \mathbb{F} , and still obtain a B with entries in \mathbb{F} . A finite field extension can be constructed efficiently (see Section 14.9 in [39]). This feature of the algorithm is explained in [7].

► **Remark 6.** The theorem gives an ET algorithm for NW (see Theorem 50). The algorithm also works over \mathbb{Q} , where ET for NW was not known to be decidable.

► **Remark 7.** A random sparse polynomial is a $(n, d, s, d/3)$ design polynomial with high probability (see Lemma 36). Thus, we have ET for random sparse polynomials.

In Section 3, we prove Theorem 3 and elaborate on how to handle non-homogeneous design polynomials and transforms of the form $A\mathbf{x} + \mathbf{b}$, where $A \in \text{GL}(n, \mathbb{F})$ and $\mathbf{b} \in \mathbb{F}^n$.

► **Definition 8** (Symmetries of a polynomial). *Let $f \in \mathbb{F}[\mathbf{x}]$ be an n -variate, degree- d polynomial. The set $\mathcal{G}_f := \{A \in \text{GL}(n, \mathbb{F}) : f(A\mathbf{x}) = f\}$ is the group of symmetries of f .*

The authors of [20] studied the symmetries of NW by examining the Lie algebra associated with it, while these corollaries of Theorem 3 (which is proved using a different technique) hold for general design polynomials.

► **Corollary 9** (Symmetries of design polynomials). *Let $d \geq 3t$, f be a (n, d, t) design polynomial and $A \in \mathcal{G}_f$. Then, $A = PS$, where P is a permutation matrix and S is a scaling matrix.*

³ Here, “time” means number of field operations. Over \mathbb{Q} , the complexity is $\text{poly}((nd)^t, \beta)$, where β is the bit complexity of the coefficients of f .

⁴ also known as the DeMillo–Lipton–Schwartz–Zippel lemma [14, 41, 36].

► **Corollary 10** (Equivalent design polynomials). *Let $d \geq 3t$, $A \in \text{GL}(n, \mathbb{F})$ and f, g be (n, d, t) design polynomials such that $f = g(A\mathbf{x})$. Then, $A = PS$, where P, S are as stated above.*

Our second result, proven in Section 3.5, shows that $\text{GI} \equiv_p \text{PE}$ for design polynomials with all-one coefficients. Here, \equiv_p denotes polynomial-time, many-one equivalence.

► **Theorem 11** (GI and PE). *$\text{GI} \equiv_p \text{PE}$ for $(n, 6, 2)$ design polynomials with all-one coefficients.*

Theorem 11 holds for (n, d, t) design polynomials for any $d \geq 6$ and $t \leq d/3$, and also for (n, d, t) multilinear design polynomials with $d \geq 5$ and $t < d/2$. Thus, PE for (n, d, t) design polynomials with all-one coefficients, for $d \geq 6$ and $t \leq d/3$, polynomial-time reduces to PE for $(n, 6, 2)$ design polynomials with all-one coefficients.⁵

Our third result, proven in Section 3.5, shows that if a random sparse polynomial of sufficiently large constant degree is used in Patarin’s scheme for public and private key generations, then the private key can be recovered in quasi-polynomial time.

► **Definition 12** (Random sparse polynomial). *An n -variate, degree- d , s -sparse polynomial $f(\mathbf{x})$ is a random s -sparse polynomial if each monomial is formed by picking d variables uniformly and independently at random from \mathbf{x} ; the coefficients are then chosen arbitrarily.*

► **Theorem 13** (A vulnerability of Patarin’s scheme). *Let $n, s, d, q \in \mathbb{N}$, $n > d^8$, $n^3 \leq s < \left(\frac{n}{d^2}\right)^{d/6}$, $d \geq 25$ be a constant, and $q = n^{O(1)}$. Let f be a random s -sparse polynomial over \mathbb{F}_q . If f is used in Patarin’s scheme for key generation, then the scheme can be broken in quasi-poly(n) time.*

► **Remark 14.** The bound on s , stated in the theorem, is for simplicity. The precise bound is $n^2 \log(n) \leq s \leq \sqrt{\epsilon} \left(\frac{n}{d^2}\right)^{d/6}$, where ϵ is the constant from Lemma 36. The lower bound on d can be derived from the inequality $n^2 \log(n) \leq \sqrt{\epsilon} \left(\frac{n}{d^2}\right)^{d/6}$ and fixing $\epsilon = 0.01$.

► **Remark 15.** Patarin’s scheme was shown to be vulnerable in [23] when using a random constant-degree *multilinear* polynomial for key generation. In [23] a random polynomial was defined by selecting the coefficients of *all* multilinear monomials randomly and independently, while we allow arbitrary coefficients, non-multilinear monomials, and a lower number of monomials.

Our fourth and final result, proven in Section 4, gives an algorithm to learn random affine projections of multilinear design polynomials such as the polynomials in the NW family.

► **Definition 16** (Affine projections). *Let $m, n \in \mathbb{N}$, $m \geq n$. Let f and g be polynomials in n and m variables, respectively. If $f(\mathbf{x}) = g(A\mathbf{x} + \mathbf{b})$ for some $A \in \mathbb{F}^{m \times n}$ and $\mathbf{b} \in \mathbb{F}^m$, then f is an affine projection of g . An affine projection is random if $A \in_r \mathbb{F}^{m \times n}$, where \in_r denotes that the entries of A are chosen randomly and independently from a sufficiently large subset of \mathbb{F} .*

► **Definition 17** (Learning affine projections of design polynomials). *Given black-box access to an n -variate $f \in \mathbb{F}[\mathbf{x}]$, which is an affine projection of an unknown (m, d, s, t) design polynomial g , recover $B \in \mathbb{F}^{m \times n}$, $\mathbf{c} \in \mathbb{F}^m$ and a (m, d, s, t) design polynomial h such that $f = h(B\mathbf{x} + \mathbf{c})$.*

⁵ It is worth noting, in [2], the authors showed a reduction from PE for degree- d forms to PE for cubic forms over fields containing d -th roots. However, the reduction there does not seem to preserve the design condition.

► **Theorem 18** (Learning random affine projections of multilinear design polynomials). *Let $m, n, d, s, t \in \mathbb{N}$, $m \geq n \geq d^{4+\epsilon}$, where $\epsilon > 0$, $d \geq 3t$, $s < \left(\frac{\sqrt{n}}{d^2}\right)^{\frac{d}{13}}$. Let $\text{char}(\mathbb{F}) = 0$ or $> d$ and $|\mathbb{F}| \geq \text{poly}(sd)d^t$. There is a randomized, $\text{poly}(m, s, n^t)$ -time algorithm that takes input black-box access to an n -variate, degree- d polynomial $f \in \mathbb{F}[\mathbf{x}]$, with the promise that there exist some multilinear (m, d, s, t) design polynomial g and some $A \in_r \mathbb{F}^{m \times n}$ such that $f = g(A\mathbf{x})$, and outputs, with high probability, a $B \in \mathbb{F}^{m \times n}$ and a multilinear (m, d, s, t) polynomial h such that $B = PSA$ and $f = h(B\mathbf{x})$, where P, S are permutation and scaling matrices, respectively.*

► **Remark 19.** For $\text{NW}_{q,d,t}$ with $t \leq d/1300$, $m = qd = n^{10}$ and $n \geq d^5$, it holds that $s = q^t < \left(\frac{\sqrt{n}}{d^2}\right)^{\frac{d}{13}}$. Thus, we can learn random affine projections of $\text{NW}_{q,d,t}$ for $m = \text{poly}(n)$, assuming $n \geq d^{4+\epsilon}$. The precise bound on $|\mathbb{F}|$ is stated in [7].

► **Remark 20.** Theorem 18 *does not* imply that either NW is not VNP-complete or there is a VNP-complete family whose random affine projections are learnable. The reason is that the algorithm assumes $n \geq d^{4+\epsilon}$, but it may be the case that a VNP polynomial family is a projection of NW in the setting $n < d^4$.

► **Remark 21.** As mentioned earlier, our motivation for designing a learning algorithm for random affine projections of multilinear design polynomials originates from NW, which is also multilinear and design. We believe that a similar theorem holds for non-multilinear design polynomials as well. Theorem 3 provides evidence towards this belief since it holds for non-multilinear design polynomials as well.

In Section 4, we elaborate on how non-homogeneous multilinear design polynomials and general transforms of the form $A\mathbf{x} + \mathbf{b}$, where $A \in_r \mathbb{F}^{m \times n}$ and $\mathbf{b} \in \mathbb{F}^m$, are handled.

1.2 Proof techniques

The core underlying technique, used to prove Theorems 3 and 18, is based on the vector space decomposition framework introduced in [16, 27]. Suppose that f can be expressed as:

$$f = T_1 + T_2 + \dots + T_s, \tag{2}$$

and we wish to learn the *terms* T_1, \dots, T_s that are *simple* in some sense. For example, in our setting, each T_i is a product of linear forms (see details on next page). The authors in [16, 27] reduce the task of learning the T_i 's to the vector space decomposition (VSD) problem. We define the VSD problem first and then discuss the reduction.

Vector space decomposition (VSD) for (\mathcal{L}, U, V) . Given bases of vector spaces U, V and a set of linear maps \mathcal{L} from U to V , output a (further indecomposable) decomposition of U, V as:

$$U = U_1 \oplus \dots \oplus U_s \text{ and } V = V_1 \oplus \dots \oplus V_s, \text{ such that } \langle \mathcal{L} \circ U_i \rangle \subseteq V_i \text{ for all } i \in [s].$$

Reducing the learning problem to VSD. We choose appropriate sets of operators \mathcal{L}_1 and \mathcal{L}_2 to get spaces $U = \langle \mathcal{L}_1 \circ f \rangle$ and $V = \langle \mathcal{L}_2 \circ U \rangle$ such that the following are satisfied:

- $U = U_1 \oplus \dots \oplus U_s$ and $V = V_1 \oplus \dots \oplus V_s$, where $U_i = \langle \mathcal{L}_1 \circ T_i \rangle$ and $V_i = \langle \mathcal{L}_2 \circ U_i \rangle$.
- Each pair (U_i, V_i) is *indecomposable* with respect to \mathcal{L}_2 .
- The (further indecomposable) decomposition is *unique* up to a reordering of U_i 's and V_i 's.
- T_i 's can be recovered efficiently from the bases of the U_i 's.

If such two sets of operators can be found, then learning T_1, \dots, T_s reduces to the VSD problem for (\mathcal{L}_2, U, V) . The (U_i, V_i) 's need to be indecomposable and unique otherwise a VSD algorithm might output some other decomposition, making the recovery of the T_i 's hard.

Solving the VSD problem. The authors of [13] gave a polynomial-time algorithm for the symmetric case of the problem when $U = V$. The algorithm works over finite fields, \mathbb{C}, \mathbb{R} but not over \mathbb{Q} (if we wish to output a decomposition over \mathbb{Q}). The authors of [16] showed a reduction of VSD to the symmetric case. The authors of [9] and [16] exploited the structure of U and V (arising in their settings) to get a VSD algorithm that also works over \mathbb{Q} .

The VSD framework above gives rise to a meta algorithm for learning the “terms”. To use the algorithm for Theorems 3 and 18, we need appropriate sets of operators \mathcal{L}_1 and \mathcal{L}_2 satisfying the above-mentioned conditions which point to four technical steps in the analysis, which we state first and then discuss how to execute them for Theorems 3 and 18.

■ **Algorithm 1** Meta algorithm [16].

-
- **Input:** $f = T_1 + T_2 + \dots + T_s$, where T_i 's are unknown “simple” terms.
 - **Output:** T'_1, T'_2, \dots, T'_s such that $T'_i = T_{\pi(i)}$ for some permutation π on $[s]$.
1. Compute $U = \langle \mathcal{L}_1 \circ f \rangle$ and $V = \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ f \rangle$.
 2. Solve VSD for (\mathcal{L}_2, U, V) ; find the decompositions $U = U_1 \oplus \dots \oplus U_s$, $V = V_1 \oplus \dots \oplus V_s$.
 3. Recover T'_i from U_i .
-

1. **Direct sum structure:** This means establishing $U = U_1 \oplus \dots \oplus U_s$ and $V = V_1 \oplus \dots \oplus V_s$.
2. **Uniqueness of decomposition:** Once the direct sum holds, it needs to be shown that the decomposition of U and V is indecomposable with respect to \mathcal{L}_2 and unique up to permutations of the U_i 's and V_i 's. Inspired by the Krull-Schmidt theorem [28], [16] analysed the *adjoint algebra*⁶ associated with (\mathcal{L}_2, U, V) and pointed out a sufficient condition for uniqueness to hold. The adjoint algebra for (\mathcal{L}, U, V) is defined as $\text{Adj}(\mathcal{L}, U, V) := \{(D, E) \mid D : U \rightarrow U, E : V \rightarrow V \text{ are linear maps, and } \forall L \in \mathcal{L}, LD = EL\}$. The authors of [16] noted that if $\text{Adj}(\mathcal{L}, U, V)$ is block diagonalizable, i.e., $\forall (D, E) \in \text{Adj}(\mathcal{L}, U, V)$ if $D(U_i) \subseteq U_i$ and $E(V_i) \subseteq V_i$, then the decomposition is unique. So, we need to show that $\text{Adj}(\mathcal{L}_2, U, V)$ is block diagonalizable.
3. **Vector space decomposition:** With 1 and 2 satisfied, an algorithm is required to decompose U and V . As mentioned before, the vector space decomposition algorithm in [13] does not quite work over \mathbb{Q} . But fortunately, the adjoint algebra comes to the rescue again. Suppose, $\text{Adj}(\mathcal{L}_2, U, V)$ is block diagonalizable. If it is further block *equi-triangularizable* (refer Definition 23) and has an element (D, E) , where D has s distinct eigenvalues, then the U_i 's are the generalized eigenspaces of D which can be computed efficiently⁷. Thus, if $\text{Adj}(\mathcal{L}_2, U, V)$ is block equi-triangularizable, then computing vector space decomposition reduces to computing generalized eigenspaces.
4. **Recovery of T_i :** Finally, once $U_i = \langle \mathcal{L}_1 \circ T_i \rangle$ is obtained, we need to derive T_i from it.

⁶ The authors of [16] attributed the use of adjoint algebra in their work to a suggestion by Youming Qiao.

⁷ The existence of such an element implies that for a random $(D, E) \in \text{Adj}(\mathcal{L}_2, U, V)$, D has s distinct eigenvalues with high probability by the Schwartz-Zippel lemma.

Connecting our problems with the learning problem given by Equation (2). Let $g = \sum_{i \in [s]} c_i m_i$ be a design polynomial and $f = g(A\mathbf{x}) = \sum_{i \in [s]} T_i$, where $T_i = c_i m_i(A\mathbf{x})$ is a product of linear forms. Then, we can learn the unknown transformation A (up to permutation and scaling) by learning and factoring the terms T_1, \dots, T_s . We do so by implementing the above steps for Theorems 3 and 18; we discuss this next. We compare our work with relevant previous works in Section A.1.

1.2.1 Implementing the four steps for Theorem 3

We choose $\mathcal{L}_1 = \mathcal{L}_2 = \partial^t$. For a (n, d, s, t) design polynomial g with $d \geq 3t$, we show Step 1 in Lemma 25 by leveraging the design property, Step 2 by showing that the adjoint is block diagonalizable (Lemma 28), and Step 3 by showing further that the adjoint is block equi-triangularizable (Proposition 30). Since the input f is in the orbit of g , these properties also hold for f by Lemma 27. A term T_i of f is in the orbit of a monomial of g , and so, T_i is a product of linear forms. The recovery process for T_i from U_i is the same as that in [27]. The transform and the design polynomial are then obtained from the T_i 's (Proposition 24).

1.2.2 Implementing the four steps for Theorem 18

We choose $\mathcal{L}_1 = \partial^k$ and $\mathcal{L}_2 = \partial^2$, where $k > t$ is appropriately chosen in Lemma 45. First, we show that assuming the two non-degeneracy conditions stated in Section 4.1, all the steps can be implemented. Step 1 is immediate from the first non-degeneracy condition. Lemma 41 shows that $\text{Adj}(\partial^2, U, V)$ is block diagonalizable (in fact, block equi-triangularizable) for non-degenerate affine projections. Hence, both Steps 2 and 3 hold. The process of recovering the terms (i.e., Step 4) is the same as that for Theorem 3.

Second, we show that random affine projections of design polynomials are non-degenerate with high probability. The second non-degeneracy condition holds with high probability given the restriction on the field size. If we show that $\dim U = s \binom{d}{k}$, then the direct sum structure holds⁸. By the Schwartz-Zippel lemma, it is sufficient to show that for every design polynomial, there exists an affine projection such that $\dim U = s \binom{d}{k}$. We do this by showing that the probability that $\dim U = s \binom{d}{k}$ is non-zero if f is chosen from a specific class of affine projections as described by the *two-phase* random process in the proof of Lemma 45 (which can be found in [7]).

2 Preliminaries

2.1 Notations and definitions

For $n \in \mathbb{N}$, $[n]$ is the set $\{1, 2, \dots, n\}$. We use b.b.a to refer to black-box access. The set of $n \times n$ invertible matrices over \mathbb{F} is denoted as $\text{GL}(n, \mathbb{F})$. For two polynomials f and g , $f \sim g$ denotes that f is in the orbit of g . Variable sets are denoted as \mathbf{x}, \mathbf{y} and \mathbf{z} . Permutation and scaling matrices are denoted as P and S , respectively. A monomial in \mathbf{x} is denoted as $\mathbf{x}^\alpha := x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}$, which has total degree $|\alpha| := \sum_{i=1}^n \alpha_i$. The set of degree t derivatives in \mathbf{x} is denoted as ∂^t while $\partial^t f$ denotes the set of degree t derivatives of the polynomial f . The vector space spanned by a set of polynomials S over \mathbb{F} is denoted as $\langle S \rangle$. Typically,

⁸ as $U \subseteq U_1 + \dots + U_s$ and $\dim U_i \leq \binom{d}{k}$.

g denotes a (n, d, s, t) design polynomial $g = g_1 + g_2 + \dots + g_s$ where g_i are monomials of degree d , while f denotes the input polynomial $f = g(A\mathbf{x}) = T_1 + T_2 + \dots + T_s$, where $T_i = g_i(A\mathbf{x})$ for $A \in \text{GL}(n, \mathbb{F})$. Define the spaces $U, U_i, V, V_i, U', U'_i, V', V'_i$ as follows:

$$U := \langle \mathcal{L}_1 \circ f \rangle, \quad U' := \langle \mathcal{L}_1 \circ g \rangle, \quad V := \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ f \rangle, \quad V' := \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ g \rangle,$$

$$U_i := \langle \mathcal{L}_1 \circ T_i \rangle, \quad U'_i := \langle \mathcal{L}_1 \circ g_i \rangle, \quad V_i := \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ T_i \rangle, \quad V'_i := \langle \mathcal{L}_2 \circ \mathcal{L}_1 \circ g_i \rangle,$$

where $\mathcal{L}_1 = \mathcal{L}_2 = \partial^t$ (as defined in Section 1.2.1). These spaces will be used in Section 3. For the algorithmic preliminaries, refer Section A.2.

► **Definition 22** (Adjoint Algebra). *The adjoint algebra associated with (\mathcal{L}_2, U, V) is defined as $\text{Adj}(\mathcal{L}_2, U, V) := \{(D, E) \mid D : U \rightarrow U, E : V \rightarrow V \text{ are linear maps, and } \forall L \in \mathcal{L}_2, LD = EL\}$.*

Adjoint algebra was introduced in Section 4.3 of [40] as an associative ring to study the decompositions of bilinear maps and has been used in [11] for developing a fast isomorphism testing algorithm for a subclass of finite p -groups. A meta-framework for designing learning algorithms for arithmetic circuits was given in [16], where the learning problem was reduced to the vector space decomposition problem, and the uniqueness of vector space decomposition was proved by analyzing a certain adjoint algebra (refer [16] for details).

► **Definition 23** (Equi-triangular matrix). *An equi-triangular matrix is triangular with equal diagonal entries. Block equi-triangularizability and simultaneous block equi-triangularizability are defined similarly as block diagonalizability and simultaneous block diagonalizability.*

3 Equivalence testing for design polynomials

We state the ET algorithm in Section 3.1. The precise time complexity of the ET algorithm is $\text{poly}\left(\binom{n+t}{n}d^t\right)$ we discuss this further in Section 3.2 which analyses it. Section 3.3 analyses the adjoint algebra of a design polynomial g . Based on the structure of the adjoint, Section 3.4 develops and analyses a VSD algorithm. In Section 3.5, Theorems 11 and 13 are proven.

■ **Algorithm 2** Equivalence testing for design polynomials.

-
- **Input:** Black-box access to an $f \in \mathbb{F}[\mathbf{x}]$, where $f = g(A\mathbf{x})$ for some unknown (n, d, s, t) design polynomial g and $A \in \text{GL}(n, \mathbb{F})$.
 - **Output:** A matrix B and a (n, d, s, t) design polynomial h , where $B = PSA$ and $f = h(B\mathbf{x})$ for some permutation matrix P and scaling matrix S .
1. Compute black-box access to bases of $U = \langle \partial^t f \rangle$ and $V = \langle \partial^t U \rangle$.
 2. Perform VSD for (∂^t, U, V) using Algorithm 3. Let $U = U_1 \oplus \dots \oplus U_s$ be the decomposition returned by Algorithm 3.
 3. Express $\frac{\partial^t f}{\partial \mathbf{x}^\alpha} = u_{1\alpha} + \dots + u_{s\alpha}$, where $u_{i\alpha} \in U_i$ and obtain black-box access to $u_{i\alpha}$ for all $i \in [s]$ and \mathbf{x}^α of degree t . The black-box for T_i is given by $\frac{(d-t)!}{d!} \sum_{|\alpha|=t} \binom{t}{\alpha_1 \dots \alpha_n} \mathbf{x}^\alpha u_{i\alpha}(\mathbf{x})$.
 4. From black-box access to the T_i 's, recover and return a $B \in \text{GL}(n, \mathbb{F})$ and a (n, d, s, t) design polynomial h such that $f = h(B\mathbf{x})$ using Proposition 24.
-

3.1 The algorithm

1. **Step 1:** As discussed in Section 1.2, $\mathcal{L}_1 = \partial^t$ and $\mathcal{L}_2 = \partial^t$ are used to define U and V . Black-box access to bases of U and V is computable using Facts 46 and 47.
2. **Step 2:** The VSD algorithm of Section 3.4 gives b.b.a to bases of U_i 's.
3. **Step 3:** Each $\frac{\partial^t f}{\partial \mathbf{x}^\alpha}$ is expressible as a sum of $u_{i\alpha} \in U_i$'s, where each $u_{i\alpha}$ is $\frac{\partial^t T_i}{\partial \mathbf{x}^\alpha}$. Using Fact 48, b.b.a to $\frac{\partial^t f}{\partial \mathbf{x}^\alpha}$ and U_i 's, black-boxes to $u_{i\alpha}$ can be obtained. It can be verified using Lagrange's formula that the described black-box for T_i is the correct one.
4. **Step 4:** The proof of Proposition 24 details the recovery process for B and h .

► **Proposition 24.** *Assuming b.b.a to the T_i 's, B and h can be recovered in $\text{poly}(n, s, d)$ time.*

3.2 Analysis of the algorithm

Each step is randomized with a small probability of error. For the analysis, we assume that each step executes correctly. An implicit check is made at the end to see if h is a (n, d, s, t) design polynomial, B is invertible and $f = h(B\mathbf{x})$, failing which the algorithm is repeated.

The correctness of Algorithm 2 holds if it executes each of the four steps listed in Section 1.2 correctly. By Lemmas 25 and 27 (given below), U and V have the required direct sum structure. The correctness of the vector space decomposition follows from the correctness of Facts 46 and 47 and that of the vector space decomposition algorithm of Section 3.4. The uniqueness of decomposition follows from Proposition 30 and Lemmas 27 and 32. The correctness of the recovery of T_i 's, B and h follows from Fact 48 and Proposition 24. Let U', U'_i, V', V'_i be as defined in Section 2.1.

► **Lemma 25.** *For $d \geq 3t$, $U' = U'_1 \oplus U'_2 \cdots \oplus U'_s$ and $V' = V'_1 \oplus V'_2 \cdots \oplus V'_s$.*

The following proposition gives the time complexity of the algorithm.

► **Proposition 26.** *Algorithm 2 has a running time of $\text{poly}\left(\binom{n+t}{n}d^t\right)$.*

Typically $n > d$, in which case the running time is $\text{poly}(n^t)$.

3.3 Structure of the adjoint algebra

Once we fix bases of the spaces U, U', V and V' , a linear operator from one of these spaces to another can be naturally viewed as a matrix. Thus, $\text{Adj}(\partial^t, U, V)$ and $\text{Adj}(\partial^t, U', V')$ are sets of tuples of matrices with respect to appropriately chosen bases. We now show that $\text{Adj}(\partial^t, U, V)$ is block equi-triangularizable, i.e., the set of matrices $\{D : (D, E) \in \text{Adj}(\partial^t, U, V)\}$ is simultaneously block equi-triangularizable. By Lemma 27, it suffices to show this for $\text{Adj}(\partial^t, U', V')$. Note, after fixing bases of U' and V' , the operators $\partial^t : U' \rightarrow V'$ are also matrices. When we say $\text{Adj}(\partial^t, U, V)$ is equal to $\text{Adj}(\partial^t, U', V')$ in Lemma 27, we mean they are equal as sets of tuples of matrices with respect to appropriately chosen bases for U, V, U' and V' .

► **Lemma 27.** *Let $U, V, U_i, V_i, U', V', U'_i$ and V'_i be vector spaces as defined in Section 2.1 with $\mathcal{L}_1 = \mathcal{L}_2 = \partial^t$. Define the invertible map $\phi : \mathbb{F}[\mathbf{x}] \rightarrow \mathbb{F}[\mathbf{x}]$ as $\phi(p) := p(A\mathbf{x})$ for $p \in \mathbb{F}[\mathbf{x}]$, where $A \in \text{GL}(n, \mathbb{F})$ is as in Algorithm 2. Then,*

1. $U' \cong U$, $V' \cong V$, $U'_i \cong U_i$ and $V'_i \cong V_i$ via the map ϕ . In other words, if \mathcal{B} is a basis of U' , $\phi(\mathcal{B})$ is a basis of U . This holds similarly for the other spaces.
2. Let \mathcal{B}'_1 and \mathcal{B}'_2 be bases for U' and V' respectively, with $\phi(\mathcal{B}'_1)$ and $\phi(\mathcal{B}'_2)$ being basis of U and V respectively. Then, $\text{Adj}(\partial^t, U', V')$, with respect to bases \mathcal{B}'_1 and \mathcal{B}'_2 , is equal to $\text{Adj}(\partial^t, U, V)$, with respect to bases $\phi(\mathcal{B}'_1)$ and $\phi(\mathcal{B}'_2)$.

► **Lemma 28.** *If $(D', E') \in \text{Adj}(\partial^t, U', V')$, then $D'(U'_i) \subseteq U'_i$ and $E'(V'_i) \subseteq V'_i$, $\forall i \in [s]$.*

The direct sum structure of U' and V' implies that every $\partial^t : U' \rightarrow V'$ is block diagonal, with respect to the monomial basis of U' and V' , and by Lemma 28, $\text{Adj}(\partial^t, U', V')$ is block diagonal with respect to these bases. This and the adjoint condition imply that $\text{Adj}(\partial^t, U', V')$ comprises the adjoints of the g_i 's. Thus, it suffices to analyse the adjoint of the g_i 's. The adjoint of a monomial need not be trivial, as shown in Section add ref. We show that the adjoint algebra of a monomial is equi-triangularizable. For g_i , an arbitrary monomial of g , $\mathcal{B}'_i := \{\partial^t g_i\}$ is a basis⁹ for U'_i . For $(D', E') \in \text{Adj}(\partial^t, U', V')$, let D'_i and E'_i be the restriction of D' and E' to U'_i and V'_i respectively. Represent D'_i as a $\dim(U'_i) \times \dim(U'_i)$ matrix with respect to \mathcal{B}'_i , where $D'_i[\mathbf{x}^\alpha][\mathbf{x}^\beta]$ is the coefficient of $\frac{\partial^t g_i}{\partial \mathbf{x}^\alpha}$ in $D'_i\left(\frac{\partial^t g_i}{\partial \mathbf{x}^\beta}\right)$. Lemma 29 shows when $D'_i[\mathbf{x}^\alpha][\mathbf{x}^\beta]$ is 0 and that all entries $D'_i[\mathbf{x}^\alpha][\mathbf{x}^\alpha]$ are equal. We use the notation $\text{mon}(c \cdot \mathbf{x}^\alpha) := \mathbf{x}^\alpha$, where $c \in \mathbb{F} \setminus \{0\}$; the definition is naturally extended to a set of monomials. For e.g., $\text{mon}\{2x_1^4, 5x_1x_2\} = \{x_1^4, x_1x_2\}$.

► **Lemma 29.** *Let $|\alpha| = |\beta| = t$, $\frac{\partial^t g_i}{\partial \mathbf{x}^\alpha} \neq 0$ and $\frac{\partial^t g_i}{\partial \mathbf{x}^\beta} \neq 0$. Let D'_i be as above. Then,*

1. $D'_i[\mathbf{x}^\alpha][\mathbf{x}^\beta] = 0$, if $\text{mon}\{\partial^t(\frac{\partial^t g_i}{\partial \mathbf{x}^\alpha})\} \not\subseteq \text{mon}\{\partial^t(\frac{\partial^t g_i}{\partial \mathbf{x}^\beta})\}$, and
2. $D'_i[\mathbf{x}^\alpha][\mathbf{x}^\alpha] = D'_i[\mathbf{x}^\beta][\mathbf{x}^\beta]$.

The following proposition shows that $\text{Adj}(\partial^t, U', V')$ is block equi-triangularizable.

► **Proposition 30.** *A basis \mathcal{B}' for U' exists with respect to which any D' , where $(D', E') \in \text{Adj}(\partial^t, U', V')$ for some E' , is block equi-triangular.*

As detailed in the proof of Proposition 30, reordering each \mathcal{B}'_i and concatenating them gives \mathcal{B}' . For each \mathcal{B}'_i , a directed acyclic graph G_i is constructed with vertices as \mathcal{B}'_i . For $\frac{\partial^t g_i}{\partial \mathbf{x}^\alpha}, \frac{\partial^t g_i}{\partial \mathbf{x}^\beta} \in \mathcal{B}'_i$, if $\text{mon}\{\partial^t(\frac{\partial^t g_i}{\partial \mathbf{x}^\alpha})\} \subseteq \text{mon}\{\partial^t(\frac{\partial^t g_i}{\partial \mathbf{x}^\beta})\}$ then an edge from $\frac{\partial^t g_i}{\partial \mathbf{x}^\alpha}$ to $\frac{\partial^t g_i}{\partial \mathbf{x}^\beta}$ exists in G_i . The topological sort of G_i gives the reordering of \mathcal{B}'_i . By Lemma 29, D'_i is equi-triangular with respect to the reordered \mathcal{B}'_i implying D' is block equi-triangular with respect to \mathcal{B}' .

3.4 Vector space decomposition

■ **Algorithm 3** Vector space decomposition algorithm.

Input: B.b.a to bases of spaces U and V .

Output: B.b.a to bases of spaces W_1, \dots, W_s , where $W_i = U_{\pi(i)}$ for some permutation π .

1. Compute a basis $D^{(1)}, \dots, D^{(b)}$ of $\text{Adj}(\partial^t, U, V)_1 := \{D \mid (D, E) \in \text{Adj}(\partial^t, U, V)\}$.
2. Pick $c_1, \dots, c_b \in_r S \subseteq \mathbb{F}$, where $|S| = s^3$. Let $D := c_1 D^{(1)} + \dots + c_b D^{(b)}$.
3. Factorize the characteristic polynomial of D and obtain its eigenvalues $\lambda_1, \dots, \lambda_s$.
4. Compute $W_i := \text{Ker}((D - \lambda_i I)^{\dim U})$ for all $i \in [s]$ and output b.b.a to bases of W_1, \dots, W_s .

Step 1 is executed by solving the linear system arising from $LD = EL$, for $L \in \mathcal{L}_2$, with the entries of D and E as the variables. In Step 2, a random linear combination of the $D^{(i)}$'s gives a random operator D . The eigenvalues of D can be found by factorizing the

⁹ Assume that the set $\{\partial^t g_i\}$ consists of only the nonzero derivatives.

characteristic polynomial.¹⁰ Step 4 involves the computation of the generalized eigenspaces of D . Note, Lemma 32 proves the uniqueness of decomposition and the indecomposability of U_i and V_i with respect to ∂^t .

► **Lemma 31.** D has s distinct eigenvalues with probability $\geq 1 - \frac{\binom{s}{2}}{|S|}$.

► **Lemma 32.** Let $D \in \text{Adj}(\partial^t, U, V)$ such that it has s distinct eigenvalues $\lambda_1, \dots, \lambda_s$. Then, $\text{Ker}((D - \lambda_i I)^{\dim U}) = U_{\pi(i)}$ for all $i \in [s]$, and for some permutation π on $[s]$.

► **Proposition 33.** Algorithm 3 has a running time of $\text{poly}\left(\binom{n+t}{n} d^t\right)$.

Handling non-homogeneous polynomials and translations. For non-homogeneous (n, d, s, t) design polynomials, if the degree of *all* monomials is $\geq 3t$, Lemmas 25, 28 and Proposition 30 hold. When $f(\mathbf{x}) = g(A\mathbf{x} + \mathbf{b})$ for $\mathbf{b} \in \mathbb{F}^n$, then since this transform is also invertible, a lemma similar to Lemma 27 holds. The analysis then proceeds in the same way. Proposition 24 can also recover translations. For multilinear (n, d, s, t) design polynomials, $\mathcal{L}_2 = \partial^1$ with the adjoint $\text{Adj}(\partial^1, U, V)$ improves the bound on t to $d \geq 2t + 1$. Lemmas 25, 28 and 29 hold with some minor changes, and $\text{Adj}(\partial^1, U', V')$ can be shown to be trivial.

3.5 Applications of the equivalence test

3.5.1 $GI \equiv_p PE$ for design polynomials: Proof of Theorem 11

$GI \leq_p PE$ for design polynomials. Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two n -vertex simple graphs with e edges each. Let there be an arbitrary ordering on the edges of both graphs with \mathbf{I}_1 an index function mapping E_1 to $[e]$ and \mathbf{I}_2 similarly mapping E_2 to $[e]$. Introduce variables x_1, \dots, x_n and y_1, \dots, y_e . Construct

$$M_1 := \{x_i x_j y_{\mathbf{I}_1(i,j)}^4 : (i, j) \in E_1\} \text{ and } M_2 := \{x_l x_k y_{\mathbf{I}_2(l,k)}^4 : (l, k) \in E_2\}.$$

Let $h_1(\mathbf{z}) := \sum_{m \in M_1} m$ and $h_2(\mathbf{z}) := \sum_{m \in M_2} m$, where $\mathbf{z} = \mathbf{x} \sqcup \mathbf{y}$. Both h_1 and h_2 are $(n + e, 6, e, 2)$ design polynomials with all-one coefficients constructible in $\text{poly}(n)$ time. Proposition 34 follows from Corollary 10, the coefficients being 1, and \mathbf{y} variables having degree 4.

► **Proposition 34.** For h_1, h_2 as above, $G_1 \cong G_2 \iff h_1 \sim h_2$.

PE for design polynomials $\leq_p GI$. Let $h_1(\mathbf{x})$ and $h_2(\mathbf{x})$ be (n, d, s, t) design polynomials with all-one coefficients, satisfying $d \geq 3t$. If h_1 and h_2 are multilinear, construct hypergraphs H_1 and H_2 with \mathbf{x} as the vertices and subsets of vertices corresponding to the monomials of h_1 and h_2 as the hyperedges, respectively. Observe that $h_1 \sim h_2$ iff $H_1 \cong H_2$ as, by Corollary 10, if $h_1 \sim h_2$, then they are equivalent via a permutation matrix. It is well-known that hypergraph isomorphism reduces to GI (refer [32]). If h_1 and h_2 are non-multilinear, the argument is more elaborate: Now the monomials correspond to multisets of \mathbf{x} , while hyperedges need to be subsets of vertices. This can be handled by examining a standard reduction from hypergraph isomorphism to GI that uses bipartite graphs (see the opening paragraph of [5]). By introducing in-between vertices to handle parallel edges, graphs G_1 and G_2 can be constructed in $\text{poly}(s)$ time. For details, refer to the proof of Proposition 35 in [7].

► **Proposition 35.** For graphs G_1, G_2 as above, $G_1 \cong G_2 \iff h_1 \sim h_2$.

¹⁰Here, we need an efficient univariate polynomial factorization algorithm over \mathbb{F} .

3.5.2 Cryptanalysis of Patarin's scheme: Proof of Theorem 13

Patarin's authentication scheme [34], described in Section A.3, is based on the presumed hardness of PE for random polynomials of constant degree. The attack on the scheme is as:

1. **Invoking Theorem 3:** Invoke Theorem 3 on f_1 and f_2 to obtain $h_1 \sim f_1$ and $h_2 \sim f_2$.
2. **Recovering P:** Use Theorem 11 to construct graphs G_1 and G_2 corresponding to h_1 and h_2 and use Babai's algorithm [6] for GI to recover a permutation matrix P .
3. **Recovering S:** Solve the system of monomial equations arising from $h_2(\mathbf{x}) = h_1(P\mathbf{S}\mathbf{x})$, with the entries of S as variables.

The attack relies on Lemma 36, Propositions 37, 38, Theorems 3, 11 and Corollary 10.

► **Lemma 36.** *A random s -sparse polynomial, as per Definition 12, is a (n, d, s, t) design polynomial with probability at least $1 - \epsilon$, if $n > d^2$ and $s \leq \sqrt{\epsilon} \left(\frac{n}{d^2}\right)^{\frac{1}{2}}$ where $0 < \epsilon < 1$.*

► **Proposition 37.** *If f is a random s -sparse polynomial as per Lemma 36 with $s \geq n^3$, $n > d^8$, $d \geq 25$, $t = \frac{d}{3}$ and $\epsilon = 0.01$, f has no non-trivial permutation symmetry with high probability.¹¹*

► **Proposition 38.** *Let f be as in Proposition 37. For $h_1(\mathbf{x}) = f(P_1 S_1 \mathbf{x})$ and $h_2(\mathbf{x}) = f(P_2 S_2 \mathbf{x})$, where P_1, P_2 are permutation matrices and S_1, S_2 are scaling matrices, there exists a unique permutation matrix P such that $h_2(\mathbf{x}) = h_1(P\mathbf{S}\mathbf{x})$ for some scaling matrix S .*

Invoking Theorem 3. For $n > d^8$, $\epsilon = 0.01$, $t = d/3$, $d \geq 25$ and $n^3 \leq s \leq 0.1 \left(\frac{n}{d^2}\right)^{\frac{d}{6}}$, Lemma 36 implies f is, with high probability, a $(n, d, s, d/3)$ design polynomial. Thus, invoking Theorem 3 on f_1 and f_2 gives $h_1, h_2, P_1 S_1 A_1$ and $P_2 S_2 A_2$ where $f_1 = h_1(P_1 S_1 A_1 \mathbf{x})$ and $f_2 = h_2(P_2 S_2 A_2 \mathbf{x})$. Clearly, $h_1 \sim h_2$ by the transform $P_1 S_1 (P_2 S_2)^{-1} = PS$ for appropriate P and S . If P and S can be recovered, then $A_1^{-1} A_2 = (P_1 S_1 A_1)^{-1} PS (P_2 S_2 A_2)$ can be recovered. As $n > d^8$ and d is a constant, this step requires $\text{poly}(n)$ time.

Recovering P. Note that P maps the monomials of h_1 to h_2 while S scales the coefficients accordingly. To recover P , treat h_1 and h_2 as design polynomials with all-one coefficients and use Theorem 11 and the GI algorithm of [6]. This step can be done in quasi-poly(s) time, and the uniqueness of P , which holds by Propositions 38 and 37, implies the correctness.

Recovering S. Let $h_1(\mathbf{x}) = \sum_{i=1}^s c_i m_i$ and $h_2(\mathbf{x}) = \sum_{i=1}^s \tilde{c}_i m_i$. Now $h_2(\mathbf{x}) = h_1(P\mathbf{S}\mathbf{x}) = h_1(S'\mathbf{P}\mathbf{x})$ for an appropriate scaling matrix S' . Treat the diagonal entries of S' as variables $\{z_1, z_2 \dots z_n\}$. Equating the coefficients of the monomials, we get $c_i m_i(z_1, \dots, z_n) = \tilde{c}_j$ where $m_j = m_i(P\mathbf{x})$. If $m_i = x_1^{\alpha_{i,1}} x_2^{\alpha_{i,2}} \dots x_n^{\alpha_{i,n}}$, we get the following monomial equations:

$$z_1^{\alpha_{i,1}} z_2^{\alpha_{i,2}} \dots z_n^{\alpha_{i,n}} = \tilde{c}_j c_i^{-1} \quad \forall i \in [s]. \quad (3)$$

There are s such equations in n variables, which is converted to a system of linear equations by taking $\log(z_j)$ as variables and $\alpha_{i,j}$ and $\log(\tilde{c}_j c_i^{-1})$ as constants. Computing $\log(a)$ over \mathbb{F}_q is finding the discrete logarithm of a with respect to a generator γ of \mathbb{F}_q^\times .¹² Since $q = O(\text{poly}(n))$, γ can be found and discrete log can be computed in $O(\text{poly}(n))$ time. We get a system of s linear equations over \mathbb{Z}_{q-1} which can be solved in $\text{poly}(s, q)$ time using the Chinese Remainder Theorem, refer Chapter 5 of [39] for details.

¹¹ meaning, for any permutation matrix P , $f(P\mathbf{x}) = f(\mathbf{x})$ implies P is the identity matrix.

¹² that is finding a $b \in [0, q-2]$ such that $\gamma^b = a$.

4 Learning random affine projections of design polynomials

In this section, Theorem 18 is proven. Section 4.1 lists the non-degeneracy conditions imposed on affine projections of design polynomials. In Sections 4.2 and 4.4, we state and analyse the learning algorithm and the vector space decomposition algorithm, respectively. The adjoint of non-degenerate affine projections is analysed in Section 4.3. In Section 4.5, we show random affine projections of multilinear design polynomials are non-degenerate with high probability.

4.1 Non-degeneracy conditions

Let $g(\mathbf{y}) = g_1 + \dots + g_s$ be a (m, d, s, t) multilinear design polynomial with g_i 's as monomials. Let $f(\mathbf{x}) = g(l_1, l_2 \dots l_m) = T_1 + \dots + T_s$ be an n -variate affine projection of g with $T_i = g_i(l_1, \dots, l_m)$, a product of d linear forms and L_i be the set of linear forms in T_i . We say f is a random affine projection if the coefficients of the l_i 's are randomly chosen from \mathbb{F} . Define:

$$U := \langle \partial^k f \rangle, \quad V := \langle \partial^{k+2} f \rangle, \quad U_i := \langle \partial^k T_i \rangle, \quad V_i := \langle \partial^{k+2} T_i \rangle,$$

where k is as in Lemma 45 and $d \geq 2k + 2$. We say f is non-degenerate if the following holds:

1. $U = U_1 \oplus \dots \oplus U_s$ and $V = V_1 \oplus \dots \oplus V_s$.
2. The set L_i is \mathbb{F} -linearly independent for all $i \in [s]$.

4.2 The algorithm and its analysis

Algorithm 4 is similar to Algorithm 2, except $\mathcal{L}_1 = \partial^k$ and $\mathcal{L}_2 = \partial^2$ define U and V respectively and Algorithm 5 for vector space decomposition computes eigenspaces. Each step of the algorithm is randomized with a small error probability. An implicit check is run at the end of Step 4 to see if h is a (m, d, s, t) multilinear design polynomial and the linear forms per T_i are linearly independent, failing which the algorithm is repeated.

Analysis and Time complexity. We assume that each step executes without error and f is non-degenerate, which holds for a random affine projection with high probability by Lemma 45. The correctness of Algorithm 4 holds if it executes each of the four steps listed in Section 1.2 correctly. Non-degeneracy condition 1 implies the direct sum structure of U and V . The correctness of VSD follows from that of Facts 46, 47 and Algorithm 5. The uniqueness of decomposition follows from Lemmas 41 and 43. The correctness of the recovery of T_i 's, B and h follows from Fact 48 and Proposition 39. Proposition 40 gives the time complexity.

► **Proposition 39.** *Assuming b.b.a to T_i 's, B and h are recoverable in $\text{poly}(m, n, s, d)$ time.*

► **Proposition 40.** *Algorithm 4 has a running time of $\text{poly}(m, s, (nd)^t)$.*

4.3 Structure of the adjoint algebra

Lemma 41 states that $\text{Adj}(\partial^2, U, V)$ is trivial¹³ for a non-degenerate f . The idea is to leverage that each T_i is an affine projection of a multilinear monomial. By condition 2, there exists an $A_i \in \text{GL}(n, \mathbb{F})$, such that in $f(A_i \mathbf{x})$, $T_i(A_i \mathbf{x})$ is a multilinear monomial. By Lemma 27, the

¹³This is a special case of being block equi-triangularizable.

■ **Algorithm 4** Learning random affine projections of multilinear design polynomials.

-
- **Input:** B.b.a to $f = g(A\mathbf{x})$, where $A \in \mathbb{F}^{m \times n}$ and g is some (m, d, s, t) polynomial.
 - **Output:** A matrix B and (m, d, s, t) design polynomial h , with $B = PSA$ and $f = h(B\mathbf{x})$.
1. Compute black-box access to some bases of $U = \langle \partial^k f \rangle$ and $V = \langle \partial^2 U \rangle$.
 2. Perform VSD for (∂^2, U, V) : let $U = U_1 \oplus \cdots \oplus U_s$.
 3. Express $\frac{\partial^k f}{\partial \mathbf{x}^\alpha} = u_{1\alpha} + \cdots + u_{s\alpha}$, where $u_{i\alpha} \in U_i$ and obtain black-box access to $u_{i\alpha}$ for all $i \in [s]$ and \mathbf{x}^α of degree k . The black box for T_i is $\frac{(d-k)!}{d!} \sum_{|\alpha|=k} \binom{k}{\alpha_1 \dots \alpha_n} \mathbf{x}^\alpha u_{i\alpha}(\mathbf{x})$.
 4. From black box access to the T_i 's, recover and return $B \in \mathbb{F}^{m \times n}$ and (m, d, s, t) design polynomial h using Proposition 39.
-

adjoint of $f(A_i \mathbf{x})$ and that of $f(\mathbf{x})$ are equal as sets of matrices with respect to appropriate bases of their respective derivative spaces.¹⁴ The operators in the adjoint of $f(A_i \mathbf{x})$ are shown to be invariant on the derivative spaces of $T_i(A_i \mathbf{x})$ by using the fact that the derivative space of $T_i(A_i \mathbf{x})$ is the space of multilinear polynomials in d variables. The block diagonality then holds. Because of the direct sum structure, the block diagonality of the adjoint and the block diagonality of ∂^2 operators, it suffices to analyse the adjoint of individual T_i 's. Since $T_i(A_i \mathbf{x})$ is a multilinear monomial, its adjoint is trivial and thus so is the adjoint of T_i .

► **Lemma 41.** *Let g and f be as defined in Section 4.1. If f is non-degenerate, then $\text{Adj}(\partial^2, U, V)$ is block-diagonal and is also trivial (thus, also block equi-triangular).*

4.4 Vector space decomposition algorithm

■ **Algorithm 5** Vector Space Decomposition Algorithm.

Input: B.b.a to bases of spaces U and V .

Output: B.b.a to bases of W_1, \dots, W_s where $W_i = U_{\pi(i)}$ for some permutation π .

1. Compute a basis $D^{(1)}, \dots, D^{(b)}$ of $\text{Adj}(\partial^2, U, V)_1 = \{D \mid (D, E) \in \text{Adj}(\partial^2, U, V)\}$.
 2. Select $c_1, \dots, c_b \in_r S \subseteq \mathbb{F}$, $|S| = s^3$ and let $D = c_1 D^{(1)} + \cdots + c_b D^{(b)}$.
 3. Factorize the characteristic polynomial of D to obtain eigenvalues $\lambda_1, \dots, \lambda_s$.
 4. Compute $W_i := \text{Ker}(D - \lambda_i I)$ for all $i \in [s]$ and output b.b.a to the bases of W_1, \dots, W_s .
-

Analysis and Time complexity. Algorithm 5 is similar to Algorithm 3 except it uses ∂^2 operators and computes the eigenspaces of D , instead of generalized eigenspaces. Thus, the analysis for Algorithm 5 is the same as for Algorithm 3. Lemmas 41 and 42 prove that Algorithm 5 works with high probability. Proposition 44 gives the time complexity.

► **Lemma 42.** *D has s distinct eigenvalues with probability $\geq 1 - \frac{\binom{s}{2}}{|S|}$.*

► **Lemma 43.** *Let $D \in \text{Adj}(\partial^k, U, V)$ such that it has s distinct eigenvalues denoted $\lambda_1, \dots, \lambda_s$. Then $\text{Ker}(D - \lambda_i I) = U_{\pi(i)}$ for all $i \in [s]$, where π is some permutation on $[s]$.*

► **Proposition 44.** *Algorithm 5 has a running time of $\text{poly}(n, s, d^t)$.*

¹⁴Lemma 27 holds more generally for any two polynomials equivalent by invertible linear transforms.

Handling non-homogeneous polynomials and translation. The non-degeneracy conditions are the same for non-homogeneous (m, d, s, t) multilinear design polynomials and when $f(\mathbf{x}) = g(A\mathbf{x} + \mathbf{b})$ for $A \in_r \mathbb{F}^{m \times n}$ and $\mathbf{b} \in \mathbb{F}^m$. For the non-homogeneous case, if the degree of *all* monomials is $\geq 2k + 2$, then Lemmas 41 and 45 hold, and the analysis proceeds in the same way. Proposition 39 can also recover translations.

4.5 Random affine projections are non-degenerate

Lemma 45 states that a random affine projection f is non-degenerate with high probability. Showing f is non-degenerate reduces to showing certain matrices, with entries as the coefficients of the l_i 's, are full rank with high probability when the entries are chosen randomly. The main technical challenge is in proving condition 1 because the g_i 's share variables, thus T_i 's share the l_i 's. A two-stage random process is used to show the existence of an affine projection which satisfies condition 1. The Schwartz-Zippel lemma then implies that a random affine projection also satisfies condition 1 with high probability.

► **Lemma 45.** *Let g be a polynomial as defined in Section 4.1 and f be its random affine projection. For $k = t + \left\lfloor \frac{2 \log(s)}{\log\left(\frac{\sqrt{n}}{d^2}\right)} \right\rfloor + 1$, f is non-degenerate with high probability.*

5 Conclusion

In this work, we design an ET algorithm for general design polynomials (Theorem 3) and a learning algorithm for random affine projections of multilinear design polynomials (Theorem 18). As an application of the ET algorithm, we show that GI is polynomial-time equivalent to PE for design polynomials with all-one coefficients (Theorem 11). As another application, we show that Patarin's authentication scheme can be broken if it uses a higher degree sparse polynomial for key generation (Theorem 13). We also give an ET algorithm for the NW design polynomial using Theorem 3 (Theorem 50). Theorem 18 is a significant generalization of the main result in [27] that gave a learning algorithm for random affine projections of the sum-product polynomial, which is a special multilinear design polynomial.

Both the algorithms are based on the vector space decomposition framework of [27, 16]. This work's main technical contributions include analysing a non-trivial adjoint algebra associated with design polynomials and developing a VSD algorithm based on generalized eigenspaces. We end by listing some related questions:

1. **ET for sparse polynomial:** What is the complexity of ET for the class of sparse polynomials? That is, given black-box access to a polynomial f and a parameter s , what is the complexity of testing whether f is in the orbit of some s -sparse polynomial? The authors of [12] showed that the shift equivalence problem for sparse polynomials (i.e., when $f = g(\mathbf{x} + \mathbf{b})$ for some sparse polynomial g and $\mathbf{b} \in \mathbb{F}^n$) is undecidable over \mathbb{Z} .
2. **Weakening $n \geq d^{4+\epsilon}$:** Can the condition $n \geq d^{4+\epsilon}$ in Theorem 18 be changed to $n \geq d^\delta$ for arbitrary $\delta > 0$? Doing so would give stronger evidence that NW is not VNP-complete.
3. **Efficient ET for NW:** Theorem 50 gives an ET for NW, but it is not a polynomial-time algorithm. Is there a polynomial-time ET algorithm for NW? Our ET algorithm is for general design polynomials; it is possible that analyzing the properties of the NW polynomial may yield an efficient ET algorithm specifically for NW.

References

- 1 Scott Aaronson. Arithmetic natural proofs theory is sought. <http://www.scottaaronson.com/blog/?p=336>, 2008.
- 2 Manindra Agrawal and Nitin Saxena. Automorphisms of finite rings and applications to complexity of problems. In *Proceedings of the 22nd Annual Conference on Theoretical Aspects of Computer Science, STACS'05*, pages 1–17, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/978-3-540-31856-9_1.
- 3 Manindra Agrawal and Nitin Saxena. Equivalence of F-algebras and cubic forms. In *Proceedings of the 23rd Annual Conference on Theoretical Aspects of Computer Science, STACS'06*, pages 115–126, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11672142_8.
- 4 Manuel Araújo. Classification of quadratic forms. <https://www.math.tecnico.ulisboa.pt/~ggranja/manuel.pdf>, 2011.
- 5 Vikraman Arvind, Bireswar Das, Johannes Köbler, and Seinosuke Toda. Colored Hypergraph Isomorphism is Fixed Parameter Tractable. *Algorithmica*, 71(1):120–138, 2015. Conference version appeared in the proceedings of FSTTCS 2010. doi:10.1007/s00453-013-9787-y.
- 6 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.
- 7 Omkar Baraskar, Agrim Dewan, and Chandan Saha. Testing equivalence to design polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, 2024. URL: <https://eccc.weizmann.ac.il/report/2024/004/>.
- 8 Elwyn R. Berlekamp. Factoring polynomials over large finite fields. In Stanley R. Petrick, Jean E. Sammet, Robert G. Tobey, and Joel Moses, editors, *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, SYMSAC 1971, Los Angeles, California, USA, March 23-25, 1971*, page 223. ACM, 1971. doi:10.1145/800204.806290.
- 9 Vishwas Bhargava, Ankit Garg, Neeraj Kayal, and Chandan Saha. Learning generalized depth three arithmetic circuits in the non-degenerate case. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPICs*, pages 21:1–21:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.APPROX/RANDOM.2022.21.
- 10 Markus Bläser, B. V. Raghavendra Rao, and Jayalal Sarma. Testing polynomial equivalence by scaling matrices. In Ralf Klasing and Marc Zeitoun, editors, *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, volume 10472 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2017. doi:10.1007/978-3-662-55751-8_10.
- 11 Peter A. Brooksbank, Joshua Maglione, and James B. Wilson. A fast isomorphism test for groups whose lie algebra has genus 2. *Journal of Algebra*, Volume 473, Pages 545-590, ISSN 0021-8693, 2017. doi:10.1016/j.jalgebra.2016.12.007.
- 12 Suryajith Chillara, Coral Grichener, and Amir Shpilka. On hardness of testing equivalence to sparse polynomials under shifts. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPICs*, pages 22:1–22:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.STACS.2023.22.
- 13 Alexander Chistov, Gábor Ivanyos, and Marek Karpinski. Polynomial time algorithms for modules over finite dimensional algebras. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, ISSAC '97*, pages 68–74, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258726.258751.

- 14 Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.
- 15 Ankit Garg, Nikhil Gupta, Neeraj Kayal, and Chandan Saha. Determinant equivalence test over finite fields and over \mathbb{Q} . In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 62:1–62:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.62.
- 16 Ankit Garg, Neeraj Kayal, and Chandan Saha. Learning sums of powers of low-degree polynomials in the non-degenerate case. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 889–899. IEEE, 2020. doi:10.1109/FOCS46700.2020.00087.
- 17 Joshua A. Grochow. *Symmetry and equivalence relations in classical and geometric complexity theory*. PhD thesis, University of Chicago, Chicago, IL, 2012. URL: <https://home.cs.colorado.edu/~jgrochow/grochow-thesis.pdf>.
- 18 Joshua A. Grochow and Youming Qiao. On the complexity of isomorphism problems for tensors, groups, and polynomials I: tensor isomorphism-completeness. *SIAM J. Comput.*, 52(2):568–617, 2023. Conference version appeared in the proceedings of ITCS 2021. doi:10.1137/21M1441110.
- 19 Joshua A. Grochow and Youming Qiao. On the complexity of isomorphism problems for tensors, groups, and polynomials IV: linear-length reductions and their applications. *CoRR*, abs/2306.16317, 2023. doi:10.48550/arXiv.2306.16317.
- 20 Nikhil Gupta and Chandan Saha. On the symmetries of and equivalence test for design polynomials. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPICs*, pages 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.MFCS.2019.53.
- 21 Nikhil Gupta, Chandan Saha, and Bhargav Thankey. Equivalence test for read-once arithmetic formulas. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 4205–4272. SIAM, 2023. doi:10.1137/1.9781611977554.ch162.
- 22 Erich L. Kaltofen and Barry M. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Comput.*, 9(3):301–320, 1990. Conference version appeared in the proceedings of FOCS 1988. doi:10.1016/S0747-7171(08)80015-6.
- 23 Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1409–1421. SIAM, 2011. doi:10.1137/1.9781611973082.108.
- 24 Neeraj Kayal. Affine projections of polynomials: extended abstract. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 643–662. ACM, 2012. doi:10.1145/2213977.2214036.
- 25 Neeraj Kayal, Vineet Nair, and Chandan Saha. Average-case linear matrix factorization and reconstruction of low width algebraic branching programs. *Comput. Complex.*, 28(4):749–828, 2019. doi:10.1007/S00037-019-00189-0.
- 26 Neeraj Kayal, Vineet Nair, Chandan Saha, and Sébastien Tavenas. Reconstruction of full rank algebraic branching programs. *ACM Trans. Comput. Theory*, 11(1):2:1–2:56, 2019. Conference version appeared in the proceedings of CCC 2017. doi:10.1145/3282427.
- 27 Neeraj Kayal and Chandan Saha. Reconstruction of non-degenerate homogeneous depth three circuits. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 413–424. ACM, 2019. doi:10.1145/3313276.3316360.
- 28 Krull-Schmidt Theorem. <https://mathstrek.blog/2015/01/17/krull-schmidt-theorem/>.

- 29 T. Y. Lam. *Introduction To Quadratic Forms Over Fields*. American Mathematical Society, 2004.
- 30 Arjen K Lenstra, Hendrik W Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. doi:10.1007/BF01457454.
- 31 Dori Medini and Amir Shpilka. Hitting sets and reconstruction for dense orbits in $\text{vp}_{\{e\}}$ and $\Sigma\Pi\Sigma$ circuits. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPICs*, pages 19:1–19:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CCC.2021.19.
- 32 Gary L. Miller. Graph isomorphism, general remarks. *J. Comput. Syst. Sci.*, 18(2):128–142, 1979. doi:10.1016/0022-0000(79)90043-6.
- 33 Janaky Murthy, Vineet Nair, and Chandan Saha. Randomized polynomial-time equivalence between determinant and trace-imm equivalence tests. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPICs*, pages 72:1–72:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.MFCS.2020.72.
- 34 Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 1996. doi:10.1007/3-540-68339-9_4.
- 35 Nitin Saxena. *Morphisms of rings and applications to complexity*. PhD thesis, Indian Institute of Technology, Kanpur, 2006. URL: https://www.cse.iitk.ac.in/users/manindra/Students/thesis_saxena.pdf.
- 36 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.
- 37 Thomas Thierauf. The isomorphism problem for read-once branching programs and arithmetic circuits. *Chic. J. Theor. Comput. Sci.*, 1998, 1998. URL: <http://cjtcs.cs.uchicago.edu/articles/1998/1/contents.html>.
- 38 Leslie G. Valiant. Completeness classes in algebra. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979. doi:10.1145/800135.804419.
- 39 Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra (2. ed.)*. Cambridge University Press, 2003.
- 40 James B. Wilson. Decomposing p-groups via Jordan algebras. *Journal of Algebra, Volume 322, Issue 8, Pages 2642-2679, ISSN 0021-8693*, 2009. doi:10.1016/j.jalgebra.2009.07.029.
- 41 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposium on Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, pages 216–226, 1979. doi:10.1007/3-540-09519-5_73.

A Appendix

A.1 Comparison with previous work

As mentioned earlier, ET has been studied for various polynomial families. ET algorithms for power symmetric polynomials [23] and read-once formulas [21] were given by analyzing the factors of the Hessian determinant. Analyzing the Lie algebra of the determinant [24, 15], the permanent [24], and the iterated matrix multiplication [26, 33] polynomials led to ET algorithms for these families. For the elementary symmetric polynomials, the maximal

dimension of the space of second-order partials gave an ET algorithm [23]. It was shown in [17] that over algebraically closed fields of characteristic 0, ET for polynomials characterized by the continuous part of their symmetries is equivalent to testing matrix isomorphism to their Lie algebras, implying over such fields an efficient ET exists for such polynomials, provided matrix isomorphism to their Lie algebra can be solved efficiently. However, for design polynomials, these techniques *do not* work. For example, the $(2, d, 2, 3)$ design polynomial $x_1 x_2^{d-1} + x_1^{d-1} x_2$, where $d \geq 3$ is odd, has a trivial Lie algebra and an irreducible Hessian determinant over \mathbb{Q} . For $d \geq 3$, the $(2d, d, 2, 1)$ design polynomial $\prod_{i=1}^d x_i + \prod_{i=d+1}^{2d} x_i$ has second-order partials dimension $2\binom{d}{2}$, which is less than the maximum possible dimension $\binom{2d}{2}$. Design polynomials, particularly NW, are *not* characterized by the continuous part of their symmetries (see [20]); hence the ET algorithm implied from [17] does not apply. Thus, a different technique must be used for design polynomials.

The VSD framework was used previously in [27, 16, 9] to design learning algorithms. The authors of [31] showed that polynomials in the orbit of the sum-product polynomial satisfy the non-degeneracy conditions of [27], and so, we have an ET algorithm for this family. But note that a sum-product polynomial has a read-once formula; an ET algorithm can also be obtained via the Hessian determinant [21]. To our knowledge, our work is the first to use the VSD framework to develop an ET algorithm (in Theorem 3) for a natural family of polynomials for which none of the other three techniques work. Also, there is a notable difference between the analysis in [16, 9] and the analysis here. In [16, 9], the relevant adjoint algebra is diagonalizable, while in our case the adjoint is block equi-triangularizable. The VSD algorithms of previous works recovered the component spaces by computing eigenspaces, while we compute generalized eigenspaces to do so.

We learn affine projections of design polynomials (in Theorem 18) under certain non-degeneracy conditions similar to those of [27]. However, proving the non-degeneracy of random affine projections of design polynomials requires a more involved analysis than proving the non-degeneracy of random depth-3 circuits in [27]. The reason is, unlike the terms of the circuit models studied in [27, 16, 9], the terms in our case have *shared randomness* as the same random affine form can appear in multiple terms. Theorem 18 is a significant generalization of the main result in [27] that handles random affine projections of the sum-product polynomial, which is a special design polynomial.

The learning algorithm in Theorem 18 is proper as it outputs an appropriate affine map. It is also an average-case algorithm for learning affine projections of design polynomials as the input is a random affine projection. This average-case, proper learning algorithm exploits the property that the space of partial derivatives of an affine projection of a design polynomial is *low* dimensional (under the technical conditions mentioned in the theorem statement) to reduce the learning problem to VSD. A natural question arises at this point: is it always possible to design an average-case, proper learning algorithm (via a reduction to VSD) for affine projections of a model satisfying such low dimensional partial derivatives space property? The answer is unclear. The related version [7] gives an example of affine projections of low width algebraic branching programs (ABPs) satisfying the technical assumptions of Theorem 18 and with low dimensional partial derivatives spaces, and for which (to the best of our knowledge) no average-case, proper learning algorithm is known. The authors of [25] gave such an algorithm assuming that the widths are the same across the layers of the ABP.

A.2 Algorithmic preliminaries

► **Fact 46.** *Given black-box access to an n -variate degree d polynomial f , we can compute black-box access to $\frac{\partial^k f}{\partial \mathbf{x}^\alpha}$ in $\text{poly}(n, d^k)$ time, where $|\alpha| = k$. (Refer [26] for a proof idea.)*

► **Fact 47.** Given black-boxes to n -variate degree d polynomials f_1, f_2, \dots, f_l , there is a randomized $\text{poly}(n, l, d)$ time algorithm that computes a basis for the vector space

$$(f_1, f_2 \dots f_l)^\perp := \{(\alpha_1, \alpha_2, \dots, \alpha_l) \in \mathbb{F}^l : \sum_{i=1}^l \alpha_i f_i = 0\}$$

Refer [23] for a proof of the above fact. A corollary of Fact 47 is:

► **Fact 48.** Given black-box access to linearly independent polynomials f_1, \dots, f_l and an $f = \sum_{i=1}^l \beta_i f_i$, where $\beta_i \in \mathbb{F}$, the β_i 's can be computed in randomized $\text{poly}(n, l, d)$ time.

► **Fact 49.** Let $d \in \mathbb{N}$, $\text{char}(\mathbb{F}) = 0$ or $> d$ and $|\mathbb{F}| \geq d^6$. Given black-box access to an n -variate degree d polynomial f , black-box access to its irreducible factors can be computed in randomized $\text{poly}(n, d)$ time. (Refer [22] for details.)

A.3 Description of Patarin's scheme

Patarin's scheme is a provably perfect zero-knowledge authentication scheme; thus Alice can prove to Bob that she knows a secret without revealing *any* information about the secret. The key generation process is as follows:

1. Select an n -variate degree d polynomial $f(\mathbf{x}) \in_r \mathbb{F}_q[\mathbf{x}]$, where d is a constant.
2. Select two matrices $A_1, A_2 \in_r \mathbb{F}_q^{n \times n}$.¹⁵
3. Compute the public key $(f_1, f_2) := (f(A_1 \mathbf{x}), f(A_2 \mathbf{x}))$ and the private key $C = A_2^{-1} A_1$.

The authentication procedure is as follows:

1. Alice selects an $R \in_r \mathbb{F}_q^{n \times n}$ and computes $g := f_1(R\mathbf{x})$ and sends it to Bob.
2. Bob receives g and picks $h := f_1$ or f_2 with probability $\frac{1}{2}$, challenging Alice to show $h \sim g$.
3. Alice receives h . If $h = f_1$, she sends R . If $h = f_2$, she sends CR .

A.4 Equivalence Testing for NW

The PS -equivalence testing problem for NW is as follows: given a polynomial f , check if $f = \text{NW}_{q,d,t}(PS\mathbf{x})$ for some permutation P and scaling S , and recover them if so. Theorem 50 (the proof can be found in [7]) follows from Theorems 3 and 11, and the GI algorithm in [6].

► **Theorem 50.** Let $q, d, t \in \mathbb{N}$, $q \geq d$, $t \leq d/3$, $|\mathbb{F}| \geq q^{3t}$, $\text{char}(\mathbb{F}) = 0$ or $> d$. ET for $\text{NW}_{q,d,t}$ reduces to PS -equivalence testing for $\text{NW}_{q,d,t}$ in $\text{poly}(q^t)$ time. Further, PS -equivalence testing for $\text{NW}_{q,d,t}$ reduces to S -equivalence testing for $\text{NW}_{q,d,t}$ in quasi- $\text{poly}(q^t)$ time.

The S -equivalence testing problem for NW is as follows: Given a polynomial f , check if $f = \text{NW}_{q,d,t}(S\mathbf{x})$ for some scaling S , and recover it if so. Over \mathbb{R} and \mathbb{F}_p , S -equivalence testing for NW can be done by the algorithm of [10] in $\text{poly}(q, \beta)$ time, where β is the bit complexity of the coefficients of f , and also by an algorithm of [20]. Over \mathbb{Q} , S -equivalence testing of NW can be done in $\text{poly}(q^t, \beta)$ time, assuming oracle access to integer-factoring. This combined with Theorem 50 gives a quasi- $\text{poly}(q^t)$ time algorithm for ET for NW. Here is a proof sketch of S -equivalence test for $\text{NW}_{q,d,t}$ over \mathbb{Q} : If f is S -equivalent to $\text{NW}_{q,d,t}$, then $f = \sum_{h \in \mathbb{F}_q[y], \deg h < t} c_h \prod_{i=0}^{d-1} x_{i,h(i)}$, where $c_h \in \mathbb{F}$. With the entries of S as \mathbf{z} variables, we get the following equations:

¹⁵A random matrix is invertible with high probability.

9:22 Testing Equivalence to Design Polynomials

$$z_{0,h(0)}z_{1,h(1)} \cdots z_{d-1,h(d-1)} = c_h^{-1} \quad \text{for } h \in \mathbb{F}_q[y]_{<t} .$$

There are q^t many equations in qd variables. Since $c_h^{-1} \in \mathbb{Q}$, $c_h^{-1} = a/b$ for some $a, b \in \mathbb{Z}$. Using the integer-factoring oracle, factor the integers a, b into primes $p_1, p_2 \dots p_l$. Now, reduce it to solving an appropriate Diophantine linear system by taking logarithms to the base p_i , with $\log_{p_i}(z_{j,h(j)})$ as variables $w_{j,h(j),i}$. This Diophantine linear system has lqd variables and lq^t equations. Treating this system as a matrix, check its consistency and then determine the linearly independent rows (say there are k of them). A $k \times k$ submatrix with non-zero determinant m can be formed from these rows, which corresponds to expressing the Diophantine linear system as linear equations in k of the $w_{j,h(j),i}$ variables with constants as affine forms in the remaining $w_{j,h(j),i}$ variables. By Cramer's rule, a solution to such a system is a fraction with the numerator as the affine forms and the denominator as m . The problem then further reduces to solving a linear system determined by the affine forms over the ring \mathbb{Z}_m since $w_{j,h(j),i}$ must be integers. This whole process can be done in $\text{poly}(q^t, \beta)$ time.

Therefore, ET for NW can be solved in time quasi-polynomial in the sparsity of $\text{NW}_{q,d,t}$.

Expressive Quantale-Valued Logics for Coalgebras: An Adjunction-Based Approach

Harsh Beohar  

University of Sheffield, UK

Barbara König  


Universität Duisburg-Essen, Germany

Jonas Forster  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Paul Wild  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Sebastian Gurke  

Universität Duisburg-Essen, Germany

Karla Messing  

Universität Duisburg-Essen, Germany

Lutz Schröder  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

We address the task of deriving fixpoint equations from modal logics characterizing behavioural equivalences and metrics (summarized under the term *conformances*). We rely on an earlier work that obtains Hennessy-Milner theorems as corollaries to a fixpoint preservation property along Galois connections between suitable lattices. We instantiate this to the setting of coalgebras, in which we spell out the compatibility property ensuring that we can derive a behaviour function whose greatest fixpoint coincides with the logical conformance. We then concentrate on the linear-time case, for which we study coalgebras based on the machine functor living in Eilenberg-Moore categories, a scenario for which we obtain a particularly simple logic and fixpoint equation. The theory is instantiated to concrete examples, both in the branching-time case (bisimilarity and behavioural metrics) and in the linear-time case (trace equivalences and trace distances).

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases modal logics, coalgebras, behavioural equivalences, behavioural metrics, linear-time semantics, Eilenberg-Moore categories

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.10

Related Version *Full Version*: <https://arxiv.org/abs/2310.05711> [5]

Funding The first author was supported by the EPSRC NIA Grant EP/X019373/1, while the remaining authors were supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 434050016 (SpeQt).

1 Introduction

Behavioural equivalences (such as bisimilarity and trace equivalence) are an important technique to identify states with the same behaviour in a transition system [39]. They have been complemented by notions of behavioural metrics [14, 38, 10] measuring the distance between states, in particular in a quantitative setting. We work in a coalgebraic setting [32] that allows us to answer generic questions about behavioural equivalences and metrics, parametrized over various branching types (non-deterministic, probabilistic, weighted, etc.).

There are various ways to characterize behavioural equivalences or metrics, which we illustrate using trace equivalence as an example: (i) *direct specification*: Two states x, y are trace equivalent if they admit the same traces; (ii) *logic*: x, y are trace equivalent if they cannot be distinguished in a modal logic based on diamond modalities and the constant



© Harsh Beohar, Sebastian Gurke, Barbara König, Karla Messing, Jonas Forster, Lutz Schröder, and Paul Wild;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 10; pp. 10:1–10:19



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



true; (iii) *fixpoint equation*: x, y are trace equivalent if the pair $(\{x\}, \{y\})$ is contained in the greatest fixpoint of the bisimulation function on the determinized transition system; (iv) *games*: there is an attacker-defender game characterizing the equivalence.

Our focus is on (ii) and (iii), and our take is quite different from the usual approach: Instead of first defining the behavioural equivalence/metric and then setting up an expressive logic for it, we start by defining the logic and derive a fixpoint equation from the logic. Fixpoint equations are of great interest since efficient algorithms for computing behavioural conformances are almost always based on fixpoint characterizations; in future work, we aim to exploit such characterizations for algorithmic purposes. However, for a given logic, corresponding fixpoint equations do not always exist, and we give conditions for ensuring that they do. We use the Galois connection approach from [4] as a starting point, and instead of instantiating it for each case study, we propose a generic coalgebraic framework. By employing fibrations (resp. indexed categories) [17, 18], we parameterize over the notion of *conformance* (e.g. equivalence, metric) that is our focus of attention. Moreover, we parameterize over a quantale in which both conformances and formulae take their values.

One interest, particularly, is in linear-time notions of conformance (such as trace/language equivalences and their quantitative cousins), for which we work in an Eilenberg-Moore category where the coalgebras live. We exploit the generalized powerset construction [20, 34] and characterize those (trace) logics that can be turned into suitable fixpoint equations on the determinized coalgebra, using a notion of compatibility [4] that has its roots in up-to techniques [30]. We also study the relation of compatibility to the notion of depth-1 separation used in (quantitative) graded logics [11, 13].

More concretely, we work with coalgebras of the form $c: X \rightarrow FTX$, living in some category \mathbf{C} , where a monad T intuitively specifies the implicit branching (or side effects) and a functor F describes the explicit branching type. For instance, for a non-deterministic automaton we choose $T = \mathcal{P}$ and $F = _ \Sigma \times \mathbf{2}$. We fix an *EM-law* $\zeta: TF \Rightarrow FT$ [20] allowing us to obtain a determinized coalgebra, i.e., a coalgebra of the form $c^\#: TX \rightarrow FTX$ that can be viewed as a coalgebra in the Eilenberg-Moore category of T . We can then define a logic function log that is defined on sets of \mathcal{V} -valued predicates on X and whose least fixpoint induces a behavioural conformance on TX . Alternatively, given the determinization $c^\#$, we can – in a fibrational style – define a conformance on TX as the greatest fixpoint of a Kantorovich lifting followed by a reindexing via $c^\#$. The aim is to show that both conformances coincide.

We allow arbitrary constants in the logic, which – in particular in the linear-time case – are able to add extra distinguishing power to the logic. Along the way we give an answer to the question of why, unlike branching-time logics, linear-time logics often do not need any additional (boolean) operators, only modalities and constants.

As examples we consider bisimilarity and branching-time pseudometrics for probabilistic transition systems, as well as linear-time conformances such as trace equivalence and trace distance.

Roadmap. After reviewing preliminaries in Section 2, we summarize the approach based on Galois connections (adjunctions) in Section 3. The instantiation to generic coalgebras is presented in Section 4 and a concrete quantale-valued branching-time logic spelled out in Section 5. Section 6 specializes to coalgebras in Eilenberg-Moore categories, leading to results strengthening those for the general case. Finally, Section 7 details the linear-time case studies mentioned above, and we conclude in Section 8.

The proofs can be found in the full version of this paper [5].

2 Preliminaries

We recall some basic definitions and facts on lattices, quantales, generalized distances, coalgebra, monads and their algebras and on indexed categories. We do assume basic familiarity with category theory (e.g. [1]).

2.1 Lattices, Fixpoints and Galois Connections

A *complete lattice* $(\mathbb{L}, \sqsubseteq)$ consists of a set \mathbb{L} with a partial order \sqsubseteq such that each $Y \subseteq \mathbb{L}$ has a least upper bound $\bigsqcup Y$ (also called supremum, join) and a greatest lower bound $\bigsqcap Y$ (also called infimum, meet). The Knaster-Tarski theorem [36] guarantees that any monotone function $f: \mathbb{L} \rightarrow \mathbb{L}$ on a complete lattice \mathbb{L} has a *least fixpoint* μf and a *greatest fixpoint* νf .

Let \mathbb{L}, \mathbb{B} be two lattices. A *Galois connection* from \mathbb{L} to \mathbb{B} is a pair $\alpha \dashv \gamma$ of monotone functions $\alpha: \mathbb{L} \rightarrow \mathbb{B}, \gamma: \mathbb{B} \rightarrow \mathbb{L}$ such that for all $\ell \in \mathbb{L}, b \in \mathbb{B}$: $\alpha(\ell) \sqsubseteq b \iff \ell \sqsubseteq \gamma(b)$.

A *closure* $\text{cl}: \mathbb{L} \rightarrow \mathbb{L}$ is a monotone, idempotent and extensive (i.e. $\forall x \in \mathbb{L} \ x \sqsubseteq \text{cl}(x)$) function on a lattice. A *co-closure* is monotone, idempotent and extensive wrt. \sqsupseteq . Given a Galois connection $\alpha \dashv \gamma$, $\gamma \circ \alpha$ is always a closure and $\alpha \circ \gamma$ a co-closure.

2.2 Quantales and Generalized Distances

► **Definition 1.** A (*unital, commutative*) *quantale* $(\mathcal{V}, \otimes, 1)$, or just \mathcal{V} , is a complete lattice with an associative, commutative operation $\otimes: \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{V}$ with unit 1 that distributes over arbitrary (possibly infinite) joins \bigvee . If 1 is the top element of \mathcal{V} , then \mathcal{V} is *integral*.

In a quantale \mathcal{V} , the functor $- \otimes y$ has a right adjoint $[y, -]$ for every $y \in \mathcal{V}$; that is, $x \otimes y \leq z \iff x \leq [y, z]$ for all $x, y, z \in \mathcal{V}$.

► **Example 2.**

1. The Boolean algebra $\mathbf{2}$ with $\otimes = \wedge$ and unit 1 is an integral quantale; for $y, z \in \mathbf{2}$, we have $[y, z] = y \rightarrow z$.
2. The complete lattice $[0, 1]$ ordered by the reversed order of the reals, i.e. $\leq = \geq_{\mathbb{R}}$, and equipped with truncated addition $r \otimes s = \min(r + s, 1)$, is an integral quantale; for $r, s \in [0, 1]$, we have $[r, s] = s \dot{-} r = \max(s - r, 0)$ (truncated subtraction).

Given a quantale \mathcal{V} , a *directed (\mathcal{V} -valued) pseudometric (on X)* is a function $d: X \times X \rightarrow \mathcal{V}$ where (i) $\forall x \in X \ d(x, x) \geq 1$ (reflexivity); (ii) $\forall x, y, z \in X \ d(x, z) \geq d(x, y) \otimes d(y, z)$ (transitivity/triangle inequality). Moreover, d is a *pseudometric* if additionally, (iii) $\forall x, y \in X \ d(x, y) = d(y, x)$ (symmetry). We write $\mathbf{DPMet}_{\mathcal{V}}(X)$ to denote the lattice of all directed pseudometrics on X , while for pseudometrics we use $\mathbf{PMet}_{\mathcal{V}}(X)$. Given $d_X \in \mathbf{DPMet}_{\mathcal{V}}(X), d_Y \in \mathbf{DPMet}_{\mathcal{V}}(Y)$, a function $f: X \rightarrow Y$ is *non-expansive* (wrt. d_X, d_Y) if $d_X(x, x') \leq d_Y(f(x), f(x'))$ for all $x, x' \in X$. Note that due to the choice of order in the quantale, the inequality in the definitions above is reversed wrt. the standard definitions that are typically given in the order on the reals. As originally observed by Lawvere [26], one may see directed \mathcal{V} -valued pseudometrics as \mathcal{V} -enriched categories, or just \mathcal{V} -categories, and non-expansive functions as \mathcal{V} -functors.

2.3 Coalgebras and Eilenberg-Moore Categories

Given a functor $F: \mathbf{C} \rightarrow \mathbf{C}$, an *F-coalgebra* (X, c) (or simply c) consists of an object $X \in \mathbf{C}$ and a \mathbf{C} -arrow $c: X \rightarrow FX$. In the paradigm of *universal coalgebra* [32], we understand X as the state space of a transition system, F as specifying the branching type of the system,

and c as a transition map that assigns to each state a collection of successors structured according to F . For instance, when $\mathbf{C} = \mathbf{Set}$ is the category of sets and functions, then the powerset functor \mathcal{P} assigns to each set its powerset, and \mathcal{P} -coalgebras are just sets equipped with a transition relation. On the other hand, the (finitely supported) distribution functor \mathcal{D} assigns to each set X the set of finitely supported probability distributions on X , given in terms of maps $p: X \rightarrow [0, 1]$ with finite support such that $\sum_{x \in X} p(x) = 1$. A \mathcal{D} -coalgebra thus is precisely a Markov chain.

Recall that a *monad* $(T, \eta: \text{Id} \Rightarrow T, \mu: TT \Rightarrow T)$ on \mathbf{C} , usually denoted by just T , consists of a functor $T: \mathbf{C} \rightarrow \mathbf{C}$ and natural transformations $\eta: \text{Id} \Rightarrow T$ (the *unit*) and $\mu: TT \Rightarrow T$ (the *multiplication*), subject to certain coherence laws. Monads abstractly capture notions of algebraic theory, with TX being thought of as terms modulo provable equality over variables in X . Correspondingly, a T -*algebra* (X, a) consists of an object X of \mathbf{C} and a \mathbf{C} -arrow $a: TX \rightarrow X$ such that $a \circ \eta_X = \text{id}_X$ and $a \circ Ta = a \circ \mu_X$; we may think of T -algebras as algebras for the algebraic theory encapsulated by T . A homomorphism between T -algebras $(X, a), (Y, b)$ is a \mathbf{C} -arrow $f: X \rightarrow Y$ such that $b \circ Tf = f \circ a$. The *Eilenberg-Moore category* of T , denoted $\mathbf{EM}(T)$, is the category of T -algebras and their homomorphisms. There is a free-forgetful adjunction $L \dashv R: \mathbf{C} \rightarrow \mathbf{EM}(T)$, where the forgetful functor R maps an algebra to its underlying \mathbf{C} -object and L maps an object $X \in \mathbf{C}$ to the free algebra (TX, μ_X) .

A (*monad-over-functor*) *distributive law* (or *EM-law*) of a monad T over a functor F is a natural transformation $\zeta: TF \Rightarrow FT$ satisfying $\zeta_X \circ \eta_{FX} = F\eta_X$ and $\zeta_X \circ \mu_{FX} = F\mu_X \circ \zeta_{TX} \circ T\zeta_X$. This is equivalent to saying that the assignment $\tilde{F}(X, a) = (FX, Fa \circ \zeta_X)$ defines a *lifting* $\tilde{F}: \mathbf{EM}(T) \rightarrow \mathbf{EM}(T)$ of F (where *lifting* means that $R\tilde{F} = FR$). Then, the *determinization* [34] of a coalgebra $c: X \rightarrow FTX$ in \mathbf{C} is the transpose $c^\#: LX \rightarrow \tilde{F}LX$ of c under $L \dashv R$. More concretely the determinization can be obtained as $c^\# = F\mu_X \circ \zeta_{TX} \circ Tc$. For instance, when $FX = X^\Sigma \times \mathbf{2}$ and $T = \mathcal{P}$, then this yields exactly the standard powerset construction for the determinization of non-deterministic automata.

2.4 Indexed Categories and Fibrations

Our aim is to equip objects of a category with additional information, e.g., consider sets with (equivalence) relations or metrics. Formally, this is done by working with fibrations, in particular we will consider fibrations arising from the Grothendieck construction for indexed categories [17, 18]. For us it is sufficient to consider as indexed categories functors $\Phi: \mathbf{C}^{\text{op}} \rightarrow \mathbf{Pos}$, where \mathbf{Pos} is the category of posets (ordered by \preceq) with monotone maps. Such functors induce a fibration $U: \int \Phi \rightarrow \mathbf{C}$ where U is the forgetful functor and $\int \Phi$ is the category whose objects and arrows are characterized as follows:

$$\frac{X \in \mathbf{C} \wedge d \in \Phi X}{(X, d) \in \int \Phi} \quad \frac{X \xrightarrow{f} Y \in \mathbf{C} \wedge d \preceq (\Phi f)d'}{(X, d) \xrightarrow{f} (Y, d') \in \int \Phi}$$

Here, $f^* = \Phi f$ is also called *reindexing operation* and d is called a *conformance*.

Typical examples are functors $\Phi: \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Pos}$ mapping a set X to the lattice of equivalence relations or pseudometrics on X .

3 Adjoint Logic: the General Framework

We summarize previous results on relating logical and behavioural conformances using Galois connections [4]. These results are based on the following well-known property that shows how fixpoints are preserved by Galois connections (e.g. [3, 8, 9]). The formulation of these properties involves a notion of compatibility studied for coinductive up-to techniques [30].

► **Definition 3.** Let $\log, \text{cl}: \mathbb{L} \rightarrow \mathbb{L}$ be monotone endofunctions on a partial order $(\mathbb{L}, \sqsubseteq)$. Then \log is *cl-compatible* if $\log \circ \text{cl} \sqsubseteq \text{cl} \circ \log$.

► **Theorem 4** ([4]). Let $\alpha: \mathbb{L} \rightarrow \mathbb{B}$, $\gamma: \mathbb{B} \rightarrow \mathbb{L}$ be a Galois connection between complete lattices \mathbb{L}, \mathbb{B} , and let $\log: \mathbb{L} \rightarrow \mathbb{L}$, $\text{beh}: \mathbb{B} \rightarrow \mathbb{B}$ be monotone. Then the following holds.

1. If $\text{beh} = \alpha \circ \log \circ \gamma$ then $\alpha(\mu \log) \sqsubseteq \mu \text{beh}$.
2. If $\alpha \circ \log = \text{beh} \circ \alpha$, then $\alpha(\mu \log) = \mu \text{beh}$. If \log reaches its fixpoint in ω steps, i.e., $\mu \log = \log^\omega(\perp)$, then so does beh .
3. Let $\text{cl} = \gamma \circ \alpha$ be the closure operator of the Galois connection, and suppose that $\text{beh} = \alpha \circ \log \circ \gamma$. If \log is cl-compatible, then $\alpha(\mu \log) = \mu \text{beh}$.

► **Remark 5.** In fact, there is a weaker notion than compatibility that ensures the same result, i.e., $\alpha(\mu \log) = \mu \text{beh}$. In particular, it is sufficient to show the following condition:

- $\log(\text{cl}(\ell)) \sqsubseteq \text{cl}(\log(\ell))$ for all $\ell \in \mathbb{S} \subseteq \mathbb{L}$ where \mathbb{S} is an invariant of \log , i.e. $\log[\mathbb{S}] \subseteq \mathbb{S}$, $\perp \in \mathbb{S}$, and \mathbb{S} is closed under directed joins. (If the fixpoint is reached in ω steps, closure under directed joins is not required.)

We apply this in a scenario where \mathbb{L} consists of logical formulas (or more precisely their semantics, in the shape of sets of definable predicates) and \mathbb{B} consists of conformances. These Galois connections will be contra-variant when we consider the quantalic ordering in \mathbb{B} . Then, \log is the “logic function” that adds a layer of modalities and propositional operators to a given set of predicates, so that $\mu \log$ is the semantics of the set of formulas of the logic. On the other hand, beh is the “behaviour function”, whose greatest fixpoint νbeh (remember the contra-variance) is behavioural conformance.

► **Example 6.** The simplest Galois connection used in [4] for characterizing behavioural equivalence is between $\mathbb{L} = \mathcal{P}(2^X)$ (sets of predicates on X) and $\mathbb{B} = \mathbf{PMet}_2(X)$ (equivalences on X), where α maps every set of predicates to the equivalence relation induced by it and γ maps an equivalence to the set of predicates closed under it.

Moving to pseudometrics we obtain a Galois connection between $\mathbb{L} = \mathcal{P}([0, 1]^X)$ (sets of real-valued predicates on X) and $\mathbb{B} = \mathbf{PMet}_{[0,1]}(X)$ (pseudometrics on X) where α maps every set of functions $X \rightarrow [0, 1]$ to the least pseudometric making all these functions non-expansive and γ takes a pseudometric and produces all its non-expansive functions.

So first, define a logical universe \mathbb{L} and a logic function $\log: \mathbb{L} \rightarrow \mathbb{L}$. Second, choose a suitable Galois connection $\alpha \dashv \gamma$ to a behaviour universe \mathbb{B} and show that \log is cl-compatible. Third, derive the behaviour function $\text{beh} = \alpha \circ \log \circ \gamma: \mathbb{B} \rightarrow \mathbb{B}$. From the results above, we automatically obtain the equality $\alpha(\mu \log) = \mu \text{beh}$, which tells us that logical and behavioural equivalence respectively distance coincide (Hennessy-Milner theorem).

4 Adjoint Logic for Coalgebras

In this section we will describe a general framework where the adjoint logic is instantiated to the setting of coalgebraic modal logic.

4.1 Setting up the Adjunction

One can generalize from Example 6 and instead of a set X take an object in a (locally small) category \mathbf{C} . Furthermore we fix an object $\Omega \in \mathbf{C}$ (the *truth value* object), which in all our applications will be a quantale. Predicates are represented by the indexed category $\mathbf{C}(_, \Omega)$; thus, sets of predicates (lattice \mathbb{L}) are given by the indexed category $\mathcal{P} \circ \mathbf{C}(_, \Omega)$ (where the order is inclusion). In addition, we use an indexed category Φ specifying the notion of conformance on X (lattice \mathbb{B}) and work with the following assumptions:

A1 Each fibre ΦX is a poset having arbitrary meets (thus, a complete lattice) and the reindexing map preserves these meets (i.e. $\int \Phi$ has fibred limits).

A2 Let $d_\Omega \in \Phi \Omega$ be a fixed conformance on the truth value object Ω .

For an arrow $f \in \mathbf{C}(X, Y)$, we write f^\bullet for the reindexing in $\mathbf{C}(_, \Omega)$ ($f^\bullet g = g \circ f$, where $g \in \mathbf{C}(Y, \Omega)$) and f^* for the reindexing in Φ ($f^* = \Phi f$).

► **Theorem 7.** *Let X be an object of \mathbf{C} . Under Assumptions **A1** and **A2**, there is a dual adjoint situation (contravariant Galois connection) $\alpha_X \dashv \gamma_X$ between the underlying fibres:*

$$\begin{aligned} \alpha_X : \mathcal{P}(\mathbf{C}(X, \Omega)) &\rightarrow \Phi(X)^{op} & S \subseteq \mathbf{C}(X, \Omega) &\mapsto \bigwedge_{k \in S} k^*(d_\Omega) \\ \gamma_X : \Phi(X)^{op} &\rightarrow \mathcal{P}(\mathbf{C}(X, \Omega)) & d \in \Phi X &\mapsto \{k \in \mathbf{C}(X, \Omega) \mid d \preceq k^*(d_\Omega)\}. \end{aligned}$$

More concretely: α_X, γ_X are both antitone ($S \subseteq S' \implies \alpha_X(S) \succeq \alpha_X(S')$, $d \preceq d' \implies \gamma_X(d) \supseteq \gamma_X(d')$) and we have $d \preceq \alpha_X(S) \iff S \subseteq \gamma_X(d)$ for $d \in \Phi X$ and $S \in \mathcal{P}\mathbf{C}(X, \Omega)$.

Thus, for $X \in \mathbf{C}$, the fibres $\mathcal{P}\mathbf{C}(X, \Omega)$ and $(\Phi X)^{op}$ will take the role of \mathbb{B} and \mathbb{L} (respectively) as in [4, Theorem 3.2]. Moreover, Theorem 7 will be instantiated to obtain the desired Galois connections between predicates and conformances for our case studies.

► **Example 8.** Let $\mathbf{C} = \mathbf{Set}$ and $\Omega = \mathcal{V}$ be a quantale. We consider $\Phi X = \mathbf{DPMet}_{\mathcal{V}}(X)$ (resp., $\Phi X = \mathbf{PMet}_{\mathcal{V}}(X)$) with the order \preceq on ΦX induced by the pointwise lifting of the order \leq on \mathcal{V} . The reindexing functor f^* for a function $f: X \rightarrow Y$ is given by $f^*d = d \circ (f \times f)$; thus, satisfying **A1**. As conformance d_Ω on \mathcal{V} we take the internal hom $[_, _]$ (resp., its symmetrization: $d_\Omega(x, y) = [x, y] \wedge [y, x]$); thus, satisfying **A2**. Then we have $\alpha_X \dashv \gamma_X : \mathcal{P}(\mathbf{Set}(X, \mathcal{V})) \rightleftarrows \mathbf{DPMet}_{\mathcal{V}}(X)$, where:

$$\begin{aligned} \alpha_X(S)(x, x') &= \bigwedge_{h \in S} d_\Omega(h(x), h(x')) \\ \gamma_X(d) &= \{h: X \rightarrow \mathcal{V} \mid \forall_{x, x' \in X} d(x, x') \leq d_\Omega(h(x), h(x'))\} \end{aligned}$$

In both cases, α_X assigns to a set of maps the greatest (directed) pseudometric making all these maps non-expansive, while γ_X maps a pseudometric all its non-expansive maps.

4.2 Characterizing Closure

Given that the key condition imposed on the logic function in Theorem 4 is compatibility with the closure of the Galois connection, it is important to understand how this closure operates. In the setting of Theorem 7 we can characterize the closure in terms of non-expansive propositional operators, provided that γ is natural. We note first that α is always natural:

► **Proposition 9.** *In Theorem 7, the transformation α is natural in $X \in \mathbf{C}$, that is, for $f \in \mathbf{C}(X, Y)$, we have $\alpha_X \circ \mathcal{P}(f^\bullet) = f^* \circ \alpha_Y$.*

For the right adjoint γ , naturality need not hold in general. It does hold for \mathbf{Set} and generalized (directed) metrics over the quantales in Example 2. A counterexample can however be constructed for $\mathbf{C} = \mathbf{EM}(\mathcal{P})$ and $\mathcal{V} = [0, 1]$ (see [4]).

If γ is natural, then we can characterize the closure $\gamma \circ \alpha$ using the internal language of indexed categories. To this end, suppose that \mathbf{C} has (small) products. Then for every $S \subseteq \mathbf{C}(X, \Omega)$, we have a unique tupling $\langle S \rangle: X \rightarrow \Omega^S$ such that $\pi_k \circ \langle S \rangle = k$ for all $k \in S$, where $\pi_k: \Omega^S \rightarrow \Omega$ is the product projection for k .

► **Lemma 10.** *The right adjoint γ in Theorem 7 is lazily natural, i.e. $\mathcal{P}(f^\bullet) \circ \gamma_Y \subseteq \gamma_X \circ f^*$ for $f: X \rightarrow Y \in \mathbf{C}$. If γ is natural (i.e. the inclusion is an equality) and \mathbf{C} has products, then $\gamma_X(\alpha_X(S)) = \mathcal{P}(\langle S \rangle^\bullet)(\gamma_{\Omega^S}(d_{\Omega^S}))$, where $d_{\Omega^S} = \bigwedge_{k \in S} \pi_k^* d_\Omega$.*

This result can be interpreted as follows: $\gamma_{\Omega^S}(d_{\Omega^S})$ is the set of all non-expansive functions $\Omega^S \rightarrow \Omega$, hence all non-expansive operators of arbitrary arity on Ω . Reindexing via $\langle S \rangle$ means to combine all predicates in S via those operators, hence we describe the closure under all non-expansive operations on Ω .

4.3 Towards a Generic Logic Function

Since our slogan is to generate the behaviour function from the logic function, we start by setting up our logical framework first. Following [4, 12], we adopt a semantic approach to defining a (modal) logic, i.e., we specify the operators (including modalities) as a transformation of predicates; formally, as a (natural) transformation \log of type $\mathcal{PC}(_, \Omega) \Rightarrow \mathcal{PC}(_, \Omega)$. The idea is that the logic function \log_X adds one “layer” of modal depth; in particular, the least fixpoint $\mu \log_X$ of \log_X can be seen as the set of (interpretations of) all modal formulas.

In particular, we require

A3 Fix a family $(ev_\lambda \in \mathbf{C})_{\lambda \in \Lambda}$ of evaluation maps $ev_\lambda: F\Omega \rightarrow \Omega$.

As noted in [33], such evaluation maps – commonly used in coalgebraic modal logic – correspond to natural transformations of type $\mathbf{C}(_, \Omega) \rightarrow \mathbf{C}(F_, \Omega)$ by the Yoneda lemma.

► **Proposition 11.** *A family of evaluation maps $(ev_\lambda)_{\lambda \in \Lambda}$ induces a natural transformation $\Lambda: \mathcal{PC}(_, \Omega) \Rightarrow \mathcal{PC}(F_, \Omega)$ given by $S \mapsto \{\lambda_X(h) \mid \lambda \in \Lambda, h \in S\}$, where $\lambda_X(h) = ev_\lambda \circ Fh$.*

Apart from modalities, a logic typically needs operators and constants. We do not consider constants as 0-ary operators, which allows us to distinguish between operators that arise as in Lemma 10 from the closure of the Galois connection (that is, non-expansive operators) and the remaining (constant) operators that bring additional distinguishing power. This is, for instance, needed in the case of trace equivalence on a determinized transition system to distinguish the empty set of states from sets of states having no transitions. We need an additional (constant) predicate for this task that can neither be provided by the closure nor by a constant modality (see Appendix A.1).

A4 We assume a set $\Theta_X \subseteq \mathbf{C}(X, \Omega)$ of *constants* (which is later restricted to consist of free extensions of constant maps).

To model the propositional operators, we introduce a closure cl'_X :

A5 For each $X \in \mathbf{C}$ we assume that there is a closure $cl'_X: \mathcal{PC}(X, \Omega) \rightarrow \mathcal{PC}(X, \Omega)$ (not necessarily natural), specifying the propositional operators.

We say that cl'_X is a *subclosure* of cl_X whenever $cl'_X \subseteq cl_X$, which means that the propositional operators implemented by cl'_X are already contained in the closure induced by the Galois connection (cf. Lemma 10).

Now we can define the logic function for a coalgebra c as

$$\log_X = \mathcal{P}(c^\bullet) \circ \Lambda_X \circ cl'_X \cup \Theta_X: \mathcal{PC}(X, \Omega) \rightarrow \mathcal{PC}(X, \Omega).$$

Its least fixpoint contains all predicates that can be described by modal formulas.¹

¹ In our setup a formula is either a constant or starts with a modality, which still results in an expressive logic. One could slightly modify \log_X and obtain all formulas by adding another closure cl' .

► **Example 12.** Let $\mathbf{C} = \mathbf{Set}$, $c: X \rightarrow FX$ and $\Phi X = \mathbf{PMet}_{\mathcal{V}}(X)$. Recall the Galois connection from Example 8 and consider the following two examples, where in both cases no constants are needed, i.e., Θ_X is empty; thus, **A4** vacuously holds.

1. *Bisimilarity on (unlabelled) transition systems:* we let $F = \mathcal{P}_{\text{fin}}$ (finite powerset functor), $\mathcal{V} = 2$, and consider the evaluation map $ev_{\diamond}: \mathcal{P}_{\text{fin}}2 \rightarrow 2$ encoding the usual diamond modality: $ev_{\diamond}(U) = 1 \iff 1 \in U$. This can be extended to a logic by choosing as cl' (Assumption **A5**) the closure under all (finitary) Boolean operators.
2. *Behavioural metrics for probabilistic transition systems with termination:* we let $FX = \mathcal{D}X + 1$ (where $1 = \{\checkmark\}$) and $\mathcal{V} = ([0, 1], \geq_{\mathbb{R}})$. Define two evaluation maps: $ev_E: \mathcal{D}[0, 1] + 1 \rightarrow [0, 1]$ corresponds to expectation, i.e. $ev_E(p) = \sum_{r \in [0, 1]} r \cdot p(r)$ if $p \in \mathcal{D}X$ (0 otherwise). Furthermore, $ev_*: \mathcal{D}[0, 1] + 1 \rightarrow [0, 1]$ with $ev_*(p) = 1$ if $p = \checkmark$ (0 otherwise). We extend this to a logic by defining cl' (Assumption **A5**): we add as operators the constant 1, $\min(\varphi, \varphi')$, $1 - \varphi$ and $\varphi \dot{-} q$ for a rational q (where φ is a formula), as in similar logics for probabilistic transition systems [38].

To ensure that the requirements of Section 3 are met, we have to show compatibility of the logic function. To this end, we introduce the notion of compability of cl' .

► **Definition 13.** Given a closure $\text{cl}'_X: \mathcal{PC}(X, \Omega) \rightarrow \mathcal{PC}(X, \Omega)$, we say that cl'_X is *compatible* if the map $\Lambda_X \circ \text{cl}'_X$ (for each $X \in \mathbf{C}$) is compatible with the closure $\text{cl}_X = \gamma_X \circ \alpha_X$ induced by the adjoint situation in Theorem 7, i.e., $\Lambda_X \circ \text{cl}'_X \circ \text{cl}_X \subseteq \text{cl}_{FX} \circ \Lambda_X \circ \text{cl}'_X$.

The following results hold under Assumptions **A1-A5** and thus, we avoid stating them in various lemma/theorem statements.

► **Proposition 14.** For a given compatible closure cl'_X , the above logic function \log_X is *cl_X-compatible*, i.e., $\log_X \circ \text{cl}_X \subseteq \text{cl}_X \circ \log_X$.

We will now study equivalent conditions and special cases in which compatibility holds. First, it is easy to see that $\text{cl}' = \text{cl}$ is always compatible, but typically introduces infinitary operators. Moreover, if cl' is the identity (that is, there are no propositional operators), then compatibility of cl' reduces to cl -compatibility of Λ .

► **Lemma 15.** Let cl'_X be a subclosure of cl_X . It holds that cl'_X is compatible if and only if $\alpha_{FX} \circ \Lambda_X \circ \text{cl}'_X \preceq \alpha_{FX} \circ \Lambda_X \circ \text{cl}_X$.

We next adapt the separation property establishing expressiveness of *graded logics* w.r.t. *graded semantics* [11, 13] to the present setting, an additional twist being that the conformance w.r.t. which modalities must be separating is the one induced by the modalities themselves.

► **Definition 16** (Depth-1 self-separation). A set $S \subseteq \mathbf{C}(X, \Omega)$ of predicates is *initial* for $d \in \Phi X$ if $\alpha_X(S) = d$. Let cl'_X be a subclosure of cl_X . The *depth-1 self-separation* property holds if for every S that is closed under cl'_X (i.e., $S = \text{cl}'_X(S)$) and initial for d , it follows that $\Lambda_X(S)$ is initial for $\kappa_X(d)$ where $\kappa_X = \alpha_{FX} \circ \Lambda_X \circ \gamma_X$.

► **Lemma 17.** Let cl'_X be a subclosure of cl_X . Then cl'_X is compatible if and only if the *depth-1 self-separation property* holds.

Finally, we study a sufficient condition on evaluation maps ensuring cl -compatibility of Λ .

► **Lemma 18.** If each evaluation map ev_{λ} arises as a natural transformation $\eta: F \Rightarrow \text{Id}$ or $\eta: F \Rightarrow \Omega$ (Ω is the constant functor mapping every object to Ω), that is $ev_{\lambda} = \eta_{\Omega}$, then Λ is compatible with cl .

► **Example 19.** We establish compatibility for the logics considered in Example 12. In branching-time logics in general, depth-1 self-separation usually boils down to establishing a Stone-Weierstraß type property saying that if $S \subseteq \mathbf{C}(X, \Omega)$ is initial and closed under cl'_X , then S is dense in $\mathbf{C}(X, \Omega)$, for suitably restricted X [12]. For finitary set functors such as \mathcal{P}_{fin} or \mathcal{D} , it suffices to prove self-separation on finite X . Additional details are as follows.

1. In the case of unlabelled transition systems, we are given an equivalence relation R on a finite set X , a set $S \subseteq \mathbf{Set}(X, 2)$ that is initial for R and closed under Boolean combinations, and $A, B \in \mathcal{P}_{\text{fin}}(X)$ that are distinguished by some predicate $\diamond_X f: \mathcal{P}_{\text{fin}}(X) \rightarrow 2$ where f is invariant under R . We then have to show that A, B are distinguished by $\diamond g$ for some $g \in S$. But by functional completeness of Boolean logic and because X is finite, S is in fact the set of *all* R -invariant functions $X \rightarrow 2$, so we can just take $g = f$.
2. The argument is similar for probabilistic transition systems, with some additional considerations necessitated by the quantitative setting. We are now given a set $S \subseteq \mathbf{Set}(X, [0, 1])$ that is initial for d and closed under propositional operators as per Example 12.2. By a variant of the Stone-Weierstraß theorem, this implies that S is dense in the space of non-expansive maps $(X, d) \rightarrow [0, 1]$ (see [43]), which means that in an argument as in the previous item, we can take g to range over functions in S that approximate the given non-expansive function $f: (X, d) \rightarrow [0, 1]$ arbitrarily closely, using additionally that the predicate lifting induced by ev_E as in Example 12(2) is non-expansive [42].

4.4 Towards a Generic Behaviour Function

Building on the previous section, we define the behaviour function $\text{beh}_X: \Phi X \rightarrow \Phi X$ as $\text{beh}_X = \alpha_X \circ \log_X \circ \gamma_X$ and – under the assumption of compatibility – we have² $\alpha_X(\mu \log_X) = \nu \text{beh}_X$. In other words, the notions of logical and behavioural conformances coincide.

This motivates a closer investigation of νbeh_X : in what sense does it coincide with known behavioural equivalences or metrics? Defining a behavioural conformance (that is, an element of ΦX) in a fibrational setting is typically done by taking the greatest fixpoint of a function defined in two steps: the lifting of a conformance ΦX to $\Phi(FX)$, followed by a reindexing via c . Here, we consider Kantorovich-style [2] or codensity [22] liftings based on the evaluation maps. Kantorovich liftings have originally been used to lift metrics on a set X to metrics of probability distributions over X . In the probabilistic case, an alternative characterization is given via optimal transport plans in transportation theory (earth mover’s distance) [40].

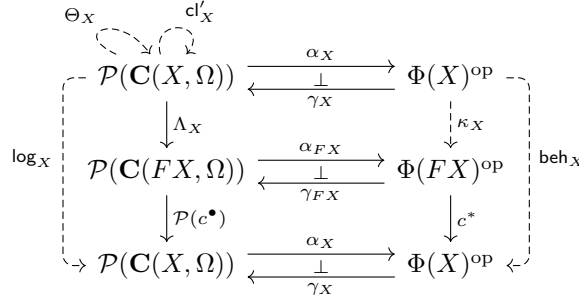
We use the natural transformation Λ introduced earlier and consider the composite $\kappa_X = \alpha_{FX} \circ \Lambda_X \circ \gamma_X$, the mentioned Kantorovich lifting. If $F = \mathcal{D}$ and the evaluation map is expectation, then we obtain exactly the classical Kantorovich lifting.

Given a coalgebra $c: X \rightarrow FX$, we use the *behaviour function* $\text{beh}_X = c^* \circ \kappa_X \wedge \alpha_X(\Theta_X)$; here, Θ_X is a set of constants as in Section 4.3 (Assumption **A4**).

► **Example 20.** We derive the behaviour functions for Examples 12 and 19.

1. In the case $F = \mathcal{P}$ and $\mathcal{V} = 2$, the lifting $\kappa_X(R) \subseteq \mathcal{P}X \times \mathcal{P}X$ of an equivalence relation $R \subseteq X \times X$ is the Egli-Milner lifting, i.e. $U \kappa_X(R) V \iff \forall x \in U \exists y \in V x R y \wedge \forall y \in V \exists x \in U x R y$. It is well-known that the greatest fixpoint of $\text{beh}_X = c^* \circ \kappa_X$ is precisely Park-Milner bisimilarity.

² Note that the adjunction defined in Theorem 7 is contravariant. Hence the least fixpoint of beh from Theorem 4 becomes the greatest fixpoint νbeh_X wrt. the lattice order \preceq .



■ **Figure 1** The adjoint setup with $\log_X = \mathcal{P}(c^*) \circ \Lambda_X \circ \text{cl}'_X \cup \Theta_X$ and $\text{beh}_X = c^* \circ \kappa_X \cup \alpha_X(\Theta_X)$.

2. In the case $FX = \mathcal{D}X + 1$ and $\mathcal{V} = [0, 1]$, we obtain the lifting $\kappa_X(d) \in \mathbf{PMet}(\mathcal{D}X + 1)$ of a pseudometric $d \in \mathbf{PMet}(X)$. It is easy to see that $\kappa_X(d)(p_1, p_2)$ is the distance given by the classical Kantorovich lifting of d if $p_1, p_2 \in \mathcal{D}X$. If $p_1 = \surd = p_2$, then the distance is 0, otherwise 1. The least fixpoint (under the usual order on $[0, 1]$) of the behaviour function $\text{beh}_X = c^* \circ \kappa_X$ agrees with standard notions of bisimulation distance (e.g. [38]).

We conclude the section by showing that behaviour functions defined in this way are actually the ones obtained from the logic function. For the diagram underlying the proof see Figure 1.

► **Theorem 21.** *Assume that cl'_X is a subclosure of cl_X and compatible. For a set Θ_X of constants and a coalgebra $c: X \rightarrow FX$, the logic function $\log_X = \mathcal{P}(c^*) \circ \Lambda_X \circ \text{cl}'_X \cup \Theta_X$ induces $\text{beh}_X = c^* \circ \kappa_X \wedge \alpha_X(\Theta_X)$, i.e., $\alpha_X \circ \log_X \circ \gamma_X = \text{beh}_X$.*

Putting everything together via Theorem 4, if cl'_X is a compatible closure, then we have $\alpha_X(\mu \log_X) = \nu \text{beh}_X$, that is, logical conformance coincides with behavioural conformance.

5 Logics for Quantale-valued Simulation Distances

We next consider a quantitative modal logic \mathcal{L}_Λ that we show to be expressive for similarity distance (a behavioural directed metric) under certain conditions. Throughout this section, our working category \mathbf{C} is \mathbf{Set} , we have a fixed functor F on \mathbf{Set} , and a fixed quantale $\Omega = \mathcal{V}$ with distance $d_\mathcal{V} = [_, _]$ being the internal hom. Furthermore $\Phi X = \mathbf{DPMet}_\mathcal{V}(X)$. In this section we assume naturality of γ , which holds for the quantales given in Example 2.

$$\varphi \in \mathcal{L}_\Lambda ::= \bigwedge_{i \in I} \varphi_i \mid \varphi \otimes v \mid d_\mathcal{V}(v, \varphi) \mid [\lambda]\varphi \quad (\text{for } v \in \mathcal{V}, \lambda \in \Lambda, I \in \mathbf{Set})$$

This logic is the positive fragment of quantale-valued coalgebraic modal logic [41, 12], and generalizes logics for real-valued simulation distance [42] to the quantalic setting. The first three operators are regarded as propositional operators of cl' , while the $[\lambda]$ are the modalities. We do not use explicit constants ($\Theta_X = \emptyset$), but note that constant truth \top is included as the empty meet. On a coalgebra $c: X \rightarrow FX \in \mathbf{Set}$, we interpret a formula φ as a function $\llbracket \varphi \rrbracket: X \rightarrow \mathcal{V}$ as usual (by structural induction on terms). Note that we do not allow negation, as we aim to characterize similarity distance. Disjunction could be included but, as in the two-valued case [39], is not needed to characterize simulation. Meet is infinitary, so the logic function \log does not reach its least fixpoint in ω steps.

The next results show that the three operators (as well as join) are all non-expansive and hence cl' is a subclosure of cl (cf. Lemma 10), which moreover is compatible.

► **Proposition 22.** *Infinitary meets, infinitary joins, negative scaling $d_\mathcal{V}(v, _)$, and positive scaling $_ \otimes v$ (for $v \in \mathcal{V}$) are non-expansive.*

► **Proposition 23.** *If each $\lambda \in \Lambda$ is sup-preserving (i.e. $\lambda_X(\bigvee P) = \bigvee \mathcal{P}(\lambda_X)(P)$, for every subset $P \subseteq \mathbf{Set}(X, \mathcal{V})$), then the sub-closure cl' of cl as above is compatible.*

Diamond-like modalities (for powerset or fuzzy powerset) are typically sup-preserving. In such cases, Theorem 21 yields expressiveness of the above logic for similarity distance, defined as the greatest fixpoint of $\text{beh}_X(d) = c^* \circ \kappa_X(d)$ where κ_X is the directed Kantorovich lifting.

6 The Adjoint Setup in an Eilenberg-Moore Category

While we have seen in the examples of the previous sections that the framework can be instantiated to coalgebras living in \mathbf{Set} , thus providing Hennessy-Milner theorems for bisimilarity, we are now interested in tackling trace equivalences and trace metrics. To this end, we work in Eilenberg-Moore categories [34], which also allows us to determinize a coalgebra using the generalized powerset construction (cf. 2.3).

In particular, we instantiate the adjoint setup to the category $\mathbf{EM}(T)$ of T -algebras (for some monad T on \mathbf{C}), provide conditions guaranteeing compatibility, and characterize the behaviour function. Furthermore, taking inspiration from [33], we also introduce a general syntax for modal formulas that can be interpreted over coalgebras living in $\mathbf{EM}(T)$. As introduced in Section 2.3, we fix a coalgebra $c: X \rightarrow FTX$ living in \mathbf{C} and its determinization $c^\#: LX \rightarrow \tilde{F}LX$ in $\mathbf{EM}(T)$ via a distributive law $\zeta: TF \Rightarrow FT$.

We assume an indexed category $\Psi: \mathbf{C}^{\text{op}} \rightarrow \mathbf{Pos}$ (that has fibred limits) and lift it to the category $\mathbf{EM}(T)$ of T -algebras by postcomposition, that is, $\Phi = \Psi \circ R$ (thus ensuring **A1**):

$$\mathbf{EM}(T)^{\text{op}} \xrightarrow{R} \mathbf{C}^{\text{op}} \xrightarrow{\Psi} \mathbf{Pos}.$$

Here, R is the forgetful functor in the free-forgetful adjunction $L \dashv R: \mathbf{C} \rightarrow \mathbf{EM}(T)$ from Section 2.3. To handle **A2**, we fix a truth value object $\Omega \in \mathbf{C}$ equipped with a T -algebra structure $o: T\Omega \rightarrow \Omega$ and $d_\Omega \in \Phi\Omega$. These assumptions ensure that Theorem 7 becomes applicable. We will denote the reindexing for Φ by $_*$, while we overload the notation and specify the reindexing in both \mathbf{C} and $\mathbf{EM}(T)$ by $_*$.

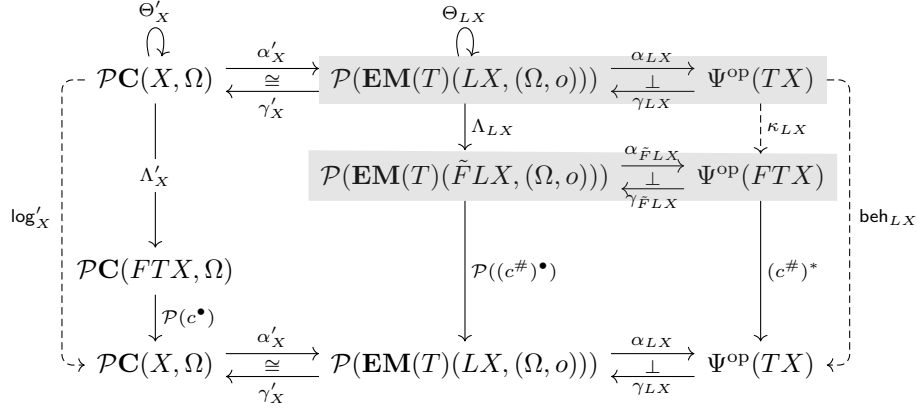
We focus on free algebras $LX = (TX, \mu_X)$ (over $X \in \mathbf{C}$) and apply Theorem 7 to the above-mentioned indexed category $\Psi \circ R$, which gives the adjoint situations depicted by shaded rectangles in Figure 2. We note that the middle hom-set $\mathbf{EM}(T)(LX, (\Omega, o))$ is isomorphic to $\mathbf{C}(X, \Omega)$ at the top left – due to the free-forgetful adjunction – with the bijection between the respective powersets witnessed by α', γ' . This allows us to define the logic function on the lattice $\mathcal{P}(\mathbf{C}(X, \Omega))$, which is a simpler structure than $\mathcal{P}(\mathbf{EM}(T)(LX, (\Omega, o)))$. In particular, formulas can then be evaluated directly on the state space X .

6.1 Logic and Behaviour Function for Coalgebras in Eilenberg-Moore

Recall from Section 4.3 that we need evaluation maps in the working category to define a logic function. So, to ensure **A3**, we assume a set Λ of evaluation maps for \tilde{F} , i.e., a family $(\tilde{F}(\Omega, o) \xrightarrow{ev_\lambda} (\Omega, o) \in \mathbf{EM}(T))_{\lambda \in \Lambda}$ of algebra homomorphisms. More concretely, a \mathbf{C} -arrow $ev_\lambda: F\Omega \rightarrow \Omega$ is such an algebra homomorphism if it satisfies $o \circ T ev_\lambda = ev_\lambda \circ F o \circ \zeta_\Omega$.

As in Section 4, this induces a natural transformation Λ . Since every homomorphism is also a map in \mathbf{C} , we can define Λ'_X , the predicate lifting on $\mathcal{PC}(X, \Omega)$:

$$\mathcal{PC}(X, \Omega) \xrightarrow{\alpha'_X} \mathcal{PEM}(T)(LX, (\Omega, o)) \xrightarrow{\Lambda_{LX}} \mathcal{PEM}(T)(\tilde{F}LX, (\Omega, o)) \xrightarrow{\mathcal{P}(R)} \mathcal{PC}(FTX, \Omega).$$



■ **Figure 2** The adjoint setup for algebras, where $\text{beh}_{LX} = (c^\#)^* \circ \kappa_{LX} \wedge \alpha_{LX}(\Theta_{LX})$ and $\text{log}'_X = \mathcal{P}(c^\bullet) \circ \Lambda'_X \cup \Theta_{LX}$.

Note that Λ' is a natural transformation (since α, Λ are natural transformations and R is a functor); on components it can be easily characterized as follows:

► **Lemma 24.** *We have that $\Lambda'_X(S) = \{ev_\lambda \circ Fo \circ FTh \mid h \in S\}$ where $S \subseteq \mathbf{C}(X, \Omega)$.*

That is, $\Lambda'_X(S)$ is obtained by first lifting the predicates from $\mathbf{C}(X, \Omega)$ to $\mathbf{C}(TX, \Omega)$ via the evaluation map $o: T\Omega \rightarrow \Omega$ and then to $\mathbf{C}(FTX, \Omega)$ via $ev_\lambda: F\Omega \rightarrow \Omega$. This process can be seen as applying a “double modality” for T and F .

We can now invoke the results of the previous chapter and assume that Λ is compatible with the closure induced by the adjunction, that is, we work without propositional operators (hence cl' , as mentioned in Assumption **A5**, is the identity), only constants, at first sight a strong property. We will however see in the next section that this always holds when F is a machine functor and we choose suitable evaluation maps.

The next theorem focusses on free algebras and is partly a corollary of Proposition 14 and Theorem 21. However there is a new component: instead of defining the logic function on (free) Eilenberg-Moore categories, reindexing via the determinized coalgebra $c^\#$, it is possible – as indicated above – to define it directly on arrows of type $X \rightarrow \Omega$ living in \mathbf{C} , reindexing with c . This coincides with the view that formulas should be evaluated on states in X rather than elements of TX . The diagram in Figure 2 outlines how to show this result.

► **Theorem 25.** *We fix a coalgebra $(c: X \rightarrow FTX) \in \mathbf{C}$. Assume that Λ_{LX} is compatible with the closure cl_{LX} and fix $\Theta_{LX} \subseteq \mathbf{EM}(T)(LX, \Omega)$ to ensure that **A4** holds.*

1. *Then the logic function $\text{log}_{LX} = \mathcal{P}((c^\#)^*) \circ \Lambda_{LX} \cup \Theta_{LX}$ is cl_{LX} -compatible.*
2. *For the behaviour function $\text{beh}_{LX} = (c^\#)^* \circ \kappa_{LX} \wedge \alpha_{LX}(\Theta_{LX})$ (where $\kappa_{LX} = \alpha_{\tilde{F}LX} \circ \Lambda_{LX} \circ \gamma_{LX}$), we have $\alpha_{LX}(\mu \text{log}_{LX}) = \nu \text{beh}_{LX}$.*
3. *Now define another logic function $\text{log}'_X = \mathcal{P}(c^\bullet) \circ \Lambda'_X \cup \Theta_{LX}$ with $\Theta_{LX} = \alpha'_X(\Theta'_X)$. It holds that $\alpha'_X \circ \text{log}'_X \circ \gamma'_X = \text{log}_{LX}$ and we obtain $\alpha_{LX}(\alpha'_X(\mu \text{log}'_X)) = \nu \text{beh}_X$.*

We hence consider a simple logic \mathcal{L}_{EM} for $\mathbf{EM}(T)$, where T is a monad on \mathbf{Set} :

$$\varphi \in \mathcal{L}_{\text{EM}} ::= \theta \mid [\lambda]\varphi \quad (\text{where } \theta \in \Theta, \lambda \in \Lambda)$$

Given a coalgebra $c: X \rightarrow FTX \in \mathbf{Set}$, each formula $\varphi \in \mathcal{L}_{\text{EM}}$ is interpreted as a function $\llbracket \varphi \rrbracket: X \rightarrow \Omega$, which is defined by structural induction as follows:

- Let $\varphi = \theta$. Then $\llbracket \varphi \rrbracket$ is given by a predefined constant $X \rightarrow \Omega$.
- Let $\varphi = [\lambda]\varphi'$. Then $\llbracket \varphi \rrbracket = ev_\lambda \circ Fo \circ FT\llbracket \varphi' \rrbracket \circ c$ (see definition of Λ'_X in Lemma 24).

► **Corollary 26.** *Under the requirements of Theorem 25, the logic \mathcal{L}_{EM} is expressive for the behavioural conformance beh_{LX} , i.e., $\alpha_{LX}(\alpha'_X(\{\llbracket \varphi \rrbracket \mid \varphi \in \mathcal{L}_{EM}\})) = \nu \text{beh}_{LX}$.*

6.2 The Machine Functor

Our next aim is to show that the machine functor has certain natural evaluation maps ensuring that the predicate lifting is cl-compatible (one of the conditions of Theorem 25). Throughout this section, we restrict ourselves to a monad T on **Set** and fix the *machine functor* $M = _{}^\Sigma \times B$ with $\Sigma \in \mathbf{Set}$ and $(B, b) \in \mathbf{EM}(T)$. Since all monads in **Set** are strong and B is endowed with a T -algebra structure $b: TB \rightarrow B$, there is a canonical distributive law ζ [19, Exercise 5.4.4]:

$$\zeta_X: T(X^\Sigma \times B) \xrightarrow{\langle a \mapsto T(\pi_a \circ \pi_1), b \circ T\pi_2 \rangle} (TX)^\Sigma \times B, \quad (1)$$

where $(\pi_i)_{i \in \{1,2\}}$ are the usual projections and $\pi_a: X^\Sigma \rightarrow X$ is the evaluation map ($\pi_a(g) = g(a)$ where $g: \Sigma \rightarrow X$). Now let \tilde{M} be the lifting of M to $\mathbf{EM}(T)$, induced by the ζ . We observe that the evaluation maps suggested by it arise from natural transformations in the sense of Lemma 18.

► **Proposition 27.**

1. *Let $a \in \Sigma$. Then $\eta_a: \tilde{M} \Rightarrow \text{Id}$ given by the composite $MX \xrightarrow{\pi_1} X^\Sigma \xrightarrow{\pi_a} X$ is a natural transformation.*
2. *Let $f: (B, b) \rightarrow (\Omega, o)$ be a homomorphism. Then $\eta'_f: \tilde{M} \Rightarrow \Omega$ given by the composite $MX \xrightarrow{\pi_2} B \xrightarrow{f} \Omega$ is a natural transformation.*

Thus, $ev_a = \eta_\Omega$, $ev_f = \eta'_\Omega$ satisfy the properties of Lemma 18, and if each evaluation map is of this form, then Λ is cl-compatible.

6.3 Alternative Formulation of Kantorovich Lifting

The behaviour function in Section 6.1 is based on the generalized Kantorovich lifting κ [2], which works as follows: given a pseudometric d on Y (here $Y = TX$), generate *all* non-expansive functions $Y \rightarrow \Omega$ wrt. d , lift these functions to $FY \rightarrow \Omega$ and from there generate a pseudometric on FY . However, κ_{LX} – since it is defined in an Eilenberg-Moore category – works subtly differently: it takes *all non-expansive functions that are algebra homomorphisms*. This looks natural in the categorical setting, but may pose problems if we implement the procedures. The standard (probabilistic) Kantorovich lifting can for instance be computed based on the Kantorovich-Rubinstein duality, by determining optimal transport plans [40].

Here, both types of liftings coincide at least on relevant metrics. To show this result, we first define an alternative way of lifting, as opposed to defining the lifting on T -algebra maps. Applying Theorem 7 on Ψ (rather than on $\Psi \circ R$) gives the adjunction $\alpha_X^C \dashv \gamma_X^C: \mathcal{PC}(X, \Omega) \rightleftarrows \Psi X^{\text{op}}$. Now consider the lifting $\kappa_{TX}^C = \alpha_{FTX}^C \circ \Lambda_{TX} \circ \gamma_{TX}^C$, where $\Lambda_{TX}: \mathcal{PC}(TX, \Omega) \rightarrow \mathcal{PC}(FTX, \Omega)$ is defined identically to Λ_{LX} .

► **Theorem 28.** *Assume that d is preserved by the co-closure, i.e. $d = \alpha_{LX}(\gamma_{LX}(d))$, the co-closure $\alpha_X^C \circ \gamma_X^C$ is the identity, and each evaluation map ev_λ arises from some natural transformation either of type $F \Rightarrow \text{Id}$ or $F \Rightarrow \Omega$. Then the two liftings $\kappa_{LX}, \kappa_{TX}^C$ coincide on d , i.e., $\kappa_{LX}(d) = \kappa_{TX}^C(d)$.*

The condition $d = \alpha_{LX}(\gamma_{LX}(d))$ is a necessary, but not a serious restriction: this property is typically satisfied by the \top metric and is preserved by the behaviour function. Hence during fixpoint iteration this invariant is preserved and the greatest fixpoints of beh_X based on either version of the Kantorovich lifting coincide (if the fixpoint is reached in ω steps).

In addition, if $\mathbf{C} = \mathbf{Set}$ and \mathcal{V} is an integral quantale, it can easily be shown that the co-closure $\alpha_X^{\mathbf{C}} \circ \gamma_X^{\mathbf{C}}$ is the identity. This enables us to concretely spell out the behaviour function for the case of the machine functor $M = _{}^{\Sigma} \times B$, provided that the conformances ΨX are (directed) pseudometrics.

► **Theorem 29.** *Assume that $\mathbf{C} = \mathbf{Set}$, $\Psi X = \mathbf{DPMet}_{\mathcal{V}}(X)$ (resp. $\Psi X = \mathbf{PMet}_{\mathcal{V}}(X)$) for an integral quantale \mathcal{V} and let $d_{\mathcal{V}} = [_, _]$ (resp. the symmetrized variant of $[_, _]$). Let $d: LX \times LX \rightarrow \mathcal{V}$ be a pseudometric that is preserved by the co-closure $\alpha_{LX} \circ \gamma_{LX}$. Assume that M is the machine functor and the family of evaluation maps is*

$$\{ev_a \mid a \in \Sigma\} \cup \{ev_f \mid f \in \mathcal{F} \subseteq \mathbf{EM}(T)((B, b), (\mathcal{V}, o))\}.$$

Then the corresponding behaviour function $\text{beh}_{LX}: \Psi(LX) \rightarrow \Psi(LX)$ is defined as follows: let $t_1, t_2 \in LX$ with $c^{\#}(t_i) = (b_i, g_i) \in B \times LX^{\Sigma}$:

$$\text{beh}_{LX}(d)(t_1, t_2) = \bigwedge_{a \in \Sigma} d(g_1(a), g_2(a)) \wedge \bigwedge_{f \in \mathcal{F}} d_{\mathcal{V}}(f(b_1), f(b_2)) \wedge \bigwedge_{\theta \in \Theta_{LX}} d_{\mathcal{V}}(\theta(t_1), \theta(t_2)),$$

The above function beh is co-continuous and fixpoint iteration terminates after ω steps.

7 Case Studies for the Linear-time Case

7.1 Workflow

We recall the parameters of our framework and set out a workflow that we follow in our case studies. Let F be a machine functor and T a monad on a category \mathbf{C} .

- Model systems as coalgebras of type $c: X \rightarrow FTX$ with a distributive law $\zeta: TF \Rightarrow FT$.
- Fix a truth value object $(\Omega, o) \in \mathbf{EM}(T)$ and $d_{\Omega} \in \Psi\Omega$.
- Define a fibration (indexed category) $\Phi = \Psi \circ R: \mathbf{EM}(T)^{\text{op}} \rightarrow \mathbf{Pos}$ by fixing an indexed category $\Psi: \mathbf{C}^{\text{op}} \rightarrow \mathbf{Pos}$ to define the conformances.
- Fix a set Λ of evaluation maps (predicate liftings) as homomorphisms $\tilde{F}(\Omega, o) \rightarrow (\Omega, o)$.
- Fix a set of constants $\Theta_X \subseteq \mathbf{C}(X, \Omega)$.

Note that the last four conditions correspond to Assumptions **A1-A5**, which are necessary to set up a logic and a behaviour function beh as defined in Section 6.1 and guarantee expressiveness of the resulting logic. Whenever we choose $\Psi X = \mathbf{DPMet}_{\mathcal{V}}(X)$ or $\Psi X = \mathbf{PMet}_{\mathcal{V}}(X)$ for an integral quantale \mathcal{V} , we can rely on the characterization of the fixpoint equation in Theorem 29.

We present one worked out case studies based on this workflow, two others are given in Appendix A.

7.2 Trace Distance for Probabilistic Automata

A *probabilistic automaton* [31] is a quadruple (X, Σ, μ, p) where for each state $x \in X$ and each possible action $a \in \Sigma$ there is a probability distribution $\mu_{x,a}$ on the possible successors in X , and where each state $x \in X$ has a payoff value $p(x) \in [0, 1]$. Following [34, 35], we model them as coalgebras in the Eilenberg-Moore setting as detailed in the table below.

$\mathbf{C} = \mathbf{Set}$, $F = _{}^\Sigma \times [0, 1]$, $T = \mathcal{D}$ $(B = [0, 1], b \text{ expectation})$ $c: X \rightarrow (\mathcal{D}X)^\Sigma \times [0, 1]$ $c^\# : \mathcal{D}X \rightarrow (\mathcal{D}X)^\Sigma \times [0, 1]$	Logic: evaluation maps: ev_a (Prop. 27(1)) $ev_*(f, r) = r$ (Prop. 27(2)) constants $\Theta_X = \emptyset$ formulas: $\varphi = [a_1] \cdots [a_n]^*$
$\Omega = [0, 1]$ (Ex. 2(2)) $o: \mathcal{D}[0, 1] \rightarrow [0, 1]$ expectation $\Psi(X) = \mathbf{PMet}_{[0,1]}(X)$ $d_\Omega(r, s) = r - s $	Behaviour function: $\text{beh}_{\mathcal{D}X}: \Psi(\mathcal{D}X) \rightarrow \Psi(\mathcal{D}X)$ $\text{beh}_{\mathcal{D}X}(d)(p_1, p_2) = \max\{\sup_{a \in \Sigma} d(g_1(a), g_2(a)), d_\Omega(r_1, r_2)\}$

Thus, given a formula $\varphi = [a_1] \cdots [a_n]^*$ and a state $x \in X$, $\llbracket \varphi \rrbracket(x)$ gives us the expected payoff after choosing actions according to the word $a_1 \dots a_n$. The distance of two states x_1, x_2 is hence the supremum of the difference of payoffs, over all words.

Expressiveness again follows from Corollary 26.

8 Conclusion

Related work. By now there is a large number of papers considering coalgebraic semantics beyond branching-time, for instance [15, 34, 27, 7]. Furthermore in the same period quite a wealth of results on the treatment of behavioural metrics in coalgebraic generality has been published [38, 2, 23, 12]. However, there is little work combining both linear-time semantics and behavioural metrics in the setting of coalgebra. In that respect we want to mention [13] that is based on the graded monad framework [27] and which investigates exactly this combination. However, different from the present paper, the focus is on the expressiveness of the logics with respect a graded semantics (that intuitively specifies the traces of a state). Hence, using the classification of the introduction, it studies the relationship of (i) and (ii).

Unlike other approaches, our main focus is on exploiting an adjunction (Galois connection) and fixpoint preservation results to obtain Hennessy-Milner theorems “for free”. We start by setting up a logic, characterizing the behavioural equivalence, and investigate under which circumstances we can derive a corresponding fixpoint characterization. The fixpoint equation might be defined on an infinite state space, but often there are finitary techniques that can be employed, such as reducing the state space to a finite subset, linear programming, up-to techniques, etc. In particular, for systems as in Appendix A.2 we are working on promising results (based on [3]) for deriving bounds for behavioural distances via finite witnesses using up-to techniques, even for infinite state spaces. The algorithmic angle of our approach is not yet fully worked out in the present paper but establishing fixpoint equations as we do here is a necessary first step in this direction.

Note that our concept deviates from the dual adjunction approach [21, 24, 25, 28] to coalgebraic modal logic. There the functor on the “logic universe” characterizes the *syntax* of the logics, while the semantics is instead given by a natural transformation. Nevertheless, it complements (at least when restricted to the classical case of Boolean predicates) the recent approach [37] that combines fibrations in the dual adjunction setup since having contravariant Galois connections between fibres (at a “local” level) is equivalent to having dual adjunctions between certain fibred categories (at a “global” level). It is unclear how to establish this correspondence in the setting of quantitative \mathcal{V} -valued predicates.

Future work. Currently the operators of the logic, given by cl' , are rather generic, although we instantiated them in special cases to ensure expressiveness (see in particular Sections 5 and 6.2). We envision a general theory to ensure expressiveness of the logics, similar to

Post’s functional completeness theorem [29], which characterizes complete sets of operators for the boolean case. This question is strongly related to the notion of an approximating family in [23] that has again close connections to compatibility as discussed in [4].

We will also study the condition requiring that a conformance (pseudometric) is preserved by the co-closure ($d = \alpha_{LX}(\gamma_{LX}(d))$). Previous results [4] suggest that this is related to the notion of (metric) congruence, as e.g. defined in [6], but the connection seems to be non-trivial.

Another avenue of research is to further investigate the quantale-valued logic for the branching case introduced in Section 5, to extend it to the undirected case and restrict to finitary operators.

References

- 1 Jiří Adámek, Horst Herrlich, and George E. Strecker. *Abstract and concrete categories: The joy of cats*. Wiley, 1990. Republished in: Reprints in Theory and Applications of Categories, No. 17 (2006) pp. 1–507. URL: <http://tac.mta.ca/tac/reprints/articles/17/tr17abs.html>.
- 2 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Coalgebraic behavioral metrics. *Logical Methods in Computer Science*, 14(3), 2018. Selected Papers of the 6th Conference on Algebra and Coalgebra in Computer Science (CALCO 2015).
- 3 Paolo Baldan, Barbara König, and Tommaso Padoan. Abstraction, up-to techniques and games for systems of fixpoint equations. In *Proc. of CONCUR ’20*, volume 171 of *LIPICs*, pages 25:1–25:20. Schloss Dagstuhl – Leibniz Center for Informatics, 2020.
- 4 Harsh Beohar, Sebastian Gurke, Barbara König, and Karla Messing. Hennessy-Milner theorems via Galois connections. In *Proc. of CSL ’23*, volume 252 of *LIPICs*, pages 12:1–12:18. Schloss Dagstuhl – Leibniz Center for Informatics, 2023.
- 5 Harsh Beohar, Sebastian Gurke, Barbara König, Karla Messing, Jonas Forster, Lutz Schröder, and Paul Wild. Expressive quantale-valued logics for coalgebras: an adjunction-based approach, 2023. arXiv:2310.05711. URL: <https://arxiv.org/abs/2310.05711>.
- 6 Filippo Bonchi, Alexandra Silva, and Ana Sokolova. The Power of Convex Algebras. In *Proc. of CONCUR ’17*, volume 85 of *LIPICs*, pages 23:1–23:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.
- 7 Corina Cîrstea. From branching to linear time, coalgebraically. *Fundamenta Informaticae*, 150(3-4):379–406, 2017.
- 8 Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proc. of POPL ’79*, pages 269–282. ACM Press, 1979.
- 9 Patrick Cousot and Radhia Cousot. Temporal abstract interpretation. In Mark N. Wegman and Thomas W. Reps, editors, *Proc. of POPL ’00*, pages 12–25. ACM, 2000.
- 10 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318:323–354, 2004.
- 11 Ulrich Dorsch, Stefan Milius, and Lutz Schröder. Graded monads and graded logics for the linear time - branching time spectrum. In Wan J. Fokkink and Rob van Glabbeek, editors, *Concurrency Theory, CONCUR 2019*, volume 140 of *LIPICs*, pages 36:1–36:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.36.
- 12 Jonas Forster, Sergey Goncharov, Dirk Hofmann, Pedro Nora, Lutz Schröder, and Paul Wild. Quantitative Hennessy-Milner Theorems via Notions of Density. In *Proc. of CSL ’23*, volume 252 of *LIPICs*, pages 22:1–22:20, 2023.
- 13 Jonas Forster, Lutz Schröder, Paul Wild, Harsh Beohar, Sebastian Gurke, Barbara König, and Karla Messing. Graded semantics and graded logics for Eilenberg-Moore coalgebras, 2023. arXiv:2307.14826. URL: <https://arxiv.org/abs/2307.14826>.
- 14 Alessandro Giacalone, Chi-Chang Jou, and Scott A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In Manfred Broy and Cliff B. Jones, editors, *Programming concepts and methods*, pages 443–458. North-Holland, 1990.

- 15 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4:11):1–36, 2007.
- 16 Dirk Hofmann, Gavin J. Seal, and Walter Tholen, editors. *Lax algebras*, pages 145–283. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2014. doi: 10.1017/CB09781107517288.005.
- 17 Bart Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1st edition, January 1999.
- 18 Bart Jacobs. Predicate logic for functors and monads. Available from author’s website, 2010. URL: <http://www.cs.ru.nl/~bart/PAPERS/predlift-indcat.pdf>.
- 19 Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. doi: 10.1017/CB09781316823187.
- 20 Bart Jacobs, Alexandra Silva, and Ana Sokolova. Trace semantics via determinization. In *Proc. of CMCS ’12*, pages 109–129. Springer, 2012. LNCS 7399.
- 21 Bartek Klin. Coalgebraic modal logic beyond sets. In *Proc. of MFPS ’07*, volume 173 of *ENTCS*, pages 177–201, 2007.
- 22 Yuichi Komorida, Shin-ya Katsumata, Nick Hu, Bartek Klin, and Ichiro Hasuo. Codensity games for bisimilarity. In *Proc. of LICS ’19*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785691.
- 23 Yuichi Komorida, Shin-ya Katsumata, Clemens Kupke, Jurriaan Rot, and Ichiro Hasuo. Expressivity of quantitative modal logics: Categorical foundations via codensity and approximation. In *Proc. LICS ’21*, pages 1–14. IEEE, 2021.
- 24 Clemens Kupke and Dirk Pattinson. Coalgebraic semantics of modal logics: An overview. *Theoretical Computer Science*, 412:5070–5094, 2011.
- 25 Clemens Kupke and Jurriaan Rot. Expressive logics for coinductive predicates. In *Proc. of CSL ’20*, volume 152 of *LIPICs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz Center for Informatics, 2020.
- 26 F. William Lawvere. Metric spaces, generalized logic, and closed categories. *Rendiconti del Seminario Matematico e Fisico di Milano*, 43(1):135–166, 1973. Republished in *Reprints in Theory and Applications of Categories* 1 (2002), 1–37. doi:10.1007/bf02924844.
- 27 Stefan Milius, Dirk Pattinson, and Lutz Schröder. Generic trace semantics and graded monads. In *Proc. of CALCO ’15*, volume 35 of *LIPICs*, pages 253–269. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015.
- 28 Dusko Pavlovic, Michael Mislove, and James Worrell. Testing semantics: Connecting processes and process logics. In *Proc. of AMAST ’06*, pages 308–322. Springer, 2006. LNCS 4019.
- 29 Francis Jeffrey Pelletier and Norman M. Martin. Post’s functional completeness theorem. *Notre Dame Journal of Formal Logic*, 31(2), 1990.
- 30 Damien Pous. Complete lattices and up-to techniques. In *Proc. of APLAS ’07*, pages 351–366. Springer, 2007. LNCS 4807.
- 31 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.
- 32 Jan Rutten. Universal coalgebra: A theory of systems. *Theoretical Computer Science*, 249:3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.
- 33 Lutz Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theoretical Computer Science*, 390(2):230–247, 2008. Foundations of Software Science and Computational Structures. doi:10.1016/j.tcs.2007.09.023.
- 34 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing the powerset construction, coalgebraically. In *Proc. of FSTTCS ’10*, volume 8, pages 272–283. Schloss Dagstuhl – Leibniz Center for Informatics, 2010. LIPICs.
- 35 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9(1:09), 2013.

- 36 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
- 37 Ruben Turkenburg, Harsh Beohar, Clemens Kupke, and Jurriaan Rot. Forward and Backward Steps in a Fibration. In Paolo Baldan and Valeria de Paiva, editors, *10th Conference on Algebra and Coalgebra in Computer Science (CALCO 2023)*, volume 270 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CALCO.2023.6.
- 38 Franck van Breugel and James Worrell. A behavioural pseudometric for probabilistic transition systems. *Theoretical Computer Science*, 331:115–142, 2005.
- 39 Rob van Glabbeek. The linear time – branching time spectrum I. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001.
- 40 Cédric Villani. *Optimal Transport – Old and New*, volume 338 of *A Series of Comprehensive Studies in Mathematics*. Springer, 2009.
- 41 Paul Wild and Lutz Schröder. A quantified coalgebraic van Benthem theorem. In *Proc. of FOSSACS '21*, volume 12650 of *LNCS*, pages 551–571. Springer, 2021.
- 42 Paul Wild and Lutz Schröder. Characteristic logics for behavioural hemimetrics via fuzzy lax extensions. *Logical Methods in Computer Science*, 18(2), 2022. doi:10.46298/lmcs-18(2:19)2022.
- 43 Paul Wild, Lutz Schröder, Dirk Pattinson, and Barbara König. A van Benthem theorem for fuzzy modal logic. In Anuj Dawar and Erich Grädel, editors, *Logic in Computer Science, LICS 2018*, pages 909–918. ACM, 2018. doi:10.1145/3209108.3209180.

A Case Studies for the Linear-time Case

A.1 Trace Equivalence for Labelled Transition Systems

We spell out a simple case study: trace equivalence [39] for labelled transition systems. The main ingredients are summarized in the table below.

$\mathbf{C} = \mathbf{Set}$, $F = _{}^{\Sigma}$, $T = \mathcal{P}$ $(B = 1)$ $c: X \rightarrow (\mathcal{P}X)^{\Sigma}$ $c^{\#}: \mathcal{P}X \rightarrow (\mathcal{P}X)^{\Sigma}$	Logic: evaluation maps: ev_a (Prop. 27(1)) constants $\Theta_X = \{1\}$, constant 1-function formulas: $\varphi = [a_1] \cdots [a_n]1$
$\Omega = \mathbf{2}$ (Ex. 2(1)) $o: \mathcal{P}\mathbf{2} \rightarrow \mathbf{2}$ supremum $\Psi(X)$: equivalences on X d_{Ω} : equality on Ω	Behaviour function: $\text{beh}_{\mathcal{P}X}: \Psi(\mathcal{P}X) \rightarrow \Psi(\mathcal{P}X)$ $\text{beh}_{\mathcal{P}X}(R)(U, V) = (U = \emptyset \Leftrightarrow V = \emptyset) \wedge$ $\quad \forall a \in \Sigma (c^{\#}(U)(a), c^{\#}(V)(a)) \in R$

The modality $[a]$ boils down to the standard diamond modality (due to the definition of o); a state $x \in X$ satisfies $\varphi = [a_1] \cdots [a_n]1$ iff there exists a trace $a_1 \cdots a_n$ from x . The constant 1 is needed to start building formulas and to distinguish the empty set from a non-empty set on $LX = \mathcal{P}X$. (Note that $\Theta_{LX} = \alpha'_X(\Theta_X) = \{\tilde{1}\}$ with $\tilde{1}(Y) = 0$ iff $Y = \emptyset$.) Its role cannot be taken by a constant modality or an operator, since those have to be homomorphisms in $\mathbf{EM}(\mathcal{P})$, hence sup-preserving.

Expressiveness of trace logic $\mathcal{L}_{\mathbf{EM}}$ now directly follows from Corollary 26.

A.2 Directed Fuzzy Trace Distance

We now consider directed trace distances for weighted transition systems over a generic quantale.

We work with the “fuzzy” monad $T = \mathcal{P}_{\mathcal{V}}$ (aka \mathcal{V} -valued powerset monad [16, Remark 1.2.3]) on \mathbf{Set} that is defined as $\mathcal{P}_{\mathcal{V}} = \mathcal{V}^X$ on objects and as $Tf(g)(y) = \bigvee_{f(x)=y} g(x)$ (for $f: X \rightarrow Y$) on arrows. Its unit $\eta_X: X \rightarrow \mathcal{P}_{\mathcal{V}}X$ is given by $\eta_X(x)(x') = 1$ if $x = x'$

and 0 (the empty join) otherwise. Multiplication $\mu_X: \mathcal{P}_V \mathcal{P}_V X \rightarrow \mathcal{P}_V X$ is defined as $\mu_X(G)(x) = \bigvee_{g \in \mathcal{P}_V X} G(g) \otimes g(x)$. Note that for $V = 2$ (cf. Example 2(1)) T corresponds to the powerset monad \mathcal{P} .

$\mathbf{C} = \mathbf{Set}, F = _ \Sigma, T = \mathcal{P}_V$ $(B = 1)$ $c: X \rightarrow (\mathcal{P}_V X)^\Sigma$ $c^\#: \mathcal{P}_V X \rightarrow (\mathcal{P}_V X)^\Sigma$	Logic: evaluation maps: ev_a (Prop. 27(1)) constants $\Theta_X = \{1\}$, constant 1-function formulas: $\varphi = [a_1] \cdots [a_n]1$
$\Omega = V, o: \mathcal{P}_V V \rightarrow V$ $g \mapsto \bigvee_{v \in V} g(v) \otimes v$ $\Psi(X) = \mathbf{DPMet}_V(X)$ $d_\Omega(v, v') = [v, v']$	Behaviour function: $\text{beh}_{\mathcal{P}_V X}: \Psi(\mathcal{P}_V X) \rightarrow \Psi(\mathcal{P}_V X)$ $\text{beh}_{\mathcal{P}_V X}(d)(g_1, g_2) = \bigwedge_{a \in \Sigma} d(c^\#(g_1)(a), c^\#(g_2)(a)) \wedge$ $\quad \left[\bigvee_{x \in X} g_1(x), \bigvee_{x \in X} g_2(x) \right]$

Evaluating a formula $\varphi = [a_1] \dots [a_n]1$ on a state $x_0 \in X$ results in

$$\llbracket \varphi \rrbracket(x_0) = \bigvee \left\{ \bigotimes_{0 \leq i < n} c(x_i)(a_{i+1})(x_{i+1}) \mid x_1 \dots x_n \in X^n \right\}.$$

This follows directly from structural induction on φ , from distributivity and from the evaluation of the modality $[a]\varphi'$:

$$\llbracket [a]\varphi' \rrbracket(x) = \bigvee_{y \in X} c(x)(a)(y) \otimes \llbracket \varphi' \rrbracket(y).$$

Intuitively we check how well x can match the trace $a_1 \dots a_n$, where $c(x)(a)(y)$ measures the degree to which x can make an a -transition to y .

The second part of the minimum in the definition of beh stems from the constants $\Theta_X = \{1\}$, since $\Theta_{LX} = \alpha'_X(\Theta_X) = \{\tilde{1}\}$ with $\tilde{1}(h) = \bigvee_{x \in X} h(x)$ for $h: X \rightarrow V$. Without it, the fixpoint iteration would stabilize at the constant 1-pseudometric.

Expressiveness again follows from Corollary 26. Expressiveness of a logic for *symmetric* fuzzy trace distance has already been shown in previous work [13].

A Characterization of Efficiently Compilable Constraint Languages

Christoph Berkholz  

Technische Universität Ilmenau, Germany

Stefan Mengel  

Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France

Hermann Wilhelm  

Technische Universität Ilmenau, Germany

Abstract

A central task in *knowledge compilation* is to compile a CNF-SAT instance into a succinct representation format that allows efficient operations such as testing satisfiability, counting, or enumerating all solutions. Useful representation formats studied in this area range from ordered binary decision diagrams (OBDDs) to circuits in decomposable negation normal form (DNNFs).

While it is known that there exist CNF formulas that require exponential size representations, the situation is less well studied for other types of constraints than Boolean disjunctive clauses. The *constraint satisfaction problem* (CSP) is a powerful framework that generalizes CNF-SAT by allowing arbitrary sets of constraints over any finite domain. The main goal of our work is to understand for which type of constraints (also called the *constraint language*) it is possible to efficiently compute representations of polynomial size. We answer this question completely and prove two tight characterizations of efficiently compilable constraint languages, depending on whether target format is structured.

We first identify the combinatorial property of “strong blockwise decomposability” and show that if a constraint language has this property, we can compute DNNF representations of linear size. For all other constraint languages we construct families of CSP-instances that provably require DNNFs of exponential size. For a subclass of “strong *uniformly* blockwise decomposable” constraint languages we obtain a similar dichotomy for *structured* DNNFs. In fact, strong (uniform) blockwise decomposability even allows efficient compilation into multi-valued analogs of OBDDs and FBDDs, respectively. Thus, we get complete characterizations for all knowledge compilation classes between O(B)DDs and DNNFs.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Theory of computation → Complexity theory and logic

Keywords and phrases constraint satisfaction, knowledge compilation, dichotomy, DNNF

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.11

Related Version *Full Version*: <https://arxiv.org/abs/2311.10040> [5]

Funding *Christoph Berkholz*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - project number 414325841.

Stefan Mengel: This work has been partially supported by the ANR project EQUUS ANR-19-CE48-0019.

Hermann Wilhelm: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - project number 414325841.



© Christoph Berkholz, Stefan Mengel, and Hermann Wilhelm;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 11; pp. 11:1–11:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

One of the main aims of *knowledge compilation* is to encode solution sets of computational problems into a succinct but usable form [25]. Typical target formats for this compilation process are different forms of decision diagrams [44] or restricted classes of Boolean circuits. One of the most general representation formats are circuits in *decomposable negation normal form* (DNNF), which have been introduced in [23] as a compilation target for Boolean functions. Related notions, which also rely on the central decomposability property, have been independently considered in databases [38, 37] and circuit complexity [41, 1]. Besides these, DNNF circuits and related compilation classes have also been proven useful in areas like probabilistic inference [26], constraint satisfaction [31, 6, 4, 34], MSO evaluation [2], QBF solving [15], to name a few. The DNNF representation format has become a popular data structure because it has a particularly good balance between generality and usefulness [25]. There has also been a large amount of practical work on compiling solution sets into DNNF or its fragments, see e.g. [33, 24, 36, 17, 39, 31]. In all these works, it is assumed that the solutions to be compiled are given as a system of constraints, often as a set of disjunctive Boolean clauses, i. e., in conjunctive normal form (CNF).

In this setting, however, strong lower bounds are known: it was shown that there are CNF-formulas whose representation as DNNF requires exponential size [7, 8, 14, 3]. The constraints out of which the hard instances in [7, 3, 14] are constructed are very easy – they are only monotone clauses of two variables. Faced with these negative results, it is natural to ask if there are any classes of constraints that guarantee efficient compilability into DNNF.

We answer this question completely and prove a tight characterization for every constraint language Γ . We first examine the combinatorial property of *strong blockwise decomposability* and show that if a constraint language Γ has this property, any system of constraints over Γ can be compiled into a DNNF representation of linear size within polynomial time. Otherwise, there are systems of constraints that require exponential size DNNF representations. In the tractable case, one can even compile to the restricted fragment of *free decision diagrams* (FDD) that are in general known to be exponentially less succinct than DNNF [44, 25].

We also consider the important special case of so-called structured DNNF [40] which are a generalization of the well-known ordered binary decision diagrams [9]. We show that there is a restriction of strong blockwise decomposability that determines if systems of constraints over a set Γ can be compiled into structured DNNF in polynomial time. In the tractable case, we can in fact again compile into a restricted fragment, this time ordered decision diagrams (ODD).

Let us stress that all our lower bounds are unconditional and do not depend on any unproven complexity assumptions.

Further related work. Our work is part of a long line of work in constraint satisfaction, where the goal is to precisely characterize those constraint languages that are “tractable”. Starting with the groundbreaking work of Schaefer [42] which showed a dichotomy for deciding consistency of systems of *Boolean* constraints, there has been much work culminating in the dichotomy for general constraint languages [11, 45]. Beyond decision, there are dichotomies for counting [19, 10, 27], enumeration [21], optimization [18, 30, 20] and in the context of valued CSPs [43, 13]. Note that the hardness part (showing that certain constraint languages do not admit efficient algorithms) always relies on some complexity theoretic assumption.

The complexity of constraint satisfaction has also been studied “from the other side”, where the constraint language is unrestricted and the structure of the *constraint network* (i. e. how the constraints are arranged) has been analyzed. In this setting, characterizations

of (bounded arity) constraint networks have been obtained for the deciding the existence of solutions [29] and counting solutions [22] (again, under some complexity theoretic assumption), while for enumerating solutions only partial results exist [12]. Tractability classifications of the constraint network have also been obtained in the context of valued CSPs [16]. Recently, an unconditional characterization of (bounded arity) constraint networks that allow efficient compilation into DNNFs has been proven [6].

Out of these works, our results are most closely related to the counting dichotomy of [27]. The tractable classes we obtain in our dichotomies are also tractable for counting, and in fact we use the known counting algorithm in our compilation algorithms as a subroutine. Also, the tractability criterion has a similar flavor, but wherever in [27] the count of elements in certain relations is important, for our setting one actually has to understand their structure and how exactly they decompose. That said, while the tractability criterion is related to the one in [27], the techniques to show our results are very different. In particular, where [27] use reductions from $\#P$ -complete problems to show hardness, we make an explicit construction and then use communication complexity to get strong, unconditional lower bounds. Also, our two algorithms for construction ODDs and FDDs are quite different to the counting algorithm.

Outline of the paper. After some preliminaries in Section 2, we introduce the decomposability notions that we need to formulate our results in Section 3. We also give the formal formulation of our main results there. In Section 4, we show some of the properties of the constraints we defined in the section before. These properties will be useful throughout the rest of the paper. In Section 5, we present the algorithms for the positive cases of our dichotomies, then, in Section 6, we show the corresponding lower bounds. We specialize our results to the case of Boolean relations in Section 7, showing that the tractable cases are essentially only equalities and disequalities. Finally, we conclude in Section 8.

2 Preliminaries

Constraints. Throughout the paper we let D be a finite *domain* and X a finite set of *variables* that can take values over D . We typically denote variables by u, v, w, x, y, z and domain elements by a, b, c, d . A k -tuple $(x_1, \dots, x_k) \in X^k$ of variables is also denoted by $\vec{x} = x_1x_2 \cdots x_k$ and we let $\tilde{x} := \{x_1, \dots, x_k\}$ be the set of variables occurring in \vec{x} . We use the same notation for tuples of domain elements. A k -ary *relation* R (over D) is a set $R \subseteq D^k$. A k -ary *constraint* (over X and D), denoted by $R(\vec{x})$, consists of a k -tuple of (not necessarily distinct) variables $\vec{x} \in X^k$ and a k -ary relation $R \subseteq D^k$. For a constraint $R(\vec{x})$ we call \tilde{x} the *scope* of the constraint and R the *constraint relation*. The *solution set* of a constraint $R(x_1, \dots, x_k)$ is defined by $\text{sol}(R(x_1, \dots, x_k)) := \{\beta \mid \beta: \{x_1, \dots, x_k\} \rightarrow D; (\beta(x_1), \dots, \beta(x_k)) \in R\}$. Since the order of the columns in a constraint relation is not of great importance for us, we often identify constraints with their solution set and treat sets \mathcal{S} of mappings from $Y \subseteq X$ to D as a constraint with scope Y and solution set \mathcal{S} . Moreover, for readability we will often write, e. g., “a constraint $R(x, y, \vec{w})$, such that x and $y \dots$ ” when we do not strictly require x and y to be at the first and second position and actually refer to any constraint having x and y in its scope.

CSP-instance and constraint language. A *constraint satisfaction instance* $I = (X, D, C)$ consists of a finite set of variables X , a finite domain D and a finite set C of constraints. The solution set $\text{sol}(I) := \{\alpha \mid \alpha: X \rightarrow D; \alpha|_{\tilde{x}} \in \text{sol}(R(\vec{x})) \text{ for all } R(\vec{x}) \in C\}$ of an instance I is

the set of mappings from X to D that satisfies all constraints. A *constraint language* Γ is a finite set of relations (over some finite domain D). A constraint satisfaction instance I is a $\text{CSP}(\Gamma)$ -instance if every constraint relation R occurring in I is contained in the constraint language Γ .

Conjunction and projection. A *conjunction* $R(\vec{u}) = S(\vec{v}) \wedge T(\vec{w})$ of two constraints $S(\vec{v})$ and $T(\vec{w})$ defines a constraint $R(\vec{u})$ over $\vec{u} = u_1 \cdots u_\ell$ with scope $\tilde{u} = \tilde{v} \cup \tilde{w}$ and constraint relation $R := \{(\beta(u_1), \dots, \beta(u_\ell)) \mid \beta: \tilde{u} \rightarrow D; \beta|_{\tilde{v}} \in \text{sol}(S(\vec{v})) \text{ and } \beta|_{\tilde{w}} \in \text{sol}(T(\vec{w}))\}$.¹ If $\tilde{v} \cap \tilde{w} = \emptyset$, then a conjunction is called a (*Cartesian*) *product* and written $R(\vec{u}) = S(\vec{v}) \times T(\vec{w})$. For a constraint $R(\vec{x})$ and a set $Y \subseteq \tilde{x}$ we let the *projection* $\pi_Y R(\vec{x})$ be the constraint $S(\vec{y})$ with scope $\tilde{y} = Y$ obtained by projecting the constraint relation to corresponding coordinates, i. e., for $\vec{x} = x_1 \cdots x_k$ let $1 \leq i_1 < \cdots < i_\ell \leq k$ s. t. $\{i_1, \dots, i_\ell\} = \{i \mid x_i \in Y\}$, $\vec{y} := x_{i_1} \cdots x_{i_\ell}$, and $S := \{(a_{i_1}, \dots, a_{i_\ell}) \mid (a_1, \dots, a_k) \in R\}$.

Formulas and pp-definability. A Γ -*formula* $F(\vec{x}) = \bigwedge_i S_i(\vec{x}_i)$ is a conjunction over several constraints whose constraint relations are in Γ . To avoid notational clutter, we use “ $F(\vec{x})$ ” for both, the conjunction as syntactic expression *and* the constraint defined by this formula. Note that any $\text{CSP}(\Gamma)$ -instance $I = (X, D, C)$ corresponds to a Γ -formula $F(\vec{x}) = \bigwedge_{R_i(\vec{x}_i) \in C} R_i(\vec{x}_i)$ with $\tilde{x} = X$ and $\text{sol}(F(\vec{x})) = \text{sol}(I)$. Thus, we can treat $\text{CSP}(\Gamma)$ -instances as Γ -formulas and vice versa. A *primitive positive (pp)* formula over Γ is an expression $F(\vec{y}) = \pi_{\tilde{y}}(\bigwedge_i S_i(\vec{x}_i) \wedge \bigwedge_{(j,k)} x_j = x_k)$ consisting of a projection applied to a $\Gamma \cup \{(a, a) \mid a \in D\}$ -formula, which uses constraint relations from Γ and the equality constraint. Note that conjunctions of pp-formulas can be written as pp-formula by putting one projection at the front and renaming variables. A constraint is *pp-definable* over Γ , if it can be defined by a pp-formula over Γ . Moreover, a relation $R \subseteq D^k$ is *pp-definable* over Γ if it is the constraint relation of a pp-definable constraint $R(x_1, \dots, x_k)$ for pairwise distinct x_1, \dots, x_k . The co-clone $\langle \Gamma \rangle$ of Γ is the set of all pp-definable relations over Γ .

Selection. It is often helpful to use additional unary relations $S \subseteq D$ and to write $U_S(x)$ for the constraint $S(x)$ with scope $\{x\}$ and constraint relation S . We use $U_a(x)$ as an abbreviation for $U_{\{a\}}(x)$. For a constraint $R(\vec{u})$, a variable $x \in \tilde{u}$, and a domain element $a \in D$ we define the *selection* $R(\vec{u})|_{x=a}$ be the constraint obtained by forcing x to take value a , that is, $R(\vec{u})|_{x=a} := R(\vec{u}) \wedge U_a(x)$. Similarly, for a set $S \subseteq D$ we write $R(\vec{u})|_{x \in S} := R(\vec{u}) \wedge U_S(x)$.

DNNF. We will be interested in circuits representing assignments of variables to a finite set of values. To this end, we introduce a multi-valued variant of DNNF; we remark that usually DNNF are only defined over the Boolean domain $\{0, 1\}$ [23], but the extension we make here is straightforward and restricted variants have been studied e.g. in [31, 4, 35, 28] under different names.

Let X be a set of variables and D be a finite set of values. A circuit over the operations \times and \cup is a directed acyclic graph with a single sink, called output gate, and whose inner nodes, called gates, are labeled with \times or \cup . The source-nodes, called inputs of the circuit, are labeled by expressions of the form $x \mapsto a$ where $x \in X$ and $a \in D$. We say that \times -gate v is *decomposable* if there are no two inputs labeled with $x \mapsto a$ and $x \mapsto b$ (with possibly

¹ Again, we may just write “ $S(\vec{v}) \wedge T(\vec{w})$ ” instead of “ $R(\vec{u}) = S(\vec{v}) \wedge T(\vec{w})$ ” if the order or multiple occurrences of variables in \vec{u} does not matter (the solution set $\text{sol}(R(\vec{u}))$ is always the same).

$a = b$) that have a path to v going through different children of v . A DNNF is a circuit in which all \times -gates are decomposable. Note that in a DNNF, for every \times -gate v , every variable $x \in X$ can only appear below one child of v .

For every DNNF O we define the set $S(O)$ of assignments *captured* by O inductively:

- The set captured by an input v with label $x \mapsto d$ is the single element set $S(v) = \{x \mapsto d\}$.
- For a \cup -gate with children v_1, v_2 , we set $S(v) := S(v_1) \cup S(v_2)$.
- For a \times -gate with children v_1, v_2 , we set $S(v) := S(v_1) \times S(v_2)$ where for two assignments $a : X_1 \rightarrow D$ and $b : X_2 \rightarrow D$, we interpret (a, b) as the joined assignment $c : X_1 \cup X_2 \rightarrow D$ with $c(x) := \begin{cases} a(x), & \text{if } x \in X_1 \\ b(x), & \text{if } x \in X_2 \end{cases}$.

We define $S(O) := S(v_o)$ where v_o is the output gate of O . Note that $S(O)$ is well-defined in the case of \times , since X_1 and X_2 are disjoint because of decomposability. Finally, we say that O accepts an assignment α to X if there is an assignment $\beta \in S(O)$ such that α is an extension of β to all variables in X . We say that O represents a constraint $C(\vec{x})$ if O accepts exactly the assignments in $\text{sol}(C(\vec{x}))$.

A v -tree of a variable set X is a rooted binary tree whose leaves are in bijection to X . For a v -tree T of X and a node t of T we define $\text{var}(t)$ to be the variables in X that appear as labels in the subtree of T rooted in t . We say that a DNNF O over X is *structured by* T if for every sub-representation O' of O there is a node t in T such that O' is exactly over the variables $\text{var}(t)$ [40]. We say that O is *structured* if there is a v -tree T of X such that O is structured by T .

We will use the following basic results on DNNF which correspond to projection and selection on constraints.

► **Lemma 1.** *Let $C(\vec{u})$ be a constraint for which there exists a DNNF of size s , $x \in \vec{u}$ and $A \subseteq D$. Then there exists a DNNF for the selection $C(\vec{u})|_{x \in A}$ of size $\leq s$.*

► **Lemma 2.** *Let $C(\vec{u})$ be a constraint for which there exists a DNNF of size s and $\tilde{v} \in \vec{u}$. Then there exists a DNNF for the projection $\pi_{\tilde{v}}(C(\vec{u}))$ of size $\leq s$.*

Decision Diagrams. A decision diagram O over a variable set X is a directed acyclic graph with a single source and two sinks and in which all non-sinks have $|D|$ outgoing edges. The sinks are labeled with 0 and 1, respectively, while all other nodes are labeled with variables from x . For every non-sink, the $|D|$ outgoing edges are labeled in such a way that every value in D appears in exactly one label. Given an assignment α to X , the value computed by O is defined as follows: we start in the source and iteratively follow the edge labeled by $\alpha(x)$ where x is the label of the current node. We continue this process until we end up in a leaf whose label then gives the value of O on α . Clearly, this way O computes a constraint over X with relation $\{\alpha \mid O \text{ computes } 1 \text{ on input } \alpha\}$.

We are interested in decision diagrams in which on every source-sink-path every variable appears at most once as a label. We call these diagrams *free decision diagrams (FDD)*. An FDD for which there is an order π such that when a variable x appears before y on a path then x also appears before y in π is called *ordered decision diagrams*. We remark that FDD and ODD are in the literature mostly studied for the domain $\{0, 1\}$ in which case they are called FBDD and OBDD, respectively, where the ‘‘B’’ stands for binary. Note also that there is an easy linear time translation of FDD into DNNF and ODD into structured DNNF, see e.g. [25]. In the other direction there are no efficient translations, see again [25].

Rectangles. Let \vec{u} be a variable vector, $\tilde{u} = \tilde{x} \cup \tilde{y}$ and $\tilde{x} \cap \tilde{y} = \emptyset$. Then we say that the constraint $R(\vec{u})$ is a (*combinatorial*) *rectangle* with respect to the partition (\tilde{x}, \tilde{y}) if and only if $R(\vec{u}) = \pi_{\tilde{x}}(R(\vec{u})) \times \pi_{\tilde{y}}(R(\vec{u}))$. Let $Z \subseteq \tilde{u}$. Then we call the partition (\tilde{x}, \tilde{y}) *Z-balanced* if $\frac{|Z|}{3} \leq |\tilde{x} \cap Z| \leq \frac{2|Z|}{3}$. A constraint $R(\vec{u})$ is called a *Z-balanced rectangle* if it is a rectangle with respect to a Z-balanced partition. A *Z-balanced rectangle cover* \mathcal{R} of a constraint $R(\vec{u})$ is defined to be a set of Z-balanced rectangles such that $\text{sol}(R(\vec{u})) := \bigcup_{\tau \in \mathcal{R}} \text{sol}(\tau(\vec{u}))$. The size of \mathcal{R} is the number of rectangles in it.

3 Blockwise decomposability

In this section we introduce our central notion of blockwise and uniformly blockwise decomposable constraints and formulate our main theorems that lead to a characterization of efficiently representable constraint languages.

The first simple insight is the following. Suppose two constraints $S(\vec{v})$ and $T(\vec{w})$ with disjoint scopes are efficiently representable, e.g., by a small ODD. Then their Cartesian product $R(\vec{v}, \vec{w}) = S(\vec{v}) \times T(\vec{w})$ also has a small ODD: given an assignment (\vec{a}, \vec{b}) , we just need to check independently whether $\vec{a} \in S$ and $\vec{b} \in T$, for example, by first using the ODD for $S(\vec{v})$ and then using the ODD for $T(\vec{w})$. Thus, if a constraint can be expressed as a Cartesian product of two constraints, we only have to investigate whether the two parts are easy to represent. This brings us to our first definition.

► **Definition 3.** Let $R(\vec{u})$ be a constraint and (V_1, \dots, V_ℓ) be a partition of its scope. We call $R(\vec{u})$ decomposable w.r.t. (V_1, \dots, V_ℓ) if $R(\vec{u}) = \pi_{V_1}(R(\vec{u})) \times \dots \times \pi_{V_\ell}(R(\vec{u}))$. A constraint $R(\vec{u})$ is indecomposable if it is only decomposable w.r.t. trivial partitions (V_1, \dots, V_ℓ) where $V_i = \emptyset$ or $V_i = \tilde{u}$ for $i \in [\ell]$.

Next, we want to relax this notion to constraints that are “almost” decomposable. Suppose we have four relations S_1, S_2 of arity s and T_1, T_2 of arity t and let a, b be two distinct domain elements. Let

$$R := (\{(a, a)\} \times S_1 \times T_1) \cup (\{(b, b)\} \times S_2 \times T_2). \quad (1)$$

The constraint $R(x, y, \vec{v}, \vec{w})$ may now not be decomposable in any non-trivial variable partition. However, after fixing values for x and y the remaining selection $R(x, y, \vec{v}, \vec{w})|_{x=c, y=d}$ is decomposable in (\vec{v}, \vec{w}) for any pair $(c, d) \in D^2$. Thus, an ODD could first read values for x, y and then use ODDs for $S_1(\vec{v})$ and $T_1(\vec{w})$ if $x = y = a$, ODDs for $S_2(\vec{v})$ and $T_2(\vec{w})$ if $x = y = b$, or reject otherwise. This requires, of course, that $S_1(\vec{v})$ and $S_2(\vec{v})$, as well as $T_1(\vec{w})$ and $T_2(\vec{w})$, have small ODDs over the *same* variable order. For FDDs and DNNFs, however, we would not need this requirement on the variable orders.

To reason about the remaining constraints after two variables have been fixed, it is helpful to use the following matrix notation. Let $R(\vec{u})$ be a constraint and $x, y \in \tilde{u}$ be two variables in its scope. The *selection matrix* $M_{x,y}^R$ is the $|D| \times |D|$ matrix where the rows and columns are indexed by domain elements $a_i, a_j \in D$ and the entries are the constraints

$$M_{x,y}^R[a_i, a_j] := \pi_{\tilde{u} \setminus \{x, y\}}(R(\vec{u})|_{x=a_i, y=a_j}). \quad (2)$$

► **Example 4.** Let $D = \{a, b, c\}$ and $R(x, y, z, v)$ a constraint with constraint relation $R = \{(a, a, a, a), (b, b, a, b), (b, b, a, c), (b, b, c, c), (c, b, c, a)\}$. The selection matrix in x and y is depicted below, where the first line and column are the indices from D and the matrix entries contain the constraint relations of the corresponding constraints $M_{x,y}^R[a_i, a_j](z, v)$:

$$\left(\begin{array}{c|ccc} x \setminus y & a & b & c \\ \hline a & \{(a, a)\} & \emptyset & \emptyset \\ b & \emptyset & \{(a, b), (a, c), (c, c)\} & \emptyset \\ c & \emptyset & \{(c, a)\} & \emptyset \end{array} \right) \quad (3)$$

A *block* in the selection matrix is a subset of rows $A \subseteq D$ and columns $B \subseteq D$. We also associate with a block (A, B) the corresponding constraint $R(\vec{u})|_{x \in A, y \in B}$. A selection matrix is a *proper block matrix*, if there exist pairwise disjoint $A_1, \dots, A_k \subseteq D$ and pairwise disjoint $B_1, \dots, B_k \subseteq D$ such that for all $a_i, a_j \in D$:

$$\text{sol}(M_{x,y}^R[a_i, a_j]) \neq \emptyset \iff \text{there is } \ell \in [k] \text{ such that } a_i \in A_\ell \text{ and } a_j \in B_\ell. \quad (4)$$

The selection matrix in Example 4 is a proper block matrix with $A_1 = \{a\}$, $A_2 = \{b, c\}$, $B_1 = \{a\}$, $B_2 = \{b\}$. We will make use of the following alternative characterization of proper block matrices. The simple proof is similar to [27, Lemma 1].

► **Lemma 5.** *A selection matrix $M_{x,y}^R$ is a proper block matrix if and only if it has no 2×2 -submatrix with exactly one empty entry.*

Now we can define our central tractability criterion for constraints that have small ODDs, namely that any selection matrix is a proper block matrix whose blocks are decomposable over the same variable partition that separates x and y .

► **Definition 6** (Uniform blockwise decomposability). *A constraint $R(\vec{u})$ is uniformly blockwise decomposable in x, y if $M_{x,y}^R$ is a proper block matrix with partitions (A_1, \dots, A_k) , (B_1, \dots, B_k) and there is a partition (V, W) of \tilde{u} with $x \in V$ and $y \in W$ such that each block $R(\vec{u})|_{x \in A_i, y \in B_i}$ is decomposable in (V, W) . A constraint $R(\vec{u})$ is uniformly blockwise decomposable if it is uniformly decomposable in any pair $x, y \in \tilde{u}$.*

In the non-uniform version of blockwise decomposability, it is allowed that the blocks are decomposable over different partitions. This property will be used to characterize constraints having small FDD representations.

► **Definition 7** (Blockwise decomposability). *A constraint $R(\vec{u})$ is blockwise decomposable in x, y if $M_{x,y}^R$ is a proper block matrix with partitions (A_1, \dots, A_k) , (B_1, \dots, B_k) and for each $i \in [k]$ there are $V_i, W_i \subseteq \tilde{u}$ with $x \in V_i$ and $y \in W_i$ such that each block $R(\vec{u})|_{x \in A_i, y \in B_i}$ is decomposable in (V_i, W_i) . A constraint $R(\vec{u})$ is blockwise decomposable if it is decomposable in any pair $x, y \in \tilde{u}$.*

Note that every uniformly blockwise decomposable relation is also blockwise decomposable. The next example illustrates that the converse does not hold.

► **Example 8.** Consider the 4-ary constraint relations

$$R_1 := \{(a, a, a, a), (a, b, a, b), (b, a, b, a), (b, b, b, b)\} \quad (5)$$

$$R_2 := \{(c, c, c, c), (c, d, d, c), (d, c, c, d), (d, d, d, d)\} \quad (6)$$

$$R := R_1 \cup R_2 \quad (7)$$

Then the selection matrix $M_{x,y}^R$ of the constraint $R(x, y, u, v)$ has two non-empty blocks:

$$M_{x,y}^R = \left(\begin{array}{c|cccc} x \setminus y & a & b & c & d \\ \hline a & \{(a, a)\} & \{(a, b)\} & \emptyset & \emptyset \\ b & \{(b, a)\} & \{(b, b)\} & \emptyset & \emptyset \\ c & \emptyset & \emptyset & \{(c, c)\} & \{(d, c)\} \\ d & \emptyset & \emptyset & \{(d, c)\} & \{(d, d)\} \end{array} \right) \quad (8)$$

The first block $R(x, y, u, v)|_{x \in \{a,b\}, y \in \{a,b\}} = R_1(x, y, u, v)$ is decomposable in $\{x, u\}$ and $\{y, v\}$, while the second block $R(x, y, u, v)|_{x \in \{c,d\}, y \in \{c,d\}} = R_2(x, y, u, v)$ is decomposable in $\{x, v\}$ and $\{y, u\}$. Thus, the constraint is blockwise decomposable in x, y , but not uniformly blockwise decomposable.

Finally, we transfer these characterizations to relations and constraint languages: A k -ary relation R is (uniformly) blockwise decomposable, if the constraint $R(x_1, \dots, x_k)$ is (uniformly) blockwise decomposable for pairwise distinct variables x_1, \dots, x_k . A constraint language Γ is (uniformly) blockwise decomposable if every relation in Γ is (uniformly) blockwise decomposable. A constraint language Γ is *strongly (uniformly) blockwise decomposable* if its co-clone $\langle \Gamma \rangle$ is (uniformly) blockwise decomposable.

Now we are ready to formulate our main theorems. The first one states that the strongly uniformly blockwise decomposable constraint languages are precisely those that can be efficiently compiled to a structured representation format (anything between ODDs and structured DNNFs).

► **Theorem 9.** *Let Γ be a constraint language.*

1. *If Γ is strongly uniformly blockwise decomposable, then there is a polynomial time algorithm that constructs an ODD for a given CSP(Γ)-instance.*
2. *If Γ is not strongly uniformly blockwise decomposable, then there is a family (I_n) of CSP(Γ)-instances such that any structured DNNF for I_n has size $2^{\Omega(\|I_n\|)}$.*

Our second main theorem states that the larger class of strongly blockwise decomposable constraint languages captures CSPs that can be efficiently compiled in an unstructured format between FDDs and DNNFs.

► **Theorem 10.** *Let Γ be a constraint language.*

1. *If Γ is strongly blockwise decomposable, then there is a polynomial time algorithm that constructs an FDD for a given CSP(Γ)-instance.*
2. *If Γ is not strongly blockwise decomposable, then there is a family (I_n) of CSP(Γ)-instances such that any DNNF for I_n has size $2^{\Omega(\|I_n\|)}$.*

4 Properties of the Decomposability Notions

In this section we state important properties about (uniform) blockwise decomposability – all omitted proofs are contained in the full version [5]. We start by observing that these notions are closed under projection and selection.

► **Lemma 11.** *Let $R(\vec{u})$ be a blockwise decomposable, resp. uniformly blockwise decomposable, constraint and let $Y \subseteq \vec{u}$, $z \in \vec{u}$, and $S \subseteq D$. Then the projection $\pi_Y R(\vec{u})$ as well as the selection $R(\vec{u})|_{z \in S}$ are also blockwise decomposable, resp. uniformly blockwise decomposable.*

► **Corollary 12.** *If a constraint language Γ is strongly (uniformly) blockwise decomposable, then its individualization $\Gamma^\bullet := \Gamma \cup \{\{a\} : a \in D\}$ is also strongly (uniformly) blockwise decomposable.*

Next, we will show that blockwise decomposable relations allow for efficient counting of solutions by making a connection to the work of Dyer and Richerby [27]. To state their dichotomy theorem, we need the following definitions. A constraint $R(\vec{x}, \vec{y}, \vec{z})$ is *balanced* (w.r.t. $\vec{x}; \vec{y}; \vec{z}$), if the $|D|^{|\vec{x}|} \times |D|^{|\vec{y}|}$ matrix $M_{\vec{x}, \vec{y}}^\#$ defined by $M_{\vec{x}, \vec{y}}^\#[\vec{a}, \vec{b}] = |\text{sol}(R(\vec{x}, \vec{y}, \vec{z})|_{\vec{x}=\vec{a}, \vec{y}=\vec{b}})|$ is a block diagonal matrix (after permuting rows/columns), where each block has rank one. A constraint language Γ is *strongly balanced* if every at least ternary pp-definable constraint is balanced.

► **Theorem 13** (Effective counting dichotomy [27]). *If Γ is strongly balanced, then there is a polynomial time algorithm that computes $|\text{sol}(I)|$ for a given $\text{CSP}(\Gamma)$ -instance I . If Γ is not strongly balanced, then counting solutions for $\text{CSP}(\Gamma)$ -instances is $\#P$ -complete. Moreover, there is a polynomial time algorithm that decides if a given constraint language Γ is strongly balanced.*

Our next lemma connects blockwise decomposability with strong balance and leads to a number of useful corollaries. We sketch the proof for the case when $R(\vec{x}, \vec{y}, \vec{z})$ is ternary, a full proof can be found in [5].

► **Lemma 14.** *Every strongly blockwise decomposable constraint language is strongly balanced.*

Proof sketch. Let Γ be strongly blockwise decomposable and $R(x, y, z)$ a pp-defined ternary constraint. Then the selection matrix $M_{x,y}^{R(x,y,z)}$ is a block matrix, where each block (A, B) is decomposable in some (V, W) , either $(\{x, z\}, \{y\})$ or $(\{x\}, \{y, z\})$. In any case, for the corresponding block (A, B) in $M_{x,y}^\#$ we have $M_{x,y}^\#[a, b] = s_a \cdot t_b$ where $s_a = |\text{sol}(\pi_V(R(x, y, z)|_{x=a, y \in B}))|$ and $t_b = |\text{sol}(\pi_W(R(x, y, z)|_{x \in A, y=b}))|$. Thus, the block has rank 1. ◀

► **Corollary 15.** *Let Γ be a strongly blockwise decomposable constraint language.*

1. *Given a Γ -formula $F(\vec{u})$ and a (possibly empty) partial assignment α , the number $|\text{sol}(F(\vec{u})|_\alpha)|$ of solutions that extend α can be computed in polynomial time.*
2. *Given a pp-formula $F(\vec{u})$ over Γ and $x, y \in \tilde{u}$, the blocks of $M_{x,y}^{F(\vec{u})}$ can be computed in polynomial time.*
3. *Given a pp-formula $F(\vec{u})$ over Γ , the indecomposable factors of $F(\vec{u})$ can be computed in polynomial time.*

Proof sketch. Claim 1 follows immediately from the combination of Lemma 14 with Theorem 13 and the fact that strongly blockwise decomposable constraint languages are closed under selection (Corollary 12). For Claim 2 let $F(\vec{u}) = \pi_{\tilde{u}}(F'(\vec{v}))$ for some Γ -formula $F'(\vec{v})$. To compute the blocks of $M_{x,y}^{F(\vec{u})}$, we can use Claim 1 to compute for every $x = a$ and $y = b$ whether $|\text{sol}(M_{x,y}^{F'(\vec{v})}[a, b])| > 0$ and hence $M_{x,y}^{F(\vec{u})} \neq \emptyset$. Claim 3 is a bit more subtle and deferred to the full version [5]. ◀

We close this section by stating the following property that applies only to *uniformly* decomposable constraints.

► **Lemma 16.** *Let $R(\vec{u})$ be a constraint that is uniformly blockwise decomposable in x and y . Then there exist \vec{v} and \vec{w} with $\tilde{u} = \{x, y\} \dot{\cup} \tilde{v} \dot{\cup} \tilde{w}$ such that*

$$R(\vec{u}) = \pi_{x,\tilde{v}}R(\vec{u}) \wedge \pi_{y,\tilde{w}}R(\vec{u}) \wedge \pi_{x,y}R(\vec{u}). \quad (9)$$

Furthermore, if R is defined by a pp-formula F over a strongly uniformly blockwise decomposable Γ , then \vec{v} and \vec{w} can be computed from F in polynomial time.

5 Algorithms

5.1 Polynomial time construction of ODDs for strongly uniformly blockwise decomposable constraint languages

The key to the efficient construction of ODD for uniformly blockwise decomposable constraints is the following lemma, which states that any such constraint is equivalent to a treelike conjunction of binary projections of itself.

11:10 A Characterization of Efficiently Compilable Constraint Languages

► **Lemma 17** (Tree structure lemma). *Let $R(\vec{u})$ be a constraint that is uniformly blockwise decomposable. Then there is an undirected tree T with vertex set $V(T) = \vec{u}$ such that*

$$R(\vec{u}) = \bigwedge_{\{p,q\} \in E(T)} \pi_{\{p,q\}}(R(\vec{u})).$$

Furthermore, T can be calculated in polynomial time in $\|F\|$, if R is uniformly blockwise decomposable and given as pp-formula F .

Note that from Lemma 17 it follows in particular that any uniformly decomposable constraint is a conjunction of binary constraints. Thus, it follows from Theorem 9, our main result for ODD-representations, that any constraint language that allows efficient ODD-representations can essentially only consist of binary constraints. Hence, for example affine constraints, for which it is known that they allow efficient counting [19], are hard in our setting.

Proof of Lemma 17. We first fix x and y arbitrarily and apply Lemma 16 to obtain a tri-partition $(\tilde{v}, \{x, y\}, \tilde{w})$ of \vec{u} such that $R(\vec{u}) = \pi_{x,\tilde{v}}(R(\vec{u})) \wedge \pi_{x,y}(R(\vec{u})) \wedge \pi_{y,\tilde{w}}(R(\vec{u}))$. We add the edge $\{x, y\}$ to T . By Lemma 11, $\pi_{x,\tilde{v}}(R(\vec{u}))$ and $\pi_{y,\tilde{w}}(R(\vec{u}))$ are uniformly blockwise decomposable, so Lemma 16 can be recursively applied on both projections. For (say) $\pi_{x,\tilde{v}}(R(\vec{u}))$ we fix x , choose an arbitrary $z \in \tilde{v}$, apply Lemma 16, and add the edge $\{x, z\}$ to T . Continuing this construction recursively until no projections with more than two variables are left yields the desired result. ◀

From this tree structure we will construct small ODDs by starting with a centroid, i. e., a variable whose removal splits the tree into connected components of at most $n/2$ vertices each. From the tree structure lemma it follows that we can handle the (projection on the) subtrees independently. A recursive application of this idea leads to an ODD of size $O(n^{\log |D|+1})$.

Proof of part 1 in Theorem 9. Let I be a CSP(Γ)-instance and $F_I(\vec{u})$ the corresponding Γ -formula. By Lemma 17 we can compute a tree T such that $F_I(\vec{u}) = \bigwedge_{\{p,q\} \in E(T)} \pi_{\{p,q\}}(F_I(\vec{u}))$. By Corollary 15.1 we can explicitly compute, for each $\{p, q\} \in E(T)$, a binary relation $R_{\{p,q\}} \subseteq D^2$ such that $R_{\{p,q\}}(p, q) = \pi_{\{p,q\}}(F_I(\vec{u}))$. Now we define the formula $F_T(\vec{u}) = \bigwedge_{\{p,q\} \in E(T)} R_{\{p,q\}}(p, q)$ and note that $\text{sol}(F_T(\vec{u})) = \text{sol}(F_I(\vec{u}))$. It remains to show that such tree-CSP instances can be efficiently compiled to ODDs. This follows from the following inductive claim, where for technical reasons we also add unary constraints $U_{D_v}(v)$ for each vertex v (setting $D_v := D$ implies the theorem).

▷ **Claim.** Let T be a tree on n vertices and $F_T(\vec{u}) = \bigwedge_{\{v,w\} \in E(T)} R_{\{v,w\}}(v, w) \wedge \bigwedge_{v \in \vec{u}} U_{D_v}(v)$ be a formula. Then there is an order $<$, depending only on T , such that an ODD $_{<}$ of size at most $f(n) := n|D|^{\log(n)}$ deciding $F_T(\vec{u})$ can be computed in $n^{O(1)}$.

We prove the claim by induction on n and the start $n = 1$ is trivial. If $n \geq 2$ let z be a centroid in this tree, that is a node whose removal splits the tree into $\ell \geq 1$ connected components T_1, \dots, T_ℓ of at most $n/2$ vertices each. Let $\vec{v}_1, \dots, \vec{v}_\ell$ be vectors of the variables in these components, so $(\{z\}, \vec{v}_1, \dots, \vec{v}_\ell)$ partitions \vec{u} . Let $x_i \in V(T_i)$ be the neighbors of z in T . We want to branch on z and recurse on the connected components T_i . To this end, for each assignment $z \mapsto a$ we remove for each neighbor x_i those values that cannot be extended to $z \mapsto a$. That is, $D_{x_i}^a := \{b: \{x_i \mapsto b, z \mapsto a\} \in \text{sol}(U_{D_{x_i}}(x_i) \wedge R_{\{x_i,z\}}(x_i, z) \wedge U_{D_z}(z))\}$. Now we let $F_i^a(\vec{v}_i) := \bigwedge_{\{v,w\} \in E(T_i)} R_{\{v,w\}}(v, w) \wedge \bigwedge_{v \in \vec{v}_i \setminus \{x_i\}} U_{D_v}(v) \wedge U_{D_{x_i}^a}(x_i)$ and observe that

$$\text{sol}(F_T(\vec{u})) = \bigcup_{a \in D} \text{sol}(U_a(z) \times F_1^a(\vec{v}_1) \times \dots \times F_\ell^a(\vec{v}_\ell)). \quad (10)$$

By induction assumption, for each $i \in [\ell]$ there is an order $<_i$ of \tilde{v}_i such that each $F_i^a(\vec{v}_i)$ has an $\text{ODD}_{<_i}^a$ of size $f(n_i)$ for $n_i := |\tilde{v}_i|$. Now we start our ODD for $F_T(\vec{u})$ with branching on z followed by the sequential combination of $\text{ODD}_{<_1}^a, \dots, \text{ODD}_{<_\ell}^a$ for each assignment $a \in D$ to z . This completes the inductive construction. Since its size is bounded by $1 + |D| \sum_{i \in [\ell]} f(n_i)$, the following easy estimations finish the proof of the claim (recall that $\sum_{i \in [\ell]} n_i = n - 1$): $1 + |D| \sum_{i \in [\ell]} f(n_i) \leq 1 + |D| \sum_{i \in [\ell]} n_i |D|^{\log(n/2)} = 1 + |D|^{\log(n)} (n - 1) \leq n |D|^{\log(n)} \blacktriangleleft$

5.2 Polynomial size FDDs for strongly blockwise decomposable constraint languages

For blockwise decomposable constraints that are *not* uniformly blockwise decomposable, a good variable order may depend on the values assigned to variables that are already chosen, so it is not surprising that the tree approach for ODDs does not work in this setting.

For the construction of the FDD, we first compute the indecomposable factors (this can be done by Corollary 15.3 and treat them independently. This, of course, could have also been done for the ODD construction. The key point now is how we treat the indecomposable factors: every selection matrix $M_{x,y}^R$ for a (blockwise decomposable) indecomposable constraint necessarily has two non-empty blocks. But then every row $x = a$ must have at least one empty entry $\text{sol}(M_{x,y}^R[a, b]) = \emptyset$. This in turn implies that, once we have chosen $x = a$, we can exclude b as a possible value for y ! As we have chosen y arbitrarily, this also applies to any other variable (maybe with a different domain element b). So the set of possible values for every variable shrinks by one and since the domain is finite, this cannot happen too often. Algorithm 1 formalizes this recursive idea. To bound the runtime of this algorithm, we analyze the size of the recursion tree; the details are found in [5].

6 Lower Bounds

In this section, we will prove the lower bounds of Theorem 9 and Theorem 10. In the proofs, we will use the approach developed in [8] that makes a connection between DNNF size and rectangle covers. We will use the following variant:

► **Lemma 18.** *Let O be a DNNF of size s representing a constraint $R(\vec{x})$ and let $Z \subseteq \vec{x}$. Then there is a Z -balanced rectangle cover of f of size s . Moreover, if O is structured, then the rectangles in the cover are all with respect to the same variable partition.*

The proof of Lemma 18 is very similar to existing proofs in [8] and given in the full version [5].

6.1 Lower Bound for DNNF

In this Section, we show the lower bound for Theorem 10 which we reformulate here.

► **Proposition 19.** *Let Γ be a constraint language that is not strongly blockwise decomposable. Then there Γ -formulas F_n of size $\Theta(n)$ and $\varepsilon > 0$ such that any DNNF for F_n has size at least $2^{\varepsilon \|F_n\|}$.*

In the remainder of this section, we show Proposition 19, splitting the proof into two cases. First, we consider the case where $M_{x,y}^R$ is not a proper block matrix.

► **Lemma 20.** *Let $R(x, y, \vec{z})$ be a constraint such that $M_{x,y}^R$ is not a proper block matrix. Then there is a family of $\{R\}$ -formulas F_n and $\varepsilon > 0$ such that any DNNF for F_n has size at least $2^{\varepsilon \|F_n\|}$.*

11:12 A Characterization of Efficiently Compilable Constraint Languages

■ **Algorithm 1** FDD construction algorithm.

Input: Γ -formula $F(x_1, \dots, x_n)$ for strongly blockwise decomposable Γ over domain D .
Output: An FDD deciding $F(x_1, \dots, x_n)$.

- 1: Initialize variable domains $D_{x_i} \leftarrow D$ for $i = 1, \dots, n$.
- 2: **return** CONSTRUCTFDD($F(x_1, \dots, x_n)$; D_{x_1}, \dots, D_{x_n})

- 3: **procedure** CONSTRUCTFDD($R(x_1, \dots, x_n)$; D_{x_1}, \dots, D_{x_n})
- 4: $R(x_1, \dots, x_n) \leftarrow R(x_1, \dots, x_n) \wedge \bigwedge_{i \in [n]} U_{D_{x_i}}(x_i)$
- 5: **if** $n = 1$ **then**
- 6: **return** 1-node FDD deciding $R(x_1)$, branching on all values D_{x_1} .
- 7: Compute the indecomposable factors $R_1(\vec{u}_1), \dots, R_m(\vec{u}_m)$ of $R(x_1, \dots, x_n)$.
- 8: **if** $m \geq 2$ **then**
- 9: **for** $i = 1 \dots m$ **do** FDD _{i} \leftarrow CONSTRUCTFDD($R_i(\vec{u}_i)$; D_y for $y \in \vec{u}_i$)
- 10: **return** Sequential composition of FDD₁, \dots , FDD _{m}
- 11: **else**
- 12: Introduce branching node for x_1 .
- 13: **for** $a \in D_{x_1}$ **do**
- 14: **for** $i = 2, \dots, n$ **do**
- 15: **for** $b \in D_{x_i}$ **do**
- 16: **if** $\text{sol}(M_{x_1, x_i}^R[a, b]) = \emptyset$ **then**
- 17: $D_{x_i} \leftarrow D_{x_i} \setminus \{b\}$ \triangleright This happens for at least one $b \in D_{x_i}$.
- 18: $S_a(x_2, \dots, x_n) \leftarrow \pi_{x_2, \dots, x_n}(R(x_1, \dots, x_n)|_{x_1=a})$
- 19: FDD ^{a} \leftarrow CONSTRUCTFDD($S_a(x_2, \dots, x_n)$; D_{x_2}, \dots, D_{x_n})
- 20: Connect $x \xrightarrow{a}$ FDD ^{a} for all $a \in D_{x_1}$ and **return** resulting FDD

In the proof of Lemma 20, we will use a specific family of graphs. We remind the reader that a matching is a set of edges in a graph that do not share any end-points. The matching is induced if the graph induced by the end-points of the matching contains exactly the edges of the matching.

► **Lemma 21.** *There is an integer d and a constant $\varepsilon > 0$ such that there is an infinite family (G_n) of d -regular, bipartite graphs such that for each set $X \subseteq V(G_n)$ with $|X| \leq n = |V(G)|/2$ there is an induced matching of size at least $\varepsilon|X|$ in which each edge has exactly one endpoint in X .*

Proof of Lemma 20. If $M_{x,y}^R$ is not a proper block matrix, then, by Lemma 5, the matrix $M_{x,y}^R$ has a 2×2 -submatrix with exactly three non-empty entries. So let $a, b, c, d \in D$ such that $M_{x,y}^R(b, d) = \emptyset$ and $M_{x,y}^R(a, c)$, $M_{x,y}^R(a, d)$ and $M_{x,y}^R(b, c)$ are all non-empty.

We describe a construction that to every bipartite graph $G = (A, B, E)$ gives a formula $F(G)$ as follows: for every vertex $u \in A$, we introduce a variable x_u and for every vertex $v \in B$ we introduce a variable x_v . Then, for every edge $e = uv \in E$ where $u \in A$ and $v \in V$, we add a constraint $R(x_u, x_v, \vec{z}_e)$ where \vec{z}_e consists of variables only used in this constraint. We fix the notation $X_A := \{x_v \mid v \in A\}$, $X_B := \{x_v \mid v \in B\}$ and $X := X_A \cup X_B$.

Let (F_n) be the family of formulas defined by $F_n = F(G_n)$ where (G_n) is the family from Lemma 21. Clearly, $\|F_n\| = \Theta(|E(G_n)|) = \Theta(n)$, as required. Fix n for the remainder of the proof. Let F'_n be the formula we get from F_n by restricting all variables $x_u \in X_A$ to $\{a, b\}$ and all variables $x_v \in X_B$ to $\{c, d\}$ by adding some unary constraints. Let \mathcal{R} be an X -balanced rectangle cover of F'_n . We claim that the size of \mathcal{R} is at least $2^{\varepsilon' n}$, where

$$\varepsilon' = \frac{1}{3} \cdot \varepsilon \cdot \log_2 \left(1 + \frac{1}{2^{d+1}|R|^{d^2}} \right)$$

and d is the degree of G_n . To prove this, we first show that for every $\mathfrak{r} \in \mathcal{R}$, $2^{\varepsilon'|X|} \cdot |\text{sol}(\mathfrak{r})| \leq |\text{sol}(F'_n)|$. So let $\mathfrak{r}(\vec{v}, \vec{w}) = \mathfrak{r}_1(\vec{v}) \times \mathfrak{r}_2(\vec{w}) \in \mathcal{R}$. Since \mathfrak{r} is an X -balanced rectangle, we may assume $|X|/3 \leq |\vec{v} \cap V| \leq 2|X|/3$. By choice of G_n , we have that there is an induced matching \mathcal{M} in G_n of size at least $\varepsilon|X|/3$ consisting of edges that have one endpoint corresponding to a variable in \vec{v} and one endpoint corresponding to a variable in \vec{w} . Consider an edge $e = uv \in \mathcal{M}$. Assume that $x_u \in \vec{v}$ and $x_v \in \vec{w}$. Since we have $\mathfrak{r}(\vec{v}, \vec{w}) = \mathfrak{r}(\vec{v}) \times \mathfrak{r}(\vec{w})$, we get

$$\pi_e \mathfrak{r}(\vec{v}, \vec{w}) = \pi_{\vec{v} \cap e}(\mathfrak{r}(\vec{v})) \times \pi_{\vec{w} \cap e}(\mathfrak{r}(\vec{w})).$$

By construction $\pi_e \mathfrak{r}(\vec{v}, \vec{w}) \subseteq \{(a, c), (a, d), (b, c)\}$, so it follows that either $\pi_e \mathfrak{r}(\vec{v}, \vec{w}) \subseteq \{(a, c), (a, d)\} = \{a\} \times \{c, d\}$ or $\pi_e \mathfrak{r}(\vec{v}, \vec{w}) \subseteq \{(a, c), (b, c)\} = \{a, b\} \times \{c\}$. Assume w.l.o.g. that $\{b, c\} \notin \pi_e \mathfrak{r}(\vec{v}, \vec{w})$ (the other case can be treated analogously). It follows that for each solution $\beta \in \text{sol}(r)$, we get a solution $q(\beta) \in \text{sol}(F_n) \setminus \text{sol}(\mathfrak{r}(\vec{v}, \vec{w}))$ by setting

- $q(\beta)(x_u) := b$,
- For all $x_\ell \in N(x_u)$, we set $q(\beta)(x_\ell) := c$,
- For all $x_\ell \in N(x_u)$ and all $x_m \in N(y_\ell)$ we set $q(\beta)(z_{m\ell}) := \vec{g}$ where \vec{g} is such that $(b, c, \vec{g}) \in R$.

Note that values \vec{g} exist because $M_{x,y}^R(b, c)$ is non-empty. Observe that for two different solutions β and β' the solutions $q(\beta)$ and $q(\beta')$ may be the same. However, we can bound the number $|q^{-1}(q(\beta))|$, giving a lower bound on the set of solutions not in r . To this end, suppose that $q(\beta) = q(\beta')$. Since q only changes the values of x_u , exactly d x_ℓ -variables and at most d^2 vectors of z -variables (the two latter bounds come from the degree bounds on G_n), $q(\beta) = q(\beta')$ implies that β and β' coincide on all other variables. This implies

$$|q^{-1}(q(\beta))| \leq 2^{d+1}|R|^{d^2},$$

because there are only that many possibilities for the variables that q might change. By considering $\{x_u, y_v\}$, we have shown that

$$|\text{sol}(F_n)| \geq |\text{sol}(r)| + \frac{1}{2^{d+1}|R|^{d^2}} |\text{sol}(r)|.$$

So we have constructed $\frac{|\text{sol}(r)|}{2^{d+1}|R|^{d^2}}$ solutions not in r . Now we consider not only one edge but all possible subsets I of edges in \mathcal{M} : for a solution $\beta \in \text{sol}(\mathfrak{r}(\vec{v}, \vec{w}))$, the assignment $q_I(\beta)$ is constructed as the q above, but for all edges $e \in I$. Reasoning as above, we get

$$|q_I^{-1}(q_I(\beta))| \leq \left(2^{d+1}|R|^{d^2}\right)^{|I|}.$$

It is immediate to see that $q_I(\beta) \neq q_{I'}(\beta)$ for $I \neq I'$. Thus we get

$$\begin{aligned} |\text{sol}(F'_n)| &\geq \sum_{m=0}^{\frac{1}{3}\varepsilon|X|} \binom{\frac{1}{3}\varepsilon|X|}{m} \left(\frac{1}{2^{d+1}|R|^{d^2}}\right)^m |\text{sol}(r)| \\ &= \left(1 + \frac{1}{2^{d+1}|R|^{d^2}}\right)^{\frac{1}{3}\varepsilon|X|} |\text{sol}(r)| = 2^{\varepsilon'n} |\text{sol}(r)|. \end{aligned}$$

It follows that every X -balanced rectangle cover has to have a size of at least $2^{\varepsilon'|X|}$. With Lemma 18 and Lemma 1 it follows that any DNNF for F_n has to have a size of at least $2^{\varepsilon'n}$. \blacktriangleleft

Now we consider the case that $M_{x,y}^R$ is a proper block matrix, but R is not blockwise decomposable in some pair of variables x, y .

► **Lemma 22.** *Let $R(x, y, \vec{z})$ be a relation such that $M_{x,y}^R$ is a proper block matrix but $R(x, y, \vec{z})$ is not blockwise decomposable in x and y . Then there is a family of formulas F_n and $\varepsilon > 0$ such that a DNNF for F_n needs to have a size of at least $2^{\varepsilon \|F_n\|}$.*

The proof of Lemma 22 is a slight variant of that of Lemma 20, see [5].

We now have everything in place to prove Proposition 19.

Proof of Proposition 19. Since Γ is not strongly blockwise decomposable, there is a relation $R \in \langle \Gamma \rangle$ that is not blockwise decomposable in x and y . Then, by definition of co-clones and Lemma 11, there is a Γ -formula that defines R . If there are variables $x, y \in \tilde{x}$ such that $M_{x,y}^R$ is not a proper block matrix, then we can apply Lemma 20 to get R -formulas that require exponential size DNNF. Then by substituting all occurrences of R in these formula by the Γ -formula defining R , we get the required hard Γ -formulas. If all $M_{x,y}^R$ are proper block matrices, then there are variables x, y such that $M_{x,y}^R$ is not blockwise decomposable. Using Lemma 22 and reasoning as before, then completes the proof. ◀

6.2 Lower Bound for structured DNNF

In this section, we prove the lower bound of Theorem 9 which we formulate here again.

► **Proposition 23.** *Let Γ be a constraint language that is not strongly uniformly blockwise decomposable. Then there Γ -formulas F_n of size $\Theta(n)$ and $\varepsilon > 0$ such that any structured DNNF for F_n has size at least $2^{\varepsilon \|F_n\|}$.*

Note that for all constraint languages that are not strongly blockwise decomposable, the result follows directly from Proposition 19, so we only have to consider constraint languages which are strongly blockwise decomposable but not strongly uniformly blockwise decomposable. We start with a simple observation.

► **Observation 24.** *Let $\mathfrak{r}(\tilde{x}, \tilde{y})$ be a rectangle with respect to the partition (\tilde{x}, \tilde{y}) . Let $Z \subseteq \tilde{x} \cup \tilde{y}$, then $\pi_Z(\mathfrak{r}(\tilde{x}, \tilde{y}))$ is a rectangle with respect to the partition $(\tilde{x} \cap Z, \tilde{y} \cap Z)$.*

We start our proof of Proposition 23 by considering a special case.

► **Lemma 25.** *Let $R(x, y, \vec{z})$ be a constraint such that there are two assignments $a, b \in \text{sol}(R)$ such that for every partition \tilde{z}_1, \tilde{z}_2 of \vec{z} we have that $a|_{\{x\} \cup \tilde{z}_1} \cup b|_{\{y\} \cup \tilde{z}_2} \notin \text{sol}(R)$ or $a|_{\{y\} \cup \tilde{z}_2} \cup b|_{\{x\} \cup \tilde{z}_1} \notin \text{sol}(R)$. Consider*

$$F_n := \bigwedge_{i \in [n]} R(x_i, y_i, \vec{z}_i)$$

where the \vec{z}_i are disjoint variable vectors. Let (\tilde{x}, \tilde{y}) be a variable partition for F_n and \mathcal{R} be a rectangle cover of F_n such that each rectangle \mathfrak{r}_j in \mathcal{R} respects the partition (\tilde{x}, \tilde{y}) . If for all $i \in [n]$ we have that all $x_i \in \tilde{x}$ and $y_i \in \tilde{y}$ or $x_i \in \tilde{y}$ and $y_i \in \tilde{x}$, then \mathcal{R} has size at least 2^n .

Proof. We use the so-called fooling set method from communication complexity, see e.g. [32, Section 1.3]. To this end, we will construct a set \mathcal{S} of satisfying assignments of F_n such that every rectangle of \mathcal{R} can contain at most one assignment in \mathcal{S} .

So let a_i be the assignment to $\{x_i, y_i\} \cup \tilde{z}_i$ that assigns the variables analogously to a , so $a_i(x_i) := a(x)$, $a_i(y_i) := a(y)$, and $a_i(\vec{z}_i) := a(\vec{z})$. Define analogously b_i . Then the set \mathcal{S} consists of all assignments that we get by choosing for every $i \in [n]$ an assignment d_i as either a_i or b_i and then combining the d_i to one assignment to all variables of F_n . Note that \mathcal{S} contains 2^n assignments and that all of them satisfy F_n , so all of them must be in a rectangle of \mathcal{R} .

We claim that none of the rectangles τ_j of \mathcal{R} can contain more than one element $d \in \mathcal{S}$. By way of contradiction, assume this were not true. Then there is an τ_j that contains two assignments $d, d' \in \mathcal{S}$. Then there is an $i \in [n]$ such that in the construction of d we have chosen a_i while in the construction of d' we have chosen b_i . Let $\tilde{x}_j := \tilde{z}_j \cap \tilde{x}$ and $\tilde{y}_j := \tilde{z}_j \cap \tilde{y}$. Since $d, d' \in \text{sol}(\tau_j)$, we have that $a_i, b_i \in \text{sol}(\pi_{\{x_i, y_i\} \cup \tilde{x}_j \cup \tilde{y}_j}(\tau_j))$. Moreover, by Observation 24, $\pi_{\{x_i, y_i\} \cup \tilde{x}_j \cup \tilde{y}_j}(\tau_j)$ is a rectangle and so we have that $a_i|_{\{x_i\} \cup \tilde{x}_j} \cup b_i|_{\{y_i\} \cup \tilde{y}_j} \in \text{sol}(\pi_{\{x_i, y_i\} \cup \tilde{x}_j \cup \tilde{y}_j}(\tau_j))$ and $a_i|_{\{y_i\} \cup \tilde{y}_j} \cup b_i|_{\{x_i\} \cup \tilde{x}_j} \in \text{sol}(\pi_{\{x_i, y_i\} \cup \tilde{x}_j \cup \tilde{y}_j}(\tau_j))$. But τ consists only of solutions of F_n and thus $\text{sol}(\pi_{\{x_i, y_i\} \cup \tilde{x}_j \cup \tilde{y}_j}(\tau)) \subseteq \text{sol}(R(x_i, y_i, \tilde{z}_i))$, so $a_i|_{\{x_i\} \cup \tilde{x}_j} \cup b_i|_{\{y_i\} \cup \tilde{y}_j}, a_i|_{\{y_i\} \cup \tilde{y}_j} \cup b_i|_{\{x_i\} \cup \tilde{x}_j} \in \text{sol}(R(x_i, y_i, \tilde{z}_i))$. It follows by construction that there is a partition (\tilde{x}, \tilde{y}) of \tilde{z} such that $a|_{\{x\} \cup \tilde{x}} \cup b|_{\{y\} \cup \tilde{y}}$ and $a|_{\{y\} \cup \tilde{y}} \cup b|_{\{x\} \cup \tilde{x}}$ are in $\text{sol}(R)$. This contradicts the assumption on a and b and thus τ_j can only contain one assignment from \mathcal{S} .

Since \mathcal{S} has size 2^n and all of assignments in \mathcal{S} must be in one rectangle of \mathcal{R} , it follows that \mathcal{R} consists of at least 2^n rectangles. \blacktriangleleft

We now prove the lower bound of Proposition 23.

Proof of Proposition 23. Since Γ is not strongly uniformly blockwise decomposable, let $R(x, y, \tilde{z})$ be a constraint in (Γ) that is not uniformly blockwise decomposable in x and y . If $R(x, y, \tilde{z})$ is such that $M_{x,y}^R$ is not a proper block matrix, then the lemma follows directly from Lemma 20, so we assume in the remainder that $M_{x,y}^R$ is a proper block matrix. We denote for every block $D_1 \times D_2$ of $R(x, y, \tilde{z})$ by $R^{D_1 \times D_2}(x, y, \tilde{z})$ the sub-constraint of $R(x, y, \tilde{z})$ we get by restricting x to D_1 and y to D_2 . Since $R(x, y, \tilde{z})$ is not uniformly blockwise decomposable, for every partition $(\tilde{z}_1, \tilde{z}_2)$ of \tilde{z} there is a block $D_1 \times D_2$ such that $R^{D_1 \times D_2}(x, y, \tilde{z}) \neq \pi_{\{x\} \cup \tilde{z}_1} R^{D_1 \times D_2}(x, y, \tilde{z}) \times \pi_{\{y\} \cup \tilde{z}_2} R^{D_1 \times D_2}(x, y, \tilde{z})$.

Given a bipartite graph $G = (A, B, E)$, we construct the same formula $F(G)$ as in the proof of Lemma 20. Consider again the graphs G_n of the family from Lemma 21 and let $F_n := F(G_n)$. Fix n in the remainder of the proof. Let O be a structured DNNF representing F_n of size s . Then there is by Proposition 18 a balanced partition (X_1, X_2) of $X_A \cup X_B$ such that there is a rectangle cover of F_G of size at most s and such that all rectangles respect the partition (X_1, X_2) . Let E' be the set of edges $uv \in E$ such that x_u and x_v are in different parts of the partition (X_1, X_2) . By the properties of G_n , there is an induced matching \mathcal{M} of size $\Omega(|A| + |B|)$ consisting of edges in E' .

For every edge $e = uv \in \mathcal{M}$ let $\tilde{z}_{e,1} := \tilde{z}_e \cap X_1$ and $\tilde{z}_{e,2} := \tilde{z}_e \cap X_2$. Assume that $x_u \in X_1$ and $y_v \in X_2$ (the other case is treated analogously). Then we know that there is a block $D_1 \times D_2$ of $M_{x,y}^R$ such that

$$R^{D_1 \times D_2}(x_u, y_v, \tilde{z}_e) \neq \pi_{\{x_u\} \cup \tilde{z}_{e,1}} R^{D_1 \times D_2}(x_u, y_v, \tilde{z}_e) \times \pi_{\{y_v\} \cup \tilde{z}_{e,2}} R^{D_1 \times D_2}(x_u, y_v, \tilde{z}_e). \quad (11)$$

Since there are only at most $|D|$ blocks in $M_{x,y}^R$, there is a block $D_1 \times D_2$ such that for at least $\Omega\left(\frac{|A|+|B|}{|D|}\right) = \Omega(|A| + |B|)$ edges Equation (11) is true. Call this set of edges E^* .

Let $X^* := \{x_u \mid u \text{ is an endpoint of an edge } e \in E^*\}$. We construct a structured DNNF O' from O by existentially quantifying all variables not in a constraint $R(x_u, y_v, \tilde{z}_e)$ for $e = uv \in E^*$ and for all $x_u \in X^*$ restricting the domain to D_1 if $u \in A$ and to D_2 if $u \in B$. Note that every assignment to X^* that assigns every variable x_v with $v \in A$ to a value in D_1 and every x_v with $v \in B$ to a value in D_2 can be extended to a satisfying assignment of F_n , because $D_1 \times D_2$ is a block. Thus, O' is a representation of

$$F^* := \bigwedge_{e=uv \in E^*} R^{D_1 \times D_2}(x_u, y_v, \tilde{z}_e).$$

11:16 A Characterization of Efficiently Compilable Constraint Languages

We now use the following simple observation.

▷ **Claim 26.** Let $R'(\vec{x}, \vec{y})$ be a constraint such that $R'(\vec{x}, \vec{y}) \neq \pi_{\vec{x}}(R'(\vec{x}, \vec{y})) \times \pi_{\vec{y}}(R'(\vec{x}, \vec{y}))$. Then there are assignments $a, b \in \text{sol}(R'(\vec{x}, \vec{y}))$ such that $a|_{\vec{x}} \cup b|_{\vec{y}} \notin \text{sol}(R'(\vec{x}, \vec{y}))$ or $a|_{\vec{x}} \cup b|_{\vec{y}} \notin \text{sol}(R'(\vec{x}, \vec{y}))$.

Proof. Since $R(\vec{x}, \vec{y}) \neq \pi_{\vec{x}}(R(\vec{x}, \vec{y})) \times \pi_{\vec{y}}(R(\vec{x}, \vec{y}))$ and thus $R(\vec{x}, \vec{y}) \subsetneq \pi_{\vec{x}}(R(\vec{x}, \vec{y})) \times \pi_{\vec{y}}(R(\vec{x}, \vec{y}))$, we have that there is $a_x \in \text{sol}(\pi_{\vec{x}}(R(\vec{x}, \vec{y})))$ and $b_y \in \text{sol}(\pi_{\vec{y}}(R(\vec{x}, \vec{y})))$ such that $a|_{\vec{x}} \cup b|_{\vec{y}} \notin \text{sol}(R(\vec{x}, \vec{y}))$. Simply extending a_x and b_y to an assignment in $R(\vec{x}, \vec{y})$ yields the claim. ◀

Since Claim 26 applies to all constraints in F^* , we are now in a situation where we can use Lemma 25 which shows that any rectangle cover respecting the partition (X_1, X_2) for F^* has size $2^{|E^*|} = 2^{\Omega(|A|+|B|)}$. With Lemma 18, we know that $\|O'\| = 2^{\Omega(|A|+|B|)}$ and since the construction of O' from O does not increase the size of the DNNF, we get $s = \|O\| = 2^{\Omega(|A|+|B|)} = 2^{\Omega(\|F^*\|)} = 2^{\Omega(\|F\|)}$. ◀

7 The Boolean Case

In this section, we will specialize our dichotomy results for the Boolean domain $\{0, 1\}$. A relation R over $\{0, 1\}$ is called *bijunctive affine* if it can be written as a conjunction of the relations $x = y$ and $x \neq y$ and unary relations, so $R \subseteq E$ with $E = \{=, \neq, U_0, U_1\}$ where $U_0 = \{0\}$ and $U_1 = \{1\}$. A set of relations Γ is called *bijunctive affine* if all $R \in \Gamma$ are *bijunctive affine*. We will show the following dichotomy for the Boolean case:

► **Theorem 27.** *Let Γ be a constraint language. If all relations in Γ are *bijunctive affine*, then there is a polynomial time algorithm that, given a Γ -formula F , constructs an OBDD for F . If not, then there are formulas F_n and $\varepsilon > 0$ such that an DNNF for F_n needs to have a size of at least $2^{\varepsilon \|F_n\|}$.*

Let us remark here that, in contrast to general domains D , there is no advantage of FDD over ODD in the Boolean case: either a constraint language allows for efficient representation by ODD or it is hard even for DNNF. So in a sense, the situation over the Boolean domain is easier. Also note that the tractable cases over the Boolean domain are very restricted, allowing only equalities and disequalities.

8 Conclusion

We have seen that there is a dichotomy for compiling systems of constraints into DNNF based on the constraint languages. It turns out that the constraint languages that allow efficient compilation are rather restrictive, in the Boolean setting they consist essentially only of equality and disequality. From a practical perspective, our results are thus largely negative since interesting settings will most likely lie outside the tractable cases we have identified.

One question that we leave for future work is that of decidability. Given a constraint language, can one decide if it is (uniformly) blockwise decomposable? We remark that for the similar property in counting complexity from [27], it is known that the dichotomy criterion is decidable. But we do currently not yet understand if a similar result is possible in our setting.

References

- 1 Noga Alon, Mrinal Kumar, and Ben Lee Volk. Unbalancing sets and an almost quadratic lower bound for syntactically multilinear arithmetic circuits. *Comb.*, 40(2):149–178, 2020. doi:10.1007/s00493-019-4009-0.
- 2 Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 111:1–111:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.111.
- 3 Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting knowledge compilation classes and width parameters. *Theory Comput. Syst.*, 64(5):861–914, 2020. doi:10.1007/s00224-019-09930-2.
- 4 Jérôme Amilastre, H el ene Fargier, Alexandre Niveau, and C edric Pralet. Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. *Int. J. Artif. Intell. Tools*, 23(4), 2014. doi:10.1142/S021821301460015X.
- 5 Christoph Berkholz, Stefan Mengel, and Hermann Wilhelm. A characterization of efficiently compilable constraint languages. *CoRR*, abs/2311.10040, 2023. doi:10.48550/ARXIV.2311.10040.
- 6 Christoph Berkholz and Harry Vinall-Smeeth. A dichotomy for succinct representations of homomorphisms. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 113:1–113:19. Schloss Dagstuhl - Leibniz-Zentrum f ur Informatik, 2023. doi:10.4230/LIPICs.ICALP.2023.113.
- 7 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Expander CNFs have exponential DNNF size. *CoRR*, abs/1411.1995, 2014. arXiv:1411.1995.
- 8 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Knowledge compilation meets communication complexity. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1008–1014. IJCAI/AAAI Press, 2016. URL: <http://www.ijcai.org/Abstract/16/147>.
- 9 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.
- 10 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5):34:1–34:41, 2013. doi:10.1145/2528400.
- 11 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.
- 12 Andrei A. Bulatov, V ictor Dalmau, Martin Grohe, and D aniel Marx. Enumerating homomorphisms. *J. Comput. Syst. Sci.*, 78(2):638–650, 2012. doi:10.1016/J.JCSS.2011.09.006.
- 13 Jin-Yi Cai and Xi Chen. Complexity of counting CSP with complex weights. *J. ACM*, 64(3):19:1–19:39, 2017. doi:10.1145/2822891.
- 14 Florent Capelli. Understanding the complexity of #SAT using knowledge compilation. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–10. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005121.
- 15 Florent Capelli and Stefan Mengel. Tractable QBF by Knowledge Compilation. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPICs.STACS.2019.18.

- 16 Clément Carbonnel, Miguel Romero, and Stanislav Zivný. The complexity of general-valued constraint satisfaction problems seen from the other side. *SIAM J. Comput.*, 51(1):19–69, 2022. doi:10.1137/19M1250121.
- 17 Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*, pages 187–194. AAAI Press, 2013. doi:10.1609/aaai.v27i1.8690.
- 18 Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. Syst. Sci.*, 51(3):511–522, 1995. doi:10.1006/jcss.1995.1087.
- 19 Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996. doi:10.1006/inco.1996.0016.
- 20 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of boolean constraint satisfaction problems*. SIAM, 2001.
- 21 Nadia Creignou, Frédéric Olive, and Johannes Schmidt. Enumerating all solutions of a boolean CSP by non-decreasing weight. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 2011. doi:10.1007/978-3-642-21581-0_11.
- 22 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.
- 23 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001. doi:10.1145/502090.502091.
- 24 Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In Ramón López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 328–332. IOS Press, 2004.
- 25 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002. doi:10.1613/jair.989.
- 26 Guy Van den Broeck and Dan Suciu. Query processing on probabilistic data: A survey. *Found. Trends Databases*, 7(3-4):197–341, 2017. doi:10.1561/19000000052.
- 27 Martin E. Dyer and David Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM J. Comput.*, 42(3):1245–1274, 2013. doi:10.1137/100811258.
- 28 Hélène Fargier and Pierre Marquis. On the use of partially ordered decision graphs in knowledge compilation and quantified boolean formulae. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 42–47. AAAI Press, 2006. URL: <http://www.aaai.org/Library/AAAI/2006/aaai06-007.php>.
- 29 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), March 2007. doi:10.1145/1206035.1206036.
- 30 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000. doi:10.1137/S0097539799349948.
- 31 Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Compiling constraint networks into multivalued decomposable decision graphs. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 332–338. AAAI Press, 2015. URL: <http://ijcai.org/papers15/Abstracts/IJCAI15-053.html>.
- 32 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.
- 33 Jean-Marie Lagniez and Pierre Marquis. An improved decision-dnnf compiler. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 667–673. ijcai.org, 2017. doi:10.24963/ijcai.2017/93.

- 34 Robert Mateescu and Rina Dechter. Compiling constraint networks into AND/OR multi-valued decision diagrams (aomdds). In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of *Lecture Notes in Computer Science*, pages 329–343. Springer, 2006. doi:10.1007/11889205_25.
- 35 Robert Mateescu, Rina Dechter, and Radu Marinescu. AND/OR multi-valued decision diagrams (aomdds) for graphical models. *J. Artif. Intell. Res.*, 33:465–519, 2008. doi:10.1613/jair.2605.
- 36 Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-dnnf compilation with sharpSAT. In Leila Kosseim and Diana Inkpen, editors, *Advances in Artificial Intelligence - 25th Canadian Conference on Artificial Intelligence, Canadian AI 2012, Toronto, ON, Canada, May 28-30, 2012. Proceedings*, volume 7310 of *Lecture Notes in Computer Science*, pages 356–361. Springer, 2012. doi:10.1007/978-3-642-30353-1_36.
- 37 Dan Olteanu. Factorized databases: A knowledge compilation perspective. In Adnan Darwiche, editor, *Beyond NP, Papers from the 2016 AAI Workshop, Phoenix, Arizona, USA, February 12, 2016*, volume WS-16-05 of *AAAI Technical Report*. AAAI Press, 2016. URL: <http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12638>.
- 38 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, 2015. doi:10.1145/2656335.
- 39 Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3141–3148. AAAI Press, 2015. URL: <http://ijcai.org/Abstract/15/443>.
- 40 Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 517–522. AAAI Press, 2008. URL: <http://www.aaai.org/Library/AAAI/2008/aaai08-082.php>.
- 41 Ran Raz, Amir Shpilka, and Amir Yehudayoff. A lower bound for the size of syntactically multilinear arithmetic circuits. *SIAM J. Comput.*, 38(4):1624–1647, 2008. doi:10.1137/070707932.
- 42 Thomas J. Schaefer. The complexity of satisfiability problems. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226. ACM, 1978. doi:10.1145/800133.804350.
- 43 Johan Thapper and Stanislav Zivný. The complexity of finite-valued csp. *J. ACM*, 63(4):37:1–37:33, 2016. doi:10.1145/2974019.
- 44 Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000. URL: <http://ls2-www.cs.uni-dortmund.de/monographs/bdd/>.
- 45 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

Modal Logic Is More Succinct Iff Bi-Implication Is Available in Some Form

Christoph Berkholz ✉ 

Technische Universität Ilmenau, Germany

Dietrich Kuske ✉

Technische Universität Ilmenau, Germany

Christian Schwarz ✉

Technische Universität Ilmenau, Germany

Abstract

Is it possible to write significantly smaller formulae, when using more Boolean operators in addition to the De Morgan basis (and, or, not)? For propositional logic a negative answer was given by Pratt: every formula with additional operators can be translated to the De Morgan basis with only polynomial increase in size.

Surprisingly, for modal logic the picture is different: we show that adding bi-implication allows to write exponentially smaller formulae. Moreover, we provide a complete classification of finite sets of Boolean operators showing they are either of no help (allow polynomial translations to the De Morgan basis) or can express properties as succinct as modal logic with additional bi-implication. More precisely, these results are shown for the modal logic T (and therefore for K). We complement this result showing that the modal logic S5 behaves as propositional logic: no additional Boolean operators make it possible to write significantly smaller formulae.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases succinctness, modal logic

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.12

Funding *Christoph Berkholz*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - project number 414325841.

1 Introduction

Many classical logics such as propositional logic, first-order and second-order logic, temporal and modal logics incorporate a complete set of Boolean operators in their definitions – mostly the De Morgan basis (\wedge , \vee , \neg). While for the expressiveness it is clearly irrelevant which complete operator set is used, this choice may have an impact on the succinctness of formulae. The main aim of this paper is to understand which additional operators allow to express properties with strictly more succinct formulae.

A first simple observation is that if an additional operator defines a *read-once function*, i.e., it can be expressed in the De Morgan basis in such a way that every variable occurs at most once, then it can easily be eliminated without blowing-up the formula too much. Thus, read-once operators such as $x \rightarrow y \equiv \neg x \vee y$ are really just syntactic sugar. For operators that are not read-once, such as bi-implication $x \leftrightarrow y$ or the ternary majority operator $\text{maj}(x, y, z)$, the situation is less clear, because mindlessly replacing them with any equivalent De Morgan formula may lead to an exponential explosion of the formula size. So can it be that such additional operators actually allow to write exponentially more succinct formulae? For propositional logic, a negative answer was given by Pratt [11]: first balance the formula so that it has logarithmic depth and then replace the additional operators by any De Morgan translation. This clearly leads to a linear increase in formula depth and therefore only to a polynomial increase in formula size.



© Christoph Berkholz, Dietrich Kuske, and Christian Schwarz;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 12; pp. 12:1–12:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Balancing a formula is, however, not possible for logics that contain quantifiers. For such logics it is still possible to efficiently remove certain operators that are not read-once. We show that if an operator $\text{op}(x_1, \dots, x_k)$ is “locally read-once”, that is, has for every $i \in [k]$ an equivalent De Morgan formula in which x_i appears only once, then it can be efficiently eliminated. While we prove this result explicitly for modal logic in Section 2, it actually holds for other logics with quantifiers as well. An example of an operator that is locally read-once, but not read-once, is $\text{maj}(x, y, z)$ (see Example 3).

One of our main results is that this characterisation is tight for modal logic: for operators that are not locally read-once, there is no way of removing them without increasing the formula size exponentially in the worst-case. Thus, adding any operator that is not locally read-once to the De Morgan basis allows to write exponentially more succinct formulae. One example of such a useful operator is bi-implication $x \leftrightarrow y$, for which it is not hard to show that any equivalent expression over (\wedge, \vee, \neg) contains x and y twice and hence it is not locally read-once. Furthermore, we analyse the succinctness classes of modal logic wrt. polynomial translations. Can it be that adding even more Boolean operators allows to express properties even more succinct? Here we show that *any* extension of modal logic by a set of operators containing at least one that is not locally read-once has the same succinctness. Thus there are exactly two succinctness classes that are exponentially separated: one containing standard modal logic and the other containing its extension by bi-implication.

Since this dichotomy is in contrast with propositional logic, where only one succinctness class exists, we also investigate what happens for fragments of modal logic defined by restrictions on the Kripke structures. Here we obtain the same dichotomy for structures with a reflexive accessibility relation. But upon considering equivalence relations only, we can show that the two succinctness classes collapse, as they do in propositional logic.

Related work. It seems that this paper is the first to consider the influence of Boolean operators to the succinctness of modal logics. Other aspects have been studied in detail.

Pratt [11] studied the effect of complete bases of binary operators on the size of propositional formulae and proved in particular that there are always polynomial translations. Wilke [15] proved a succinctness gap between two branching time temporal logics, Adler and Immerman [1] developed a game-theoretic method and used it to improve Wilke’s result and to show other succinctness gaps. The succinctness of further temporal logics was considered, e.g., in [2, 10].

Lutz et al. [9, 8] study the succinctness and complexity of several modal logics. French et al. [3] consider multi-modal logic with an abbreviation that allows to express “for all $i \in \Gamma$ and all i -successors, φ holds” where Γ is some set of modalities. Using Adler-Immerman-games, they prove (among other results in similar spirit) that this abbreviation allows exponentially more succinct formulae than plain multi-modal logic.

Grohe and Schweikardt [5] study the succinctness of first-order logic with a bounded number of variables and, for that purpose, develop extended syntax trees as an alternative view on Adler-Immerman-games. These extended syntax trees were used by van Ditmarsch et al. [14] to prove an exponential succinctness gap between a logic of contingency (public announcement logic, resp.) and modal logic.

Hella and Vilander [6] define a formula size game (modifying the Adler-Immerman-game) and use it to show that bisimulation invariant first-order logic is non-elementarily more succinct than modal logic.

2 Where “all” logics coincide

Suppose \mathcal{L} is some classical logic like propositional logic, predicate logic, second-order predicate logic, temporal logic (linear or branching), or modal logic. All these logics use Boolean connectives, usually $\{\vee, \neg\}$, $\{\vee, \wedge, \neg\}$, $\{\rightarrow\}$, or $\{\vee, \wedge, \neg, \rightarrow, \leftrightarrow\}$. One could also allow connectives like majority (“at least two of the statements s_1, s_2 , and s_3 are true”) or divisibility by three (“all or none of the statements s_1, s_2 , and s_3 are true”) without changing the expressive power. But what about the succinctness? More precisely: if, in addition to the De Morgan basis $\{\vee, \wedge, \neg\}$, we allow Boolean connectives from the sets F and G , respectively, is the resulting logic $\mathcal{L}[F]$ more succinct than $\mathcal{L}[G]$? The main result of this section demonstrates that there are at most two “succinctness classes” (up to a polynomial), namely the one containing plain logic \mathcal{L} and the other one containing the extension of \mathcal{L} with bi-implication \leftrightarrow .

Formulae. Let P be a countably infinite set of propositional variables and let $\mathbb{B} = \{\top, \perp\}$ be the Boolean domain where we assume $\top > \perp$. For a set F of Boolean functions, we let $\text{ML}[F]$ be the set of all formulae in modal logic that may use operators from F in addition to the constants \top and \perp as well as the Boolean operators \neg, \wedge , and \vee . Formally, $\text{ML}[F]$ is defined by the syntax

$$\varphi ::= \perp \mid \top \mid p \mid \neg\varphi \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid f(\varphi, \dots, \psi) \mid \diamond\varphi,$$

for propositional variables $p \in P$ and operators $f \in F^1$. We write ML for $\text{ML}[\emptyset]$, the set of formulae in standard modal logic, and $\text{ML}[f]$ for $\text{ML}[\{f\}]$. Furthermore, $\text{PL}[F] \subseteq \text{ML}[F]$ denotes the set of propositional formulae that may use functions from F as well; more precisely, $\text{PL}[F]$ is the set of formulae from $\text{ML}[F]$ that do not use the operator \diamond . Since we always include the null-ary functions \top and \perp and the unary functions p and $\neg p$, we can assume that all functions in F are of arity at least two.

The *size* $|\varphi|$ of a formula from $\text{ML}[F]$ is the number of nodes in its syntax tree.

Semantics. Formulae are interpreted over *pointed Kripke structures*, i.e., over tuples $S = (W, R, V, \iota)$, consisting of a set W of possible worlds, a binary accessibility relation $R \subseteq W \times W$, a valuation $V: W \rightarrow \mathcal{P}(P)$, assigning to every world in W the set of propositional variables that are declared to be true at this world, and an initial world $\iota \in W$.

The satisfaction relation \models between a world w of S and an $\text{ML}[F]$ -formula is defined inductively, where

- $S, w \models p$ if $p \in V(w)$,
 - $S, w \models \diamond\varphi$ if $S, w' \models \varphi$ for some $w' \in W$ with $(w, w') \in R$, and
 - $S, w \models f(\alpha_1, \dots, \alpha_k)$ if $f(b_1, \dots, b_k) = \top$ where, for all $i \in [k]$, $b_i = \top$ iff $S, w \models \alpha_i$
- (the definitions of $S, w \models \varphi$ for $\varphi \in \{\top, \perp, \neg\alpha, \alpha \vee \beta, \alpha \wedge \beta\}$ are as expected). A pointed Kripke structure S is a *model* of φ ($S \models \varphi$) if φ holds in its initial world, i.e., $S, \iota \models \varphi$.

Now let \mathcal{C} be some class of pointed Kripke structures. A formula φ is *satisfiable in \mathcal{C}* if it has a model in \mathcal{C} and φ *holds in \mathcal{C}* if every structure from \mathcal{C} is a model of φ . The formula φ *entails* the formula ψ *in \mathcal{C}* (written $\varphi \models_{\mathcal{C}} \psi$) if any model of φ from \mathcal{C} is also a model of ψ ; φ and ψ are *equivalent over \mathcal{C}* (denoted $\varphi \equiv_{\mathcal{C}} \psi$) if $\varphi \models_{\mathcal{C}} \psi$ and $\psi \models_{\mathcal{C}} \varphi$.

¹ Depending on the context, we consider an element $f \in F$ as a Boolean function or as a symbol in a formula.

Classes of Kripke structures. For different application areas (i.e., interpretations of the operator \diamond), the following classes of Kripke structures have attracted particular interest. For convenience, we define them as classes of *pointed* Kripke structures.

- The class \mathcal{S}_K of all pointed Kripke structures.
- The class \mathcal{S}_T of all pointed Kripke structures with reflexive accessibility relation.
- The class \mathcal{S}_{S5} of pointed Kripke structures where the accessibility relation is an equivalence relation.

Suppose φ is a propositional formula. Then $S, \iota \models \varphi$ only depends on the set $V(\iota)$ of variables that hold in the world ι . Thus, instead of evaluating propositional formulae (as special modal formulae) in Kripke structures, it suffices to evaluate them (as is usually done) in sets of propositional variables or, equivalently, mappings from the set of variables into the Boolean domain \mathbb{B} .

► **Definition 1** (Translations). *Let F and G be sets of Boolean functions, \mathcal{C} a class of pointed Kripke structures, and $\kappa: \mathbb{N} \rightarrow \mathbb{N}$ some function. Then $\text{ML}[F]$ has κ -translations wrt. \mathcal{C} in $\text{ML}[G]$ if, for every formula $\varphi \in \text{ML}[F]$, there exists a formula $\psi \in \text{ML}[G]$ with $\varphi \equiv_{\mathcal{C}} \psi$ and $|\psi| \leq \kappa(|\varphi|)$.*

The logic $\text{ML}[F]$ has polynomial translations wrt. \mathcal{C} in $\text{ML}[G]$ if it has κ -translations wrt. \mathcal{C} for some polynomial function κ ; sub-exponential and exponential translations are defined similarly.

In this section, we aim at sufficient conditions for the existence of polynomial translations wrt. \mathcal{S}_K of $\text{ML}[F]$ in $\text{ML}[G]$, i.e., we will always consider the class of *all* pointed Kripke structures. For notational convenience, we will regularly omit the explicit reference to the class \mathcal{S}_K , e.g., “equivalent” means “equivalent over \mathcal{S}_K ”, $\varphi \models \psi$ means $\varphi \models_{\mathcal{S}_K} \psi$, and “ κ -translations” means “ κ -translations wrt. \mathcal{S}_K ”.

In this paper, $[n] = \{1, 2, \dots, n\}$ for all $n \in \mathbb{N}$.

2.1 Polynomial translations

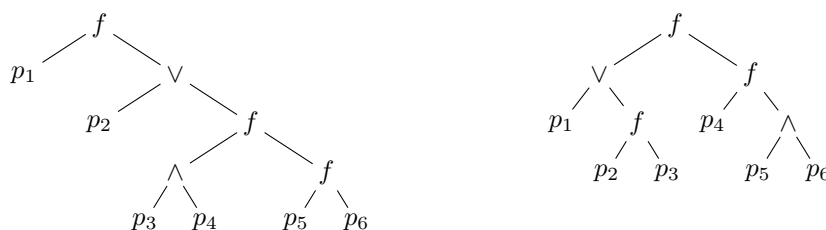
Suppose F and G are sets of Boolean functions. Recall that formulae from $\text{ML}[F]$ can use operators from F as well as \neg, \wedge , and \vee (and similarly for $\text{ML}[G]$). Since the De Morgan basis is complete, for every function $f \in F$, there is some formula $\omega \in \text{PL} \subseteq \text{ML}[G]$ such that the formulae $f(p_1, \dots, p_k)$ and $\omega(p_1, \dots, p_k)$ are equivalent. Consequently, to translate a formula $\varphi \in \text{ML}[F]$ into a formula $\psi \in \text{ML}[G]$, we only need to replace every sub-formula $f(\alpha_1, \dots, \alpha_k)$ in φ by $\omega(\alpha_1, \dots, \alpha_k)$. In general, this translation leads to an exponential size increase. But if, in the formula ω , every variable p_i appears only once, we obtain a linear translation. In this section, we provide polynomial (and in general non-linear) translations of $\text{ML}[F]$ in $\text{ML}[G]$ under the following weaker assumption.

► **Definition 2** (Representations). *Let G be a set of Boolean functions, f a Boolean operator of arity k , and $i \in [k]$.*

A $\text{PL}[G]$ -representation of (f, i) is a $\text{PL}[G]$ -formula $\omega_i(p_1, \dots, p_k)$ that is equivalent to the $\text{PL}[f]$ -formula $f(p_1, \dots, p_k)$ and uses the variable p_i at most once.

A set F of Boolean functions has $\text{PL}[G]$ -representations if there are $\text{PL}[G]$ -representations for all $f \in F$ and $i \in [\text{ar}(f)]$.

► **Example 3.** Consider the majority function $\text{maj}(p, q, r)$ that is true iff at least two arguments are true. Then $(\text{maj}, 1)$ has the PL -representation $(p \wedge (q \vee r)) \vee (q \wedge r)$. Using the symmetry of maj , it follows that $\{\text{maj}\}$ has PL -representations.



■ **Figure 1** Syntax trees of $\varphi = f(p_1, p_2 \vee f(p_3 \wedge p_4, f(p_5, p_6)))$ and $\psi = f(p_1 \vee f(p_2, p_3), f(p_4, p_5 \wedge p_6))$.

Next, consider bi-implication \leftrightarrow , where the following observation seems folklore: if the PL-formula $\psi(p, q)$ is equivalent to $p \leftrightarrow q$ and mentions p only once (say, under an even number of negations), then $\top \equiv (\perp \leftrightarrow \perp) \equiv \psi(\perp, \perp) \leq \psi(\top, \perp) \equiv \perp$, a contradiction.

Assuming that F has $\text{PL}[G]$ -representations, we will construct, from a formula in $\text{ML}[F]$, an equivalent formula in $\text{ML}[G]$ of polynomial size. Since this will be done inductively, we will have to deal with formulae from $\text{ML}[F \cup G]$ and the task then is better described as elimination of functions $f \in F$ from formulae in $\text{ML}[F \cup G]$.

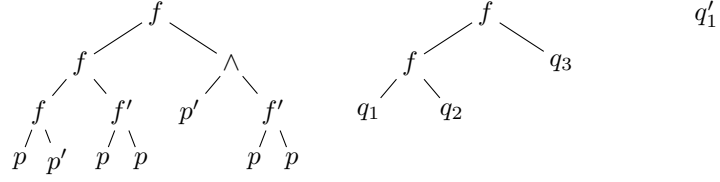
Before we present the details of our construction, we briefly demonstrate the main idea behind the proof (for $F = \{f\}$ and $G = \emptyset$). The main results (Lemma 7 and Proposition 8) will appear at the end of the section.

Assume that f is of arity 2 and consider the two formulae

$$\varphi = f(p_1, p_2 \vee f(p_3 \wedge p_4, f(p_5, p_6))) \text{ and } \psi = f(p_1 \vee f(p_2, p_3), f(p_4, p_5 \wedge p_6)),$$

whose syntax trees are depicted in Fig. 1. A distinguishing property of the left tree is the existence of a branch (a path from the root to a leaf) that contains all F -vertices. Assuming f to have $\text{PL}[G]$ -representations (with $G = \emptyset$), there exist Boolean combinations $\omega_1(x, y) \equiv \omega_2(x, y) \equiv f(x, y)$ of the variables x and y , such that x occurs only once in $\omega_1(x, y)$ and y only once in $\omega_2(x, y)$. Proceeding bottom-up, we now replace each F -vertex $f(\alpha, \beta)$ in the syntax tree of φ by either $\omega_1(\alpha, \beta)$ or $\omega_2(\alpha, \beta)$, depending on whether we have previously modified the left or the right sub-tree (regarding $f(p_5, p_6)$, we are free to choose between ω_1 and ω_2). Note that, although ω_1 and ω_2 may duplicate some parts of φ , our choice ensures that we never duplicate such parts whose size has already changed. Consequently, this procedure results in a linear increase $\ell \cdot |\varphi|$ in the size of φ , where the coefficient ℓ essentially depends on how often y occurs in $\omega_1(x, y)$ and how often x occurs in $\omega_2(x, y)$. The resulting formula φ' belongs to $\text{ML}[G]$ and is equivalent to φ . Hence $\text{ML}[F \cup G]$ -formulae for which all F -vertices lie on some common branch have $\text{ML}[G]$ -translations of linear size.

The formula ψ on the other hand does not have the property that all F -vertices lie on some common branch, but the two sub-formulae α and β rooted at the children of the root do. Hence we can apply the above transformation to them separately, yielding equivalent $\text{ML}[G]$ -formulae α' and β' whose size increases at most by a factor of ℓ . Then $\psi' = f(\alpha', \beta') \equiv \psi$ is of size $|\psi'| \leq \ell \cdot |\psi|$. Note that this step reduces the total number of F vertices – in particular, ψ' now contains only a single operator from F . Applying the step again yields an equivalent $\text{ML}[G]$ -formula ψ'' of size $|\psi''| \leq \ell \cdot |\psi'| \leq \ell^2 \cdot |\psi|$. For arbitrary $\text{ML}[F \cup G]$ -formulae, the number of steps relates to the “nesting depth” D of those F -vertices that have at least two arguments in $\text{ML}[F \cup G] \setminus \text{ML}[G]$, thus resulting in a formula of size $\ell^D \cdot |\psi|$. In this section, we will show that D is at most logarithmic in the size of ψ , thus giving a polynomial bound and establishing the main part of the succinctness result.



■ **Figure 2** Syntax trees of $\varphi = f(f(f(p, p'), f'(p, p)), p' \wedge f'(p, p))$, $d_F(\varphi) = f(f(q_1, q_2), q_3)$, and $d_F^2(\varphi) = q_1$.

Let F and G be disjoint sets of Boolean functions and let $N_{F,G} \subseteq \text{ML}[F \cup G]$ be described by the syntax

$$\varphi ::= \psi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \neg\varphi \mid f(\psi, \dots, \psi, \varphi, \psi, \dots, \psi) \mid g(\varphi, \dots, \varphi) \mid \Diamond\varphi,$$

where $\psi \in \text{ML}[G]$, $f \in F$, and $g \in G$. The formulae from $N_{F,G}$ have a slightly more general property than ψ in the example above. More precisely, a formula φ belongs to $N_{F,G}$ if, and only if, for any sub-formula $f(\alpha_1, \dots, \alpha_k)$ of φ with $f \in F$, at most one α_i contains some operator from F . Conjunction, disjunction, and operators from G on the other hand may have occurrences of operators from F in any number of arguments.

► **Definition 4 (Derivative).** Let $\varphi = \varphi(\bar{p}) \in \text{ML}[F \cup G]$ with $\bar{p} = (p_1, \dots, p_m)$. The F -derivative $d_F(\varphi)$ of φ is the smallest $\text{ML}[F \cup G]$ -formula $\gamma(\bar{p}, q_1, \dots, q_n)$ (up to renaming of the variables q_1, \dots, q_n), such that

- q_i occurs exactly once in $\gamma(\bar{p}, q_1, \dots, q_n)$ for all $i \in [n]$ and
- there exist $\alpha_1, \dots, \alpha_n \in N_{F,G} \setminus \text{ML}[G]$ such that φ and $\gamma(\bar{p}, \alpha_1, \dots, \alpha_n)$ are identical.

Intuitively, $\gamma(\bar{p}, q_1, \dots, q_n)$ is obtained from $\varphi(\bar{p})$ by simultaneously replacing all “maximal ($N_{F,G} \setminus \text{ML}[G]$)-formulae” by distinct fresh variables q_1, \dots, q_n (where multiple occurrences of the same formula are replaced by different variables). An example is depicted in Fig. 2 (with $F = \{f, f'\}$ and $G = \emptyset$).

Let $\varphi \in \text{ML}[F \cup G] \setminus \text{ML}[G]$. Then $d_F(\varphi)$ contains fewer occurrences of operators from F than φ . Hence there exists a smallest integer $r \geq 0$ for which the r -th derivative $d_F^r(\varphi) = d_F(d_F(\dots d_F(\varphi) \dots))$ is an $\text{ML}[G]$ -formula, where $d_F^0(\varphi) = \varphi$.

► **Definition 5 (Rank).** Let $\varphi \in \text{ML}[F \cup G]$. The F -rank $\text{rank}_F(\varphi)$ of φ is the smallest integer $r \geq 0$ for which $d_F^r(\varphi) \in \text{ML}[G]$.

We first show that a formula with high F -rank must also be large.

► **Lemma 6.** Let F and G be disjoint sets of Boolean functions and $\varphi \in \text{ML}[F \cup G]$. Then $|\varphi| \geq 2^{\text{rank}_F(\varphi)}$.

Proof. Recall that we can assume that all functions in F have arity at least 2. Since we only refer to the F -rank of a formula, we will simply speak of the rank of a formula.

We prove the stronger claim that the syntax tree of a formula φ of positive rank has at least $2^{\text{rank}_F(\varphi)-1}$ sub-trees of the form $f(\beta_1, \dots, \beta_k)$ with $f \in F$ and $\beta_1, \dots, \beta_k \in \text{ML}[G]$ (in the following, we call such a sub-tree an F -leaf).

Since counting the F -leaves in all derivatives of φ results in a lower bound on the total number of operators from F in φ , it follows that φ contains at least $1 + 2 + \dots + 2^{\text{rank}_F(\varphi)-1} = 2^{\text{rank}_F(\varphi)} - 1$ operators from F . Hence $|\varphi| \geq 2^{\text{rank}_F(\varphi)}$ since none of the functions in F has arity zero.

It now remains to prove the bound on the number of F -leaves. Recall that we consider formulae of rank at least one. If $\text{rank}_F(\varphi) = 1$, φ contains at least one operator from F and hence has at least $2^{1-1} = 1$ F -leaf. Now, assume that $\varphi = \varphi(\bar{p})$ is of rank at least two. Let $\gamma(\bar{p}, q_1, \dots, q_n)$ be the derivative of φ , and let $\alpha_1, \dots, \alpha_n \in N_{F,G}$ such that $\varphi = \gamma(\bar{p}, \alpha_1, \dots, \alpha_n)$. Then, for every F -leaf $f(\beta_1, \dots, \beta_k)$ of $\gamma(\bar{p}, q_1, \dots, q_n)$, there exist indices $i \neq j$ such that $\beta_i, \beta_j \in \{q_1, \dots, q_n\}$. By induction hypothesis, $\gamma(\bar{p}, q_1, \dots, q_n)$ has at least $2^{\text{rank}_F(\varphi)-2}$ F -leaves, each of which contains at least two of the fresh variables $\{q_1, \dots, q_n\}$. Since each α_i contains at least one operator from F , $\varphi = \gamma(\bar{p}, \alpha_1, \dots, \alpha_n)$ has at least twice the number of F -leaves compared to $\gamma(\bar{p}, q_1, \dots, q_n)$, i.e., $2^{\text{rank}_F(\varphi)-1}$ F -leaves. \blacktriangleleft

We can now turn to the main ingredient for our succinctness result.

► **Lemma 7.** *Let F and G be disjoint sets of Boolean functions and, for $f \in F$ and $i \in [\text{ar}(f)]$, let $\omega_{f,i} \in \text{PL}[G]$ be a $\text{PL}[G]$ -representation of (f, i) . Let, furthermore, $\kappa: \mathbb{N} \rightarrow \mathbb{N}$ be a monotone function such that $1 \leq \kappa(1)$ and $|\omega_{f,i}| \leq \kappa(\text{ar}(f))$ for any $f \in F$ and $i \in [\text{ar}(f)]$. Finally, let $\kappa': \mathbb{N} \rightarrow \mathbb{N}: n \mapsto \kappa(n)^{\log_2 n} \cdot n$.*

Then $\text{ML}[F \cup G]$ has κ' -translations in $\text{ML}[G]$.

Proof. For $\varphi \in \text{ML}[F \cup G]$, let K_φ denote the maximal arity of any $f \in F$ occurring in φ (or 1 if $\varphi \in \text{ML}[G]$). Note that $\kappa(K_\varphi) \geq 1$ for all $\varphi \in \text{ML}[F \cup G]$.

We prove the following claim: every $\varphi \in \text{ML}[F \cup G]$ is equivalent to an $\text{ML}[G]$ -formula of size at most $(\kappa(K_\varphi))^{\text{rank}_F(\varphi)} \cdot |\varphi|$. Since $K_\varphi \leq |\varphi|$ and $\text{rank}_F(\varphi) \leq \log_2 |\varphi|$ by Lemma 6, this claim ensures that every $\text{ML}[F \cup G]$ -formula φ of size n has an equivalent $\text{ML}[G]$ -formula of size at most $\kappa(n)^{\log_2 n} \cdot n = \kappa'(n)$.

The proof of the claim proceeds by induction on the F -rank of φ . Since F remains fixed, we will refer to the F -rank simply as rank. As before, we assume that all operators in F are at least binary.

Let $\varphi \in \text{ML}[F \cup G]$ be of rank at most one. We show by induction on the structure of φ that there exists an equivalent $\text{ML}[G]$ -formula φ' of size $|\varphi'| \leq \kappa(K_\varphi) \cdot |\varphi|$. If φ is a propositional variable or one of the constants \top or \perp , $|\varphi| = 1 \leq \kappa(1) \cdot |\varphi|$ since $\kappa(1) \geq 1$.

Now, assume that $\varphi = \alpha_1 \wedge \alpha_2$. By induction hypothesis, there exist $\text{ML}[G]$ -formulae β_1, β_2 with $|\beta_j| \leq \kappa(K_{\alpha_j}) \cdot |\alpha_j|$ and $\beta_j \equiv \alpha_j$ for $j \in [2]$. Since κ is monotone and $K_{\alpha_j} \leq K_\varphi$, $|\beta_j| \leq \kappa(K_{\alpha_j}) \cdot |\alpha_j| \leq \kappa(K_\varphi) \cdot |\alpha_j|$ for $j \in [2]$. Set $\varphi' = \beta_1 \wedge \beta_2$. Then φ' is an $\text{ML}[G]$ -formula, equivalent to φ , and of size

$$|\varphi'| = |\beta_1| + |\beta_2| + 1 \leq \kappa(K_\varphi) \cdot (|\alpha_1| + |\alpha_2|) + 1 \leq \kappa(K_\varphi) \cdot |\varphi|, \quad \text{since } \kappa(K_\varphi) \geq 1.$$

A similar argument establishes the cases $\alpha_1 \vee \alpha_2$, $\neg\alpha$, $\diamond\alpha$, and $g(\alpha_1, \dots, \alpha_n)$ for $g \in G$.

Finally, assume that φ is of the form $f(\alpha_1, \dots, \alpha_k)$ with $f \in F$. Since φ is of rank one and therefore a formula in $N_{F,G}$, there exists an index $i \in [k]$ such that no argument other than α_i contains an operator from f , i.e., with $\alpha_j \in \text{ML}[G]$ for all $j \neq i$ (and $\alpha_i \in \text{ML}[F \cup G]$). By induction hypothesis, α_i is equivalent to an $\text{ML}[G]$ -formula β_i of size $|\beta_i| \leq \kappa(K_{\alpha_i}) \cdot |\alpha_i| \leq \kappa(K_\varphi) \cdot |\alpha_i|$. Set $\beta_j = \alpha_j$ for all $j \neq i$. Then $\beta_1, \dots, \beta_k \in \text{ML}[G]$. Recall that $\omega_{f,i}(p_1, \dots, p_k)$ is a $\text{PL}[G]$ -representation of (f, i) that uses the variable p_i at most once and has size $\leq \kappa(k) \leq \kappa(K_\varphi)$. Set $\varphi' = \omega_{f,i}(\beta_1, \dots, \beta_k)$. Then φ' is equivalent to $f(\alpha_1, \dots, \alpha_k) = \varphi$ and belongs to $\text{ML}[G]$. Furthermore, φ' is obtained from $\omega_{f,i}(p_1, \dots, p_k)$ by replacing one variable with β_i and all others with formulae of size at most $\sum_{j \neq i} |\beta_j|$. Since all functions in F are at least binary, it follows that $\sum_{j \neq i} |\beta_j| \geq 1$. Hence

$$\begin{aligned}
 |\varphi'| &= |\omega_{f,i}(\beta_1, \dots, \beta_k)| \\
 &\leq |\beta_i| + \kappa(k) \cdot \sum_{j \in [k] \setminus \{i\}} |\beta_j| && , \text{ since } \sum_{j \in [k] \setminus \{i\}} |\beta_j| \geq 1 \\
 &\leq \kappa(K_\varphi) \cdot |\alpha_i| + \kappa(K_\varphi) \cdot \sum_{j \in [k] \setminus \{i\}} |\alpha_j| \\
 &\leq \kappa(K_\varphi) \cdot |f(\alpha_1, \dots, \alpha_k)| = \kappa(K_\varphi) \cdot |\varphi|.
 \end{aligned}$$

This shows the claim for formulae of rank at most one.

We proceed by induction on the rank of φ . Let $\varphi = \varphi(\bar{p}) \in \text{ML}[F \cup G]$ be of rank $r \geq 2$. Let $\gamma(\bar{p}, q_1, \dots, q_n)$ be the derivative of $\varphi(\bar{p})$ and $\alpha_1, \dots, \alpha_n$ be $N_{F,G}$ -formulae, such that $\varphi = \gamma(\bar{p}, \alpha_1, \dots, \alpha_n)$. Since the α_i are of rank one, there exist $\beta_1, \dots, \beta_n \in \text{ML}[G]$ with $\alpha_i \equiv \beta_i$ and $|\beta_i| \leq \kappa(K_{\alpha_i}) \cdot |\alpha_i| \leq \kappa(K_\varphi) \cdot |\alpha_i|$ for $i \in [n]$. Set $\psi = \psi(\bar{p}) = \gamma(\bar{p}, \beta_1, \dots, \beta_n)$. Then ψ is equivalent to φ and of size $|\psi| \leq \kappa(K_\varphi) \cdot |\varphi|$. Intuitively, ψ is obtained from φ by replacing the “maximal” $\text{ML}[F \cup G]$ sub-formulae of rank 1 by equivalent $\text{ML}[G]$ -formulae. Since ψ is of rank $r - 1$, it follows by induction hypothesis that ψ is equivalent to a formula $\varphi' \in \text{ML}[G]$ of size $|\varphi'| \leq \kappa(K_\psi)^{r-1} \cdot |\psi| \leq \kappa(K_\varphi)^r \cdot |\varphi|$. Since $\varphi \equiv \psi \equiv \varphi'$, this finishes the verification of the claim from the beginning of this proof. ◀

From Lemma 7, we can get the main result of this section, stating that $\text{ML}[F \cup G]$ is not more succinct than $\text{ML}[G]$, provided F is a finite set of Boolean functions with $\text{PL}[G]$ -representations.

► **Proposition 8.** *Let F and G be disjoint finite sets of Boolean functions such that F has $\text{PL}[G]$ -representations. Then $\text{ML}[F \cup G]$ has polynomial translations in $\text{ML}[G]$.*

Since $\text{ML}[F] \subseteq \text{ML}[F \cup G]$, this implies in particular that $\text{ML}[F]$ has polynomial translations in $\text{ML}[G]$. In view of Example 3, it ensures specifically that $\text{ML}[\text{maj}]$ has polynomial translations in plain ML .

Proof. Since F is finite, there is some constant c such that any $f \in F$ and $i \in [\text{ar}(f)]$ have a $\text{PL}[G]$ -representation of size at most c . By the previous lemma, any $\varphi \in \text{ML}[F \cup G]$ is thus equivalent to an $\text{ML}[G]$ -formula of size at most $c^{\log_2 |\varphi|} \cdot |\varphi| = |\varphi|^{1 + \log_2 c} \leq |\varphi|^d$ for some constant d . ◀

2.2 A decidable characterisation of representations

Proposition 8 gives a condition (“has $\text{PL}[G]$ -representations”) for the existence of polynomial translations. In this section, we demonstrate that this condition is decidable.

► **Definition 9** (Local monotonicity). *A Boolean function $f: \mathbb{B}^k \rightarrow \mathbb{B}$ is monotone in the i -th argument if either*

(M1) *for all $\bar{a} \in \mathbb{B}^{i-1}$, $\bar{b} \in \mathbb{B}^{k-i}$, $f(\bar{a}, \perp, \bar{b}) \leq f(\bar{a}, \top, \bar{b})$ or*

(M2) *for all $\bar{a} \in \mathbb{B}^{i-1}$, $\bar{b} \in \mathbb{B}^{k-i}$, $f(\bar{a}, \perp, \bar{b}) \geq f(\bar{a}, \top, \bar{b})$.*

That is, when changing the i -th argument from \perp to \top , while keeping the remaining ones fixed, the truth value of f uniformly increases or decreases (where, in both cases, the value may also remain unchanged). A function is called *locally monotone* if it is monotone in every argument and *non-locally-monotone* otherwise. Then conjunction, disjunction, negation, implication, as well as majority are locally monotone functions, while bi-implication is not.

► **Proposition 10.** *Let F and G be disjoint sets of Boolean functions. Then F has $\text{PL}[G]$ -representations if, and only if, all functions in F are locally monotone or some function from G is non-locally-monotone.*

Proof. First, suppose that all functions in F are locally monotone. We prove that, under this assumption, F even has PL -representations, which ensures the claim since $\text{PL} \subseteq \text{PL}[G]$. So, let $f \in F$ be of arity k (as before, we can assume $k \geq 2$). To simplify notation, we will only construct a PL -representation of (f, k) . In addition, we assume that f is increasing in the k -th argument, i.e., $f(\bar{a}, \perp) \leq f(\bar{a}, \top)$ for all $\bar{a} \in \mathbb{B}^{k-1}$. There exists a PL -formula $\omega(x_1, \dots, x_k)$ that is equivalent to $f(x_1, \dots, x_k)$. Since $f(\bar{a}, \perp) \leq f(\bar{a}, \top)$ for any $\bar{a} \in \mathbb{B}^{k-1}$, it follows that

$$f(x_1, \dots, x_k) \equiv \left(\omega(x_1, \dots, x_{k-1}, \top) \wedge x_k \right) \vee \omega(x_1, \dots, x_{k-1}, \perp).$$

In particular, the formula on the right uses the variable x_k only once and therefore forms a PL -representation of (f, k) .

Next, suppose there is some $g \in G$ that is non-locally-monotone. We have to provide $\text{PL}[G]$ -representations for all functions $f \in F$. So let $f \in F$ be arbitrary and of arity k ; for notational simplicity, we prove that there is some $\text{PL}[G]$ -representation of (f, k) .

Let A denote the set of tuples $\bar{a} \in \mathbb{B}^{k-1}$ with $f(\bar{a}, \top) = f(\bar{a}, \perp) = \top$ and let B denote the set of tuples $\bar{b} \in \mathbb{B}^{k-1}$ with $f(\bar{b}, \top) > f(\bar{b}, \perp)$. Let $\bar{x} \in A$ abbreviate the PL -formula

$$\bigvee_{\bar{a} \in A} \left(\bigwedge_{i \in [k-1], a_i = \top} x_i \wedge \bigwedge_{i \in [k-1], a_i = \perp} \neg x_i \right)$$

and let $\bar{x} \in B$ be defined likewise. Then the formula $f(\bar{x}, x_k)$ is equivalent to the formula

$$\bar{x} \in A \vee (\bar{x} \notin A \wedge (\bar{x} \in B \leftrightarrow x_k)), \tag{1}$$

which belongs to $\text{PL}[\leftrightarrow]$ (and mentions the variable x_k only once).

Let ℓ be the arity of the function $g \in G$ and suppose, for notational simplicity, that it is not monotone in its last argument. Hence there are $\bar{a}, \bar{b} \in \mathbb{B}^{\ell-1}$ such that $g(\bar{a}, \perp) < g(\bar{a}, \top)$ and $g(\bar{b}, \perp) > g(\bar{b}, \top)$. For $i \in [\ell - 1]$, set

$$\theta_i = \begin{cases} a_i & \text{if } a_i = b_i \\ \bar{x} \in B & \text{if } a_i > b_i \\ \bar{x} \notin B & \text{if } a_i < b_i \end{cases}$$

and write $\bar{\theta}$ for the tuple $(\theta_i)_{i \in [\ell-1]}$. Note that, for all $i \in [\ell - 1]$, θ_i is equivalent to a_i if $\bar{x} \in B$, and to b_i if $\bar{x} \notin B$. By choice of \bar{a} and \bar{b} it follows that the formulae $(\bar{x} \in B \leftrightarrow x_k) \in \text{PL}[\leftrightarrow]$ and $g(\bar{\theta}, x_k) \in \text{PL}[G]$ are equivalent. We finally replace $(\bar{x} \in B \leftrightarrow x_k)$ in the formula (1) by $g(\bar{\theta}, x_k)$ which yields the $\text{PL}[G]$ -representation $\bar{x} \in A \vee (\bar{x} \notin A \wedge g(\bar{\theta}, x_k))$ of (f, k) .

It remains to be shown that the existence of $\text{PL}[G]$ -representations implies that (i) or (ii) holds. So assume that F has $\text{PL}[G]$ -representations and that (ii) does not hold, i.e., that all functions from G are locally monotone. We prove by induction on the size of a formula $\varphi(p_1, \dots, p_k) \in \text{PL}[G]$ the following: if the variable p_k appears only once in φ , then the function represented by φ is monotone in its k -th argument. The claim is trivial for formulae of the form \top, \perp , and p_i .

12:10 Succinctness of Modal Logic

For the induction step, let $\varphi = g(\alpha_1, \dots, \alpha_\ell)$ with $g \in G$. Since the formula φ contains the variable p_k only once, it appears in at most one of the arguments α_i ; for notational simplicity, we assume it appears in α_ℓ . By the induction hypothesis, there is $\otimes \in \{\leq, \geq\}$ such that

$$\forall \bar{a} \in \mathbb{B}^{\ell-1}: \alpha_\ell(\bar{a}, \perp) \otimes \alpha_\ell(\bar{a}, \top). \quad (2)$$

Since we assumed all functions from G to be locally monotone, there also is $\odot \in \{\leq, \geq\}$ such that

$$\forall \bar{a} \in \mathbb{B}^{\ell-1}: g(\alpha_1(\bar{a}), \dots, \alpha_{\ell-1}(\bar{a}), \perp) \odot g(\alpha_1(\bar{a}), \dots, \alpha_{\ell-1}(\bar{a}), \top). \quad (3)$$

Putting (2) and (3) together, we obtain

$$\begin{aligned} \forall \bar{a} \in \mathbb{B}^{\ell-1}: \varphi(\bar{a}, \perp) &\leq \varphi(\bar{a}, \top) && \text{if } \otimes = \odot, \text{ and} \\ \forall \bar{a} \in \mathbb{B}^{\ell-1}: \varphi(\bar{a}, \perp) &\geq \varphi(\bar{a}, \top) && \text{if } \otimes \neq \odot. \end{aligned}$$

Intuitively, \odot indicates whether φ increases (when going from \perp to \top in the last argument of g , i.e., in α_ℓ), while \otimes may flip the direction if the truth-value of α_ℓ decreases, when increasing p_k . Hence, the formula φ represents a function that is locally monotone in its last argument provided φ is of the form $g(\alpha_1, \dots, \alpha_\ell)$ for some $g \in G$. The arguments are similar if φ is of the form $\alpha_1 \wedge \alpha_2$, $\alpha_1 \vee \alpha_2$, or $\neg \alpha_1$.

This finishes the inductive proof.

Recall that F has $\text{PL}[G]$ -representations. Since each $\text{PL}[G]$ -representation of (f, i) describes a function (namely f) that is monotone in the i -th argument, we obtain that all functions from F are locally monotone, which completes the proof. \blacktriangleleft

Now Propositions 8 and 10 yield the following central result.

► Theorem 11. *Let \mathcal{C} be some class of pointed Kripke structures. Let F and G be disjoint finite sets of Boolean functions such that all functions from F are locally monotone or some function from G is non-locally-monotone. Then $\text{ML}[F]$ has polynomial translations wrt. \mathcal{C} in $\text{ML}[G]$.*

Proof. By the assumptions on F and G , F has $\text{PL}[G]$ -representations by Proposition 10. Hence, by Proposition 8, $\text{ML}[F]$ has polynomial translations wrt. \mathcal{S}_K in $\text{ML}[G]$, i.e., for any formula $\varphi \in \text{ML}[F]$, there exists a formula $\psi \in \text{ML}[G]$ of polynomial size with $\varphi \equiv_{\mathcal{S}_K} \psi$. Since $\mathcal{C} \subseteq \mathcal{S}_K$, this implies $\varphi \equiv_{\mathcal{C}} \psi$. Hence, indeed, $\text{ML}[F]$ has polynomial translations wrt. \mathcal{C} in $\text{ML}[G]$. \blacktriangleleft

As we have mentioned at the beginning of this section, all the results also hold for other logics \mathcal{L} such as predicate logic, second-order predicate logic, temporal logic (linear or branching), and probably many more. Suppose that some function from G is non-locally-monotone. Since \leftrightarrow is non-locally-monotone, it follows that $\mathcal{L}[G]$ and $\mathcal{L}[\leftrightarrow]$ have polynomial translations in each other, i.e., they are equally succinct (up to a polynomial). Similarly, if all functions from G are locally monotone, then $\mathcal{L}[G]$ and $\mathcal{L} = \mathcal{L}[\emptyset]$ have polynomial translations in each other since all functions from \emptyset are locally monotone. In other words, for any set of Boolean functions G , $\mathcal{L}[G]$ is as succinct as $\mathcal{L}[\leftrightarrow]$ or as \mathcal{L} .

Thus, the situation looks similar for many logics: there are at most two “succinctness classes”. For propositional logic, it was shown by Pratt [11] that also $\text{PL}[\leftrightarrow]$ has polynomial translations in PL , i.e., that there is just one such class. In the following section, we will show that, whether or not modal logic has two “succinctness classes”, depends on the class of Kripke structures.

3 Where modal logics diverge

So far, we saw that for many logics \mathcal{L} , there are at most two “succinctness classes” (up to a polynomial), namely those of \mathcal{L} and $\mathcal{L}[\leftrightarrow]$, respectively. In this section, we will show that these classes differ for the modal logic T (and hence also for K) but coincide for the modal logic S5. We will therefore, from now on, be precise and return to the original notation, i.e., write $\equiv_{\mathcal{S}_K}$ instead of \equiv etc.

3.1 Where the “succinctness classes” differ

Our aim in this section is to show that $\text{ML}[\leftrightarrow]$ is exponentially more succinct than ML. Formally, we prove $\text{ML}[\leftrightarrow]$ does not have sub-exponential translations wrt. \mathcal{S}_T in ML (recall that \mathcal{S}_T is the class of pointed Kripke structures with reflexive accessibility relation).

► **Lemma 12.** *The logic $\text{ML}[\leftrightarrow]$ does not have sub-exponential translations wrt. \mathcal{S}_T in ML.*

Proof. Let $\varphi_0 = p_0$ and $\varphi_{n+1} = p_{n \oplus 1} \wedge (p \leftrightarrow \diamond \varphi_n)$ for $n \geq 0$, where $m \oplus n := (m + n) \bmod 2$. We will prove that $|\psi| \geq 2^n$ for any $n \geq 0$ and any $\psi \in \text{ML}$ with $\psi \equiv_{\mathcal{S}_T} \varphi_n$. Since $|\varphi_n|$ is linear in n , this ensures the lemma’s claim.

In this proof, we use the following notation. Let $\psi \in \text{ML}$ be any ML-formula. Then ψ is a Boolean combination of formulae of the form \top , \perp , $p \in P$, and $\diamond \lambda$ with $\lambda \in \text{ML}$. By E_ψ , we denote the set of all formulae $\lambda \in \text{ML}$ such that $\diamond \lambda$ appears in this Boolean combination with even negation depth. Similarly, O_ψ denotes the set of all formulae $\lambda \in \text{ML}$ such that $\diamond \lambda$ appears in this Boolean combination with odd negation depth.

A more formal definition proceeds as follows by induction:

$$E_\psi = \begin{cases} \emptyset & \text{if } \psi \in \{\top, \perp\} \cup P \\ \{\lambda\} & \text{if } \psi = \diamond \lambda, \lambda \in \text{ML} \\ O_\alpha & \text{if } \psi = \neg \alpha \\ E_\alpha \cup E_\beta & \text{if } \psi \in \{\alpha \wedge \beta, \alpha \vee \beta\} \end{cases} \quad O_\psi = \begin{cases} \emptyset & \text{if } \psi \in \{\top, \perp\} \cup P \\ \emptyset & \text{if } \psi = \diamond \lambda, \lambda \in \text{ML} \\ E_\alpha & \text{if } \psi = \neg \alpha \\ O_\alpha \cup O_\beta & \text{if } \psi \in \{\alpha \wedge \beta, \alpha \vee \beta\} \end{cases}$$

We now show, by induction on n , that any ML-formula ψ with $\psi \equiv_{\mathcal{S}_T} \varphi_n$ satisfies $|\psi| \geq 2^n$. Since φ_n uses (at most) the propositional variables p , p_0 , and p_1 , we can assume the same about ψ .

The case $n = 0$ is easy to see since any formula has size at least $1 = 2^0$. Let now $n \geq 0$ and consider the formula $\varphi_{n+1} = p_{n \oplus 1} \wedge (p \leftrightarrow \diamond \varphi_n)$ and assume there were an ML-formula $\psi \equiv_{\mathcal{S}_T} \varphi_{n+1}$ that satisfies $|\psi| < 2^{n+1}$.

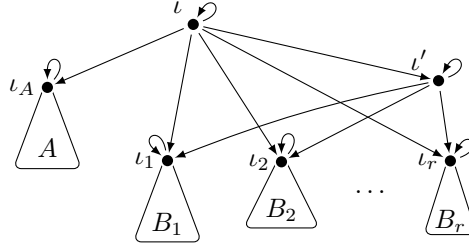
Note that, although the sets E_ψ and O_ψ may have non-empty intersection,

$$\left| \bigvee E_\psi \right| + \left| \bigvee O_\psi \right| \leq |\psi| < 2^{n+1},$$

hence at least one of these disjunctions must be of size $< 2^n$.

Case 1: $|\bigvee O_\psi| < 2^n$. Then, by the induction hypothesis, $\varphi_n \not\equiv_{\mathcal{S}_T} \bigvee O_\psi$. Let m denote the size of the formula $\varphi_n \wedge \neg \bigvee O_\psi$.

Let α be a formula from ML that is satisfiable in \mathcal{S}_T , of size at most m , and that uses no propositional variables other than p, p_0, p_1 . Then α has a model $A_\alpha \in \mathcal{S}_T$ that uses, at most, the propositional variables from α . Since there are only finitely many such formulae α , there exists a finite set $\mathcal{C}_m \subseteq \mathcal{S}_T$ of pointed Kripke structures such that every ML-formula has a model in \mathcal{C}_m , provided it is satisfiable in \mathcal{S}_T , of size at most m , and uses at most the propositional variables from $\{p, p_0, p_1\}$.



■ **Figure 3** Schematic representation of Kripke structure S with $V(\iota) = V(\iota') = \{p_{n \oplus 1}\}$.

Let O_ψ^+ be the set of formulae λ from O_ψ with $\lambda \models_{\mathcal{S}_T} \varphi_n$. Since $|\bigvee O_\psi^+| \leq |\bigvee O_\psi| < 2^n$, the induction hypothesis ensures $\bigvee O_\psi^+ \not\models_{\mathcal{S}_T} \varphi_n$. On the other hand, $\bigvee O_\psi^+ \models_{\mathcal{S}_T} \varphi_n$, hence the formula $\varphi_n \wedge \neg \bigvee O_\psi^+$ is satisfiable in \mathcal{S}_T and of size at most m . Consequently, there exists a structure $A = (W_A, V_A, R_A, \iota_A) \in \mathcal{C}_m$ with $A \models \varphi_n$ but $A \not\models \bigvee O_\psi^+$.

Let $B_1, \dots, B_r \in \mathcal{C}_m$ with $B_i = (W_i, R_i, V_i, \iota_i)$ be the models of $\neg \varphi_n$ from \mathcal{C}_m . We assume that the sets W_i for $i \in [r]$ and W_A are mutually disjoint.

We now define a Kripke structure $S = (W, R, V) \in \mathcal{S}_T$ as follows (cf. Fig. 3):

$$W = \{\iota, \iota'\} \uplus \bigcup_{i \in [r]} W_i \cup W_A$$

$$R = \{(\iota, \iota), (\iota, \iota'), (\iota', \iota'), (\iota, \iota_A)\} \cup \{(\iota, \iota') \times \{\iota_i \mid i \in [r]\}\} \cup \bigcup_{i \in [r]} R_i \cup R_A$$

$$V(w) = \begin{cases} \{p_{n \oplus 1}\} & \text{if } w \in \{\iota, \iota'\} \\ V_i(w) & \text{if } w \in W_i \text{ for some } i \in [r] \\ V_A(w) & \text{if } w \in W_A \end{cases}$$

From S , we obtain the pointed Kripke structures (S, ι) and (S, ι') by choosing the initial world as ι and ι' respectively. Note that both structures belong to \mathcal{S}_T since the accessibility relation R is reflexive. Our aim is to prove $(S, \iota) \models \neg \varphi_{n+1} \wedge \psi$, which contradicts the equivalence of φ_{n+1} and ψ .

First, we show $(S, \iota) \models \neg \varphi_{n+1}$. Recall that $A \models \varphi_n$ and therefore $(S, \iota_A) \models \varphi_n$. From $(\iota, \iota_A) \in R$, we obtain $(S, \iota) \models \neg p \wedge \diamond \varphi_n$, implying $(S, \iota) \models \neg \varphi_{n+1}$.

To prove $(S, \iota) \models \psi$, we first show $(S, \iota') \models \varphi_{n+1}$, implying $(S, \iota') \models \psi$ since we assumed $\varphi_{n+1} \equiv_{\mathcal{S}_T} \psi$. From this, we will infer that also $(S, \iota) \models \psi$.

Recall that $B_i \models \neg \varphi_n$ and therefore $(S, \iota_i) \models \neg \varphi_n$ for all $i \in [r]$. Furthermore, $(S, \iota') \models \neg p_{n \bmod 2}$ implies $(S, \iota') \models \neg \varphi_n$. Since $\{\iota', \iota_i \mid i \in [r]\}$ is the set of worlds accessible from ι' , this implies $(S, \iota') \models \neg \diamond \varphi_n$. Since, in addition, $(S, \iota') \models p_{n \oplus 1} \wedge \neg p$, we obtain $(S, \iota') \models \varphi_{n+1}$. Now $\varphi_{n+1} \equiv_{\mathcal{S}_T} \psi$ and $(S, \iota') \in \mathcal{S}_T$ imply $(S, \iota') \models \psi$.

The final step in our proof is the verification of $(S, \iota) \models \psi$. Recall that ψ is a Boolean combination of atomic formulae and of formulae $\diamond \lambda$ with $\lambda \in O_\psi \cup E_\psi$. Note that (S, ι) and (S, ι') agree in the atomic formulae holding there. Since O_ψ and E_ψ are the formulae $\diamond \lambda$ appearing in the Boolean combination with odd and even, respectively, negation depth, it suffices to show the following:

1. If $\lambda \in E_\psi$ with $(S, \iota') \models \diamond \lambda$, then $(S, \iota) \models \diamond \lambda$.
2. If $\lambda \in O_\psi$ with $(S, \iota') \not\models \diamond \lambda$, then $(S, \iota) \not\models \diamond \lambda$.

To demonstrate the former, suppose $\lambda \in E_\psi$ with $(S, \iota') \models \diamond \lambda$. Then $(S, \iota') \models \lambda$ or there is $i \in [r]$ with $(S, \iota_i) \models \lambda$. Since $(\iota, \iota') \in R$ and $(\iota, \iota_i) \in R$, we obtain $(S, \iota) \models \diamond \lambda$ in either case.

To demonstrate the second claim regarding $\lambda \in O_\psi$, we proceed by contraposition: so let $\lambda \in O_\psi$ with $(S, \iota) \models \diamond\lambda$. Here, we distinguish two cases.

- Suppose $\lambda \models_{\mathcal{S}_T} \varphi_n$, i.e., $\lambda \in O_\psi^+$. From $(S, \iota) \models \diamond\lambda$, we obtain that the formula λ holds in one of the worlds ι, ι', ι_A , or ι_i for some $i \in [r]$. But $(S, \iota_A) \models \lambda$ implies $A \models \lambda$, which is impossible since $A \not\models \bigvee O_\psi^+$ and $\lambda \in O_\psi^+$. But also $(S, \iota_i) \models \lambda$ and therefore $B_i \models \lambda$ is impossible since $\lambda \models_{\mathcal{S}_T} \varphi_n$, $B_i \in \mathcal{S}_T$, and $B_i \models \neg\varphi_n$. Consequently, the formula λ holds in one of the worlds ι and ι' . Again using $\lambda \models_{\mathcal{S}_T} \varphi_n$, we obtain that also φ_n holds in ι or in ι' . But this cannot be the case since $p_{n \bmod 2}$ does not hold in either of the two worlds – a contradiction. This finishes the verification of the second claim above in case $\lambda \in O_\psi^+ \subseteq O_\psi$.
- Finally, the case $\lambda \not\models_{\mathcal{S}_T} \varphi_n$ remains to be considered. But then the formula $\lambda \wedge \neg\varphi_n$ is satisfiable in \mathcal{S}_T . Since the size of this formula is bounded by m , there is some pointed Kripke structure $B \in \mathcal{C}_m$ with $B \models \lambda \wedge \neg\varphi_n$. The choice of the pointed Kripke structures B_1, \dots, B_r implies $B = B_i$ for some $i \in [r]$. Hence $(S, \iota_i) \models \lambda$. From $(\iota', \iota_i) \in R$, we obtain $(S, \iota') \models \diamond\lambda$.

This finishes the proof of the two numbered claims above. As explained there, they imply $(S, \iota) \models \psi$.

So, we proved $(S, \iota) \models \neg\varphi_{n+1} \wedge \psi$ in case $|\bigvee O_\psi| < 2^n$ contradicting the equivalence of φ_{n+1} and ψ .

Case 2: $|\bigvee E_\psi| < 2^n$. We consider the formula $\neg p_{n \oplus 1} \vee ((\neg p) \leftrightarrow \diamond\varphi_n) \equiv_{\mathcal{S}_K} \neg\varphi_{n+1} \equiv_{\mathcal{S}_T} \neg\psi$. Observe that $O_{\neg\psi} = E_\psi$, such that $|\bigvee O_{\neg\psi}| < 2^n$. Hence we can use the same argument as before for $\neg\psi$ to obtain a contradiction: simply label the worlds ι and ι' with $\{p, p_{n \oplus 1}\}$ instead of $\{p_{n \oplus 1}\}$. ◀

We now come to the classification of modal logics $\text{ML}[F]$ that have polynomial translations in ML.

► **Theorem 13.** *Let \mathcal{C} be either of the classes \mathcal{S}_K or \mathcal{S}_T . Let F be a finite set of Boolean functions. Then the following are equivalent:*

- (1) *All functions from F are locally monotone.*
- (2) *The set F has PL-representations.*
- (3) *$\text{ML}[F]$ has polynomial translations wrt. \mathcal{C} in ML.*
- (4) *$\text{ML}[\leftrightarrow]$ does not have sub-exponential translations wrt. \mathcal{C} in $\text{ML}[F]$.*
- (5) *$\text{ML}[\leftrightarrow]$ does not have polynomial translations wrt. \mathcal{C} in $\text{ML}[F]$.*

Proof. The implication (1) \Rightarrow (2) follows from Proposition 10 (with $G = \emptyset$), the implication (2) \Rightarrow (3) is Proposition 8 (again, with $G = \emptyset$). Now suppose (3) and assume, towards a contradiction, that (4) does not hold. Then $\text{ML}[\leftrightarrow]$ has sub-exponential translations in $\text{ML}[F]$ and $\text{ML}[F]$ has polynomial translations in ML. Since $f(n)^k$ is sub-exponential for any sub-exponential function f and any constant k , we get that $\text{ML}[\leftrightarrow]$ has sub-exponential translations in ML, contradicting Lemma 12. Thus, the implication (3) \Rightarrow (4) holds. The implication (4) \Rightarrow (5) is trivial. Finally, suppose (5) holds. Then, by Proposition 8 again, $\{\leftrightarrow\}$ does not have $\text{PL}[F]$ -representations. Hence, by Proposition 10 (with $F = \{\leftrightarrow\}$ and $G = F$), all functions in F are locally monotone. ◀

3.2 Where the “succinctness classes” collapse

In this section we show that, for every finite set of operators F , $\text{ML}[F]$ has polynomial translations wrt. \mathcal{S}_{S5} in ML. Recall that \mathcal{S}_{S5} is the set of pointed Kripke structures whose accessibility relation is an equivalence relation. Note that two formulae φ and ψ are equivalent wrt. \mathcal{S}_{S5} if, and only if, they are equivalent wrt. the class of pointed Kripke structures S from \mathcal{S}_{S5} whose accessibility relation is total. These pointed Kripke structures have the pleasant property that a \diamond -quantified formula either holds in every world or in none, i.e.,

$$S, w \models \diamond\varphi \quad \text{iff} \quad S, w' \models \diamond\varphi. \quad (4)$$

We use the above equivalence to prove that $\text{ML}[\leftrightarrow]$ -formulae can be “balanced” when considering structures from \mathcal{S}_{S5} only. That $\text{ML}[\leftrightarrow]$ has polynomial translations wrt. \mathcal{S}_{S5} in ML then forms an easy corollary. The general result for any set F then follows from Theorem 11.

For a formula $\varphi \in \text{ML}[F]$, let $\|\varphi\|$ denote the number of leaves of the syntax tree of φ , i.e., the total number of occurrences of propositional variables and constants \top and \perp . Furthermore, let $d(\varphi)$ be the depth of the syntax tree of φ , that is, the length of a longest path from the root to a leaf. In particular, $d(\varphi) = 0$ if, and only if, $\varphi \in P \cup \{\top, \perp\}$. If all operators in φ have arity at most r , the number of leaves, the size, and the depth of φ satisfy $\|\varphi\| \leq |\varphi| \leq r^{d(\varphi)+1}$.

► **Lemma 14.** *For every $\varphi \in \text{ML}[\leftrightarrow]$ there exists a formula $\varphi' \in \text{ML}[\leftrightarrow]$ with $\varphi' \equiv_{\mathcal{S}_{S5}} \varphi$ and $d(\varphi') \leq 8 \cdot (1 + \log_2 \|\varphi\|)$.*

► **Remark 15.** Since $\text{PL}[\leftrightarrow] \subseteq \text{ML}[\leftrightarrow]$, the above lemma implies that each formula $\varphi \in \text{PL}[\leftrightarrow]$ is equivalent to some $\text{PL}[\leftrightarrow]$ -formula of depth logarithmic in $\|\varphi\|$. According to Gashkov and Sergeev [4], a more general form of this result for propositional logic was known to Khrapchenko in 1967, namely that it holds for any complete basis of Boolean functions (e.g., for $\{\wedge, \vee, \neg, \leftrightarrow\}$ as here). They also express their regret that the only source for this is a single paragraph in a survey article by Yablonskii and Kozyrev [16], see also [7]. Often, it is referred to as Spira’s theorem who published it in 1971 [13], assuming that all at most binary Boolean functions are allowed in propositional formulae. Khrapchenko’s general form was then published by Savage [12].

Proof. Throughout the proof, we consider the logic $\text{ML}[\leftrightarrow]$ only and therefore simply speak of formulae when referring to $\text{ML}[\leftrightarrow]$ -formulae.

The proof proceeds by induction on $\|\varphi\|$. First assume that $\|\varphi\| = 1$. Then $\varphi = \text{op}_1 \text{op}_2 \cdots \text{op}_r \lambda$ with $r \geq 0$, $\text{op}_i \in \{\neg, \diamond\}$ for all $i \in [r]$, and $\lambda \in P \cup \{\top, \perp\}$. Using the equivalences $\neg\neg\alpha \equiv_{\mathcal{S}_{S5}} \alpha$, $\diamond\diamond\alpha \equiv_{\mathcal{S}_{S5}} \diamond\alpha$, and $\diamond\neg\diamond\alpha \equiv_{\mathcal{S}_{S5}} \neg\diamond\alpha$ (the latter two following from (4)), the formula φ is equivalent over \mathcal{S}_{S5} to a formula ψ of depth at most $3 \leq 8 \cdot (1 + \log_2 \|\varphi\|)$. This establishes the case $\|\varphi\| = 1$.

Otherwise, $\|\varphi\| \geq 2$ and φ contains some binary operator. Let $m = \|\varphi\|$. Intuitively, we split the formula φ into two parts, each containing about half the leaves from φ . Formally, there are formulae $\alpha(x)$ with only one occurrence of x and β such that $\varphi = \alpha(\beta)$,

- $\|\beta\| > \frac{m}{2}$, and
- $\beta = \text{op}(\beta_1, \beta_2)$ with $\text{op} \in \{\wedge, \vee, \leftrightarrow\}$ and $\|\beta_1\|, \|\beta_2\| \leq \frac{m}{2}$.

It is not difficult to find such formulae: simply start at the root of the syntax tree of φ and proceed towards the leaves in the direction of the child that contains more than half the leaves of φ (while there is one). The vertex, in which the procedure stops, corresponds to the operator op in $\beta = \text{op}(\beta_1, \beta_2)$ above. Note that $\|\alpha(x)\| = m - \|\beta\| + 1 < m - \frac{m}{2} + 1$, i.e., $\|\alpha(x)\| \leq \frac{m}{2}$.

First assume that x does not occur under a \diamond -operator in $\alpha(x)$. Then

$$\varphi = \alpha(\beta) \equiv_{\mathcal{S}_{55}} (\alpha(\perp) \wedge \neg\beta) \vee (\alpha(\top) \wedge \beta) \quad (5)$$

since, in both formulae, β is interpreted in the initial-world (hence the formulae are even equivalent over \mathcal{S}_K). By induction hypothesis, there exist formulae $\alpha'(x)$, β'_1 , and β'_2 with

- $\alpha'(x) \equiv_{\mathcal{S}_{55}} \alpha(x)$ and $d(\alpha'(x)) \leq 8 \cdot (1 + \log \frac{m}{2}) = 8 \cdot \log_2 m$ as well as
- $\beta'_i \equiv_{\mathcal{S}_{55}} \beta_i$ and $d(\beta'_i) \leq 8 \cdot (1 + \log \frac{m}{2}) = 8 \cdot \log_2 m$ for $i \in \{1, 2\}$.

Set $\beta' = \text{op}(\beta'_1, \beta'_2)$. Then β' is equivalent to β over \mathcal{S}_{55} and of depth at most $1 + 8 \cdot \log_2 m$. From (5), it follows that $\varphi \equiv_{\mathcal{S}_{55}} (\alpha'(\perp) \wedge \neg\beta') \vee (\alpha'(\top) \wedge \beta') =: \varphi'$. Furthermore, the depth of φ' satisfies

$$\begin{aligned} d(\varphi') &= \max \{2 + d(\alpha'(x)), 3 + d(\beta')\} \\ &= \max \{2 + 8 \cdot \log_2 m, 3 + 1 + 8 \cdot \log_2 m\} \leq 4 + 8 \cdot \log_2 m \leq 8 \cdot (1 + \log_2 m), \end{aligned}$$

which completes the first case, where x does not occur under a \diamond in $\alpha(x)$.

Otherwise, the variable x occurs in the scope of a \diamond -operator. Now, we split $\alpha(x)$ at the last such \diamond , i.e., there exist formulae $\alpha_1(y)$ with only one occurrence of y and $\alpha_2(x)$ with only one occurrence of x that, furthermore, does not lie under a \diamond -operator, such that $\alpha(x) = \alpha_1(\diamond\alpha_2(x))$. In particular, $\|\alpha_1(y)\|, \|\alpha_2(x)\| \leq \|\alpha(x)\| \leq \frac{m}{2}$. By induction hypothesis, there exist $\alpha'_1(y)$, $\alpha'_2(x)$, β'_1 , and β'_2 with

- $\alpha'_i(z) \equiv_{\mathcal{S}_{55}} \alpha_i(z)$ and $d(\alpha'_i(z)) \leq 8 \cdot \log_2 m$ for $i \in \{1, 2\}$ as well as
- $\beta'_i \equiv_{\mathcal{S}_{55}} \beta_i$ and $d(\beta'_i) \leq 8 \cdot \log_2 m$ for $i \in \{1, 2\}$.

As before, let $\beta' = \text{op}(\beta'_1, \beta'_2)$ with $\beta' \equiv_{\mathcal{S}_{55}} \beta$ and $d(\beta') \leq 1 + 8 \cdot \log_2 m$. Now, consider the formulae

$$\psi' = (\alpha'_2(\perp) \wedge \neg\beta') \vee (\alpha'_2(\top) \wedge \beta') \quad \text{and} \quad \varphi' = (\alpha'_1(\perp) \wedge \neg\psi') \vee (\alpha'_1(\top) \wedge \psi').$$

Then $\psi' \equiv_{\mathcal{S}_{55}} \alpha_2(\beta)$, since x does not occur under a \diamond in $\alpha_2(x)$, hence β is interpreted in the initial world in both formulae. But also $\varphi' \equiv_{\mathcal{S}_{55}} (\alpha_1(\perp) \wedge \neg\psi') \vee (\alpha_1(\top) \wedge \psi') \equiv_{\mathcal{S}_{55}} \alpha_1(\diamond\psi')$ since, whether or not $\diamond\psi'$ holds in a particular world, does not depend on the choice of the world (see (4)). Hence $\varphi' \equiv_{\mathcal{S}_{55}} \alpha_1(\diamond\psi') \equiv_{\mathcal{S}_{55}} \alpha_1(\diamond\alpha_2(\beta)) = \varphi$. Similar to the first case, one can show

$$\begin{aligned} d(\psi') &= \max \{2 + d(\alpha'_2(x)), 3 + d(\beta')\} \leq 4 + 8 \cdot \log_2 m \quad \text{and} \\ d(\varphi') &= \max \{2 + d(\alpha'_1(y)), 4 + d(\psi')\} \\ &\leq \max \{2 + 8 \cdot \log_2 m, 8 + 8 \cdot \log_2 m\} \leq 8 \cdot (1 + \log_2 m), \end{aligned}$$

which completes the second case and hence the inductive proof. \blacktriangleleft

Let $\varphi \in \text{ML}[\leftrightarrow]$. By the previous lemma, there exists an $\text{ML}[\leftrightarrow]$ -formula ψ that is equivalent to φ over \mathcal{S}_{55} and has depth $d(\psi) \leq 8 \cdot (1 + \log_2 \|\varphi\|)$. Let φ' be obtained from ψ by replacing each sub-formula $\alpha \leftrightarrow \beta$ by $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$. Then φ' belongs to ML , is equivalent to φ over \mathcal{S}_{55} , and the depth increases at most by a factor of three, i.e., $d(\varphi') \leq 3 \cdot d(\psi) \leq 3 \cdot 8 \cdot (1 + \log_2 \|\varphi\|) = 24 \cdot (1 + \log_2 \|\varphi\|)$. Since all operators in φ' are at most binary, it follows that

$$|\varphi'| \leq 2^{d(\varphi')+1} \leq 2^{24 \cdot (1 + \log_2 \|\varphi\|)+1} \leq c \cdot \|\varphi\|^{c'} \leq c \cdot |\varphi|^{c'} \quad \text{for some constants } c, c' > 0.$$

Hence we verified the following claim.

► **Lemma 16.** $\text{ML}[\leftrightarrow]$ has polynomial translations wrt. \mathcal{S}_{S5} in ML.

Therefore, the modal logic S5 has only one “succinctness class”.

► **Theorem 17.** Let F be a finite set of Boolean functions. Then $\text{ML}[F]$ has polynomial translations wrt. \mathcal{S}_{S5} in ML.

Proof. Since \leftrightarrow is non-locally-monotone, it follows by Theorem 11 that $\text{ML}[F]$ has polynomial translations wrt. \mathcal{S}_{S5} in $\text{ML}[\leftrightarrow]$ and therefore in ML by Lemma 16 above. ◀

4 Conclusion

This paper considers the question whether or not the use of additional Boolean functions allows for more succinct formulae. For many logics, elimination of locally monotone functions is possible with polynomial size increase; for arbitrary functions, this holds if we allow bi-implication to appear in the resulting formula. Regarding propositional logic, it is known that also bi-implication can be eliminated with polynomial size increase. The same applies for modal logic if we restrict the class of Kripke structures to equivalence relations. When considering all reflexive Kripke structures however, this is no longer the case – bi-implication cannot be eliminated in modal logic without introducing an exponential size increase when considering a class of Kripke structures that contains all reflexive structures. It remains open, where exactly the change from polynomial to exponential size increase occurs, e.g., whether bi-implication can be eliminated with polynomial size increase when considering all reflexive and symmetric or all reflexive and transitive Kripke structures.

References

- 1 M. Adler and N. Immerman. An $n!$ lower bound on formula size. *ACM Trans. Comput. Log.*, 4(3):296–314, 2003.
- 2 K. Etessami, M. Vardi, and T. Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.
- 3 T. French, W. van der Hoek, P. Iliev, and B.P. Kooi. On the succinctness of some modal logics. *Artif. Intell.*, 197:56–85, 2013.
- 4 S.B. Gashkov and I.S. Sergeev. О значении работ В. М. Храпченко. *Прикладная дискретная математика*, 2020(48):109–124, 2020. doi:10.17223/20710410/48/10.
- 5 M. Grohe and N. Schweikardt. The succinctness of first-order logic on linear orders. *Log. Methods Comput. Sci.*, 1(1), 2005.
- 6 L. Hella and M. Vilander. Formula size games for modal logic and μ -calculus. *J. Log. Comput.*, 29(8):1311–1344, 2019.
- 7 S. Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012.
- 8 C Lutz. Complexity and succinctness of public announcement logic. In *AAMAS’06*, pages 137–143. ACM, 2006.
- 9 C. Lutz, U. Sattler, and F. Wolter. Modal logic and the two-variable fragment. In *CSL’01*, Lecture Notes in Computer Science vol. 2142, pages 247–261. Springer, 2001.
- 10 N. Markey. Temporal logic with past is exponentially more succinct. *Bull. EATCS*, 79:122–128, 2003.
- 11 V.R. Pratt. The effect of basis on size of Boolean expressions. In *16th Annual Symposium on Foundations of Computer Science*, pages 119–121, 1975. doi:10.1109/SFCS.1975.29.
- 12 J.E. Savage. *The Complexity of Computing*. Wiley, 1976.
- 13 P.M. Spira. On time-hardware tradeoffs for Boolean functions. In *Proc. of 4th Hawaii Intern. Symp. on System Sciences*, pages 525–527, 1971.

- 14 H. van Ditmarsch, Jie Fan, W. van der Hoek, and P. Iliev. Some exponential lower bounds on formula-size in modal logic. In Rajeev Goré, Barteld P. Kooi, and Agi Kurucz, editors, *Advances in Modal Logic 2014*, pages 139–157. College Publications, 2014.
- 15 T. Wilke. Ctl^+ is exponentially more succinct than CTL. In *FSTTCS'99*, Lecture Notes in Computer Science vol. 1738, pages 110–121. Springer, 1999.
- 16 S.V. Yablonskii and V.P. Kozurev. Математические вопросы кибернетики. Информационные материалы, Академия наук СССР научный совет по комплексной проблеме "кибернетика", 19a:3–15, 1968.

Temporalizing Digraphs via Linear-Size Balanced Bi-Trees

Stéphane Bessy ✉

LIRMM, Univ Montpellier, CNRS, Montpellier, France

Stéphan Thomassé ✉

Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

Laurent Viennot ✉

Inria Paris, DI ENS, Paris, France

Abstract

In a directed graph D on vertex set v_1, \dots, v_n , a *forward arc* is an arc $v_i v_j$ where $i < j$. A pair v_i, v_j is *forward connected* if there is a directed path from v_i to v_j consisting of forward arcs. In the **Forward Connected Pairs Problem (FCPP)**, the input is a strongly connected digraph D , and the output is the maximum number of forward connected pairs in some vertex enumeration of D . We show that FCPP is in APX, as one can efficiently enumerate the vertices of D in order to achieve a quadratic number of forward connected pairs. For this, we construct a linear size balanced bi-tree T (an out-branching and an in-branching with same size and same root which are vertex disjoint in the sense that they share no vertex apart from their common root). The existence of such a T was left as an open problem (Brunelli, Crescenzi, Viennot, Networks 2023) motivated by the study of temporal paths in temporal networks. More precisely, T can be constructed in quadratic time (in the number of vertices) and has size at least $n/3$. The algorithm involves a particular depth-first search tree (Left-DFS) of independent interest, and shows that every strongly connected directed graph has a balanced separator which is a circuit. Remarkably, in the request version **RFCPP** of FCPP, where the input is a strong digraph D and a set of requests R consisting of pairs $\{x_i, y_i\}$, there is no constant $c > 0$ such that one can always find an enumeration realizing $c \cdot |R|$ forward connected pairs $\{x_i, y_i\}$ (in either direction).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases digraph, temporal graph, temporalization, bi-tree, {in-branching, out-branching, in-tree, out-tree}, forward connected pairs, left-maximal DFS

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.13

Related Version *Full Version*: <https://arxiv.org/abs/2304.03567>

Funding This research was partially supported by the ANR project Digraphs ANR-19-CE48-0013 and the ANR project Tempogral ANR-22-CE48-0001.

1 Introduction

Motivated by network design applications, the following problem of scheduling the arcs of a multi-digraph was mentioned as an open problem in [8] and formally introduced in [2]. The **Maximum Reachability Edge Temporalisation (MRET)** consists in assigning a time label $t_{xy} \in \mathbb{N}$ to each arc xy of a digraph $D = (V, A)$ so as to maximize the number of pairs x, z of vertices connected by a *temporal path*, that is a path from x to z where time labels strictly increase along the path. In the acyclic case, all existing paths can be made temporal using a topological ordering of the vertices, and transferring the index of a node x to every arc leaving x . The problem becomes particularly intriguing in the strongly connected case in which every pair of vertices are connected by a path. It was shown in [2] that MRET is NP-hard



© Stéphane Bessy, Stéphan Thomassé, and Laurent Viennot;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 13; pp. 13:1–13:12



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



even restricted to strongly connected digraphs. In the same paper, the authors suggest that MRET is in APX by conjecturing, in any strongly connected digraph, the existence of some arc-disjoint in-branching T^- and out-branching T^+ (allowing arbitrary overlapping in terms of vertices), both with linear size and rooted at the same vertex r .

We show that this conjecture holds, and since the branchings can be constructed in polynomial time, that MRET is indeed in APX in the strongly connected case. The reason is that given such branchings T^- and T^+ , it is then straightforward to schedule the arcs of T^- from leaves to r , and then the arcs of T^+ from r to leaves, to obtain an arc scheduling temporally connecting $|T^-| \cdot |T^+|$ pairs of vertices. If both branchings span a fraction c of vertices, for some $c > 0$, then this scheduling temporally connects at least a fraction c^2 of all pairs which guarantees approximation ratio at most $1/c^2$.

Related work

The undirected version of the MRET problem is studied in [10] where it is proven that it is NP-complete to decide whether an undirected graph has an edge scheduling connecting all pairs. However, one can easily produce an edge scheduling connecting a constant fraction of pairs by decomposing a spanning tree at a centroid c so as to produce two disjoint trees T, T' with the same root c and each of them covering one third of the vertices. It is then straightforward to schedule edges so as to connect a constant fraction of pairs. A similar (undirected) problem where an edge can be scheduled several times is considered in a series of papers [1, 11, 12]. The goal is then to minimize the total number of time labels used while respecting some constraints with respect to connectivity or the maximum time label used in particular. Note that scheduling each edge twice is sufficient for temporally connecting every pair of nodes of a connected undirected graph (a first set of labels can allow each vertex to reach the root of a spanning tree, while a second set of labels can allow the root to reach every node). The MRET problem can be seen as a simplified version of the problem of scheduling buses in a public transit network [8]. Related problems [9, 13] target the minimization of temporally connected pairs and are driven by applications to mitigation of epidemic propagation.

It is shown in [3] that it is NP-complete to decide whether a strongly connected digraph contains an in-branching and an out-branching with same root, which are edge disjoint, and that both span all vertices. The same paper also relates the conjecture that such branchings exist if the digraph is c -edge-connected for sufficiently large c . The conjecture holds for $c = 2$ in digraphs with independence number at most 2 [4]. It is also shown in [5] that it is NP-complete to decide whether a strongly connected digraph has a partition of its vertices into two parts of size at least 2, such that the first part is spanned by an in-branching and the second part is spanned by an out-branching.

Main results

Our approach is to address these problems from a vertex point of view, which is enough to obtain an approximation algorithm for MRET. Specifically, we consider two maximization problems: In the **Forward Connected Pairs Problem (FCPP)** the goal is to find an enumeration (or ordering) v_1, \dots, v_n of a strongly connected digraph $D = (V, A)$ such that the number of pairs v_i, v_j joined by a directed path with increasing indices (called *forward pairs*) is maximized. In the **Balanced Bi-Tree Problem (BBTP)** the goal is to find an in-tree and an out-tree only intersecting at their root (thus making a *bi-tree*) with equal size (to be maximized). (We use in-tree and out-tree instead of in-branching and out-branching to stress

the requirement that they share no vertex apart from their common root, and hence do not span the digraph.) Notice that the former problem, FCPP, is NP-hard, as finding an enumeration joining $n(n-1)/2$ pairs of vertices with forward paths is equivalent to finding a Hamilton path in the instance (and Hamilton path in general digraphs can easily be reduced to Hamilton path in strong digraphs).

The main result of this paper is that one can find a solution T of BBTP in time $O(n^2)$ with size at least $n/3 - 1$. Therefore each in-tree and out-tree has size at least $n/6$, and by considering any enumeration of V extending a topological ordering of T , we obtain a solution of FCPP of size at least $n^2/36$. Since the maximum possible solution of FCPP is $n^2/2$, this gives a $1/18$ approximation for the **Forward Connected Pairs Problem**. Our construction can be extended to a weighted digraph where each vertex u is associated to a weight w_u . It then produces a bi-tree where the in-tree and the out-tree both have total weight at least $W/6$, where $W = \sum_{u \in V} w_u$ is the total weight of the digraph.

We also consider a covering version CFCPP of FCPP where we look for a minimal set of orderings of the vertex set such that for every pair x, y , one of xy or yx is a forward pair in one of the orderings. Since we can cover a positive fraction of pairs, it is natural to wonder if CFCPP always admits a solution with a constant number of orderings (or at most $\log n$). To this end, we consider a request variant of the problem, called RFCPP, where we ask to connect by forward paths a maximum number of pairs among a given set $R \subseteq \binom{V}{2}$ of requested pairs. We provide a family of instances of RFCPP where the number of needed ordering to satisfy all the requests of R is more than a constant fraction of $|R|$. We do not know if CFCPP and RFCPP can be efficiently approximated. This still leaves open the existence of an $O(\log n)$ solution for CFCPP. Note that both CFCPP and RFCPP are NP-hard for the same reason as FCPP. The approximation of the variant of FCPP extended to general digraphs is also left open. We think that solving the strong case is a key step towards this more ambitious goal since the acyclic case can easily be solved exactly as mentioned previously.

Main techniques: left-maximal DFS and balanced circuit separators

From an algorithmic perspective, our solution relies on finding a *cyclic balanced separator* C of D . Specifically, the vertex set of D is partitioned into three parts I, C, O such that C spans a circuit, no arc links a node from I to a node in O , and which is balanced in the sense that both $I \cup C$ and $C \cup O$ have size at least $n/3$. Note that I and O can be empty, as this is the case when D is a circuit.

Such a partition can be computed in linear time from a left-maximal depth-first-search (DFS) tree, that is a DFS tree such that the children of any node are ordered from left-to-right by non-increasing sub-tree size. Both of these structures could be of independent interest in the field of digraph algorithms. The computation of a left-maximal DFS is the (quadratic) complexity bottleneck of our algorithm. We feel that a linear-time algorithm for finding a cyclic balanced separator should be achievable either by other means, or by relaxing the requirement on left-maximal DFS (we just need it to be “not too much unbalanced to the right”). However, actually computing in linear time a left-maximal DFS tree could prove more challenging. Could decremental SCC help [7]?

2 Definitions

In this paper we consider directed graphs $D = (V, A)$ (*digraphs*) in which cycles of length two are allowed. The set V is the set of *vertices* (usually n of them) and A is the set of *arcs*. We say that x, y are *connected* in D if there exists a directed path from x to y . A digraph is *strongly connected*, or simply *strong*, if all x, y are connected. In particular, if D has n

13:4 Temporalizing Digraphs via Linear-Size Balanced Bi-Trees

vertices, the number of connected couples x, y is n^2 . The *out-section* generated by a vertex x of D is the set of vertices y for which x, y are connected. We say that x *out-generates* D if the out-section of x is V .

A *schedule* is an injective mapping f of A into the positive integers. In a scheduled digraph (D, f) we say that a couple of vertices x, y is *connected* if there is a directed path $x = x_1, x_2, \dots, x_k = y$ such that $f(x_i x_{i+1}) < f(x_{i+1} x_{i+2})$ for all $i = 1, \dots, k - 2$. We denote by $c(D, f)$ the number of connected couples and by $s(D)$ the maximum over all choices of f of $c(D, f)$. We now define our scheduled ratio r_s which is the infimum of $s(D)/n^2$ over strongly connected digraphs D on n vertices, when n goes to infinity.

This ratio r_s is based on scheduling arcs, and a similar ratio r_t can be defined via a total order on vertices. We consider for this a digraph D and a total ordering $<$ on its vertices. An arc xy of D is *forward* if $x < y$. We say that a couple (x, y) is *connected* in $(D, <)$ if there is a directed path from x to y consisting of forward arcs, and we then call (x, y) a *forward couple*. We denote by $c(D, <)$ the number of forward couples and by $t(D)$ the maximum over all choices of $<$ of $c(D, <)$. The ordered ratio r_t is the infimum of $t(D)/n^2$ over strongly connected digraphs D on n vertices, when n goes to infinity.

The main problem we address in this paper is to show that both r_s and r_t are positive. Let us first prove that these questions are related.

► **Theorem 1.** $r_t \leq r_s$

Proof. Given $(D, <)$, we can consider that $<$ is a bijective mapping g from V to $0, \dots, n - 1$ respecting the order (that is $x < y$ whenever $g(x) < g(y)$). Now observe that if one define $f(xy) = n \cdot g(x) + g(y)$ for every arc, then every forward couple in $(D, <)$ is a connected pair in (D, f) . ◀

Thus we can focus on the following problem.

► **Problem 2.** *What is the value of r_t ?*

The fact that $r_t > 0$ is not obvious and is indeed our central result. Observe that we can assume that D is *minimally strongly connected*, i.e. every arc xy of D is the unique directed x, y -path in D . A classical result using ear-decompositions asserts that the number of arcs of a minimally strongly connected digraph is at most $2n - 2$ (see [6] for instance). Unfortunately we were unable to use these decompositions to prove the positivity of r_t . Our strategy instead is to find inside D a particular type of oriented tree.

An *out-tree* T^+ is an orientation of a tree in which one *root* vertex out-generates T^+ . Reversing all arcs, we obtain an *in-tree*. When identifying the root r of an in-tree T^- and the root of an out-tree T^+ , we obtain a tree orientation called a *bi-tree* T where r is the *center*. Note that x, y are connected for every $x \in T^-$ and $y \in T^+$. We say that a bi-tree is *balanced* if $|T^+| = |T^-|$. In particular, if every strongly connected D contains a balanced bi-tree of linear size, one directly obtains that $r_t > 0$ for Problem 2.

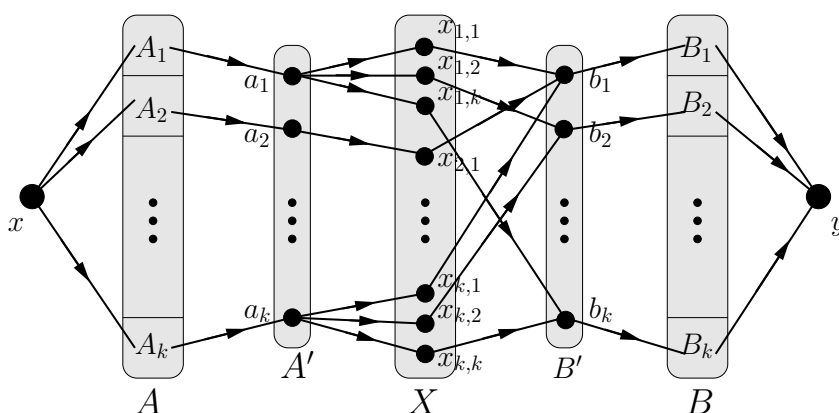
► **Problem 3.** *What is the maximum c_b for which every strongly connected directed graph on n vertices has a balanced bi-tree of size at least $c_b \cdot n$?*

We show in Theorem 11 that $c_b \geq 1/3$, where both $|T^+|$ and $|T^-|$ have size at least $1/6$. One can naturally ask if the enumeration problem directly implies the bi-tree problem. But the following example shows that this is not true in general.

► **Proposition 4.** For every integer k , there exists a minimally strongly connected digraph D on $n = 3k^2 + 2k + 2$ vertices admitting an enumeration $E = v_1, \dots, v_n$ such that:

- the number of forward couples in E is quadratic in n (at least k^4), and
- the maximal size of a balanced bi-tree only using forward arcs in E is $2k + 5 = O(\sqrt{n})$.

Proof. Let D be the digraph with vertex set $\{x\} \cup A \cup A' \cup X \cup B' \cup B \cup \{y\}$ where A' , B' have size k and A , B and X all have size k^2 . Consider $(A_i)_{1 \leq i \leq k}$ a partition of A into k sets of size k and $(B_i)_{1 \leq i \leq k}$ a partition of B into k sets of size k . Moreover, we denote by a_1, \dots, a_k the vertices of A' , b_1, \dots, b_k the vertices of B' , and $(x_{i,j})_{1 \leq i,j \leq k}$ the vertices of X . Now, add to D the following sets of arcs: $\{xa : a \in A\}$, $\{aa_i : a \in A_i\}$ for all $1 \leq i \leq k$, $\{a_i x_{i,j} : j = 1, \dots, k\}$ for all $1 \leq i \leq k$, $\{x_{i,j} b_j : i = 1, \dots, k\}$ for all $1 \leq j \leq k$, $\{b_j b : b \in B_j\}$ for all $1 \leq j \leq k$, $\{by : b \in B\}$ and $\{yx\}$. The construction is depicted in Figure 1.



■ **Figure 1** The digraph D in the proof of Proposition 4. An arc between a block and a particular vertex stands for all the arcs between each vertex of the block and the particular vertex. The arc yx is not drawn.

The digraph D is strongly connected and has $n = 3k^2 + 2k + 2$ vertices. Furthermore, D is minimally strongly connected, as for every arc uv we have either $d^+(u) = 1$ (if $uv = yx$ or $u \in A \cup X \cup B$) or $d^-(v) = 1$ (if $v \in A \cup X \cup B$).

Consider now any enumeration of D where x is the first vertex, then A is before A' , then A' is before X , then X is before B' , then B' is before B , and finally y is the last vertex. For such an enumeration all the arcs of D are forward except yx . For every $1 \leq i, j \leq k$, any vertex of A_i has a path to any vertex of B_j using the vertex $x_{i,j}$. So the number of forward couples is at least $|A| \cdot |B| = k^4$, which is quadratic in n . However, the largest balanced bi-tree only using forward arcs has its center in X and has size $2k + 5 = O(\sqrt{n})$. Indeed, any node $x_{i,j} \in X$ has only one in-neighbor which has itself k in-neighbors, and only one out-neighbor which has itself k out-neighbors, resulting in $2k + 3$ nodes which can further connect to x and y only. ◀

3 Computing a left-maximal depth first search tree

Given an out-tree T and a node x of T , we denote by T_x the *subtree* rooted at x consisting of all vertices in the out-section of x in T . A *child* of x is an out-neighbor of x in T . Let $D = (V, A)$ be a directed graph. A *depth first search tree* T of D (*DFS-tree* for short) is an out-tree which is a spanning subgraph of D with the following properties:

13:6 Temporalizing Digraphs via Linear-Size Balanced Bi-Trees

- For every node x , there is a total order \leq_x on the children of x , which is called a *left-right order*.
- If y, z are two children of x and there exists an arc of D from T_z to T_y , then $y \leq_x z$ (i.e. arcs between disjoint subtrees goes from right to left).

We are interested in a particular type of DFS-tree called *left-maximal* in which we have $|T_y| \geq |T_z|$ for every node x and children y, z such that $y \leq_x z$. In other words the size of the child subtrees of any vertex is non-increasing from left to right.

► **Theorem 5.** *If x out-generates D , then there exists a left-maximal DFS-tree T rooted at x . Moreover T can be computed in quadratic time in minimally strongly connected digraphs.*

Proof. We construct a left-maximal DFS-tree rooted at x as follows. Compute the strongly connected components of $D \setminus \{x\}$ and their sizes. Consider the acyclic digraph D' between components where an arc (C, C') indicates that there is an arc from C to C' in D . By traversing D' according to a reverse topological order, we obtain the size of the out-section of each node in $D \setminus \{x\}$. Let y be a node with out-section S_y having maximum size. Note that y belongs to some strongly connected component C which is a source in D' . Since D is strongly connected, C contains an out-neighbor of x . Free to choose y in C , we assume for simplicity that xy is indeed an arc. Construct recursively a left-maximal DFS-tree T' rooted at x in $D \setminus S_y$, and a left-maximal DFS-tree T'' rooted at y in the digraph induced by S_y . The final out-tree T is obtained by inserting T'' as the leftmost child of x in T' . It is indeed a DFS-tree as there exists no arc from S_y to any node in $D \setminus T'$ by the definition of outsection. To realize that it is also left-maximal, consider the leftmost child z of x in T' . The outsection S_z of z in $D \setminus \{x\}$ has size at most $|S_y|$, and we thus have $|T_z| \leq |S_z| \leq |S_y| = |T_y|$.

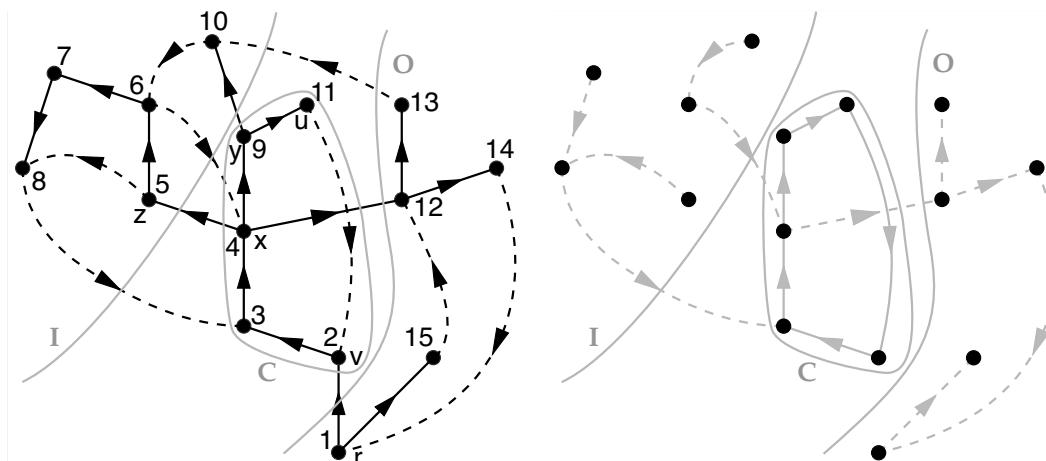
As the computation of strongly connected components, digraph D' , and outsection sizes can be done in linear time, the whole computation can be done in $O(n^2)$ time in minimally strongly connected graphs (where the number of arcs is linear in the number of vertices). ◀

► **Problem 6.** *Can we compute a left-maximal DFS-tree in $o(n^2)$ time in minimally strongly connected digraphs? What about the complexity in the general case?*

Let us now introduce our key-definition which is a particular type of partition of a strongly connected digraph D . To get beforehand a bit of intuition, one can picture a connected (undirected) graph G with a depth first search tree T drawn on the plane. The key-feature here is that any path P from the root to a leaf partitions the rest of the graph into two subsets L and R which are respectively the vertices of $V \setminus P$ to the left and to the right of P . In other words, G has a cutset $V(P)$ (i.e. its removal splits the graph into several connected components) with a remarkable property since it is spanned by a path. Observe also that *any* root-leaf path can be used, so by a classical left-right sweeping argument, one can find P such that both L and R have size at most $2n/3$ (so that the cut is balanced). We now generalize this argument to strongly connected digraphs, where a directed cycle takes over the role of P .

An (I, C, O) -decomposition of a strongly connected digraph is defined as:

- a partition of V into three subsets I, C, O ,
- C is spanned by a directed cycle,
- there is no arc from I to O .



■ **Figure 2** Left: a strong digraph (all arcs) with a left-maximal DFS-tree T (plain arcs) and the (I, C, O) decomposition associated to $T_{x,y}$. Nodes are numbered according to the corresponding DFS traversal. Right: the cycle spanning C (plain arcs), two in-trees spanning I and one out-tree spanning O .

Observe that, by strong connectivity and the fact that there is no arc from I to O , for every vertex x in I , there exists a directed path from x to C with internal vertices inside I . Similarly, for every vertex x in O , there exists a directed path from C to x with internal vertices inside O . We now show how to get a (I, C, O) -decomposition which is additionally *balanced*, that is such that both $I \cup C$ and $O \cup C$ have size greater than $n/3$.

Given a DFS-tree T , we call *left path* L_T the path starting at the root of T and which iteratively selects the leftmost child of the current vertex as the next vertex of the path. In particular, in a planar drawing respecting the left-right order of children, L_T is the path from the root to the leftmost leaf. Let x be a vertex of the left path of L_T . Given a child y of x , we define the subtree $T_{x,y}$ of T as the subtree of T_x containing x and all T_z for $z \leq x y$. We call $T_{x,y}$ a *left subtree* of T . Note that T_x is the left subtree $T_{x,y}$ obtained by selecting y as its rightmost child. Also, if y is the leftmost child, x has only one child in $T_{x,y}$.

► **Proposition 7.** *For every left subtree $T_{x,y}$, there exists an (I, C, O) -decomposition such that $T_{x,y}$ is included in $I \cup C$ and $V \setminus T_{x,y}$ is included in $O \cup C$.*

Proof. As T is a DFS-tree, any arc outgoing from $T_{x,y} \setminus \{x\}$ reaches a vertex in $T_{x,y}$ or in L_T . By strong connectivity, some arc uv with $u \in T_{x,y} \setminus \{x\}$ reaches some vertex in L_T between the root and x (possibly the root or x), and we select uv such that v has minimum distance from the root in T (see Figure 2 (Left) for an example). Note that v can be equal to x (some arc $u'v'$ with $u' \in T_x \setminus T_{x,y}$ must then lead to a vertex $v' \in L_T$ closer to the root by strong connectivity). We define C as the cycle formed the path P_{vu} from v to u in T and the arc uv . We now set $I := V(T_{x,y}) \setminus C$ and $O = V \setminus (C \cup I)$. Every arc leaving $T_{x,y} \setminus \{x\}$ reaches a vertex w of the left-path between v and x by the choice of v . Thus w is in C , and hence there is no arc from I to O . ◀

► **Proposition 8.** *Every strongly connected directed graph with $n \geq 4$ vertices has a DFS-tree T which has a left subtree $T_{x,y}$ such that $n/3 < |T_{x,y}| < 2n/3$.*

Proof. We consider a left-maximal DFS-tree T of D . Scanning the left-path of T from the root, we consider the first node z such that $|T_z| \leq n/3$. Let x be the parent of z . If $|T_z| = n/3$, the left subtree $T_{x,z}$ with size $n/3 + 1$ is the solution we are looking for. Let y be the rightmost child of x such that $|T_{x,y}| < 2n/3$. Assume for contradiction that $|T_{x,y}| \leq n/3$. By the definition of z , we have $|T_x| > n/3$, hence y is not the rightmost child of x . In particular, y has a (next) right sibling y' . As $|T_{y'}| \leq |T_z| < n/3$ by left-maximality, we reach a contradiction to the choice of y since $|T_{x,y'}| < 2n/3$. Thus $T_{x,y}$ is our solution. ◀

► **Corollary 9.** *Every strongly connected directed graph has a balanced (I, C, O) -decomposition. (i.e. both $I \cup C$ and $O \cup C$ have size strictly greater than $n/3$).*

Proof. If $n \geq 4$, then this is a direct consequence of Proposition 7 and Proposition 8. The case $n \leq 3$ follows by enumerating the cases. ◀

We now compute a linear size bi-tree from it.

4 Bi-labels

We now consider a directed graph D equipped with a *bi-label*, that is every vertex x receives a couple $(i(x), o(x))$ of positive integers. The *weight* of D is $(i(V), o(V))$, the sum of all i and o values respectively. Assume that D is a bi-labelled digraph which is the union of a digraph D' and an out-tree T^+ rooted at $r \in V(D')$ such that $D' \cap T^+ = \{r\}$. We can *transfer* the weight $o(T^+)$ to D' by adding $o(T^+ \setminus r)$ to $o(r)$ and removing all the vertices of $T^+ \setminus r$. We define similarly the transfer operation of i for an in-tree.

Given a bi-tree B of D , the *value* of B is the pair (a, b) where a is the sum of all $i(x)$ for x in B^- and b is the sum of all $o(x)$ for x in B^+ . In other words, the value of B is the label of the center after the transfers of B^+ and B^- . Observe that if D' is obtained from D by some transfer and D' has a bi-tree with value (a, b) , then D also has a bi-tree with value (a, b) .

► **Theorem 10.** *If C is a cycle equipped with a bilabel (i, o) of weight (w, w) , it contains a bi-tree with value at least $(w/2, w/2)$.*

Proof. Consider a shortest path P in $C = x_1, \dots, x_n$ such that $i(P) \geq w/2$ or $o(P) \geq w/2$. Assume without loss of generality that $P = C[x_1, \dots, x_k]$. First consider the case where we have $i(P) \geq w/2$. By minimality of P , we have $o(C[x_k, \dots, x_n]) \geq w/2$, and therefore the bi-tree B centered at x_k such that $B^+ = C[x_k, \dots, x_n]$ and $B^- = C[x_1, \dots, x_k]$ satisfies the hypothesis. In the case $o(P) \geq w/2$, we proceed similarly with x_1 as center. ◀

The bound in Theorem 10 is sharp. Consider for this a directed 4-cycle in which all labels are $(w/4, w/4)$: any bi-tree has value (a, b) with $\min\{a, b\} = w/2$.

► **Theorem 11.** *Every strong digraph $D = (V, A)$ on n vertices contains a bi-tree B such that both B^+ and B^- have size at least $n/6$.*

Proof. By Corollary 9, D has an (I, C, O) -decomposition such that both $I \cup C$ and $O \cup C$ have size at least $n/3$. Observe that D is spanned by a subgraph S consisting of the directed cycle spanning C , together with a disjoint collection of in-trees rooted at some vertices of C and with other vertices in I , and a collection of out-trees rooted at some vertices of C and with other vertices in O (see Figure 2 (Right) for an example). This comes from strong connectivity which implies that any node u in I has at least one out-neighbor v and v must be either in I or in C by the definition of the (I, C, O) -decomposition which forbids

any arc from I to O . Selecting arbitrarily one out-neighbor for each node in I results in a collection of in-trees rooted at vertices in C and spanning I . Similarly, selecting arbitrarily one in-neighbor for each node in O results in a collection of out-trees rooted at vertices in C and spanning O . We consider that S is a bi-labelled digraph by setting the value $(1, 1)$ to every vertex. We can then transfer the weight of all in-trees and out-trees to their respective roots to obtain a bi-label on C with weight at least $(n/3, n/3)$. We now invoke Theorem 10 to obtain a bi-tree with value at least $(n/6, n/6)$ for the cycle, and unfold it by reversing appropriate transfers to get a bi-tree of S with same value. ◀

As mentioned in the introduction, from the previous results we obtain the following.

► **Corollary 12.** *FCCP admits a $1/18$ -approximation in quadratic time.*

Note that the above constructions of left-maximal DFS and bi-tree can be extended to a weighted digraph D where each vertex u has a non-negative weight w_u . Given a subset $U \subseteq V$ of n' vertices, we can set $w_u = 1$ for $u \in U$ and $w_u = 0$ for $u \notin U$ to compute similarly a bi-tree B such that B^- and B^+ both span $n'/6$ vertices of U . One can then easily obtain an ordering of V such that a constant fraction of pairs in $U \times U$ are forward-connected.

We now consider the more general version of the problem where we want to connect pairs in a given set $R \subseteq \binom{V}{2}$ of requests.

5 Forward connecting a set of requested pairs

In the Request Forward Connected Pairs Problem (RFCPP), the input is a strongly connected digraph $D = (V, A)$ and a set $R \subseteq \binom{V}{2}$ of requests, and the output is a vertex ordering of D maximizing the number of forward pairs $\{x, y\} \in R$, that is unordered pairs $\{x, y\}$ such that either x, y or y, x is forward. Note that FCCP is the particular instance of RFCPP satisfying $R = \binom{V}{2}$. Since FCCP is in APX, it is natural to raise the following problem:

► **Problem 13.** *Is there a polytime constant approximation algorithm for RFCPP?*

In particular, is it always possible to satisfy a linear fraction of R ? In the more restricted variant where R is a set of couples (x, y) instead of pairs (where one wants to maximize the number of forward couples x, y), one cannot expect to satisfy a large proportion of R . Indeed, if D is the circuit (v_1, \dots, v_n) and R consists of all couples (v_{i+1}, v_i) , any vertex ordering can only satisfy at most one request of R , hence only a ratio of $1/n$ can be realized as forward couples.

Surprisingly, when requests are pairs, we could not find any set of request R which is not satisfied within a ratio of $1/\log n$. However, contrary to the case of FCCP where a positive ratio is achievable, the following result shows that there are instances of RFCPP for which only a logarithmic ratio can be realized as forward pairs.

► **Proposition 14.** *For all n , there exists an instance of RFCPP where D has $2^{n+1} - 1$ vertices, R has size $n2^{n-1}$, and no more than 2^n requested pairs can be realized as forward paths.*

Proof. The graph D is the complete binary tree of height n seen as a directed graph by letting each edge to be a circuit of length two. The set of requests is recursively defined in the following way. We consider the set L_l of all leaves which are descendants of the left child of the root r and the set L_r of leaves descendants of its right child. Now we pick an arbitrary perfect matching M between L_l and L_r and set as requests all the $|M|$ pairs formed by the edges of M . We call this set R_1 , and recursively define in the same way a set of requests

13:10 Temporalizing Digraphs via Linear-Size Balanced Bi-Trees

for each of the two children of r . We stop when reaching leaves. With the right choice of matchings, the set of requests can correspond for instance to a hypercube of dimension n on the leaves. We shall not need it, but it gives a good intuition of the possible structure of requests.

To sum up, D has $2^{n+1} - 1$ nodes, 2^n of them being leaves. Each internal node creates a matching of requests in its set of descendant leaves, hence the total number of requests is $n \cdot 2^{n-1}$.

Any ordering $<$ of $V(D)$ can be considered as the sub-digraph D' of D where we keep only arcs xy satisfying $x < y$. Note that this corresponds to an orientation of the edges of the tree. Our goal is to show that the number of requests x, y which are connected by a forward path is at most 2^n . To show this, for a given node x we denote by $rf(x)$ the number of requests between descendants of x (in the tree D) which are realized by a forward path. We also denote by $in(x)$ the number of leaves y descendant of x such that there is a forward path from y to x . Finally $out(x)$ is the number of leaves z descendant of x such that there is a forward path from x to z .

We now show by induction that both $rf(x) + in(x)$ and $rf(x) + out(x)$ are upper bounded by $\ell(x)$, the number of leaves descendant of x . This is true if x has two leaves as children since either x is a source (resp. a sink) in D' and $rf(x) + \max(in(x), out(x)) = 0 + 2$, or x has both in and out-degree 1 and $rf(x) + in(x) = rf(x) + out(x) = 1 + 1$. For the induction step, we assume that x has two children y, z .

- if we have both arcs xy, xz in D' , we have $rf(x) + in(x) \leq rf(x) + out(x) = rf(y) + rf(z) + out(y) + out(z)$, which by induction is at most $\ell(y) + \ell(z) = \ell(x)$.
- if we have both arcs yx, zx , we conclude similarly.
- if we have both arcs yx, xz , note that $rf(x) \leq rf(y) + rf(z) + \min(in(y), out(z))$. Also $in(x) = in(y)$ and $out(x) = out(z)$. Thus, if $in(y) \leq out(z)$ we have $rf(x) + in(x) \leq rf(x) + out(x) \leq rf(y) + rf(z) + \min(in(y), out(z)) + out(z) = rf(y) + in(y) + rf(z) + out(z) \leq \ell(y) + \ell(z) = \ell(x)$. And the same conclusion holds when $in(y) > out(z)$.
- if we have both arcs xy, zx , we conclude as previously.

Thus the maximum number of forward requests is 2^n . ◀

Noteworthy, for the instance in Proposition 14, there is a set of $2n$ orderings such that every pair x, y of leaves is forward connected in one of the orderings. Let us call a *forward cover* of D a set of vertex orderings \leq_1, \dots, \leq_k such that for every pair $\{x, y\}$, there is some i such that x, y or y, x is a forward pair in \leq_i . This suggests the following conjecture:

► **Conjecture 15.** *Every strong digraph D on n vertices has a $O(\log n)$ size forward cover.*

Conjecture 15 holds when D is a bi-oriented graph, that is an undirected graph considered as a digraph by replacing each edge $\{u, v\}$ by two arcs uv and vu . Here is a sketch: it suffices to consider a spanning tree T of D and to fix a centroid c . We then let $T = T_1 \cup T_2$ where $T_1 \cap T_2 = \{c\}$ and both T_1, T_2 have size at least $n/3$. We consider any vertex enumeration $T_1 \leq c \leq T_2$. Now we find recursively in T_1 and T_2 two families F_1 and F_2 of vertex enumerations of size logarithmic in $2n/3$. We conclude by gluing (on c) the orderings in F_1 and F_2 by pairs. We get in total $1 + \max(|F_1|, |F_2|)$ orders, hence a logarithmic size family.

Another intriguing question is the “forward orientation” of a pair x, y . We have seen that in some cases, like the circuit, pairs v_i, v_{i+1} are (obviously!) much more likely to be forward than pairs v_{i+1}, v_i . If indeed a forward cover of logarithmic size exists, it could be that some couples x, y are more involved in a small cover than their reverse y, x . This suggests a kind of “forwardness” of a pair of vertices which might be interesting to characterize.

6 Questions

Our main open question concerns Conjecture 15. A possible way to solve it would be to find $O(\log n)$ bi-trees such that for any pair $\{x, y\}$, there exists a bi-tree T such that either $x \in T^-$ and $y \in T^+$, or $y \in T^-$ and $x \in T^+$. The reason is that each bi-tree can easily be converted to an ordering where each couple (x, y) with $x \in T^-$ and $y \in T^+$ is a forward couple. Note that Conjecture 15 would imply that any instance of RFCPP always have a solution realizing a $1/O(\log n)$ fraction of requested pairs, opening the possibility for polynomial time $O(\log n)$ -approximation.

Another question concerns the maximum size of a balanced bi-tree that can be found in any strongly connected digraph with n vertices. Our construction shows that any such digraph contains a balanced bi-tree where both trees have size $n/6$. The construction given in [2] implies that there exists strongly connected digraphs such that in any bi-tree T , either T^- or T^+ has size at most $n/3 + O(1)$. This leaves a factor 2 gap.

If we relax the bi-tree definition by requiring that the in-branching and the out-branching are edge disjoint (an *in-out-branching*) and may overlap over more than one vertex (and still share the same root). What is the maximum size of a balanced in-out-branching in any strongly connected digraph? The upper-bound of $n/3 + O(1)$ given in [2] indeed holds for in-out-branchings. Is there a gap between the maximum size of a balanced in-out-branching and that of a balanced bi-tree?

More generally, what are the exact values of r_s and r_t ? In other words, what is the maximum ratio of couples that can be connected through an ordering of the arcs, or an ordering of the vertices respectively? Is it possible to obtain an interesting lower-bound of r_t as a function of r_s ?



References

- 1 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory Comput. Syst.*, 61(3):907–944, 2017. doi:10.1007/s00224-017-9757-x.
- 2 Alkida Balliu, Filippo Brunelli, Pierluigi Crescenzi, Dennis Olivetti, and Laurent Viennot. A note on the complexity of maximizing temporal reachability via edge temporalisation of directed graphs. *CoRR*, abs/2304.00817, 2023. URL: <https://arxiv.org/abs/2304.00817>.
- 3 Jørgen Bang-Jensen. Edge-disjoint in- and out-branchings in tournaments and related path problems. *J. Comb. Theory, Ser. B*, 51(1):1–23, 1991. doi:10.1016/0095-8956(91)90002-2.
- 4 Jørgen Bang-Jensen, Stéphane Bessy, Frédéric Havet, and Anders Yeo. Arc-disjoint in- and out-branchings in digraphs of independence number at most 2. *J. Graph Theory*, 100(2):294–314, 2022. doi:10.1002/jgt.22779.
- 5 Jørgen Bang-Jensen, Nathann Cohen, and Frédéric Havet. Finding good 2-partitions of digraphs II. enumerable properties. *Theor. Comput. Sci.*, 640:1–19, 2016. doi:10.1016/j.tcs.2016.05.034.
- 6 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs - Theory, Algorithms and Applications, Second Edition*. Springer Monographs in Mathematics. Springer, 2009.
- 7 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 365–376. ACM, 2019. doi:10.1145/3313276.3316335.
- 8 Filippo Brunelli, Pierluigi Crescenzi, and Laurent Viennot. Maximizing reachability in a temporal graph obtained by assigning starting times to a collection of walks. *Networks*, 81(2):177–203, 2023. doi:10.1002/net.22123.

13:12 Temporalizing Digraphs via Linear-Size Balanced Bi-Trees

- 9 Jessica A. Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *J. Comput. Syst. Sci.*, 119:60–77, 2021. doi:10.1016/j.jcss.2021.01.007.
- 10 F. Göbel, J. Orestes Cerdeira, and Henk Jan Veldman. Label-connected graphs and the gossip problem. *Discret. Math.*, 87(1):29–40, 1991. doi:10.1016/0012-365X(91)90068-D.
- 11 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPICs*, pages 62:1–62:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.MFCS.2022.62.
- 12 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019. doi:10.1007/s00453-018-0478-6.
- 13 Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal reachability minimization: Delaying vs. deleting. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPICs*, pages 76:1–76:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.MFCS.2021.76.

A Subquadratic Bound for Online Bisection

Marcin Bienkowski  

University of Wrocław, Poland

Stefan Schmid  

TU Berlin, Germany

Weizenbaum Institute, Berlin, Germany

Abstract

The *online bisection problem* is a natural dynamic variant of the classic optimization problem, where one has to dynamically maintain a partition of n elements into two clusters of cardinality $n/2$. During runtime, an online algorithm is given a sequence of requests, each being a pair of elements: an inter-cluster request costs one unit while an intra-cluster one is free. The algorithm may change the partition, paying a unit cost for each element that changes its cluster.

This natural problem admits a simple deterministic $O(n^2)$ -competitive algorithm [Avin et al., DISC 2016]. While several significant improvements over this result have been obtained since the original work, all of them either limit the generality of the input or assume some form of resource augmentation (e.g., larger clusters). Moreover, the algorithm of Avin et al. achieves the best known competitive ratio even if randomization is allowed.

In this paper, we present the first randomized online algorithm that breaks this natural quadratic barrier and achieves a competitive ratio of $\tilde{O}(n^{23/12})$ without resource augmentation and for an arbitrary sequence of requests.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Bisection, Graph Partitioning, online balanced Repartitioning, online Algorithms, competitive Analysis

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.14

Funding Supported by Polish National Science Centre grant 2022/45/B/ST6/00559 and by the by the European Research Council (ERC), consolidator grant 864228 (AdjustNet).

1 Introduction

The clustering of elements into subsets that are related by some similarity measure is a fundamental algorithmic problem. The problem arises in multiple contexts: a well-known abstraction is *the bisection problem* [12] that asks for partitioning of n graph nodes (elements) into two clusters of size $n/2$, so that the number of graph edges in the cut is minimized. This problem is NP-hard and its approximation ratio has been improved in a long line of papers [20, 1, 8, 7, 13]; the currently best approximation ratio of $O(\log n)$ was given by Räcke [17].

Recently this problem has been studied in a *dynamic variant* [3, 18], where instead of a fixed graph, we are given a sequence of element pairs. Serving a pair of elements that are in different clusters costs one unit, while a request between two elements in the same cluster is free. After serving a request, an algorithm may modify the partition, paying a unit cost for each element that changes its cluster.

A natural motivation for this problem originates from data centers where communicating virtual machines (elements) have to be partitioned between servers (clusters) and the overall communication cost has to be minimized: by collocating virtual machines on the same server, their communication becomes free, while the communication between virtual machines in different clusters involves using network bandwidth. Modern virtualization technology



© Marcin Bienkowski and Stefan Schmid;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 14; pp. 14:1–14:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



supports the seamless migration of virtual machines between servers, but migrations still come at the cost of data transmission. The goal is to minimize data transmission (across the network), comprising inter-cluster communication and migration.

This practical application motivates yet another aspect of the problem: the communication pattern (and, in particular, the sequence of communication pairs) is typically not known ahead of time. Accordingly, we study the dynamic variant in the *online setting*, where the sequence of communication pairs is not known a priori to an online algorithm ALG: ALG has to react immediately without the knowledge of future communication requests. To evaluate its performance, we use a standard notion of *competitive ratio* [6] that compares the cost of ALG to the cost of the optimal (offline) solution OPT: the ALG-to-OPT cost ratio is subject to minimization.

Previous Results. Avin et al. introduced the online bisection problem and presented a simple deterministic online algorithm that achieves the competitive ratio of $O(n^2)$ [3]. Their algorithm belongs to a class of component-preserving algorithms (formally defined in Section 2). Roughly speaking, it splits the request sequence into epochs. Within a single epoch, it glues requested element pairs together creating components and assigns all elements of any component to the same cluster. If such an assignment is no longer feasible, i.e., components cannot be preserved (kept on the same cluster), an epoch terminates. It is easy to argue (cf. Section 2) that OPT pays at least 1 in an epoch, and any component-preserving algorithm pays at most n^2 , thus being n^2 -competitive.

Perhaps surprisingly, no better algorithm (even a randomized one) is known for the online bisection problem. On the negative side, a lower bound of $\Omega(n)$ [2] for deterministic algorithms follows by the reduction from online paging [21].

Our Contribution. We present the first algorithm for the online bisection problem that beats the quadratic competitive ratio. All previous results with better ratios required some relaxation: either used resource augmentation or restricted the generality of the input sequence. Our IMPROVED COMPONENT BASED algorithm (ICB) is randomized, follows the component-preserving framework outlined above, and achieves the competitive ratio of $O(n^{23/12} \cdot \sqrt{\log n})$.

Our Algorithmic Ideas. Assume that an algorithm follows the component-preserving framework and we want to improve its cost within a single epoch. We may look at the problem more abstractly: there is a set of “allowed” partitions (the ones that map elements to clusters in a component-preserving way), and this set is constantly shrinking. Consider an algorithm that, whenever it needs to change its partition, changes it to one chosen uniformly at random from the set of still allowed partitions. Using standard arguments, we may argue that the algorithm changes its partition at most $O(\log y)$ times within an epoch, where y is the number of “allowed” partitions at the beginning. As the cost of serving the request and the cost of changing the partition is at most $1 + n$, the overall cost of such routine is $O(n \cdot \log y)$.

At the beginning of an epoch $y = 2 \cdot \binom{n}{n/2}$, and thus $O(n \cdot \log y) = O(n^2)$. That is, the randomized routine itself would fail to beat the quadratic upper bound of [3] if it is applied to the entire epoch. However, we may execute it in the second stage of an epoch, once the number of “allowed” partitions drops appropriately.

In the first stage of an epoch, our proposed algorithm ICB carefully tracks the component sizes. In a single step, it needs to merge two components into a single one and to map all elements of the resulting component to the same cluster. To this end, it usually has to move one of the merged components to the other cluster. A crucial insight is that, most of the time, the moved component size can be expressed as a linear combination of a moderate number of existing component sizes: in such case, only limited number of existing components have to change their clusters.

Converting this intuition into an actual algorithm is not easy. To this end, we provide a way of maintaining the greatest common divisor (GCD) of a large subset of components, so that this GCD changes only a few times within an epoch. We use number-theoretic properties to argue that whenever ICB merges two components into a single one, then usually one of them is divisible by the current value of GCD, and thus the resulting repartitioning incurs the movement of only a moderate number of other components.

The low-cost argument depends, however, on the property that not only there are many components of sizes divisible by GCD, but also both clusters contain sufficiently many of them. ICB ensures this property by regularly running a “rebalancing” routine. At some point, maintaining this property is no longer possible. We prove that such failure guarantees that the total number of “allowed” partitions is appropriately low: ICB switches then to the second stage of an epoch, where it executes the randomized policy outlined above.

Related Work. The lack of progress toward improving the $O(n^2)$ upper bound motivated the investigation of simplified variants.

A natural relaxation involves resource augmentation, where each cluster of an online algorithm can accommodate $(1 + \varepsilon) \cdot (n/2)$ elements. The performance of an online algorithm is compared to OPT whose both clusters still have capacity $n/2$. Surprisingly, the competitive ratio remains $\Omega(n)$ even for large ε (but as long as $\varepsilon < 1$) [2]. On the positive side, Rajaraman and Wasim showed an $O(n \log n)$ -competitive deterministic algorithm for a fixed $\varepsilon > 0$ [19].

Another relaxation was introduced by Henzinger et al. [11] who initiated the study of the so-called *learning variant*. In this variant, there exists a fixed partition \bar{p} (unknown to an algorithm), and all requests are consistent with \bar{p} (i.e., given between same-cluster pairs). Clearly, the optimal solution simply changes its partition to \bar{p} at the very beginning. The deterministic variant is asymptotically resolved: the optimal competitive ratio is $\Theta(n)$ [15, 16]. For the model where the learning variant is combined with resource augmentation, Henzinger et al. gave a $\Theta(\log n)$ -competitive deterministic solution (for any fixed $\varepsilon > 0$) [10].

The online bisection problem has also been studied in a generalized form, where there are $\ell > 2$ clusters, each of size n/ℓ . This extension is usually referred to as *online balanced graph partitioning*. Some of the results presented above can be generalized to this variant [2, 3, 10, 11, 15, 16, 19]. This generalization was investigated also in models with a large augmentation of $\varepsilon > 1$ [2, 9, 10, 18] and in settings with small (or even constant-size) clusters [2, 4, 16].

2 Preliminaries

We have a set V of n elements and two clusters 0 and 1. A valid partition of these elements is a mapping $p : V \rightarrow \{0, 1\}$ such that $|p^{-1}(0)| = |p^{-1}(1)| = n/2$, i.e., each cluster contains exactly $n/2$ elements. For two partitions p and p' , we use $\text{dist}(p, p') = |\{v \in V : p(v) \neq p'(v)\}|$ to denote the number of elements that change their clusters when switching from partition p to p' .

Problem Definition. An input for the online bisection problem consists of an initial partition p_0 and a sequence of element pairs $((u_t, v_t))_{t \geq 1}$. In step $t \geq 1$, an online algorithm ALG is given a pair of elements (u_t, v_t) : it pays a *service cost* of 1 if $p_{t-1}(u_t) \neq p_{t-1}(v_t)$ and 0 otherwise. Afterward, ALG has to compute a new partition p_t (possibly $p_t = p_{t-1}$) and pay $\text{dist}(p_{t-1}, p_t)$ for changing partition p_{t-1} to partition p_t .

For an input \mathcal{I} and an online algorithm ALG, we use $\text{ALG}(\mathcal{I})$ to denote its total cost on \mathcal{I} , whereas $\text{OPT}(\mathcal{I})$ denotes the optimal cost of an offline solution. ALG is γ -*competitive* if there exists β , such that $\text{ALG}(\mathcal{I}) \leq \gamma \cdot \text{OPT}(\mathcal{I}) + \beta$ for any input \mathcal{I} . While β has to be independent of \mathcal{I} , it may be a function of n . For a randomized algorithm ALG, we replace $\text{ALG}(\mathcal{I})$ with its expectation $\mathbf{E}[\text{ALG}(\mathcal{I})]$, taken over all random choices of ALG.

Component-Preserving Framework. A natural way of tackling the problem is to split requests into epochs. In a single epoch, an online algorithm ALG treats requests as edges connecting requested element pairs. Edges in a single epoch induce connected components of elements. A *component-preserving* algorithm always keeps elements of each component in the same cluster. If it is no longer possible, the current epoch ends, all edges are removed (each element is now in its own singleton component), and a new epoch begins with the next step. We note that the currently best $O(n^2)$ -competitive deterministic algorithm of [3] is component-preserving.

Now, we recast the online bisection problem assuming that we analyze a component-preserving algorithm ALG. First, observe that ALG completely ignores all intra-component requests (they also incur no cost on ALG). Consequently, we may assume that the input for ALG (within a single epoch) is a sequence of component sets \mathcal{C}_t , where:

- \mathcal{C}_0 is the initial set of n singleton components;
- \mathcal{C}_t (presented in step $t \geq 1$) is created from \mathcal{C}_{t-1} by merging two of its components, denoted c_x^t and c_y^t . They are merged into a component, denoted c_z^t , i.e.

$$\mathcal{C}_t = \mathcal{C}_{t-1} \cup \{c_z^t\} \setminus \{c_x^t, c_y^t\}.$$

For a given set of components \mathcal{C} , let $\mathcal{P}(\mathcal{C})$ denote the set of all \mathcal{C} -preserving partitions of n elements into two clusters, i.e., ones that place all elements of a single component of \mathcal{C} in the same cluster. In response to \mathcal{C}_t , ALG chooses a \mathcal{C}_t -preserving partition p_t . If $\mathcal{P}(\mathcal{C}_t)$ is empty though, then ALG does not change its partition, and an epoch terminates. The following observations let us focus on ALG's behavior in a single epoch only.

► **Lemma 1.** *The epoch of any component-preserving algorithm contains at most $n - 1$ steps, and the single-step cost is at most $n + 1$.*

Proof. In each step, the number of components decreases. Thus, after $n - 1$ steps, all elements would be in the same component, and hence $\mathcal{P}(\mathcal{C}_{n-1}) = \emptyset$. In a single step, an algorithm pays at most 1 for serving the request and changes the cluster of at most n elements. ◀

► **Lemma 2.** *If a component-preserving algorithm ALG pays at most R in any epoch, then ALG is R -competitive.*

Proof. Fix any finished epoch \mathcal{E} in an input (any epoch except possibly the last one). The final step of \mathcal{E} serves as a certificate that any algorithm keeping a static partition throughout \mathcal{E} has a non-zero cost. On the other hand, changing partition costs at least 1, and thus $\text{OPT}(\mathcal{E}) \geq 1$.

The lemma follows by summing costs over all epochs except the last one. Observe that the cost of the last epoch is at most $n^2 - 1$ (by Lemma 1), and thus can be placed in the additive term β in the definition of the competitive ratio (cf. Section 2). ◀

Note that by Lemma 1 and Lemma 2 the competitive ratio of any component-preserving algorithm (including that of [3]) is at most $(n - 1) \cdot (n + 1) < n^2$.

Notation. For an integer ℓ , we define $[\ell] = \{1, 2, \dots, \ell\}$. For any finite set $A \subset \mathbb{N}_{>0}$, we use $\gcd(A)$ to denote the greatest common divisor of all integers from A ; we assume that $\gcd(\emptyset) = \infty$.

For any component c , we denote its size (number of elements) by $\text{SIZE}(c)$. Fix any component set \mathcal{C} and an integer i . Let $\text{CNT}_i(\mathcal{C}) \triangleq |\{c \in \mathcal{C} : \text{SIZE}(c) = i\}|$ denote the number of components in \mathcal{C} of size i .

Furthermore, fix a partition $p \in \mathcal{P}(\mathcal{C})$ and a cluster $y \in \{0, 1\}$. As p is \mathcal{C} -preserving, it is constant on all elements of a given component $c \in \mathcal{C}$, and thus we may extend p to components from \mathcal{C} . We define

$$\text{CNT}_i(\mathcal{C}, p, y) \triangleq |\{c \in \mathcal{C} : p(c) = y \wedge \text{SIZE}(c) = i\}|$$

as the number of components in \mathcal{C} of size i that are inside cluster y in partition p .

We extend both notions to sets of sizes, i.e., for any set A , we set $\text{CNT}_A(\mathcal{C}) \triangleq \sum_{i \in A} \text{CNT}_i(\mathcal{C})$ and $\text{CNT}_A(\mathcal{C}, p, y) \triangleq \sum_{i \in A} \text{CNT}_i(\mathcal{C}, p, y)$.

3 A Subquadratic Algorithm

Our IMPROVED COMPONENT BASED algorithm (ICB) is component-preserving. It splits an epoch into two stages. The first stage is deterministic: with a slight “rebalancing” exception that we explain later, the components are remapped to minimize the cost of changing the partition in a single step. At a carefully chosen step that we define later, ICB switches to the second stage. In any step t of the second stage, if the current partition p_{t-1} is not \mathcal{C}_t -preserving, ICB chooses p_t uniformly at random from $\mathcal{P}(\mathcal{C}_t)$.

We now focus on describing the first stage of an epoch. Our algorithm ICB uses a few integer parameters defined below.

- Parameter $q \in [n]$. A component size is *large* if it is greater than q and is *small* otherwise.
- Parameter $w \in [n]$. If $\text{CNT}_i(\mathcal{C}) \geq w$, we call size i *popular* (in \mathcal{C}).
- Parameter $d \in [n]$.

ICB works with any values of q, w, d , as long as they satisfy $6 \cdot q^4 + 3 \leq w$, $q \cdot (2 \cdot w + 1) \leq d$, and $2 \cdot d \leq n$. The parameter values yielding the competitive ratio of $O(n^{23/12} \cdot \sqrt{\log n})$ are chosen in Theorem 11.

3.1 Helper Notions

First, for any $k \in \mathbb{N}_{>0} \cup \{\infty\}$ we define

$$\langle k \rangle \triangleq \{\ell \cdot k : \ell \in \mathbb{N}\} \cap [q].$$

In particular, $\langle \infty \rangle = \emptyset$. That is, $\langle k \rangle$ contains all small component sizes that are divisible by k . Observe that

$$k = \gcd(\langle k \rangle) \quad \text{for any } k \in [q] \cup \{\infty\}. \tag{1}$$

Second, we introduce the notion of a *balanced partition*. Fix a set of components \mathcal{C} , a value k , and an integer ℓ . For a given partition $p \in \mathcal{P}(\mathcal{C})$, we say that p is (k, ℓ) -balanced if

$$\text{CNT}_{\langle k \rangle}(\mathcal{C}, p, y) \geq \ell \quad \text{for } y \in \{0, 1\}.$$

■ **Algorithm 1** The first stage of an epoch of ICB.

Input: initial partition p_0 , sequence of component sets $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_t, \dots$

Output: sequence of partitions $p_1, p_2, \dots, p_t, \dots$, where p_t is \mathcal{C}_t -preserving.

Initialization: $g_0 \leftarrow 1$.

Processing \mathcal{C}_t (step $t \geq 1$)

```

1: if  $\mathcal{P}(\mathcal{C}_t) = \emptyset$  then                                     ▷ no  $\mathcal{C}_t$ -preserving partition
2:    $p_t \leftarrow p_{t-1}$ 
3:   terminate the current epoch
4:  $B_t = \{i \in [q] : \text{CNT}_i(\mathcal{C}_t) \geq w\}$                    ▷  $B_t$  contains small popular sizes
5:  $g_t \leftarrow \text{gcd}(\langle g_{t-1} \rangle \cap B_t)$ 
6: if  $\mathcal{P}(\mathcal{C}_t, g_t, 2d) = \emptyset$  then                       ▷ no  $\mathcal{C}_t$ -preserving  $(g_t, 2d)$ -balanced partition
7:    $p_t \leftarrow p_{t-1}$ 
8:   terminate the first stage of the current epoch
9:  $p_t^* \leftarrow \arg \min_p \{\text{dist}(p_{t-1}, p) : p \in \mathcal{P}(\mathcal{C}_t)\}$    ▷ pick closest  $\mathcal{C}_t$ -preserving candidate
10: if  $p_t^* \in \mathcal{P}(\mathcal{C}_t, g_t, d)$  then                          ▷ rebalance if necessary
11:    $p_t \leftarrow p_t^*$ 
12: else
13:    $p_t \leftarrow$  any partition from  $\mathcal{P}(\mathcal{C}_t, g_t, 2d)$ 

```

That is, a (k, ℓ) -balanced partition p of \mathcal{C} keeps at least ℓ small components of sizes divisible by k in each cluster. We use $\mathcal{P}(\mathcal{C}, k, \ell) \subseteq \mathcal{P}(\mathcal{C})$ to denote the set of all (k, ℓ) -balanced \mathcal{C} -preserving partitions.

3.2 Definition of the First Stage

The pseudo-code of ICB for the first stage is given in Algorithm 1; we describe it also below.

Computing GCD Estimator. Initially, in step t , in Lines 1–3, ICB verifies whether a \mathcal{C}_t -preserving partition exists, and terminates the epoch without changing the current partition otherwise.

Next, ICB sets B_t to be the set of small popular component sizes of \mathcal{C}_t and computes the value of *GCD estimator* $g_t \in [q] \cup \{\infty\}$. The computation balances two objectives: on one hand, we want g_t to be the greatest common divisor of B_t , on the other hand, we do not want g_t to change too often. Therefore, g_t is defined by the following iterative process (cf. Lines 4–5): We initialize $g_0 = 1$ (i.e., $\langle g_0 \rangle = [q]$). In step t , we set $g_t = \text{gcd}(\langle g_{t-1} \rangle \cap B_t)$. Note that this process ensures that $g_t \in [q] \cup \{\infty\}$ for any t .

Triggering the Second Stage. Lines 6–8 ensure that there exists a $(g_t, 2d)$ -balanced partition of \mathcal{C}_t . If this is not the case, ICB terminates the first stage without changing its partition and switches to the second stage of an epoch in the next step.

Choosing a New Partition. Finally, in Lines 9–13, ICB chooses its new partition p_t . First, it computes a *candidate partition* p_t^* as a \mathcal{C}_t -preserving partition closest to p_{t-1} . If p_t^* is (g_t, d) -balanced, it simply outputs $p_t = p_t^*$. Otherwise, it discards p_t^* , and picks any $(g_t, 2d)$ -balanced partition as p_t . We call such action *rebalancing*; we later show that it occurs rarely, i.e., in most cases $p_t = p_t^*$.

4 Analysis Roadmap

In this section, we describe the framework of our analysis in a top-down approach, listing the necessary lemmas that will be proven in the next sections, and showing that their combination yields the desired competitiveness bound.

Lemma 2 allows us to focus only on the cost of ICB in a single epoch \mathcal{E} . We denote its two stages by \mathcal{E}_1 and \mathcal{E}_2 ; the second stage may be empty if ICB terminates the first stage already in Lines 1–3. We identify \mathcal{E}_1 and \mathcal{E}_2 with the sets of the corresponding steps. In particular, we use T as the number of steps in the first stage, i.e., $\mathcal{E}_1 = [T]$.

The Second Stage. We start from a simpler case, the cost analysis in \mathcal{E}_2 . The lemmas stated below are proven in Section 6. We assume that \mathcal{E}_2 is non-empty as otherwise the associated cost is trivially zero. That is, ICB switches to the second stage because the condition in Line 6 becomes true, i.e., $\mathcal{P}(\mathcal{C}_T, g_T, 2d) = \emptyset$.

By observing that in the second stage, ICB is essentially a randomized algorithm solving the metrical tasks system (MTS) problem [5] on a uniform metric of $|\mathcal{P}(\mathcal{C}_T)|$ points, we obtain the following bound.

► **Lemma 3.** $\mathbf{E}[ICB(\mathcal{E}_2)] = O(n \cdot \log |\mathcal{P}(\mathcal{C}_T)|)$.

The usefulness of the lemma above depends on how well we can bound $|\mathcal{P}(\mathcal{C}_T)|$, the number of \mathcal{C}_T -preserving partitions. The second stage is executed only when $\mathcal{P}(\mathcal{C}_T, g_T, 2d) = \emptyset$, i.e., at step T , all \mathcal{C}_T -preserving partitions have less than $2d$ components of sizes from $\langle g_T \rangle$ in one of the clusters. This, together with combinatorial counting arguments, implies the following bound.

► **Lemma 4.** $|\mathcal{P}(\mathcal{C}_T)| = \exp(O(d \cdot \log n + z))$, where $z = \text{CNT}_{[n] \setminus \langle g_T \rangle}(\mathcal{C}_T)$.

The term $\text{CNT}_{[n] \setminus \langle g_T \rangle}(\mathcal{C}_T)$ denotes the number of components of sizes outside set $\langle g_T \rangle$ and we will bound it later using the behavior of ICB in \mathcal{E}_1 .

The First Stage: Rebalancing Costs. Now we switch our attention to the core of our approach, the first stage of an epoch. Recall that in a single step t , ICB pays at most 1 for serving the request and $\text{dist}(p_{t-1}, p_t)$ for changing the partition. We may upper-bound the latter term by $\text{dist}(p_{t-1}, p_t^*) + \text{dist}(p_t^*, p_t)$; we call the corresponding summands *switching cost* and *rebalancing cost*. It turns out that the latter part can be upper-bounded using the former. We define

$$\begin{aligned} \text{ICB}^{\text{ss}}(t) &\triangleq 1 + \text{dist}(p_{t-1}, p_t^*), \\ \text{ICB}^{\text{rb}}(t) &\triangleq \text{dist}(p_t^*, p_t). \end{aligned}$$

Clearly, $\text{ICB}(t) \leq \text{ICB}^{\text{ss}}(t) + \text{ICB}^{\text{rb}}(t)$.

► **Lemma 5.** *It holds that $\text{ICB}^{\text{rb}}(\mathcal{E}_1) \leq O(n \cdot \log q) + O(n/d) \cdot \text{ICB}^{\text{ss}}(\mathcal{E}_1)$.*

The rough idea behind the lemma above (proved formally in Subsection 5.2) is that the rebalancing cost is at most n and between two consecutive rebalancing actions, ICB pays already $\Omega(d)$ of switching cost. This statement is not always true, but it fails at most for $O(\log q)$ consecutive rebalancing actions.

The First Stage: Serving and Switching Costs. By the argument above, it now suffices to estimate $\text{ICB}^{\text{ss}}(\mathcal{E}_1)$. In Subsection 5.1 we study the evolution of g_t as a function of time step t . Recall that $g_0 = 1$ and $g_t \in [q] \cup \{\infty\}$ for any t . We say that a step t is *g-updating* if $g_t \neq g_{t-1}$.

► **Lemma 6.** *The number of g-updating steps within \mathcal{E}_1 is at most $1 + \log q$. Furthermore, $\langle g_t \rangle \subseteq \langle g_{t-1} \rangle$ for each step t of \mathcal{E}_1 .*

Recall that c_x^t and c_y^t are the components merged in step t . To bound the switching cost, we distinguish between regular and irregular steps. In regular ones, at least one merged component is small and its size is divisible by g_{t-1} .

► **Definition 7.** *A step t is regular if $\text{SIZE}(c_x^t) \in \langle g_{t-1} \rangle$ or $\text{SIZE}(c_y^t) \in \langle g_{t-1} \rangle$ and irregular otherwise.*

In Subsection 5.3, we argue that the switching and serving cost in regular steps is $o(n)$. To this end, we observe that Lines 10–13 executed in step $t - 1$ ensure (by running rebalancing if necessary) that p_{t-1} is (g_{t-1}, d) -balanced partition from $\mathcal{P}(C_{t-1})$, i.e., both clusters of partition p_{t-1} contain at least d small components of sizes divisible by g_{t-1} . This property becomes useful at the beginning of step t : using number-theoretic arguments, we may bound the number of components that need to be moved between clusters, so that eventually c_x^t and c_y^t end up in the same cluster.

► **Lemma 8.** *For any regular and not g-updating step t of \mathcal{E}_1 , $\text{ICB}^{\text{ss}}(t) = O(q^4)$.*

Finally, in Subsection 5.4, we argue that there are $o(n)$ irregular steps and we also bound the number of components whose sizes are either large or not divisible by g_t .

► **Lemma 9.** *There are at most $O(q \cdot w + n/q)$ irregular steps in \mathcal{E}_1 . Moreover, at any time t of \mathcal{E}_1 , $\text{CNT}_{[n] \setminus \langle g_t \rangle}(C_t) = O(q \cdot w + n/q)$.*

Estimating the Total Cost. We may now combine the bounds presented above to prove the desired competitive ratio.

► **Lemma 10.** *For any epoch \mathcal{E} , $\mathbf{E}[\text{ICB}(\mathcal{E})] = O((n^2/d) \cdot (q^4 + q \cdot w + n/q) + n \cdot d \cdot \log n)$.*

Proof. We split epoch \mathcal{E} into two stages, \mathcal{E}_1 and \mathcal{E}_2 , and let $T = |\mathcal{E}_1|$.

We first upper-bound the cost within \mathcal{E}_1 . Let $R \subseteq [T]$ be the set of regular steps of \mathcal{E}_1 that are not *g-updating*. Then each step from $[T] \setminus R$ is either irregular or *g-updating*. By Lemma 6 and Lemma 9,

$$|[T] \setminus R| \leq (1 + \log q) + O(q \cdot w + n/q) = O(q \cdot w + n/q). \quad (2)$$

This allows us to upper-bound the serving and switching cost of ICB in \mathcal{E}_1 as

$$\begin{aligned} \text{ICB}^{\text{ss}}(\mathcal{E}_1) &= \sum_{t \in R} \text{ICB}^{\text{ss}}(t) + \sum_{t \in [T] \setminus R} \text{ICB}^{\text{ss}}(t) \\ &\leq \sum_{t \in R} O(q^4) + \sum_{t \in [T] \setminus R} (n + 1) && \text{(by Lemma 8 and Lemma 1)} \\ &= |R| \cdot O(q^4) + O(q \cdot w + n/q) \cdot (n + 1) && \text{(by (2))} \\ &= O(n \cdot (q^4 + q \cdot w + n/q)). && \text{(as } R \leq T \leq n - 1) \end{aligned}$$

The total cost in \mathcal{E}_1 (including rebalancing) is then

$$\begin{aligned} \text{ICB}(\mathcal{E}_1) &= \text{ICB}^{\text{ss}}(\mathcal{E}_1) + \text{ICB}^{\text{rb}}(\mathcal{E}_1) \\ &= \text{ICB}^{\text{ss}}(\mathcal{E}_1) + O(n \cdot \log q) + O(n/d) \cdot \text{ICB}^{\text{ss}}(\mathcal{E}_1) && \text{(by Lemma 5)} \\ &= O(n^2/d) \cdot (q^4 + q \cdot w + n/q). \end{aligned}$$

The total expected cost in \mathcal{E}_2 (assuming \mathcal{E}_2 is present) is

$$\begin{aligned} \mathbf{E}[\text{ICB}(\mathcal{E}_2)] &= O(n \cdot \log |\mathcal{P}(\mathcal{C}_T)|) && \text{(by Lemma 3)} \\ &= n \cdot O(d \cdot \log n + \text{CNT}_{[n] \setminus \langle g_T \rangle}(\mathcal{C}_T)) && \text{(by Lemma 4)} \\ &= O(n \cdot (d \cdot \log n + q \cdot w + n/q)). && \text{(by Lemma 9)} \end{aligned}$$

Summing up, the total expected cost in the whole epoch is

$$\begin{aligned} \mathbf{E}[\text{ICB}(\mathcal{E})] &= \text{ICB}(\mathcal{E}_1) + \mathbf{E}[\text{ICB}(\mathcal{E}_2)] \\ &= O((n^2/d) \cdot (q^4 + q \cdot w + n/q) + n \cdot d \cdot \log n). && \text{(as } d \leq n) \quad \blacktriangleleft \end{aligned}$$

► **Theorem 11.** *ICB is $O(n^{23/12} \cdot \sqrt{\log n})$ -competitive for the online bisection problem.*

Proof. We set $q = \lceil n^{1/6} \rceil$, $w = 6 \cdot q^4 + 3$, and $d = \lceil n^{11/12} / \sqrt{\log n} \rceil$. Note that these values satisfy $q \cdot (2 \cdot w + 1) \leq d$ and $2 \cdot d \leq n$ for sufficiently large n . Applying Lemma 10, we obtain,

$$\mathbf{E}[\text{ICB}(\mathcal{E})] = O\left((n^2/d) \cdot n^{5/6} + n \cdot d \cdot \log n\right) = O\left(n^{23/12} \cdot \sqrt{\log n}\right).$$

The theorem follows immediately by Lemma 2. ◀

5 Analysis: the First Stage of ICB

5.1 Structural Properties

For succinctness of arguments, we extend the notion of divisibility. Recall that $a \mid b$ means that b is divisible by a and is well defined for any two positive integers a and b . We extend it also to the cases where a and b are possibly infinite: $a \mid \infty$ for any $a \in \mathbb{N}_{>0} \cup \{\infty\}$ and $\infty \nmid b$ for any $b \in \mathbb{N}_{>0}$.

▷ **Claim 12.** For any sets of integers A and B , it holds that $\text{gcd}(A) \mid \text{gcd}(A \cap B)$.

Proof. The claim follows trivially if $A \cap B = \emptyset$ as in such case $\text{gcd}(A \cap B) = \infty$. Thus, we may assume that $A \cap B \neq \emptyset$ (and hence also $A \neq \emptyset$). Fix any $i \in A \cap B$: as $i \in A$, we have $\text{gcd}(A) \mid i$. Hence, $\text{gcd}(A)$ is a divisor of all numbers from $A \cap B$, and therefore $\text{gcd}(A) \mid \text{gcd}(A \cap B)$. ◁

We now show that not only is g_t monotonically non-increasing, but when it grows in a g -updating step, the new value is a multiplicity of the old one.

► **Lemma 6.** *The number of g -updating steps within \mathcal{E}_1 is at most $1 + \log q$. Furthermore, $\langle g_t \rangle \subseteq \langle g_{t-1} \rangle$ for each step t of \mathcal{E}_1 .*

Proof. Fix any step t of \mathcal{E}_1 . By Claim 12, $\text{gcd}(\langle g_{t-1} \rangle) \mid \text{gcd}(\langle g_{t-1} \rangle \cap B_t)$. Note, however, that $\text{gcd}(\langle g_{t-1} \rangle) = g_{t-1}$ by (1), and $\text{gcd}(\langle g_{t-1} \rangle \cap B_t) = g_t$ by the definition of g_t (cf. Line 4 of the algorithm). Thus, $g_{t-1} \mid g_t$ for any step t .

14:10 A Subquadratic Bound for Online Bisection

If $g_t < \infty$, then $g_{t-1} < \infty$, and consequently $\langle g_t \rangle \subseteq \langle g_{t-1} \rangle$ follows by the definition of $\langle \cdot \rangle$. Otherwise $g_t = \infty$, in which case $\langle g_t \rangle = \emptyset \subseteq \langle g_{t-1} \rangle$. Thus, the second part of the lemma follows.

For the first part, let ℓ be the number of all g -updating steps within \mathcal{E}_1 ; we denote them by $\tau(1), \tau(2), \dots, \tau(\ell)$. Let $\tau(0) = 0$. For any $i \in \{0, \dots, \ell - 1\}$, it holds that $g_{\tau(i)} \mid g_{\tau(i+1)}$ and $g_{\tau(i)} \neq g_{\tau(i+1)}$, which implies $g_{\tau(i+1)} \geq 2 \cdot g_{\tau(i)}$. While it is possible that $g_{\tau(\ell)} = \infty$, we have $g_{\tau(\ell-1)} < \infty$, and thus $g_{\tau(\ell-1)} \leq q$. Hence, $g_{\tau(\ell-1)} \geq 2^{\ell-1} \cdot g_{\tau(0)} = 2^{\ell-1}$, and therefore $2^{\ell-1} \leq q$, which concludes the first part of the lemma. \blacktriangleleft

5.2 Rebalancing Cost

We first argue that between two consecutive rebalancing events ICB accrues sufficiently large serving and switching costs.

\triangleright **Claim 13.** Let a and b be two consecutive steps where rebalancing is executed. If $g_a = g_b$, then $\sum_{t=a+1}^b \text{ICB}^{\text{ss}}(t) \geq d/3$.

Proof. For any step $t \in \{a+1, \dots, b-1\}$, there is no rebalancing in t , and thus $p_t = p_t^*$.

$$\begin{aligned} \sum_{t=a+1}^b \text{ICB}^{\text{ss}}(t) &= \sum_{t=a+1}^b (1 + \text{dist}(p_{t-1}, p_t^*)) \\ &= (b-a) + \left(\sum_{t=a+1}^{b-1} \text{dist}(p_{t-1}, p_t^*) \right) + \text{dist}(p_{b-1}, p_b^*) \\ &= (b-a) + \sum_{t=a+1}^{b-1} \text{dist}(p_{t-1}, p_t) + \text{dist}(p_{b-1}, p_b^*) \\ &\geq (b-a) + \text{dist}(p_a, p_b^*), \end{aligned}$$

where the final relation follows by the triangle inequality. If $b-a \geq d/3$, the lemma follows immediately, and thus we assume otherwise and we will show that $\text{dist}(p_a, p_b^*) \geq d/3$. Let $g = g_a = g_b$.

As rebalancing was triggered in step b , we have $p_b^* \notin \mathcal{P}(\mathcal{C}_b, g, d)$. That is, partition p_b^* has less than d components from \mathcal{C}_b of sizes from $\langle g \rangle$ in one of the clusters (say, in cluster 0). Observe that \mathcal{C}_a can be obtained from \mathcal{C}_b by going back in time and reversing component merges, i.e., performing $b-a$ splits of components. Each such split may create two extra components of size from $\langle g \rangle$. Hence, partition p_b^* keeps less than $d + 2 \cdot (b-a) < d + (2/3) \cdot d$ components of \mathcal{C}_a of sizes from $\langle g \rangle$ in cluster 0.

Due to rebalancing in step a , we have $p_a \in \mathcal{P}(\mathcal{C}_a, g, 2d)$, i.e., p_a keeps $2d$ components of \mathcal{C}_a in cluster 0. Therefore, $\text{dist}(p_a, p_b^*) \geq d/3$, which concludes the proof. \triangleleft

\blacktriangleright **Lemma 5.** *It holds that $\text{ICB}^{\text{rb}}(\mathcal{E}_1) \leq O(n \cdot \log q) + O(n/d) \cdot \text{ICB}^{\text{ss}}(\mathcal{E}_1)$.*

Proof. Let ℓ be the number of rebalancing events within \mathcal{E}_1 . Thus, there are $\ell - 1$ disjoint chunks between two consecutive steps with rebalancing events. By Lemma 6, at most $1 + \log q$ of these chunks contain g -updating steps. We may apply Claim 13 to the remaining chunks, which yields $\text{ICB}^{\text{ss}}(\mathcal{E}_1) = \sum_{t \in [T]} \text{ICB}^{\text{ss}}(t) \geq (\ell - 2 - \log q) \cdot (d/3)$. On the other hand, the cost of a single rebalancing event is at most n , and thus

$$\begin{aligned} \text{ICB}^{\text{rb}}(\mathcal{E}_1) &\leq \ell \cdot n = (2 + \log q) \cdot n + (\ell - 2 - \log q) \cdot (d/3) \cdot (3n/d) \\ &\leq O(n \cdot \log q) + (3n/d) \cdot \text{ICB}^{\text{ss}}(\mathcal{E}_1). \end{aligned} \quad \blacktriangleleft$$

5.3 Bounding Switching Costs in Regular Steps

In this section, we show that the switching cost in regular steps is small. We start with a number-theoretic bound; its proof is deferred to the appendix.

▷ **Claim 14.** Let $A = \{a_1, a_2, \dots, a_k\} \subset \mathbb{N}_{>0}$ and $B = \{b_1, b_2, \dots, b_\ell\} \subset \mathbb{N}_{>0}$ be two non-empty and disjoint sets of positive integers. Let $g = \gcd(A \uplus B)$ and $H = \max(A \uplus B)$. Then, there exist non-negative integers $r_1, r_2, \dots, r_k, s_1, s_2, \dots, s_\ell$, such that

$$\sum_{i=1}^k r_i \cdot a_i = g + \sum_{i=1}^{\ell} s_i \cdot b_i.$$

Moreover, $\sum_{i=1}^k r_i \cdot a_i \leq 3 \cdot (k + \ell) \cdot H^2$.

We now proceed to prove several helper claims showing that, in a non- g -updating step t , both clusters contain sufficiently many components whose sizes are divisible by g_{t-1} .

▷ **Claim 15.** For any step t of \mathcal{E}_1 , it holds that $g_{t-1} < \infty$.

Proof. As $g_0 = 1$, the lemma holds trivially for $t = 1$. Hence, we assume that $t > 1$. Suppose that $g_{t-1} = \infty$. Then, $\langle g_{t-1} \rangle = \emptyset$, and therefore, there is no \mathcal{C}_{t-1} -preserving $(g_{t-1}, 2d)$ -balanced partition. Thus, when ICB executes Lines 6–8 in step $t - 1$, it would terminate \mathcal{E}_1 already in step $t - 1$. \triangleleft

▷ **Claim 16.** For any non- g -updating step t , there exists a non-empty set $A \subseteq [q]$, such that

- $\gcd(A) = g_t$,
- $\text{CNT}_i(\mathcal{C}_{t-1}) \geq w - 1$ for any $i \in A$,
- $\text{CNT}_A(\mathcal{C}_{t-1}, p_{t-1}, y) \geq q \cdot w$ for any cluster $y \in \{0, 1\}$.

Proof. We will show that set $A \triangleq \langle g_{t-1} \rangle \cap B_t \subseteq [q]$ satisfies the properties of the lemma.

For the first property, observe that by Line 5 of the algorithm, $g_t = \gcd(\langle g_{t-1} \rangle \cap B_t)$, and thus $\gcd(A) = g_t$. As step t is not g -updating, $g_t = g_{t-1}$. By Claim 15, $g_{t-1} < \infty$, and thus $g_t < \infty$ as well, which implies that A is non-empty.

As $A \subseteq B_t$, the definition of B_t implies that $\text{CNT}_i(\mathcal{C}_t) \geq w$ for any $i \in A$. There is only one component, c_x^t , that is present in \mathcal{C}_t , but not present in \mathcal{C}_{t-1} . Thus, $\text{CNT}_i(\mathcal{C}_{t-1}) \geq w - 1$ for any $i \in A$. This proves the second property of the lemma.

Finally, to show the third property, we fix any $y \in \{0, 1\}$. Lines 10–13 executed in step $t - 1$ ensure that $p_{t-1} \in \mathcal{P}(\mathcal{C}_{t-1}, g_{t-1}, d)$, i.e., $\text{CNT}_{\langle g_{t-1} \rangle}(\mathcal{C}_{t-1}, p_{t-1}, y) \geq d$.

Fix any size $i \in \langle g_{t-1} \rangle \setminus A$. By the definition of A , we have $i \notin B_t$, and thus $\text{CNT}_i(\mathcal{C}_t) \leq w - 1$. As there are only two components, c_x^t and c_y^t , that are present in \mathcal{C}_{t-1} but not present in \mathcal{C}_t , we have $\text{CNT}_i(\mathcal{C}_{t-1}) \leq w + 1$. Hence,

$$\begin{aligned} \text{CNT}_A(\mathcal{C}_{t-1}, p_{t-1}, y) &= \text{CNT}_{\langle g_{t-1} \rangle}(\mathcal{C}_{t-1}, p_{t-1}, y) - \text{CNT}_{\langle g_{t-1} \rangle \setminus A}(\mathcal{C}_{t-1}, p_{t-1}, y) \\ &\geq d - |\langle g_{t-1} \rangle \setminus A| \cdot (w + 1) \\ &\geq d - q \cdot (w + 1) \\ &\geq q \cdot w. \end{aligned}$$

where the last inequality follows as we assumed that $d \geq q \cdot (2 \cdot w + 1)$ in the definition of the algorithm. \triangleleft

▷ **Claim 17.** For any non- g -updating step t , at least one of the following properties holds:

- $\text{CNT}_{g_t}(\mathcal{C}_{t-1}, p_{t-1}, y) \geq w$ for each cluster $y \in \{0, 1\}$.
- There exists two disjoint, non-empty sets $A_0, A_1 \subseteq [q]$, such that $\gcd(A_0 \uplus A_1) = g_t$ and for each $y \in \{0, 1\}$ and $i \in A_y$, it holds that $\text{CNT}_i(\mathcal{C}_{t-1}, p_{t-1}, y) \geq (w - 1)/2$.

14:12 A Subquadratic Bound for Online Bisection

Proof. Let A be the set guaranteed by Claim 16. As $|A| \leq q$, the third property of Claim 16 implies that

$$\text{CNT}_A(\mathcal{C}_{t-1}, p_{t-1}, y) \geq |A| \cdot w \quad \text{for any } y \in \{0, 1\}. \quad (3)$$

If $|A| = 1$, then $\text{gcd}(A) = g_t$ implies that $A = \{g_t\}$. In such a case, the first condition of the lemma holds.

Hence, below we assume $|A| \geq 2$ and we will partition A into A_0 and A_1 , satisfying the second property of the lemma. For any cluster $y \in \{0, 1\}$, let

$$A'_y \triangleq \{i \in A : \text{CNT}_i(\mathcal{C}_{t-1}, p_{t-1}, y) \geq (w-1)/2\}.$$

By the second property of Claim 16, $\text{CNT}_i(\mathcal{C}_{t-1}) \geq w-1$ for any $i \in A$, and thus $A'_0 \cup A'_1 = A$. By (3) both A'_0 and A'_1 are non-empty. Thus, they satisfy all conditions of the second lemma property except being possibly non-disjoint. To fix it, we consider three cases.

- If $A'_0 \setminus A'_1 \neq \emptyset$, then we set $A_0 = A'_0 \setminus A'_1$ and $A_1 = A'_1$.
- If $A'_1 \setminus A'_0 \neq \emptyset$, then we set $A_1 = A'_1 \setminus A'_0$ and $A_0 = A'_0$.
- If $A'_0 \setminus A_1 = A'_1 \setminus A'_0 = \emptyset$, then $A'_0 = A'_1 = A$. As $|A| \geq 2$, we simply take any element $j \in A$, and set $A_0 = \{j\}$ and $A_1 = A \setminus \{j\}$. \triangleleft

► **Lemma 8.** *For any regular and not g -updating step t of \mathcal{E}_1 , $\text{ICB}^{\text{SS}}(t) = O(q^4)$.*

Proof. Recall that $\text{ICB}^{\text{SS}}(t) = 1 + \text{dist}(p_{t-1}, p_t^*)$, and p_t^* is the partition from $\mathcal{P}(\mathcal{C}_t)$ closest to p_{t-1} . Thus, our goal is to construct a partition $p \in P(\mathcal{C}_t)$ (on the basis of p_{t-1}), such that $\text{dist}(p_{t-1}, p) = O(q^4)$.

Recall that c_x^t and c_y^t are the components merged in step t . We may assume that partition p_{t-1} maps c_x^t and c_y^t to two different clusters, as otherwise $p_{t-1} \in P(\mathcal{C}_t)$, and the lemma follows by simply taking $p = p_{t-1}$.

By the lemma assumption, $g_t = g_{t-1}$. As step t is regular, the size of either c_x^t or c_y^t (or both) is from $\langle g_{t-1} \rangle = \langle g_t \rangle$. Without loss of generality, we assume that $\text{SIZE}(c_x^t) \in \langle g_t \rangle$ and let $x = \text{SIZE}(c_x^t)$. As $x \in \langle g_t \rangle$, we have $g_t < \infty$ and $g_t \mid x$. Without loss of generality, we may assume that $p_{t-1}(c_x^t) = 1$, i.e., c_x^t is in cluster 1 at the beginning of step t .

We will create p from p_{t-1} by moving components between clusters so that c_x^t changes its cluster, c_y^t does not change its cluster, and in total, at most $O(q^4)$ elements change their clusters. This will ensure that $p \in \mathcal{P}(\mathcal{C}_t)$ and $\text{dist}(p_{t-1}, p) = O(q^4)$.

Assume first that p_{t-1} maps at least $x/g_t + 1$ components of size g_t to cluster 0 (i.e., $\text{CNT}_{g_t}(\mathcal{C}_{t-1}, p_{t-1}, 0) \geq x/g_t + 1$). At least x/g_t of these components are different than c_y^t , and thus, we may simply swap c_x^t with them, at a total cost of $2x \leq 2q$.

Hence, in the following, we assume that $\text{CNT}_{g_t}(\mathcal{C}_{t-1}, p_{t-1}, 0) < x/g_t + 1$. This implies $\text{CNT}_{g_t}(\mathcal{C}_{t-1}, p_{t-1}, 0) < q+1 \leq w$. As the first property of Claim 17 is false, the second one must hold. That is, there exist two disjoint, non-empty sets $A_0, A_1 \subseteq [q]$, such that $\text{gcd}(A_0 \uplus A_1) = g_t$. Furthermore, for any $y \in \{0, 1\}$ and $i \in A_y$, it holds that $\text{CNT}_i(\mathcal{C}_{t-1}, p_{t-1}, y) \geq (w-1)/2$. By Claim 14 applied to sets A_0 and A_1 , there exist non-negative integers r_i , such that

$$\sum_{i \in A_0} r_i \cdot i = g_t + \sum_{i \in A_1} r_i \cdot i.$$

and $\sum_{i \in A_0} r_i \cdot i \leq 3 \cdot (|A_0| + |A_1|) \cdot q^2 \leq 3 \cdot q^3$. This also implies that $r_i \leq 3 \cdot q^3$ for any $i \in A_0 \uplus A_1$. Multiplying both sides by x/g_t , we obtain

$$\sum_{i \in A_0} \frac{x \cdot r_i}{g_t} \cdot i = x + \sum_{i \in A_1} \frac{x \cdot r_i}{g_t} \cdot i. \quad (4)$$

We create p from p_{t-1} by executing the following actions:

- For any $i \in A_0$, move $x \cdot r_i/g_t$ components of size i (other than c_y^t) from cluster 0 to cluster 1.
- For any $i \in A_1$, move $x \cdot r_i/g_t$ components of size i (other than c_x^t) from cluster 1 to cluster 0.
- Move component c_x^t from cluster 1 to cluster 0.

We observe that these actions are feasible: For any $i \in A_0$, we have $x \cdot r_i/g_t \leq q \cdot r_i \leq 3 \cdot q^4 \leq (w-1)/2 - 1$, so cluster 0 contains an appropriate number of components of size i (different from c_y^t). Analogously, for any $i \in A_1$, cluster 1 contains an appropriate number of components of size i (different from c_x^t). Next, the resulting partition p is \mathcal{C}_t -preserving as both c_x^t and c_y^t end up in cluster 0, and (4) ensures that the total number of elements in each cluster remains unchanged.

Finally, the number of elements that change their cluster is

$$\text{dist}(p_{t-1}, p) = 2 \sum_{i \in A_0} \frac{x \cdot r_i}{g_t} \cdot i \leq 2 \cdot q \cdot \sum_{i \in A_0} r_i \cdot i \leq 6 \cdot q^4. \quad \blacktriangleleft$$

5.4 Bounding the Number of Irregular Merges

To bound the number of irregular steps, we trace the evolution of components. Recall that $\mathcal{C}_t = \mathcal{C}_{t-1} \cup \{c_z^t\} \setminus \{c_x^t, c_y^t\}$, i.e., components c_x^t and c_y^t are merged in step t into component c_z^t . We say that components c_x^t and c_y^t are *destroyed* in step t and component c_z^t is *created* in step t . We extend these notions also to the (singleton) components of \mathcal{C}_0 , where we say that they are created in step 0, and to components of \mathcal{C}_T , where we say that they are destroyed in step $T+1$.

We now fix a small component c created at time a and destroyed at time b . Note that $0 \leq a < b \leq T+1$. By Lemma 6, $\langle g_{b-1} \rangle \subseteq \langle g_a \rangle$. We say that the component c is

- *typical* if $\text{SIZE}(c) \in \langle g_{b-1} \rangle$,
- *mixed* if $\text{SIZE}(c) \in \langle g_a \rangle \setminus \langle g_{b-1} \rangle$,
- *atypical* if $\text{SIZE}(c) \notin \langle g_a \rangle$.

That is, each component is either large, typical, atypical, or mixed. In particular, in a regular merge, at least one of the merged components is typical.

▷ **Claim 18.** Assume step t is not g -updating and both c_x^t and c_y^t are typical. Then, c_z^t is not atypical.

Proof. As components are typical, $\text{SIZE}(c_x^t) \in \langle g_{t-1} \rangle$ and $\text{SIZE}(c_y^t) \in \langle g_{t-1} \rangle$, and therefore $g_{t-1} \mid \text{SIZE}(c_x^t)$ and $g_{t-1} \mid \text{SIZE}(c_y^t)$. As $\text{SIZE}(c_z^t) = \text{SIZE}(c_x^t) + \text{SIZE}(c_y^t)$, we have $g_{t-1} \mid \text{SIZE}(c_z^t)$. Finally, as step t is not g -updating, $g_t = g_{t-1}$, and hence $g_t \mid \text{SIZE}(c_z^t)$. If c_z^t is large then the lemma follows immediately. If c_z^t is small, then we have $\text{SIZE}(c_z^t) \in \langle g_t \rangle$, and thus c_z^t cannot be atypical. ◁

Merge Forest. It is convenient to consider the following *merge forest* \mathcal{F} , whose nodes correspond to all components created within \mathcal{E}_1 . We connect these nodes by edges in a natural manner: the leaves of \mathcal{F} correspond to initial singleton components of \mathcal{C}_0 , and each non-leaf node of \mathcal{F} corresponds to a component created by merging its children components. We say that the node of \mathcal{F} is large/typical/atypical/mixed if the corresponding component is of such type.

Types of Irregular Merges. To upper-bound the number of irregular merges, we subdivide them into three types.

- *All-large irregular merges:* both merged components are large (and the resulting component is clearly large as well).
- *Mixed-resulting irregular merges:* the created component is mixed.
- *Ordinary irregular merges:* all other irregular merges.

We bound the number of these merges separately in the following three lemmas.

▷ **Claim 19.** \mathcal{F} contains at most n/q large nodes whose both children are also large.

Proof. Let L be the set of large nodes of \mathcal{F} . Clearly, L is “upward-closed”, i.e., if L contains a node, then it also contains its parent. Let \mathcal{F}_L be the sub-forest of \mathcal{F} induced by nodes from L . We partition L into three sets: L_0 , L_1 and L_2 , where a component from L_i has exactly i children in \mathcal{F}_L . We need to show that $|L_2| \leq n/q$.

As L_2 contains the branching internal nodes of \mathcal{F}_L and L_0 contains the leaves of \mathcal{F}_L , we have $|L_2| < |L_0|$. All components from L_0 are large, i.e., each of them consists of at least $q + 1$ nodes. Fix any two components from L_0 . As they are leaves of \mathcal{F}_L , they are not in the ancestor-descendant relation in \mathcal{F}_L (and not in \mathcal{F}), and hence the sets of their elements are disjoint. Thus, all components of L_0 are disjoint, which implies $|L_0| \cdot (q + 1) \leq n$. Summing up, $|L_2| < |L_0| \leq n/(q + 1)$. ◁

▷ **Claim 20.** \mathcal{F} contains at most $q \cdot w$ mixed nodes.

Proof. Fix any mixed node corresponding to component c that is created at step a and destroyed at step $b > a$. By Lemma 6, $\langle g_a \rangle \supseteq \langle g_{a+1} \rangle \supseteq \dots \supseteq \langle g_{b-1} \rangle$. As $\text{SIZE}(c) \in \langle g_a \rangle$ and $\text{SIZE}(c) \notin \langle g_{b-1} \rangle$, there exists a step $t \in [a + 1, b - 1]$, such that $\text{SIZE}(c) \in \langle g_{t-1} \rangle \setminus \langle g_t \rangle$. We say that component c is t -mixed.

We now fix a step t and show that the number of t -mixed components is at most $w \cdot |\langle g_{t-1} \rangle \setminus \langle g_t \rangle|$. Fix $j \in \langle g_{t-1} \rangle \setminus \langle g_t \rangle$. We show that at step t , the number of components of size j is at most w . Suppose for a contradiction that $\text{CNT}_j(\mathcal{C}_t) \geq w$. Then, $j \in B_t$. As $j \in \langle g_{t-1} \rangle$, we have $j \in \langle g_{t-1} \rangle \cap B_t$. On the other hand, $g_t = \text{gcd}(\langle g_{t-1} \rangle \cap B_t)$, and thus $g_t \mid j$. However, as $j \in [q]$, we would then have $j \in \langle g_t \rangle$, a contradiction.

As any mixed node is t -mixed for a step $t \in [T]$, the total number of all mixed nodes is at most $\sum_{t \in [T]} w \cdot |\langle g_{t-1} \rangle \setminus \langle g_t \rangle| \leq w \cdot |\langle g_0 \rangle \setminus \langle g_T \rangle| \leq w \cdot |\langle g_0 \rangle| = w \cdot q$. ◁

It remains to bound the number of ordinary irregular merges. To this end, we define the following amounts (for any step $t \geq 0$).

- a_t is the number of components in \mathcal{C}_t that are atypical or mixed.
- I_t is the number of ordinary irregular merges in steps $1, 2, \dots, t$.

▷ **Claim 21.** For any step $t \geq 0$, it holds that $a_t = O(q \cdot w)$ and $I_t = O(q \cdot w)$.

Proof. Let R_t be the number of (regular or irregular) merges in steps $1, 2, \dots, t$ in which c_z^t (the created component) is mixed. Let U_t be the number of g -updating steps among steps $1, 2, \dots, t$. We inductively show that

$$a_t + I_t \leq 2 \cdot (R_t + U_t). \quad (5)$$

The lemma will follow as $R_t = O(q \cdot w)$ by Claim 20 and $U_t \leq 1 + \log q$ by Lemma 6.

The base case ($t = 0$) holds as both sides are then trivially equal to 0. We assume that (5) holds for step $t - 1$ and we show it for step t . Let $\Delta a = a_t - a_{t-1}$; we define ΔI , ΔR , and ΔU analogously. It suffices to show that

$$\Delta a + \Delta I \leq 2 \cdot (\Delta R + \Delta U). \quad (6)$$

Observe that $\Delta a \in \{-2, -1, 0, 1\}$ and $\Delta I \in \{0, 1\}$. Thus, if either $\Delta R \geq 1$ or $\Delta U \geq 1$, then (6) holds trivially. This happens when step t is g -updating or c_z^t is mixed.

Thus, in the remaining part of the proof, we assume that step t is not g -updating and the c_z^t is not mixed. In such case, $\Delta R = 0$ and $\Delta U = 0$, and thus it remains to show that

$$\Delta a + \Delta I \leq 0. \tag{7}$$

We consider a few cases depending on the merge type at step t . As c_z^t is not mixed, the merge cannot be mixed-resulting irregular.

- Merge is regular, i.e., at least one of the merged components, say c_x^t , is typical. Then, $\Delta I = 0$, and we will show that $\Delta a \leq 0$.
 - If c_y^t is also typical, then Claim 18 implies that c_z^t cannot be atypical. As we assumed c_z^t is not mixed, it must be either large or typical. Hence, $\Delta a = 0$.
 - If c_y^t is large, then c_z^t is large as well, and then $\Delta a = 0$.
 - If c_y^t is atypical or mixed, then $\Delta a \leq 0$.
- Merge is all-large irregular. Then, $\Delta I = 0$ and $\Delta a = 0$.
- Merge is ordinary irregular, i.e., $\Delta I = 1$. We will show that $\Delta a \leq -1$. If both c_x^t and c_y^t are atypical or mixed, then we immediately obtain $\Delta a \leq -1$. Otherwise, we note that the merge is irregular, and hence neither c_x^t nor c_y^t is typical. Thus, one of them is large and the second one is atypical or mixed. Then, c_z^t is large as well, and thus $\Delta a \leq -1$ as well.

In either case, (7) follows, which concludes the inductive proof. \triangleleft

► **Lemma 9.** *There are at most $O(q \cdot w + n/q)$ irregular steps in \mathcal{E}_1 . Moreover, at any time t of \mathcal{E}_1 , $CNT_{[n] \setminus \langle g_t \rangle}(\mathcal{C}_t) = O(q \cdot w + n/q)$.*

Proof. There are at most n/q all-large irregular steps by Claim 19, at most $q \cdot w$ mixed-resulting irregular steps by Claim 20, and $I_T = O(q \cdot w)$ ordinary irregular steps by Claim 21. This shows the first part of the lemma.

For the second part, fix any step t . Observe that components whose sizes are from $[n] \setminus \langle g_t \rangle$ are not typical, i.e., they must be either large, atypical, or mixed. Trivially, there are at most $n/(q+1)$ large components, and the number of atypical and mixed components is $a_t = O(q \cdot w)$ by Claim 21. \blacktriangleleft

6 Analysis: the Second Stage of ICB

► **Lemma 3.** $\mathbf{E}[ICB(\mathcal{E}_2)] = O(n \cdot \log |\mathcal{P}(\mathcal{C}_T)|)$.

Proof. At the beginning of \mathcal{E}_2 , there are $|\mathcal{P}(\mathcal{C}_T)|$ \mathcal{C}_T -preserving partitions. In a step t of \mathcal{E}_2 , ICB chooses a new partition only when its current partition is not \mathcal{C}_t -preserving. In such case, it chooses a new partition p_t uniformly at random from $\mathcal{P}(\mathcal{C}_t)$.

Thus, we may treat the problem as the metrical task system (MTS) on $|\mathcal{P}(\mathcal{C}_T)|$ states, where the adversary makes the states (partitions) forbidden in some specified order. ICB then basically executes (a single phase of) the known randomized algorithm for MTS on a uniform metric [5]. By the result of [5], the expected number of times when ICB is forced to choose a new partition is $O(\log |\mathcal{P}(\mathcal{C}_T)|)$. Whenever that happens, ICB pays at most $n+1$ (cf. Lemma 1), and thus $\mathbf{E}[ICB(\mathcal{E}_2)] = (n+1) \cdot O(\log |\mathcal{P}(\mathcal{C}_T)|)$. \blacktriangleleft

► **Lemma 4.** $|\mathcal{P}(\mathcal{C}_T)| = \exp(O(d \cdot \log n + z))$, where $z = CNT_{[n] \setminus \langle g_T \rangle}(\mathcal{C}_T)$.

Proof. Recall that $|\mathcal{P}(\mathcal{C}_T)|$ is the number of ways we can feasibly assign components of \mathcal{C}_T to two clusters at the end of the first stage. We partition components of \mathcal{C}_T into set A containing components of sizes from $\langle g_T \rangle$ and set B containing components of sizes outside $\langle g_T \rangle$, i.e., $|B| = z$. We separately upper-bound the number of ways of assigning components from A and components from B to two clusters.

We assume that the second stage is present as otherwise $|\mathcal{P}(\mathcal{C}_T)| = 0$. Thus, the condition in Line 6 of the algorithm guarantees that $\mathcal{P}(\mathcal{C}_T, g_T, 2d) = \emptyset$, i.e., each feasible mapping has less than $2 \cdot d$ of components from A at least on one side. Thus, the overall number of ways of partitioning sets of A among two clusters is at most

$$\sum_{i=0}^{2d-1} 2 \cdot \binom{|A|}{i} \leq \sum_{i=0}^{2d-1} 2 \cdot \binom{n}{i} \leq 4d \cdot \binom{n}{2d} \leq 4d \cdot \frac{e^{2d} \cdot n^{2d}}{(2d)^{2d}} = \exp(O(d \cdot \log n)).$$

As the components of B can be assigned to two clusters in at most $2^{|B|} = 2^z$ ways, we have $|\mathcal{P}(\mathcal{C}_t)| \leq 2^z \cdot \exp(O(d \cdot \log n))$. ◀

7 Final Remarks

In this paper, we provided the first algorithm for the online bisection problem with the competitive ratio of $o(n^2)$. Extending the result to a more general setting of online balanced graph partitioning (i.e., multiple-cluster case) is an intriguing open problem. We note that our algorithm ICB has non-polynomial running time; we conjecture that without resource augmentation, achieving a subquadratic competitive ratio in polynomial time is not possible.

References

- 1 Sanjeev Arora, David R. Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *Journal of Computer and System Sciences*, 58(1):193–210, 1999. doi:10.1006/jcss.1998.1605.
- 2 Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic balanced graph partitioning. *SIAM Journal on Discrete Mathematics*, 34(3):1791–1812, 2020. doi:10.1137/17M1158513.
- 3 Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Online balanced repartitioning. In *Proc. 30th Int. Symp. on Distributed Computing (DISC)*, pages 243–256, 2016. doi:10.1007/978-3-662-53426-7_18.
- 4 Marcin Bienkowski, Martin Böhm, Martin Koutecký, Thomas Rothvoß, Jirí Sgall, and Pavel Veselý. Improved analysis of online balanced clustering. In *Proc. 19th Workshop on Approximation and Online Algorithms (WAOA)*, pages 224–233, 2021. doi:10.1007/978-3-030-92702-8_14.
- 5 Alan Borodin, Nati Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, 1992. doi:10.1145/146585.146588.
- 6 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- 7 Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002. doi:10.1137/S0097539701387660.
- 8 Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. Approximating the minimum bisection size. In *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pages 530–536, 2000. doi:10.1145/335305.335370.
- 9 Tobias Forner, Harald Räcke, and Stefan Schmid. Online balanced repartitioning of dynamic communication patterns in polynomial time. In *2nd Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 40–54, 2021. doi:10.1137/1.9781611976489.4.

- 10 Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. Tight bounds for online graph partitioning. In *Proc. 32nd ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 2799–2818, 2021. doi:10.1137/1.9781611976465.166.
- 11 Monika Henzinger, Stefan Neumann, and Stefan Schmid. Efficient distributed workload (re-)embedding. In *Proc. SIGMETRICS/Performance Joint Int. Conf. on Measurement and Modeling of Computer Systems*, pages 43–44, 2019. doi:10.1145/3309697.3331503.
- 12 Robert Krauthgamer. Minimum bisection. In *Encyclopedia of Algorithms*, pages 1294–1297. Springer, 2016. doi:10.1007/978-1-4939-2864-4_231.
- 13 Robert Krauthgamer and Uriel Feige. A polylogarithmic approximation of the minimum bisection. *SIAM Review*, 48(1):99–130, 2006. doi:10.1137/050640904.
- 14 Bohdan S. Majewski and George Havas. The complexity of greatest common divisor computations. In *1st Symposium on Algorithmic Number Theory (ANTS-I)*, pages 184–193, 1994. doi:10.1007/3-540-58691-1_56.
- 15 Maciej Pacut, Mahmoud Parham, and Stefan Schmid. Brief announcement: Deterministic lower bound for dynamic balanced graph partitioning. In *Proc. 39th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 461–463, 2020. doi:10.1145/3382734.3405696.
- 16 Maciej Pacut, Mahmoud Parham, and Stefan Schmid. Optimal online balanced graph partitioning. In *Proc. 40th IEEE Int. Conf. on Computer Communications (INFOCOM)*, pages 1–9, 2021. doi:10.1109/INFOCOM42981.2021.9488824.
- 17 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 40th ACM Symp. on Theory of Computing (STOC)*, pages 255–264, 2008. doi:10.1145/1374376.1374415.
- 18 Harald Räcke, Stefan Schmid, and Ruslan Zabrodin. Approximate dynamic balanced graph partitioning. In *Proc. 34th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 401–409, 2022. doi:10.1145/3490148.3538563.
- 19 Rajmohan Rajaraman and Omer Wasim. Improved bounds for online balanced graph repartitioning. In *Proc. 30th European Symp. on Algorithms (ESA)*, pages 83:1–83:15, 2022. doi:10.4230/LIPIcs.ESA.2022.83.
- 20 Huzur Saran and Vijay V. Vazirani. Finding k cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–108, 1995. doi:10.1137/S0097539792251730.
- 21 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

A Proof of Claim 14

▷ **Claim 14.** Let $A = \{a_1, a_2, \dots, a_k\} \subset \mathbb{N}_{>0}$ and $B = \{b_1, b_2, \dots, b_\ell\} \subset \mathbb{N}_{>0}$ be two non-empty and disjoint sets of positive integers. Let $g = \gcd(A \uplus B)$ and $H = \max(A \uplus B)$. Then, there exist non-negative integers $r_1, r_2, \dots, r_k, s_1, s_2, \dots, s_\ell$, such that

$$\sum_{i=1}^k r_i \cdot a_i = g + \sum_{i=1}^{\ell} s_i \cdot b_i.$$

Moreover, $\sum_{i=1}^k r_i \cdot a_i \leq 3 \cdot (k + \ell) \cdot H^2$.

Proof. By the bound given by Majewski and Havas [14], there exist coefficients $\tilde{r}_1, \tilde{r}_2, \dots, \tilde{r}_k$ and $\tilde{s}_1, \tilde{s}_2, \dots, \tilde{s}_\ell$, such that $|\tilde{r}_i| \leq \max\{H/2g, 1\} \leq H$ and $|\tilde{s}_i| \leq \max\{H/2g, 1\} \leq H$ for any i , and

$$\sum_{i=1}^k \tilde{r}_i \cdot a_i = g + \sum_{i=1}^{\ell} \tilde{s}_i \cdot b_i. \tag{8}$$

14:18 A Subquadratic Bound for Online Bisection

However, these coefficients are not necessarily non-negative. To fix it, let

$$\begin{aligned} r_1 &= \tilde{r}_1 + \lceil H/b_1 \rceil \cdot b_1 + \sum_{i=1}^{\ell} \lceil H/a_1 \rceil \cdot b_i, \\ r_i &= \tilde{r}_i + \lceil H/b_1 \rceil \cdot b_1 && \text{for } i \in \{2, \dots, k\}, \\ s_1 &= \tilde{s}_1 + \lceil H/a_1 \rceil \cdot a_1 + \sum_{i=1}^k \lceil H/b_1 \rceil \cdot a_i, \\ s_i &= \tilde{s}_i + \lceil H/a_1 \rceil \cdot a_1 && \text{for } i \in \{2, \dots, \ell\}. \end{aligned}$$

We argue that the values above satisfy the lemma conditions. First, note that $r_i \geq \tilde{r}_i + H$ and $s_i \geq \tilde{s}_i + H$, and thus r_i and s_i are non-negative for every i .

To show that r_i 's and s_i 's satisfy the identity required by the lemma, we analyze the following term

$$\begin{aligned} \sum_{i=1}^k (r_i - \tilde{r}_i) \cdot a_i &= (r_1 - \tilde{r}_1) \cdot a_1 + \sum_{i=2}^k (r_i - \tilde{r}_i) \cdot a_i \\ &= \left(\lceil H/b_1 \rceil \cdot b_1 + \sum_{i=1}^{\ell} \lceil H/a_1 \rceil \cdot b_i \right) \cdot a_1 + \sum_{i=2}^k \lceil H/b_1 \rceil \cdot b_1 \cdot a_i \\ &= \lceil H/a_1 \rceil \cdot a_1 \cdot \sum_{i=1}^{\ell} b_i + \lceil H/b_1 \rceil \cdot b_1 \cdot \sum_{i=1}^k a_i. \end{aligned} \quad (9)$$

In the same way, but swapping the roles of a 's and r 's with b 's and s 's, we obtain

$$\sum_{i=1}^{\ell} (s_i - \tilde{s}_i) \cdot b_i = \lceil H/b_1 \rceil \cdot b_1 \cdot \sum_{i=1}^k a_i + \lceil H/a_1 \rceil \cdot a_1 \cdot \sum_{i=1}^{\ell} b_i. \quad (10)$$



Therefore, (9) and (10) together imply $\sum_{i=1}^k (r_i - \tilde{r}_i) \cdot a_i = \sum_{i=1}^{\ell} (s_i - \tilde{s}_i) \cdot b_i$. Combining this relation with (8), immediately yields

$$\sum_{i=1}^k r_i \cdot a_i = g + \sum_{i=1}^{\ell} s_i \cdot b_i.$$

It remains to upper-bound $\sum_{i=1}^k r_i \cdot a_i$. Note that for any $z \leq H$, it holds that $\lceil H/z \rceil \cdot z < (H/z + 1) \cdot z \leq H + z \leq 2H$. Hence, using $\tilde{r}_i \leq H$ (for every i) and (9), we obtain

$$\begin{aligned} \sum_{i=1}^k r_i \cdot a_i &= \sum_{i=1}^k \tilde{r}_i \cdot a_i + \sum_{i=1}^k (r_i - \tilde{r}_i) \cdot a_i \\ &\leq H \cdot \sum_{i=1}^k a_i + \lceil H/a_1 \rceil \cdot a_1 \cdot \sum_{i=1}^{\ell} b_i + \lceil H/b_1 \rceil \cdot b_1 \cdot \sum_{i=1}^k a_i \\ &\leq H \cdot \left(3 \cdot \sum_{i=1}^k a_i + 2 \cdot \sum_{i=1}^{\ell} b_i \right) \\ &\leq (3 \cdot k + 2 \cdot \ell) \cdot H^2. \end{aligned} \quad \triangleleft$$

An Improved Approximation Algorithm for Dynamic Minimum Linear Arrangement

Marcin Bienkowski  

University of Wrocław, Poland

Guy Even  

Tel Aviv University, Israel

Abstract

The dynamic offline linear arrangement problem deals with reordering n elements subject to a sequence of edge requests. The input consists of a sequence of m edges (i.e., unordered pairs of elements). The output is a sequence of permutations (i.e., bijective mapping of the elements to n equidistant points). In step t , the order of the elements is changed to the t -th permutation, and then the t -th request is served. The cost of the output consists of two parts per step: request cost and rearrangement cost. The former is the current distance between the endpoints of the request, while the latter is proportional to the number of adjacent element swaps required to move from one permutation to the consecutive permutation. The goal is to find a minimum cost solution.

We present a deterministic $O(\log n \log \log n)$ -approximation algorithm for this problem, improving over a randomized $O(\log^2 n)$ -approximation by Olver et al. [22]. Our algorithm is based on first solving spreading-metric LP relaxation on a time-expanded graph, applying a tree decomposition on the basis of the LP solution, and finally converting the tree decomposition to a sequence of permutations. The techniques we employ are general and have the potential to be useful for other dynamic graph optimization problems.

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases Minimum Linear Arrangement, dynamic Variant, Optimization Problems, Graph Problems, approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.15

Funding Supported by Polish National Science Centre grant 2022/45/B/ST6/00559.

1 Introduction

This paper is motivated by a growing interest in optimization problems in a *dynamic* setting [22, 26]. By dynamic we mean that constraints or penalties associated with requests are ephemeral (i.e., disappear after the request is served). Such problems are well established in the online setting (e.g., metrical task systems [21] or server problems [17]), but are also of interest in the offline case.

A special case of dynamic optimization is the dynamic version of the classic Minimum Linear Arrangement problem (MLA). In the Minimum Linear Arrangement (MLA) problem, the input consists of a graph $G = (V, E)$. The output is an ordering of elements of V (i.e., a bijection π from V to $\{1, \dots, n\}$). The cost of the solution is the total stretch of the edges, i.e., $\sum_{(u,v) \in E} |\pi(u) - \pi(v)|$, and the goal is to find an ordering with minimum cost. The MLA problem is NP-hard [11]. A large variety of ideas and approximation techniques were developed for MLA [14, 7, 28] culminating in an $O(\sqrt{\log n} \cdot \log \log n)$ -approximation [2, 8].

Recently, the MLA problem has been studied in a dynamic setting, where the input consists of a sequence of m edges, and an algorithm has to output a sequence of permutations [22]. For a given edge (u, v) (a request) appearing in the input sequence, an algorithm may first change its current permutation π of elements paying γ for each swap of adjacent elements, and then



© Marcin Bienkowski and Guy Even;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 15; pp. 15:1–15:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



it has to pay the usual price of $|\pi(u) - \pi(v)|$. The parameter $\gamma > 0$ is used to quantify the ratio between the cost of swapping an adjacent pair and serving a request between adjacent vertices. While the problem is appealing from the theoretical point of view, its solution can be also used for track management in domain wall memory [13, 22]. Olver et al. [22] proved that the dynamic variant of MLA admits a randomized $O(\log^2 n)$ -approximation for $\gamma = 1$.

As our problem definition does not specify the initial permutation¹, setting $\gamma > n \cdot m$ penalizes rearrangement to the extent that every solution with even one swap is suboptimal. Hence, in this setting, the DMLA problem reduces to the static MLA problem, thus proving that DMLA is NP-hard.

1.1 Our Result and Techniques

We present a deterministic approximation algorithm for the dynamic offline MLA problem, achieving an improved approximation ratio of $O(\log n \log \log n)$. We emphasize that the constants of the approximation ratio do not depend on the parameter γ nor the input length m . Similarly to Olver et al. [22], we work on a time-expanded graph that contains copies of the element set (one for each time step), and whose edges encode requests and rearrangements.

The $O(\log^2 n)$ -approximation algorithm of [22] is based on a non-trivial divide-and-conquer argument, where the time-expanded graph is recursively and randomly partitioned using a balanced cut routine. The translation of the recursive partitioning to a sequence of permutations requires a “delicate shuffling” argument. One of the challenges in their analysis is locally bounding the cost of rearrangements induced by the recursive decomposition. In contrast to their approach, we propose an algorithm that computes the sequence of permutations in a unified and straightforward manner directly from the decomposition of the time-expanded graph.

Our approach builds on the following ideas. First, apply a spreading LP relaxation to assign lengths to edges of the time-expanded graph (cf. Section 3). Next, apply a tree decomposition algorithm [7] to the time-expanded graph (cf. Section 4). This binary decomposition tree represents a laminar partitioning of the time-expanded graph. For every time step, the permutation is simply extracted by applying an in-order traversal of the decomposition tree intersected with the corresponding time-slice. (See Section 5 for a description of the algorithm.)

A key component of our analysis is a local charging scheme that bounds the solution cost by the cost of the decomposition tree (see Section 6). We rely on [7] to bound the latter by a function of the cost of the optimal solution to the LP relaxation.

1.2 Related Work on Dynamic Graph Problems

Dynamic MLA Problem. Our paper is among the few that study the *approximability* of graph problems, where the input is a sequence of requests in the form of edges and the output is a sequence of “configurations”. The cost of a solution consists of “serving” the request and the cost of “moving” from one configuration to the next configuration. These problems have been usually considered in the context of *online* algorithms, where the solution must be created without knowledge of future edges, and its time complexity is of secondary importance.

¹ We discuss the implication of this assumption in Section 7.

Indeed, the dynamic MLA problem has been studied also in online flavor. As every request incurs the cost of at least 1 and at most n , the competitive ratio of an online algorithm that does not change its permutation is trivially $O(n)$. Surprisingly, it is unknown whether an online algorithm that beats this ratio exists. On the negative side, Olver et al. showed that many natural online policies have competitive ratios of $\Omega(n/\log n)$; the best known lower bound is merely $\Omega(\log n)$ [22]. We believe that the techniques of this paper will help improve our understanding of the problem, and provide insights that could eventually be used to design online algorithms for dynamic MLA with non-trivial competitive ratios.

Dynamic Balanced Graph Partitioning. Another (seemingly similar) task is the *balanced graph partitioning* problem [1, 9, 19], where the goal is to split n graph vertices into ℓ clusters, each containing n/ℓ vertices, so that the number of crossing edges is minimized. For $\ell = 2$, the problem becomes the *bisection* problem [18, 25]. This problem has been recently extended to a dynamic *online* setting, where edges arrive one by one, and the algorithm may change the clustering on the fly [3, 5, 10, 15, 16, 23, 24, 27]. Large competitive ratios of online solutions motivated the study of the problem in the dynamic *offline* setting, and recently Räcke et al. [26] constructed an $O(\log n)$ -approximation for this variant.

On a high level, the framework of Räcke et al. [26] bears some similarities to ours: they also create a time-expanded graph, solve an LP-relaxation of the problem on this graph, use the edge lengths returned by the LP to partition this graph, and finally transform the partition into the final solution (sequence of graph clusterings). Similarities end here as the details of their approach are quite different. In particular, their LP relaxation is based on a distinct type of spreading metric (using knapsack-like constraints), they use Bartal's randomized decomposition [4] to partition the graph, and the final step of transforming the partition to a sequence of clusterings is completely different.

That said, we hope that their and our paper will inspire further work on dynamic graph optimization problems, and eventually a coherent set of tools will be developed to tackle such problems.

2 Preliminaries

For an integer ℓ , we use $[\ell] \triangleq \{1, \dots, \ell\}$. The union of mutually disjoint sets X and Y is denoted by $X \uplus Y$. Throughout this paper, we work with a set of elements Q and we denote its cardinality by n . A *permutation of Q* (or simply *a permutation*) is a bijection from Q to $[n]$. We say that two elements a and b are *adjacent* in permutation π if $|\pi(a) - \pi(b)| = 1$. To reduce ambiguity, we refer to members of Q as *elements*, to vertices of decomposition trees as *nodes*, and use the *vertices* only for vertices of the time-expanded graph defined in Section 3.

Permutation Distances. An unordered pair $\{x, y\}$ of distinct elements of Q is *discordant* with respect to permutations π and π' if $(\pi(x) - \pi(y)) \cdot (\pi'(x) - \pi'(y)) < 0$. We use two notions of distance between partitions π and π' :

- Kendall's tau-distance $\text{tdist}(\pi, \pi')$ equal to the minimum number of swaps of adjacent elements required to reach permutation π' from π . This distance is also equal to the number of discordant pairs between π and π' .
- Spearman's footrule distance defined as $\text{fdist}(\pi, \pi') \triangleq \sum_{v \in Q} |\pi'(v) - \pi(v)|$.

While $\text{tdist}(\pi, \pi')$ is convenient for establishing relations between decomposition trees and permutations, $\text{fdist}(\pi, \pi')$ is better suited for defining spreading metrics in LP relaxations. The Diaconis-Graham inequality [6, 20] states that these two distances can differ at most by a factor of 2, i.e.,

$$\text{tdist}(\pi, \pi') \leq \text{fdist}(\pi, \pi') \leq 2 \cdot \text{tdist}(\pi, \pi'). \quad (1)$$

Problem Definition. The Dynamic Minimum Linear Arrangement (DMLA) problem is specified by a tuple $\mathcal{I} = (n, m, \text{IN})$, where n is the number of elements (i.e., $|Q| = n$), m is the number of requests, and IN is the input sequence consisting of m requests, each being an unordered pair of distinct elements. That is, $\text{IN} \triangleq \{(a_t, b_t)\}_{t=1}^m$, where $(a_t, b_t) \in Q \times Q$ and $a_t \neq b_t$.

A feasible solution is a sequence of m permutations $\{\pi_t\}_{t=1}^m$, and its cost is

$$\text{COST}(\mathcal{I}, \{\pi_t\}_{t=1}^m) \triangleq \sum_{t=1}^m \{\gamma \cdot \text{fdist}(\pi_{t-1}, \pi_t) + |\pi_t(a_t) - \pi_t(b_t)|\}.$$

where we assume that $\pi_0 = \pi_1$, i.e., $\text{fdist}(\pi_{t-1}, \pi_t) = 0$. The goal is to find a feasible solution with minimum cost. We call $\text{fdist}(\pi_{t-1}, \pi_t)$ the *rearrangement cost* at time t and $|\pi_t(a_t) - \pi_t(b_t)|$ the *request cost* at time t . Finally, for an algorithm A , we denote its cost on \mathcal{I} by $A(\mathcal{I})$, and we use OPT to denote the optimal algorithm.

As the cost incurred in every time step is at least 1 and at most $n - 1$, we have the following trivial bounds on the value of OPT .

▷ **Claim 1.** $m \leq \text{OPT}(\mathcal{I}) \leq m \cdot (n - 1)$ for an instance $\mathcal{I} = (n, m, \text{IN})$.

Note on the Parameter γ . Consider the following greedy algorithm that at time t moves the element b_t so that it becomes adjacent to a_t . It pays at most $(n - 2) \cdot \gamma$ for rearrangement and then 1 for the request. For $\gamma < 1/n$, this amounts to at most 2. As OPT pays at least 1 for the request cost alone, the greedy algorithm is trivially an $O(1)$ -approximation for $\gamma < 1/n$. Hence, in the remaining part of the paper, we assume that $\gamma \geq 1/n$.

Consider an algorithm that does not employ rearranging. Namely, it finds an α -approximation for the (static) MLA instance that contains an edge for every request. This algorithm would also be an α -approximation for the DMLA instance if $\gamma > m \cdot (n - 1)$. Indeed, as $\text{OPT} \leq m \cdot (n - 1)$, employing even one swap would be suboptimal. Because the (static) MLA problem admits an $O(\log n \log \log n)$ -approximation, we may assume that $\gamma \leq m \cdot (n - 1)$.

Our algorithm crucially requires the assumption $\gamma \geq 1/n$ to guarantee the approximation ratio. The assumption $\gamma \leq m \cdot (n - 1)$ is used to ensure that the edge costs of our time-expanded graphs are polynomially bounded, and thus to ensure polynomial runtime.

3 Time-Expanded Graph and Linear Relaxation

In this section, we present a linear-programming relaxation for DMLA. The relaxation is defined over a time-expanded graph that represents the DMLA instance. We note that equivalent definitions of time-expanded graphs appeared in the papers of Olver et al. [22] and Räcke et al. [26].

Consider a DMLA instance $\mathcal{I} = (n, m, \text{IN})$. We represent \mathcal{I} by a weighted *time-expanded graph* $G = (V, E, c)$ as follows. Recall that Q is the set of elements. The set of vertices $V \triangleq Q \times \{0, \dots, m\}$ contains a copy of Q for every time $t \in \{0, \dots, m\}$. We refer to each such copy as a *time-slice*.

To simplify notation, we denote the vertex $(v, t) \in V$ by v^t , namely, v^t is the t -th copy of the *element* $v \in Q$. The t -th time-slice of graph G is $Q^t \triangleq Q \times \{t\}$. Furthermore, for a subset of elements $A \subseteq Q$, we use $A^t \triangleq \{v^t \mid v \in A\}$ to denote the corresponding set of vertices in the t -th time-slice.

There are two types of edges in E . First, we introduce a set of *request edges* E_{IN} containing an edge $\{a_t^t, b_t^t\}$ for every request (a_t, b_t) in the input IN . Second, we introduce $n \cdot m$ *migration edges* between copies of elements in consecutive time-slices. That is, the set of migration edges is $E_m \triangleq \{\{v^{t-1}, v^t\} \mid v \in Q, t \in [m]\}$. We identify each edge $e \in E$ with a set containing two of its endpoints.

Finally, for an edge $e \in E$, we define its *cost* by

$$c(e) \triangleq \begin{cases} 1 & \text{if } e \in E_{\text{IN}}, \\ \gamma & \text{if } e \in E_m. \end{cases}$$

We extend the function $c(e)$ to all subsets of edges $E' \subseteq E$, i.e., $c(E') = \sum_{e \in E'} c(e)$.

Naming Convention. Sometimes we want to refer to a vertex from V without specifying its time-slice. In such case, we use star instead of time superscript, i.e., we use names such as u^* or v^* , to emphasize that we refer to vertices (members of V) and not elements (members of Q).

Edge Lengths. A solution to the DMLA problem induces the assignment of *lengths* to edges of E in the following way:

- the length of a request edge $\{a_t^t, b_t^t\}$ is set to $|\pi_t(a_t) - \pi_t(b_t)|$;
 - the length of a migration edge $\{v^{t-1}, v^t\}$ is set to $|\pi_{t-1}(v) - \pi_t(v)|$.
- In particular, the total length of all (migration) edges between two consecutive time-slices Q^{t-1} and Q^t is equal to $\text{fdist}(\pi_{t-1}, \pi_t)$.

A valid solution to DMLA cannot induce an arbitrary assignment of edge lengths. As the permutation π_t assigns distinct positions to elements, the pairwise distances (shortest path distances induced by lengths) between their corresponding vertices can be lower-bounded appropriately. We make this observation more concrete when we create a linear relaxation of the problem. This hints at a possible way of tackling the problem: we first find an assignment of lengths to edges, and we use them to compute a sequence of permutations.

Spiders. To create a linear relaxation of the DMLA problem, we introduce a helper notion. For a vertex $v^* \in V$ and set $U \subseteq V \setminus \{v^*\}$, a multi-set of edges from E is called a (v^*, U) -*spider* if it is a union of $|U|$ paths from v^* to each vertex from U . (If an edge belongs to k such paths, the spider contains k copies of such edge.) We use $\mathcal{H}(v^*, U)$ to denote the set of all possible (v^*, U) -spiders.

LP Relaxation. The LP relaxation is obtained by introducing *spreading constraints* [7] to every time-slice. Consider a DMLA instance $\mathcal{I} = (n, m, \text{IN})$, and let $G = (V, E, c)$ denote the corresponding time-expanded graph. Let $S'_j \triangleq \sum_{i=1}^j i$ and $S'_k \triangleq S'_{\lfloor k/2 \rfloor} + S'_{\lceil k/2 \rceil}$.

The LP relaxation introduces a variable z_e for every edge $e \in E$ and is formulated as follows.

$$\min \sum_{e \in E} c(e) \cdot z_e \tag{2a}$$

$$\text{s.t.} \quad \sum_{e \in H} z_e \geq S_{|A|} \quad \forall v \in Q, \forall A \subseteq Q \setminus v, \forall t \in \{0, \dots, m\}, \forall H \in \mathcal{H}(v^t, A^t), \tag{2b}$$

$$z_e \geq 0 \quad \forall e \in E. \tag{2c}$$

Before we argue that the formulation above is indeed a relaxation of the original DMLA problem, we first discuss the interpretation of spreading constraints (2b). For any two vertices $v^*, u^* \in V$, let $\text{dist}_z(v^*, u^*)$ denote their shortest-path distance in G induced by edge lengths $\{z_e\}_{e \in E}$. For a fixed time t , element $v \in Q$ and a set of elements $A \subseteq Q \setminus \{v\}$, constraints (2b) state that $\sum_{e \in H} z_e \geq S_{|A|}$ for every spider $H \in \mathcal{H}(v^t, A^t)$. That is, the total length of $|A|$ paths from v^t to all vertices of A^t has to be at least $S_{|A|}$. Constraints (2b) are thus equivalent to

$$\sum_{u \in A} \text{dist}_z(v^t, u^t) \geq S_{|A|}, \quad \forall v \in Q, \forall A \subseteq Q \setminus \{v\}, \forall t \in \{0, \dots, m\}. \quad (3)$$

► **Lemma 2.** *The minimization program above is a linear relaxation of the DMLA problem.*

Proof. Fix an instance \mathcal{I} of the DMLA problem specified by (n, m, IN) and the corresponding instance of the minimization LP above. Consider a solution to \mathcal{I} , that is, a sequence of permutations $\{\pi_t\}_{t=1}^m$. We have to show that there exists a solution to the LP whose cost is at most $\text{COST}(\mathcal{I}, \{\pi_t\}_{t=1}^m)$.

For the purpose of defining variables $\{z_e\}_{e \in E}$, we set $\pi_0 \triangleq \pi_1$. For an edge $e = \{u^{t_1}, v^{t_2}\} \in E$, we set the variable $z_e = |\pi_{t_1}(u) - \pi_{t_2}(v)|$. The cost of the LP solution is then

$$\begin{aligned} \sum_{e \in E} c(e) \cdot z_e &= \sum_{e \in E_m} \gamma \cdot z_e + \sum_{e \in E_{\text{IN}}} z_e \\ &= \sum_{t=1}^m \sum_{v \in Q} \gamma \cdot |\pi_t(v) - \pi_{t-1}(v)| + \sum_{t=1}^m |\pi_t(a_t) - \pi_t(b_t)| \\ &= \text{COST}(\mathcal{I}, \{\pi_t\}_{t=1}^m). \end{aligned}$$

It remains to show that the edge length variables satisfy the constraints; the only non-trivial one is (2b).

Fix such a constraint, given by element $v \in Q$, a subset $A \subseteq Q \setminus \{v\}$, time t , and a spider $H \in \mathcal{H}(v^t, A^t)$. We decompose H into $|A|$ paths: for an element $u \in A$, let H_u be the path from v^t to u^t in spider H . By a simple induction on path length, we have

$$\sum_{e \in H_u} z_e \geq |\pi_t(u) - \pi_t(v)| \quad (4)$$

for every element $u \in A$. By summing (4) over all elements from A , we obtain that

$$\sum_{e \in H} z_e = \sum_{u \in A} \sum_{e \in H_u} z_e \geq \sum_{u \in A} |\pi_t(u) - \pi_t(v)|.$$

We split the elements of A into two disjoint parts $A^- \triangleq \{u \in A \mid \pi_t(u) < \pi_t(v)\}$ and $A^+ \triangleq \{u \in A \mid \pi_t(u) > \pi_t(v)\}$. As π_t is bijective, $\sum_{u \in A^+} |\pi_t(u) - \pi_t(v)| \geq S'_{|A^+|}$ and $\sum_{u \in A^-} |\pi_t(u) - \pi_t(v)| \geq S'_{|A^-|} = S'_{|A| - |A^+|}$. Thus,

$$\sum_{e \in H} z_e \geq S'_{|A^+|} + S'_{|A| - |A^+|} \geq S'_{\lfloor |A|/2 \rfloor} + S'_{\lceil |A|/2 \rceil} = S_{|A|}.$$

The last inequality follows as the expression $S'_\ell + S'_{|A| - \ell}$ is minimized for $\ell = \lfloor |A|/2 \rfloor$. This shows that constraints (2b) hold and concludes the proof. ◀

4 Graph Decomposition

In this section, we present a poly-time deterministic graph decomposition procedure that is based on [7]. The input for the graph decomposition consists of an undirected graph with non-negative edge costs $G = (V, E, c)$ and non-negative edge lengths $\{z_e\}_{e \in E}$. A diameter parameter $d > 0$ specifies the “granularity” of the decomposition.

Distances and Diameters. For any two vertices $u, v \in V$, let $\text{dist}_z(u, v)$ denote the shortest-path distance in G induced by the edge lengths z . Furthermore, for a non-empty set $U \subseteq V$, let $\text{diam}_z(U) = \max_{u, v \in U} \text{dist}_z(u, v)$ be the *diameter* of U . If U contains two vertices that are not connected in G , then its diameter is infinite.

Decomposition Tree. Fix a real number $d > 0$. A d -decomposition tree of the graph $G = (V, E, c)$ is a triple $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}, \alpha)$ where $(V_{\mathcal{T}}, E_{\mathcal{T}})$ is a rooted binary tree where each internal node has two children, one marked *left*, and the other one *right*. The function $\alpha : V_{\mathcal{T}} \rightarrow 2^V \setminus \{\emptyset\}$ maps tree nodes to non-empty subsets of graph vertices V and satisfies the following conditions:

- $\alpha(w_r) = V$, where w_r is the root of \mathcal{T} ;
- $\alpha(w) = \alpha(w_L) \uplus \alpha(w_R)$ for an internal node $w \in V_{\mathcal{T}}$ and its two children w_L and w_R ;
- $\text{diam}_z(\alpha(w)) \geq d$ for every internal node $w \in V_{\mathcal{T}}$ and $\text{diam}_z(\alpha(w)) < d$ for every leaf $w \in V_{\mathcal{T}}$.

Note that the decomposition tree \mathcal{T} represents a laminar decomposition of the graph vertices V .

Cuts. For any subsets of graph nodes $U, U' \subseteq V$, let $E[U] \subseteq E$ be the subset of edges with both endpoints in U , and let $E[U, U'] \subseteq E$ be the subset of edges with one endpoint in U and the other one in U' . For an internal tree node $w \in V_{\mathcal{T}}$ with children w_L and w_R , we define $\text{cut}(w) \subseteq E$ as the set of edges between $\alpha(w_L)$ and $\alpha(w_R)$, i.e.,

$$\begin{aligned} \text{cut}(w) &\triangleq E[\alpha(w_L), \alpha(w_R)] \\ &= E[\alpha(w)] \setminus (E[\alpha(w_L)] \uplus E[\alpha(w_R)]). \end{aligned}$$

For a leaf $w \in V_{\mathcal{T}}$, we define $\text{cut}(w)$ to be the empty set.

► **Definition 3.** The cost of a decomposition tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}, \alpha)$ of a graph $G = (V, E, c)$ with non-negative edge lengths $\{z_e\}_{e \in E}$ is defined as

$$\text{COST}_{G,z}(\mathcal{T}) \triangleq \sum_{w \in V_{\mathcal{T}}} c(\text{cut}(w)) \cdot \text{diam}_z(\alpha(w)).$$

In the definition above, we assume that $0 \cdot \infty = 0$.

Cheap Decomposition Tree. The following theorem is implicitly proven (in a slightly different form) in [7]. For completeness, we present its proof in the appendix.

► **Theorem 4.** Fix a real $d > 0$ and a graph $G = (V, E, c)$ with non-negative edge lengths $\{z_e\}_{e \in E}$. It is possible to construct, in polynomial time, a d -decomposition tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}, \alpha)$ of G , such that

$$\text{COST}_{G,z}(\mathcal{T}) \leq \xi \cdot O(\log(f_{c,d} \cdot \xi) \cdot \log \log(f_{c,d} \cdot \xi)),$$

where $\xi = \sum_{e \in E} c(e) \cdot z_e$, $f_{c,d} = \max\{1, 1/(d \cdot c_{\min})\}$, and $c_{\min} = \min_{e \in E} c(e)$.

We emphasize that Theorem 4 does not depend on particular properties of graph G or the edge lengths $\{z_e\}_{e \in E}$. In particular, G does not have to be a time-expanded graph of a DMLA instance, and the theorem does not assume that lengths $\{z_e\}_{e \in E}$ satisfy the spreading constraints (2b). In [7], the stopping condition for the decomposition is when one reaches an independent set with respect to an auxiliary graph defined over the same vertex set. Conceptually, our stopping condition can be viewed as reaching an independent set with respect to an auxiliary *hypergraph* defined over the same vertex set. Indeed, the auxiliary hypergraph in a d -decomposition contains a hyperedge for every subset of vertices whose diameter is at least d .

5 Approximation Algorithm for DMLA

Algorithm definition. Consider an instance $\mathcal{I} = (n, m, \text{IN})$ of the DMLA problem. Our algorithm ALG first constructs a time-expanded graph $G = (V, E, c)$ for \mathcal{I} . Next, ALG solves the LP relaxation defined by (2a)–(2c), obtaining an optimal solution $\{z_e\}_{e \in E}$ to this LP. Then, it computes a $(1/4)$ -decomposition tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}, \alpha)$ of G using the routine guaranteed by Theorem 4.

Finally, ALG decodes \mathcal{T} into a sequence of permutations $\{\pi_t\}_{t=1}^m$. To this end, let w_1, w_2, \dots, w_ℓ denote the sequence of leaves of \mathcal{T} ordered by an in-order traversal of \mathcal{T} (that scans the left subtree before the right subtree). Note that the corresponding sequence of sets $\{\alpha(w_i)\}_{i=1}^\ell$ is a partition of V . For a fixed time t , we define a sequence of sets $\{B_i^t\}_{i=1}^\ell$, where $B_i^t \triangleq \alpha(w_i) \cap Q^t$. By Lemma 9 (cf. Section 6), every set B_i^t contains either one element of Q^t or is empty. Thus, the sequence $\{B_i^t\}_{i=1}^\ell$ induces a permutation of Q^t (and hence of Q). Let π_t denote this permutation.

Handling Arbitrary Input Lengths. In the next section, we prove the following bound.

► **Theorem 5.** *On an instance \mathcal{I} , ALG returns a feasible solution of cost $O(\log(n \cdot \text{OPT}(\mathcal{I})) \cdot \log \log(n \cdot \text{OPT}(\mathcal{I})) \cdot \text{OPT}(\mathcal{I}))$, where $\text{OPT}(\mathcal{I})$ denotes the cost of the optimal solution on \mathcal{I} .*

We are interested in obtaining a smaller asymptotic approximation ratio even if $\text{OPT}(\mathcal{I})$ is super-polynomial in n .

► **Theorem 6.** *There is a poly-time deterministic approximation algorithm for DMLA that achieves an approximation ratio of $O(\log n \cdot \log \log n)$.*

Proof. By Claim 1, $\text{OPT}(\mathcal{I})$ is super-polynomial in n only if m is. In this case, we modify ALG as follows. Split the input sequence into L phases $\mathcal{I}_1, \dots, \mathcal{I}_L$, each consisting of $m' = n^2$ requests, with the last phase possibly being shorter. Apply ALG separately to each phase, and return the concatenation of the permutation sequences output by ALG for each phase.

As OPT needs to pay 1 for each request, $\text{OPT}(\mathcal{I}) \geq (L - 1) \cdot m'$. On the other hand, by Claim 1, $\text{OPT}(\mathcal{I}_\ell) \leq n \cdot m' = n^3$ for every phase \mathcal{I}_ℓ . The rearrangement cost incurred by the transition from a permutation ending a phase to the permutation beginning the next phase at most n^2 . The cost of the whole solution is then

$$\begin{aligned} \text{ALG}(\mathcal{I}) &= (L - 1) \cdot n^2 + \sum_{\ell=1}^L \text{ALG}(\mathcal{I}_\ell) \\ &\leq \text{OPT}(\mathcal{I}) + \sum_{\ell=1}^L O(\log(n \cdot \text{OPT}(\mathcal{I}_\ell)) \cdot \log \log(n \cdot \text{OPT}(\mathcal{I}_\ell))) \cdot \text{OPT}(\mathcal{I}_\ell) \\ &= \text{OPT}(\mathcal{I}) + O(\log n \cdot \log \log n) \cdot \sum_{\ell=1}^L \text{OPT}(\mathcal{I}_\ell) \\ &= O(\log n \cdot \log \log n) \cdot \text{OPT}(\mathcal{I}), \end{aligned}$$

where in the inequality above we used Theorem 5 for upper-bounding $\text{ALG}(\mathcal{I}_\ell)$ for each ℓ . ◀

Polynomial Runtime. ALG can be implemented in time polynomial in n and m . This follows trivially except for its two building blocks: solving the LP and computing the decomposition tree. The latter runs in polynomial time by Theorem 4.

For the former (solving the LP), we note that although the linear programming formulation contains exponentially many constraints, it can be solved in polynomial time by the Ellipsoid method [12] with a separation oracle that returns a violating constraint if one exists. To this end, following [7], given edge lengths $\{z_e\}_{e \in E}$, the oracle first computes all-pairs shortest paths, i.e., the value of function dist_z . To find a violated constraint among constraints (3), it suffices to check, for every time t and every element $v \in Q$, whether there exists a set $A \subseteq Q \setminus \{v\}$, such that $\sum_{u \in A} \text{dist}_z(v^t, u^t) \geq S_{|A|}$. This becomes simple once we fix the cardinality k of A : if inequality is violated, it is violated by the set A , such that A^k contains k elements of $Q^t \setminus \{v^t\}$ that are closest to v^t (with respect to dist_z).

6 Approximation Ratio

In this section, we prove the approximation ratio stated in Theorem 5. Fix a DMLA instance $\mathcal{I} = (n, m, \text{IN})$. Let $G = (V, E, c)$ be the time-expanded graph, $\{z_e\}_{e \in E}$ be the edge lengths returned by the LP relaxation, and $\mathcal{T} = (V_T, E_T, \alpha)$ be the $(1/4)$ -decomposition tree of G computed according to Theorem 4.

6.1 Relating OPT to the Decomposition Tree

► **Lemma 7.** *It holds that $\text{COST}_{G,z}(\mathcal{T}) \leq \text{OPT}(\mathcal{I}) \cdot O(\log(n \cdot \text{OPT}(\mathcal{I})) \cdot \log \log(n \cdot \text{OPT}(\mathcal{I})))$.*

Proof. Let $c_{\min} = \min_{e \in E} \{c(e)\}$. Note that $c_{\min} = \min\{1, \gamma\}$ by the definition of graph G . As we assumed (cf. Section 2) that $\gamma \geq 1/n$, it holds that $c_{\min} \geq 1/n$. Let $\xi = \sum_{e \in E} c(e) \cdot z_e$. By Theorem 4, the computed $(1/4)$ -decomposition tree satisfies

$$\text{COST}_{G,z}(\mathcal{T}) \leq \xi \cdot O(\log(4n \cdot \xi) \cdot \log \log(4n \cdot \xi)).$$

As ξ is the optimal solution to the fractional relaxation of the problem (cf. Lemma 2), $\xi \leq \text{OPT}(\mathcal{I})$, which concludes the proof. ◀

6.2 Graph Cost and its Relation to the Decomposition Tree

In the previous section, we related $\text{COST}_{G,z}(\mathcal{T})$ to $\text{OPT}(\mathcal{I})$, and thus it remains to relate $\text{ALG}(\mathcal{I})$ to $\text{COST}_{G,z}(\mathcal{T})$. As we show later, $\text{ALG}(\mathcal{I})$ can be naturally expressed as a sum of costs of particular edges in G taken with some multiplicity. On the other hand, $\text{COST}_{G,z}(\mathcal{T})$ is defined as $\sum_{w \in V_T} c(\text{cut}(w)) \cdot \text{diam}_z(\alpha(w))$, i.e., in terms of edge lengths. To facilitate a combinatorial comparison between $\text{ALG}(\mathcal{I})$ and $\text{COST}_{G,z}(\mathcal{T})$, we first provide an alternative lower bound on $\text{COST}_{G,z}(\mathcal{T})$, called *graph cost*, which is easier to relate to $\text{ALG}(\mathcal{I})$. To this end, we start with a few helper notions.

Least Common Ancestor. We define the function $\text{lca} : 2^V \setminus \{\emptyset\} \rightarrow V_T$ as follows. Fix a non-empty set of vertices $U \subseteq V$. The set of tree nodes w such that $\alpha(w) \supseteq U$ forms a path in \mathcal{T} starting at the root. Let $\text{lca}(U)$ be the last node on this path (furthest from the root).

We drop the set notation and use $\text{lca}(u_1, u_2, \dots, u_\ell)$ instead of $\text{lca}(\{u_1, u_2, \dots, u_\ell\})$. Note that for every internal tree node w and every edge $\{u, v\} \in \text{cut}(w)$, it holds that $\text{lca}(u, v) = w$.

15:10 An Improved Approximation Algorithm for Dynamic Minimum Linear Arrangement

Width. For a non-empty subset of vertices $U \subseteq V$, we define

$$\text{width}(U) \triangleq \max_{0 \leq t \leq m} |U \cap Q^t| - 1.$$

Graph Cost. The *cost* of a graph $G = (V, E, c)$ with respect to its decomposition tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}, \alpha)$ is defined as

$$\text{COST}_{\mathcal{T}}^*(G) \triangleq \sum_{e \in E} c(e) \cdot \text{width}(\text{lca}(e)).$$

Relating Graph Cost to Decomposition Tree Cost. We show that $\text{COST}_{\mathcal{T}}^*(G) \leq 4 \cdot \text{COST}_{G,z}(\mathcal{T})$. To this end, we use spreading properties of $\{z_e\}_{e \in E}$ to relate the width of a set to its diameter.

► **Lemma 8.** *For a non-empty set $U \subseteq V$, it holds that $\text{width}(U) \leq 4 \cdot \text{diam}_z(U)$.*

Proof. Let $t = \arg \max_{s \in \{0, \dots, m\}} |U \cap Q^s|$, i.e., $\text{width}(U) = |U \cap Q^t| - 1$. For succinctness, let $Y = U \cap Q^t$. Below, we prove that

$$\text{diam}_z(Y) \geq (|Y| - 1)/4. \quad (5)$$

This will imply the lemma as $\text{width}(U) = |Y| - 1 \leq 4 \cdot \text{diam}_z(Y) \leq 4 \cdot \text{diam}_z(U)$.

If $|Y| = 1$, then $\text{diam}_z(Y) = 0$, and (5) follows trivially. Thus, in the following, we assume that $|Y| > 1$. Let v^t be an arbitrary vertex from Y . As $\{z_e\}_{e \in E}$ satisfies LP constraints and thus also (3), we obtain

$$\sum_{u^t \in Y \setminus \{v^t\}} \text{dist}_z(v^t, u^t) \geq S_{|Y|-1}. \quad (6)$$

A simple calculation shows that $S_k \geq k^2/4 + k/2$ for every $k \geq 0$ (the relation is tight for even k). Thus, $S_k/k = k/4 + 1/2 > k/4$ for every $k \geq 1$. Applying the averaging argument to (6), we obtain that there exists $u^t \in Y \setminus \{v^t\}$, such that

$$\text{dist}_z(v^t, u^t) \geq \frac{S_{|Y|-1}}{|Y|-1} > \frac{|Y|-1}{4}.$$

As $\text{diam}_z(Y) \geq \text{dist}_z(v^t, u^t)$, (5) follows. ◀

► **Lemma 9.** *Fix a leaf $w \in \mathcal{T}$. Then, $\text{width}(\alpha(w)) = 0$, and consequently, $|\alpha(w) \cap Q^t| \leq 1$ for every time t .*

Proof. As \mathcal{T} is a $(1/4)$ -decomposition tree, $\text{diam}_z(\alpha(w)) < 1/4$. Thus, $\text{width}(\alpha(w)) < 1$ by Lemma 8. The first part of the lemma follows as $\text{width}(\alpha(w))$ is an integer. The second part follows as $|\alpha(w) \cap Q^t| \leq \text{width}(\alpha(w)) + 1$ by the definition of *width*. ◀

► **Lemma 10.** *It holds that $\text{COST}_{\mathcal{T}}^*(G) \leq 4 \cdot \text{COST}_{G,z}(\mathcal{T})$.*

Proof. Fix an edge $e \in E$. We first claim that

$$\text{width}(\alpha(\text{lca}(e))) = \sum_{w \in V_{\mathcal{T}} : e \in \text{cut}(w)} \text{width}(\alpha(w)). \quad (7)$$

Indeed, if $\text{lca}(e)$ is a leaf of \mathcal{T} , then by Lemma 9, the left-hand side is 0 and the sum on the right-hand side is empty. If, however, $\text{lca}(e)$ is an internal node of \mathcal{T} , then $e \in \text{cut}(\text{lca}(e))$, and thus the sum on the right-hand side contains only one element $w = \text{lca}(e)$, and the claim follows.

Now, by the definition of $\text{COST}_{\mathcal{T}}^*(G)$,

$$\begin{aligned}
\text{COST}_{\mathcal{T}}^*(G) &= \sum_{e \in E} c(e) \cdot \sum_{w \in V_{\mathcal{T}} : e \in \text{cut}(w)} \text{width}(\alpha(w)) && \text{(by (7))} \\
&= \sum_{w \in V_{\mathcal{T}}} \sum_{e \in \text{cut}(w)} c(e) \cdot \text{width}(\alpha(w)) \\
&= \sum_{w \in V_{\mathcal{T}}} c(\text{cut}(w)) \cdot \text{width}(\alpha(w)) \\
&\leq 4 \cdot \sum_{w \in V_{\mathcal{T}}} c(\text{cut}(w)) \cdot \text{diam}_z(\alpha(w)) && \text{(by Lemma 8)} \\
&= 4 \cdot \text{COST}_{G,z}(\mathcal{T}). && \blacktriangleleft
\end{aligned}$$

6.3 Relating ALG to the Graph Cost

Now we may relate $\text{ALG}(\mathcal{I})$ to the graph cost $\text{COST}_{\mathcal{T}}^*(G) = \sum_{e \in E} c(e) \cdot \text{width}(\text{lca}(e))$. Our charging scheme preserves time locality: we relate the request and the rearrangement costs in step t to the graph cost pertaining to edges of G corresponding to time t .

► **Lemma 11.** *For every time t , it holds that $|\pi_t(a_t) - \pi_t(b_t)| \leq \text{width}(\alpha(\text{lca}(a_t^t, b_t^t)))$.*

Proof. It is convenient to look at the permutation π_t output by ALG as an ordered sequence of vertices from Q^t (cf. Section 5). Recall that this sequence is obtained by an in-order traversal of \mathcal{T} restricted to vertices from Q^t . In particular, all vertices from $\alpha(\text{lca}(a_t^t, b_t^t)) \cap Q^t$ form a contiguous part of permutation π_t and this part contains both a_t^t and b_t^t . Therefore, $|\pi_t(a_t) - \pi_t(b_t)| \leq |\alpha(\text{lca}(a_t^t, b_t^t)) \cap Q^t| - 1 \leq \text{width}(\alpha(\text{lca}(a_t^t, b_t^t)))$. ◀

► **Lemma 12.** *Fix a time t and let $\{u, v\}$ be a discordant pair of elements from Q with respect to permutations π_{t-1} and π_t . Then, either $v^t \in \alpha(\text{lca}(u^{t-1}, u^t))$ or $u^t \in \alpha(\text{lca}(v^{t-1}, v^t))$ (or both).*

Proof. Let $U = \{u^{t-1}, u^t, v^{t-1}, v^t\}$ and let $w = \text{lca}(U)$. As $\alpha(w)$ contains both u^t and v^t , Lemma 9 implies that w is an internal node of \mathcal{T} . Let w_L and w_R be the children of w in \mathcal{T} . By the definition of lca , $\alpha(w_L) \cap U \neq \emptyset$ and $\alpha(w_R) \cap U \neq \emptyset$.

It is not possible that $\{u^{t-1}, u^t\} \subseteq \alpha(w_L)$ and $\{v^{t-1}, v^t\} \subseteq \alpha(w_R)$: in such case, $\{u, v\}$ would not be a discordant pair, i.e., u would be before v in both permutations π_{t-1} and π_t . For the same reason, it is not possible that $\{u^{t-1}, u^t\} \subseteq \alpha(w_R)$ and $\{v^{t-1}, v^t\} \subseteq \alpha(w_L)$.

This means that either $\{u^{t-1}, u^t\} \in \text{cut}(w)$ or $\{v^{t-1}, v^t\} \in \text{cut}(w)$ (or both). In the former case $\text{lca}(u^{t-1}, u^t) = w$ while in the latter $\text{lca}(v^{t-1}, v^t) = w$, which concludes the proof. ◀

► **Lemma 13.** *For every time t , it holds that $\text{tdist}(\pi_{t-1}, \pi_t) \leq \sum_{v \in Q} \text{width}(\alpha(\text{lca}(v^{t-1}, v^t)))$.*

Proof. We create an injective mapping M from all $\text{tdist}(\pi_{t-1}, \pi_t)$ discordant pairs (with respect to permutations π_{t-1} and π_t) to pairs in $V_{\mathcal{T}} \times V$. A discordant pair $\{u, v\}$ is mapped:

- to the pair $(\text{lca}(u^{t-1}, u^t), v^t)$ if $v^t \in \alpha(\text{lca}(u^{t-1}, u^t))$, or
- to the pair $(\text{lca}(v^{t-1}, v^t), u^t)$ if $u^t \in \alpha(\text{lca}(v^{t-1}, v^t))$.

By Lemma 12, at least one of the conditions above must hold. If both hold, then we map the discordant pair arbitrarily to one of the pairs above.

Because the domain of M contains all discordant pairs, its cardinality equals $\text{tdist}(\pi_{t-1}, \pi_t)$. Moreover, the range of M is contained in the set of the following pairs

$$\bigcup_{v \in Q} \{(\text{lca}(v^{t-1}, v^t), y^t) \mid y^t \in \alpha(\text{lca}(v^{t-1}, v^t)) \cap (Q^t \setminus \{v^t\})\}.$$

15:12 An Improved Approximation Algorithm for Dynamic Minimum Linear Arrangement

Thus, the cardinality of the range of M is at most $\sum_{v \in Q} (|\alpha(\text{lca}(v^{t-1}, v^t)) \cap Q^t| - 1)$. As M is injective, its range is not smaller than its domain, i.e.,

$$\begin{aligned} \text{tdist}(\pi_{t-1}, \pi_t) &\leq \sum_{v \in Q} |\alpha(\text{lca}(v^{t-1}, v^t)) \cap Q^t| - 1 \\ &\leq \sum_{v \in Q} \text{width}(\alpha(\text{lca}(v^{t-1}, v^t))). \end{aligned} \quad \blacktriangleleft$$

► **Lemma 14.** *It holds that $\text{ALG}(\mathcal{I}) \leq 2 \cdot \text{COST}_{\mathcal{T}}^*(G)$.*

Proof. Let $\{\pi_t\}_{t=1}^m$ be the output of ALG on instance \mathcal{I} . The total rearrangement cost of ALG is

$$\begin{aligned} \sum_{t=1}^m \gamma \cdot \text{fdist}(\pi_{t-1}, \pi_t) &= 2 \cdot \sum_{t=1}^m \gamma \cdot \text{tdist}(\pi_{t-1}, \pi_t) && \text{(by (1))} \\ &\leq 2 \cdot \sum_{t=1}^m \sum_{v \in Q} \gamma \cdot \text{width}(\alpha(\text{lca}(v^{t-1}, v^t))) && \text{(by Lemma 13)} \\ &= 2 \cdot \sum_{e \in E_m} c(e) \cdot \text{width}(\alpha(\text{lca}(e))). && \text{(by the definition of } E_m) \end{aligned}$$

Moreover, its total request cost is

$$\begin{aligned} \sum_{t=1}^m |\pi_t(a_t) - \pi_t(b_t)| &\leq \sum_{t=1}^m 1 \cdot \text{width}(\alpha(\text{lca}(a_t^t, b_t^t))) && \text{(by Lemma 11)} \\ &= \sum_{e \in E_{\text{IN}}} c(e) \cdot \text{width}(\alpha(\text{lca}(e))). && \text{(by the definition of } E_{\text{IN}}) \end{aligned}$$

Summing up, we obtain that $\text{ALG}(\mathcal{I}) \leq 2 \cdot \sum_{e \in E} c(e) \cdot \text{width}(\alpha(\text{lca}(e))) = 2 \cdot \text{COST}_{\mathcal{T}}^*(G)$. ◀

6.4 Calculating Approximation Ratio

We are now ready to prove our main result, restated for convenience below.

► **Theorem 5.** *On an instance \mathcal{I} , ALG returns a feasible solution of cost $O(\log(n \cdot \text{OPT}(\mathcal{I})) \cdot \log \log(n \cdot \text{OPT}(\mathcal{I}))) \cdot \text{OPT}(\mathcal{I})$, where $\text{OPT}(\mathcal{I})$ denotes the cost of the optimal solution on \mathcal{I} .*

Proof. Fix an instance \mathcal{I} , corresponding graph G , output $\{z_e\}_{e \in E}$ of the LP, and the decomposition tree \mathcal{T} of G . Then,

$$\begin{aligned} \text{ALG}(\mathcal{I}) &\leq 4 \cdot \text{COST}_{\mathcal{T}}^*(G) && \text{(by Lemma 14)} \\ &\leq 8 \cdot \text{COST}_{G, z}^*(G) && \text{(by Lemma 10)} \\ &\leq 8 \cdot \text{OPT}(\mathcal{I}) \cdot O(\log(n \cdot \text{OPT}(\mathcal{I})) \cdot \log \log(n \cdot \text{OPT}(\mathcal{I}))). && \text{(by Lemma 7)} \quad \blacktriangleleft \end{aligned}$$

As stated in Theorem 6, the algorithm can be easily transformed into an $O(\log n \cdot \log \log n)$ -approximation. We also note that the constants hidden in O -notation do not depend on γ .

7 Final Remarks

In this paper, we present an $O(\log n \cdot \log \log n)$ -approximation algorithm for the dynamic minimum linear arrangement (DMLA) problem, improving over $O(\log^2 n)$ bound by Olver et al. [22].

We note that in the variant considered by Olver et al. [22], the initial permutation π_0 is fixed and is the part of the input instance. A small modification of their approach yields the same approximation ratio for the case where the input does not specify the initial permutation (as in our solution).

Conversion in the other direction is trivial, albeit it comes at a certain cost. Let π_1, \dots, π_m denote the output of our algorithm without an initial permutation. Simply prepend the given initial permutation π_0 as the initial one. The extra cost of the rearrangement from permutation π_0 to π_1 is at most $O(n^2)$. This additive cost does not influence the asymptotic approximation ratio if $m = \Omega(n^2)$. We leave handling shorter instances more efficiently as further work.

References

- 1 Konstantin Andreev and Harald Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006. doi:10.1007/s00224-006-1350-7.
- 2 Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM*, 56(2):5, 2009. doi:10.1145/1502793.1502794.
- 3 Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic balanced graph partitioning. *SIAM Journal on Discrete Mathematics*, 34(3):1791–1812, 2020. doi:10.1137/17M1158513.
- 4 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proc. 37th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 184–193, 1996. doi:10.1109/SFCS.1996.548477.
- 5 Marcin Bienkowski, Martin Böhm, Martin Koutecký, Thomas Rothvoß, Jirí Sgall, and Pavel Veselý. Improved analysis of online balanced clustering. In *Proc. 19th Workshop on Approximation and Online Algorithms (WAOA)*, pages 224–233, 2021. doi:10.1007/978-3-030-92702-8_14.
- 6 Persi Diaconis and R. L. Graham. Spearman’s footrule as a measure of disarray. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(2):262–268, 1977. doi:10.1111/j.2517-6161.1977.tb01624.x.
- 7 Guy Even, Joseph Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *Journal of the ACM*, 47(4):585–616, 2000. doi:10.1145/347476.347478.
- 8 Uriel Feige and James R. Lee. An improved approximation ratio for the minimum linear arrangement problem. *Information Processing Letters*, 101(1):26–29, 2007. doi:10.1016/j.ipl.2006.07.009.
- 9 Andreas Emil Feldmann and Luca Foschini. Balanced partitions of trees and applications. *Algorithmica*, 71(2):354–376, 2015. doi:10.1007/s00453-013-9802-3.
- 10 Tobias Forner, Harald Räcke, and Stefan Schmid. Online balanced repartitioning of dynamic communication patterns in polynomial time. In *2nd Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 40–54, 2021. doi:10.1137/1.9781611976489.4.
- 11 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 12 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

- 13 Shouzhen Gu, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Yiran Chen, and Jingtong Hu. Area and performance co-optimization for domain wall memory in application-specific embedded systems. In *Proc. 52nd Annual Design Automation Conference (DAC)*, pages 20:1–20:6, 2015. doi:10.1145/2744769.2744800.
- 14 Mark D. Hansen. Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems (extended abstract). In *Proc. 30th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 604–609, 1989. doi:10.1109/SFCS.1989.63542.
- 15 Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. Tight bounds for online graph partitioning. In *Proc. 32nd ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 2799–2818, 2021. doi:10.1137/1.9781611976465.166.
- 16 Monika Henzinger, Stefan Neumann, and Stefan Schmid. Efficient distributed workload (re-)embedding. In *Proc. SIGMETRICS/Performance Joint Int. Conf. on Measurement and Modeling of Computer Systems*, pages 43–44, 2019. doi:10.1145/3309697.3331503.
- 17 Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009. doi:10.1016/j.cosrev.2009.04.002.
- 18 Robert Krauthgamer. Minimum bisection. In *Encyclopedia of Algorithms*, pages 1294–1297. Springer, 2016. doi:10.1007/978-1-4939-2864-4_231.
- 19 Robert Krauthgamer, Joseph Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proc. 20th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 942–949, 2009. doi:10.1137/1.9781611973068.102.
- 20 Ravi Kumar and Sergei Vassilvitskii. Generalized distances between rankings. In *Proc. 19th Int. World Wide Web Conference (WWW)*, pages 571–580, 2010. doi:10.1145/1772690.1772749.
- 21 Manor Mendel. Metrical task systems. In *Encyclopedia of Algorithms*, pages 1279–1282. Springer, 2016. doi:10.1007/978-1-4939-2864-4_229.
- 22 Neil Olver, Kirk Pruhs, Kevin Schewior, René Sitters, and Leen Stougie. The itinerant list update problem. In *Proc. 16th Workshop on Approximation and Online Algorithms (WAOA)*, pages 310–326, 2018. doi:10.1007/978-3-030-04693-4_19.
- 23 Maciej Pacut, Mahmoud Parham, and Stefan Schmid. Brief announcement: Deterministic lower bound for dynamic balanced graph partitioning. In *Proc. 39th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 461–463, 2020. doi:10.1145/3382734.3405696.
- 24 Maciej Pacut, Mahmoud Parham, and Stefan Schmid. Optimal online balanced graph partitioning. In *Proc. 40th IEEE Int. Conf. on Computer Communications (INFOCOM)*, pages 1–9, 2021. doi:10.1109/INFOCOM42981.2021.9488824.
- 25 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 40th ACM Symp. on Theory of Computing (STOC)*, pages 255–264, 2008. doi:10.1145/1374376.1374415.
- 26 Harald Räcke, Stefan Schmid, and Ruslan Zabrodin. Approximate dynamic balanced graph partitioning. In *Proc. 34th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 401–409. ACM, 2022. doi:10.1145/3490148.3538563.
- 27 Rajmohan Rajaraman and Omer Wasim. Improved bounds for online balanced graph repartitioning. In *Proc. 30th European Symp. on Algorithms (ESA)*, pages 83:1–83:15, 2022. doi:10.4230/LIPIcs.ESA.2022.83.
- 28 Satish Rao and Andréa W. Richa. New approximation techniques for some linear ordering problems. *SIAM Journal on Computing*, 34(2):388–404, 2004. doi:10.1137/S0097539702413197.

A Constructing Decomposition Tree

In this section, we show how to construct a d -decomposition tree satisfying the guarantees of Theorem 4. In the description below, we fix a real number $d > 0$, a graph $G = (V, E, c)$ with non-negative edge costs c , and a set of edge lengths $\{z_e\}_{e \in E}$. Recall that for all subsets of vertices $U, U' \subseteq V$, we defined $E[U] \triangleq E \cap (U \times U)$ and $E[U, U'] \triangleq E \cap (U \times U' \cup U' \times U)$. Furthermore, let $G[U]$ be the graph G restricted to set U . Note that $G[U]$ might be disconnected even if G is connected. Let

$$\text{vol } U \triangleq \sum_{e \in E[U]} c(e) \cdot z_e$$

be the *volume* of U .

In Appendix A.1, we argue that for an arbitrary subset $U \subseteq V$ satisfying $\text{diam}_z(U) \geq d$, we may efficiently partition U into two parts, and relate the cost of the cut between these two parts, the diameter of U , and the volume of U (cf. Lemma 17).

The d -decomposition tree \mathcal{T} of G is simply a binary tree, constructed by the iterative application of set partitioning procedure: we start with the whole vertex set V and terminate when the diameter of the considered set of vertices is less than d . In Appendix A.2 and Appendix A.3, we show that the resulting tree \mathcal{T} satisfies the properties of Theorem 4.

We present our construction and cost bound assuming that $\min_{e \in E} c(e) \geq 1/d$. Later, we show how to get rid of this assumption by a simple scaling.

A.1 Partitioning a Vertex Set

We start with two technical lemmas.

► **Lemma 15** (Lemma 5 of [7]). *Let $f : [r_0, r_1] \rightarrow \mathbb{R}$ be a nonnegative monotone increasing function that is differentiable almost everywhere. If the derivative f' is continuous almost everywhere, then there exists an $r \in (r_0, r_1)$ such that $f'(r)$ is defined and satisfies*

$$f'(r) \leq \frac{f(r)}{r_1 - r_0} \cdot \ln\left(\frac{e \cdot f(r_1)}{f(r)}\right) \cdot \ln\ln\left(\frac{e \cdot f(r_1)}{f(r_0)}\right).$$

► **Lemma 16.** *Fix $0 < x \leq y$. Then,*

$$\ln\left(\frac{e \cdot (x + y)}{2 \cdot x}\right) \leq \frac{1}{\ln 2} \cdot \ln\left(\frac{x + y}{x}\right).$$

Proof. Since $y \geq x$, we have $\ln((x + y)/x) \geq \ln 2$. Therefore,

$$\begin{aligned} \ln\left(\frac{e \cdot (x + y)}{2 \cdot x}\right) &= \ln\left(\frac{x + y}{x}\right) + \left(\frac{1}{\ln 2} - 1\right) \cdot \ln 2 \\ &\leq \ln\left(\frac{x + y}{x}\right) + \left(\frac{1}{\ln 2} - 1\right) \cdot \ln\left(\frac{x + y}{x}\right) \\ &= \frac{1}{\ln 2} \cdot \ln\left(\frac{x + y}{x}\right). \end{aligned} \quad \blacktriangleleft$$

► **Lemma 17.** *Fix a real $d > 0$. Fix a graph $G = (V, E, c)$ with edge costs c satisfying $c(e) \geq 1/d$ for every $e \in E$. Let $\{z_e\}_{e \in E}$ be the (non-negative) lengths of edges. For a subset of vertices $U \subseteq V$ satisfying $\text{diam}_z(U) \geq d$, it is possible to partition U , in polynomial time, into two disjoint non-empty sets U_L and U_R , satisfying the following conditions.*

15:16 An Improved Approximation Algorithm for Dynamic Minimum Linear Arrangement

- If $G[U]$ is disconnected, then $c(E[U_L, U_R]) = 0$ and $\text{vol } U_L + \text{vol } U_R = \text{vol } U$.
- If $G[U]$ is connected, then

$$c(E[U_L, U_R]) \cdot \text{diam}_z(U) \leq g \cdot \beta \cdot \ln\left(\frac{\text{vol } U}{\beta}\right) \cdot \ln \ln(\text{vol } V),$$

where g is a universal constant, and β satisfies the following constraints:

$$\begin{aligned} \beta &\geq \max\{\text{vol } U_L, 1/4\}, \\ \text{vol } U - \beta &\geq \max\{\text{vol } U_R, 1/4\}. \end{aligned}$$

Proof. If $G[U]$ is disconnected, then we simply take U_L to be an arbitrary connected component of U and set $U_R = U \setminus U_L$. The lemma follows immediately as $c(E[U_L, U_R]) = 0$ and $\text{vol } U_L + \text{vol } U_R = \text{vol } U$. Thus in the following, we assume that $G[U]$ is connected.

We first show *the existence* of U_L and U_R satisfying the constraints above, and later we argue how to find them using a polynomial-time procedure.

For any two vertices $a, b \in U$, we define $\text{dist}_z^U(a, b)$ as the shortest-path distance between a and b that uses edges only from $G[U]$; we call it U -distance. Clearly, $\text{dist}_z^U(a, b) \geq \text{dist}_z(a, b)$. Let $u, u' \in U$ be the vertices satisfying $\text{dist}_z(u, u') = \text{diam}_z(U)$, and let

$$\Delta \triangleq \text{dist}_z^U(u, u')$$

be their U -distance. Then, $\Delta \geq \text{dist}_z(u, u') = \text{diam}_z(U) \geq d$. From this point on, we focus on the graph $G[U]$ only and use U -distances exclusively.

For a vertex $a \in U$ and a real $r \geq 0$, let

$$B_a(r) \triangleq \{v \in U \mid \text{dist}_z^U(a, v) < r\}$$

be the set of vertices in U whose U -distance to a is *strictly smaller* than r , i.e., contained in a ball centered at a of radius r . For an edge $e \in E[U]$, we define its (a, r) -adjusted length as

$$z_e^a(r) \triangleq \begin{cases} z_e & \text{if } e \in B_a(r) \times B_a(r), \\ r - \text{dist}_z^U(a, v_1) & \text{if } e = (v_1, v_2) \in B_a(r) \times (U \setminus B_a(r)), \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, the (a, r) -adjusted length of an edge e is the length of e “contained” (entirely or partially) in the ball of radius r centered at a .

Next, we introduce the function $\text{vol}_a^* : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ as

$$\begin{aligned} \text{vol}_a^*(r) &\triangleq \sum_{e \in E[U]} c(e) \cdot z_e^a(r) \\ &= \text{vol}(B_a(r)) + \sum_{e=(v_1, v_2) \in E \cap (B_a(r) \times (U \setminus B_a(r)))} c(e) \cdot (r - \text{dist}_z^U(a, v_1)). \end{aligned}$$

As $\Delta = \text{dist}_z^U(u, u')$, the balls of radii $\Delta/2$ centered at u and u' are disjoint. In particular, for each edge $e \in E[U]$, it holds that $z_e^u(\Delta/2) + z_e^{u'}(\Delta/2) \leq z_e$. This implies that $\text{vol}_u^*(\Delta/2) + \text{vol}_{u'}^*(\Delta/2) \leq \text{vol } U$. Without loss of generality, we may assume that $\text{vol}_u^*(\Delta/2) \leq \text{vol } U/2$; otherwise we may swap the roles of u and u' .

As $G[U]$ is connected, $\text{vol}_u^*(r)$ is monotonically increasing in the interval $[0, \Delta]$. Moreover, except finitely many points it holds that

$$\frac{\partial \text{vol}_u^*(r)}{\partial r} = c(B_u(r) \times (U \setminus B_u(r))).$$

That is, $\text{vol}_u^*(r)$ satisfies the conditions of Lemma 15 as the function of r in the interval $[0, \Delta]$, and thus also in the interval $[\Delta/4, \Delta/2]$. By Lemma 15, there exists $r^* \in (\Delta/4, \Delta/2)$, such that

$$c(E[B_u(r^*), U \setminus B_u(r^*)]) \leq \frac{\text{vol}_u^*(r^*)}{\Delta/4} \cdot \ln\left(\frac{e \cdot \text{vol}_u^*(\Delta/2)}{\text{vol}_u^*(r^*)}\right) \cdot \ln \ln\left(\frac{e \cdot \text{vol}_u^*(\Delta/2)}{\text{vol}_u^*(\Delta/4)}\right). \quad (8)$$

Recall that $G[U]$ is connected and contains a path between u and u' of length $\Delta \geq d$. As $c(e) \geq 1/d$ for every edge e , we have $\text{vol } U \geq \Delta \cdot (1/d) \geq 1$. Similarly, $\text{vol}_u^*(\Delta/4) \geq (\Delta/4) \cdot (1/d) \geq 1/4$. We set $U_L = B_u(r^*)$ and $U_R = U \setminus U_L$. Furthermore, we set $\beta = \text{vol}_u^*(r^*)$. By plugging these values into (8), and recalling that $\text{vol}_u^*(\Delta/2) \leq \text{vol } U/2$, we obtain

$$\begin{aligned} c(E[U_L, U_R]) \cdot \Delta &\leq 4 \cdot \beta \cdot \ln\left(\frac{e \cdot \text{vol } U}{2 \cdot \beta}\right) \cdot \ln \ln(2e \cdot \text{vol } U) \\ &\leq \frac{4}{\ln 2} \cdot \beta \cdot \ln\left(\frac{\text{vol } U}{\beta}\right) \cdot \ln \ln(2e \cdot \text{vol } U). \quad (\text{by Lemma 16 and } U \subseteq V) \end{aligned}$$

It remains to argue that β satisfies the constraints of the lemma.

- By the definition of vol_u^* , we have $\beta = \text{vol}_u^*(r^*) \geq \text{vol}(B_u(r^*)) = \text{vol } U_L$.
- By the monotonicity of vol_u^* , we have $\beta = \text{vol}_u^*(r^*) \geq \text{vol}_u^*(\Delta/4) \geq 1/4$.
- By the monotonicity of vol_u^* , we have $\beta = \text{vol}_u^*(r^*) \leq \text{vol}_u^*(\Delta/2) \leq \text{vol } U/2$, and thus $\text{vol } U - \beta \geq \text{vol } U/2 \geq 1/2 > 1/4$.
- Observe that $z_e^u(r^*) = 0$ for every edge $e \in E[U_R]$. (This follows as both endpoints of e are from U_R , and thus their distance to u is at least r^* .) In effect, $\text{vol}_u^*(r^*) + \text{vol } U_R \leq \text{vol } U$, or equivalently $\text{vol } U - \beta \geq \text{vol } U_R$.

The argument above shows the *existence* of r^* , such that setting $\beta = \text{vol}_u^*(r)$ satisfies the constraints of the lemma. To make the argument constructive and efficient, we note that the left-hand side of (8) is constant except for at most $|U|$ values of r^* (more concretely, these are values from the set $D_u = \{\text{dist}_z(u, v) \mid v \in U\}$) and the right-hand side is monotonically increasing in the interval $[\Delta/4, \Delta/2]$. Thus, it is sufficient to look for r^* only in the set $D_u \cup \{\Delta/2\}$. ◀

A.2 Using Set Partitioning for Tree Decomposition

As stated earlier, the d -decomposition tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}, \alpha)$ of graph $G = (V, E, c)$ with edge lengths $\{z_e\}_{e \in E}$ is obtained by the iterative application of Lemma 17, starting at V and terminating with leaves corresponding to sets whose diameter is less than d .

Throughout this section, we use the function $F : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ defined as

$$F(x) \triangleq \max\{x \cdot \ln(4x), 0\}.$$

That is, $F(x) = 0$ for $x \leq 1/4$ and $F(x) = x \cdot \ln(4x)$ for $x \geq 1/4$. Note that F is monotonically non-decreasing. Moreover, it satisfies the following superadditivity property.

► **Lemma 18.** *Fix $x, y \geq 0$. Then, $F(x) + F(y) \leq F(x + y)$. Moreover, if $x \geq 1/4$ and $y \geq 1/4$, then $F(x) + F(y) + x \cdot \ln((x + y)/x) \leq F(x + y)$.*

Proof. If $x < 1/4$, then $F(x) = 0$, and hence the lemma follows by monotonicity of F . The case when $y < 1/4$ is analogous. Hence, we may now assume that $x \geq 1/4$ and $y \geq 1/4$. Using the definition of F ,

$$\begin{aligned} F(x + y) - F(x) - F(y) &= x \cdot (\ln(4x + 4y) - \ln(4x)) + y \cdot (\ln(4x + 4y) - \ln(4y)) \\ &\geq x \cdot \ln\left(\frac{x + y}{x}\right) + y \cdot 0. \end{aligned} \quad \blacktriangleleft$$

15:18 An Improved Approximation Algorithm for Dynamic Minimum Linear Arrangement

► **Lemma 19.** For a node $w \in V_T$, let $V_T(w)$ be the set of nodes of a subtree of \mathcal{T} rooted at w . Let

$$A(w) \triangleq \frac{1}{g \cdot \ln \ln(\text{vol } V)} \cdot \sum_{w' \in V_T(w)} c(\text{cut}(w')) \cdot \text{diam}_z(\alpha(w')),$$

where g is the universal constant of Lemma 17. Then, $A(w) \leq F(\text{vol } \alpha(w))$.

Proof. For brevity, for a node $w \in V_T$, we use $\text{vol } w$ instead of $\text{vol } \alpha(w)$.

We show the lemma by induction. For the induction base (w is a leaf), we observe that $\text{cut}(w) = \emptyset$, and thus $c(\text{cut}(w)) = 0$. Hence, $A(w) = 0 \leq F(\text{vol } w)$.

For the induction step, we consider an internal node $w \in V_T$ with children w_L and w_R . We assume that the lemma inequality follows for both w_L and w_R , and we prove it for w . By the definition of A it follows that

$$A(w) = A(w_L) + A(w_R) + \frac{1}{g \cdot \ln \ln(\text{vol } V)} \cdot c(\text{cut}(w)) \cdot \text{diam}_z(\alpha(w)). \quad (9)$$

We consider two cases depending on whether $G[\alpha(w)]$ is connected or not.

■ If $G[\alpha(w)]$ is disconnected, Lemma 17 implies $c(\text{cut}(w)) = 0$ and $\text{vol}(w) = \text{vol}(w_L) + \text{vol}(w_R)$. Thus,

$$\begin{aligned} A(w) &= A(w_L) + A(w_R) && \text{(applying } c(\text{cut}(w)) = 0 \text{ to (9))} \\ &\leq F(\text{vol } w_L) + F(\text{vol } w_R) && \text{(by the inductive assumption)} \\ &\leq F(\text{vol } w_L + \text{vol } w_R) = F(\text{vol } w). && \text{(by Lemma 18)} \end{aligned}$$

■ If $G[\alpha(w)]$ is connected, Lemma 17 guarantees that

$$c(\text{cut}(w)) \cdot \text{diam}_z(\alpha(w)) \leq g \cdot \beta_w \cdot \ln\left(\frac{\text{vol } w}{\beta_w}\right) \cdot \ln \ln(\text{vol } V), \quad (10)$$

where β_w satisfies $\max\{\text{vol } w_L, 1/4\} \leq \beta_w$ and $\max\{\text{vol } w_R, 1/4\} \leq \text{vol } w - \beta_w$. Thus,

$$\begin{aligned} A(w) &\leq A(w_L) + A(w_R) + \beta_w \cdot \ln(\text{vol } w / \beta_w) && \text{(applying (10) to (9))} \\ &\leq F(\text{vol } w_L) + F(\text{vol } w_R) + \beta_w \cdot \ln(\text{vol } w / \beta_w) && \text{(by the inductive assumption)} \\ &\leq F(\beta_w) + F(\text{vol } w - \beta_w) + \beta_w \cdot \ln(\text{vol } w / \beta_w) && \text{(by the monotonicity of } F) \\ &\leq F(\beta_w + \text{vol } w - \beta_w) = F(\text{vol } w). && \text{(by Lemma 18)} \end{aligned}$$

As in both cases $A(w) \leq F(\text{vol } w)$, this concludes the inductive proof. ◀

A.3 Bounding Cost of Decomposition Tree

Before we prove Theorem 4, we prove its variant, where we assume that $\min_{e \in E} c(e) \geq 1/d$. Theorem 4 follows then by scaling as a simple corollary.

► **Lemma 20.** Fix a real $d > 0$, a graph $G = (V, E, c)$ such that $c(e) \geq 1/d$ for every $e \in E$, and non-negative edge lengths $\{z_e\}_{e \in E}$. It is possible to construct, in polynomial time, a d -decomposition tree $\mathcal{T} = (V_T, E_T, \alpha)$ of G , such that

$$\text{COST}_{G,z}(\mathcal{T}) \leq \xi \cdot O(\log \xi \cdot \log \log \xi),$$

where $\xi = \sum_{e \in E} c(e) \cdot z_e$.

Proof. We construct a d -decomposition tree \mathcal{T} as described in the previous section. Let w_r be its root. Using Definition 3,

$$\begin{aligned} \text{COST}_{G,z}(\mathcal{T}) &= \sum_{w \in V_{\mathcal{T}}(w_r)} c(\text{cut}(w)) \cdot \text{diam}_z(\alpha(w)) \\ &\leq O(1) \cdot F(\text{vol}(\alpha(w_r))) \cdot \ln \ln(\text{vol } V) && \text{(by Lemma 19)} \\ &= \text{vol } V \cdot O(\ln(\text{vol } V) \cdot \ln \ln(\text{vol } V)). && \text{(by the definition of } F) \end{aligned}$$

The lemma follows by observing that $\text{vol } V = \xi$. ◀

► **Theorem 4.** Fix a real $d > 0$ and a graph $G = (V, E, c)$ with non-negative edge lengths $\{z_e\}_{e \in E}$. It is possible to construct, in polynomial time, a d -decomposition tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}, \alpha)$ of G , such that

$$\text{COST}_{G,z}(\mathcal{T}) \leq \xi \cdot O(\log(f_{c,d} \cdot \xi) \cdot \log \log(f_{c,d} \cdot \xi)),$$

where $\xi = \sum_{e \in E} c(e) \cdot z_e$, $f_{c,d} = \max\{1, 1/(d \cdot c_{\min})\}$, and $c_{\min} = \min_{e \in E} c(e)$.

Proof. Let $c_{\text{thr}} = 1/d$. Note that $f_{c,d} = \max\{1, c_{\text{thr}}/c_{\min}\}$. If $c_{\min} \geq c_{\text{thr}}$, then G satisfies the requirements of Lemma 20 and $f_{c,d} = 1$. The theorem follows immediately by invoking this lemma.

In the following, we thus assume that $c_{\min} < c_{\text{thr}}$. In this case, $f_{c,d} = c_{\text{thr}}/c_{\min}$. We take the graph $G' = (V, E, c')$, where $c'(e) = (c_{\text{thr}}/c_{\min}) \cdot c(e)$ for every edge $e \in E$. We construct a d -decomposition tree \mathcal{T} of G' using Lemma 20. We now argue that \mathcal{T} satisfies the theorem statement with respect to the original graph G . Let $\xi' = \sum_{e \in E} c'(e) \cdot z_e$. Then,

$$\begin{aligned} \text{COST}_{G,z}(\mathcal{T}) &= (c_{\min}/c_{\text{thr}}) \cdot \text{COST}_{G',z}(\mathcal{T}) && \text{(by Definition 3)} \\ &\leq (c_{\min}/c_{\text{thr}}) \cdot \xi' \cdot O(\log \xi' \cdot \log \log \xi') && \text{(by Lemma 20)} \\ &= \xi \cdot O(\log((c_{\text{thr}}/c_{\min}) \cdot \xi) \cdot \log \log((c_{\text{thr}}/c_{\min}) \cdot \xi)) \\ &= \xi \cdot O(\log(f_{c,d} \cdot \xi) \cdot \log \log(f_{c,d} \cdot \xi)). && \text{◀} \end{aligned}$$

Gapped String Indexing in Subquadratic Space and Sublinear Query Time

Philip Bille  

Technical University of Denmark, Lyngby, Denmark

Inge Li Gørtz  

Technical University of Denmark, Lyngby, Denmark

Moshe Lewenstein  

Bar-Ilan University, Ramat-Gan, Israel



Solon P. Pissis  

CWI, Amsterdam, The Netherlands

Vrije Universiteit, Amsterdam, The Netherlands

Eva Rotenberg  

Technical University of Denmark, Lyngby, Denmark

Teresa Anna Steiner  

Technical University of Denmark, Lyngby, Denmark

Abstract

In GAPPED STRING INDEXING, the goal is to compactly represent a string S of length n such that for any query consisting of two strings P_1 and P_2 , called *patterns*, and an integer interval $[\alpha, \beta]$, called *gap range*, we can quickly find occurrences of P_1 and P_2 in S with distance in $[\alpha, \beta]$. GAPPED STRING INDEXING is a central problem in computational biology and text mining and has thus received significant research interest, including parameterized and heuristic approaches. Despite this interest, the best-known time-space trade-offs for GAPPED STRING INDEXING are the straightforward $\mathcal{O}(n)$ space and $\mathcal{O}(n + \text{occ})$ query time or $\Omega(n^2)$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + \text{occ})$ query time.

We break through this barrier obtaining the first interesting trade-offs with polynomially subquadratic space and polynomially sublinear query time. In particular, we show that, for every $0 \leq \delta \leq 1$, there is a data structure for GAPPED STRING INDEXING with either $\tilde{\mathcal{O}}(n^{2-\delta/3})$ or $\tilde{\mathcal{O}}(n^{3-2\delta})$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + n^\delta \cdot (\text{occ} + 1))$ query time, where occ is the number of reported occurrences.

As a new fundamental tool towards obtaining our main result, we introduce the SHIFTED SET INTERSECTION problem: preprocess a collection of sets S_1, \dots, S_k of integers such that for any query consisting of three integers i, j, s , we can quickly output YES if and only if there exist $a \in S_i$ and $b \in S_j$ with $a + s = b$. We start by showing that the SHIFTED SET INTERSECTION problem is equivalent to the indexing variant of 3SUM (3SUM INDEXING) [Golovnev et al., STOC 2020]. We then give a data structure for SHIFTED SET INTERSECTION with gaps, which entails a solution to the GAPPED STRING INDEXING problem. Furthermore, we enhance our data structure for deciding SHIFTED SET INTERSECTION, so that we can support the reporting variant of the problem, i.e., outputting all certificates in the affirmative case. Via the obtained equivalence to 3SUM INDEXING, we thus give new improved data structures for the reporting variant of 3SUM INDEXING, and we show how this improves upon the state-of-the-art solution for JUMBLED INDEXING [Chan and Lewenstein, STOC 2015] for any alphabet of constant size $\sigma > 5$.

2012 ACM Subject Classification Theory of computation \rightarrow Pattern matching

Keywords and phrases data structures, string indexing, indexing with gaps, two patterns

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.16

Related Version *Full Version:* <https://arxiv.org/abs/2211.16860>



© Philip Bille, Inge Li Gørtz, Moshe Lewenstein, Solon P. Pissis, Eva Rotenberg, and Teresa Anna Steiner;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 16; pp. 16:1–16:21



Leibniz International Proceedings in Informatics

LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Funding *Philip Bille*: Supported by the Independent Research Fund Denmark (DFR-9131-00069B).

Inge Li Gørtz: Supported by the Independent Research Fund Denmark (DFR-9131-00069B).

Solon P. Pissis: Supported by the PANGAIA (No 872539) and ALPACA (No 956229) projects.

Eva Rotenberg: Supported by the Danish Research Council grants DFR-9131-00069B “Adaptive Compressed Computation” and DFR-9131-00044B “Dynamic Network Analysis”.

Teresa Anna Steiner: Supported by the VILLUM FONDEN grant VIL51463.

1 Introduction

The classic string indexing (or text indexing) problem [28, 19] is to preprocess a string S into a compact data structure that supports efficient pattern matching queries; i.e., *decide* if the pattern occurs or not in S or *report* the set of all positions in S where an occurrence of the pattern starts. An important variant of practical interest is the GAPPED STRING INDEXING problem; the goal is to preprocess a string S of length n into a compact data structure, such that for any query consisting of two patterns P_1 and P_2 , and a gap range $[\alpha, \beta]$, one can quickly find occurrences of P_1 and P_2 in S with distance in $[\alpha, \beta]$.

Searching for patterns with gaps is of great importance in computational biology [11, 30, 24, 46, 43, 48, 50]. In DNA sequences, a structured DNA motif consists of two smaller conserved sites (patterns P_1 and P_2) separated by a spacer, that is, a non-conserved spacer of mostly fixed or slightly variable length (gap range $[\alpha, \beta]$). Thus, by introducing an efficient data structure for GAPPED STRING INDEXING, one can preprocess a DNA sequence into a compact data structure that facilitates efficient subsequent searches to DNA motifs.

Searching for patterns with gaps appears also in the area of text mining [42, 37, 44, 54]. Here, the task of finding co-occurrences of words is important, because they can indicate semantic proximity or idiomatic expressions in the text. A co-occurrence is a sequence of words (patterns P_1, P_2, \dots, P_k) that occur in close proximity (gap range $[\alpha, \beta]$ or, most often, gap range $[0, \beta]$). By giving a data structure for GAPPED STRING INDEXING, one can pre-process large bodies of text into a compact data structure that facilitates efficient subsequent co-occurrence queries for the first non-trivial number of patterns, i.e., for $k = 2$.

The algorithmic version of the problem, where a string and a *single query* is given as input, is well-studied [10, 9, 24, 45, 48, 51]. The indexing version, which is arguably more useful in real-world applications, is also much more challenging: Standard techniques yield an $\mathcal{O}(n)$ -space and $\mathcal{O}(n + \text{occ})$ -query time solution (see Appendix A) or an $\mathcal{O}(n^3)$ -space and $\mathcal{O}(|P_1| + |P_2| + \text{occ})$ -query time solution, where occ is the size of the output. A more involved approach yields $\tilde{\mathcal{O}}(n^2)$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + \text{occ})$ query time (see Appendix B).

The Combinatorial Pattern Matching community has been unable to improve the above-mentioned long-standing trade-offs. Precisely because of this, practitioners have typically been engineering algorithms or studying heuristic approaches [4, 11, 12, 24, 29, 44, 45, 46, 48, 50, 54]. Theoreticians, on the other hand, have typically been solving restricted or parameterized variants [49, 5, 31, 38, 8, 35, 7, 17, 39, 25, 40]. Thus, breaking through the quadratic-space linear-time barrier for gapped string indexing is likely to have major consequences both in theory and in practice. This work is dedicated to answering the following question:

*Is there a subquadratic-space and sublinear-query time solution for
GAPPED STRING INDEXING?*

1.1 Results

We assume throughout the standard word-RAM model of computation and answer the above question in the affirmative. Let us start by formally defining the *existence variant* of GAPPED STRING INDEXING as follows:

GAPPED STRING INDEXING

Preprocess: A string S of length n .

Query: Given two strings P_1 and P_2 and an integer interval $[\alpha, \beta]$, output YES if and only if there exists a pair (i, j) such that P_1 occurs at position i of S , P_2 occurs at position $j \geq i$ of S and $j - i \in [\alpha, \beta]$.

Similarly, in the *reporting variant* of GAPPED STRING INDEXING, a query answer is all pairs satisfying the above conditions.

GAPPED STRING INDEXING W. REPORTING

Preprocess: A string S of length n .

Query: Given two strings P_1 and P_2 and an integer interval $[\alpha, \beta]$, output all pairs (i, j) such that P_1 occurs at position i of S , P_2 occurs at position $j \geq i$ of S and $j - i \in [\alpha, \beta]$.

Our main result is the following trade-offs for GAPPED STRING INDEXING with reporting:

► **Theorem 1.** *For every $0 \leq \delta \leq 1$, there is a data structure for GAPPED STRING INDEXING W. REPORTING with either:*

- (i) $\tilde{O}(n^{2-\delta/3})$ space and $\tilde{O}(n^\delta \cdot (\text{occ} + 1) + |P_1| + |P_2|)$ query time; or
- (ii) $\tilde{O}(n^{3-2\delta})$ space and $\tilde{O}(n^\delta \cdot (\text{occ} + 1) + |P_1| + |P_2|)$ query time,

where occ is the size of the output.

For the existence variant, the bounds above hold with $\text{occ} = 1$: we can terminate the querying algorithm as soon as the first witness is reported. (Note that $n^{3-2\delta}$ is smaller than $n^{2-\delta/3}$ for $\delta > 3/5$.) Hence, we achieve the first polynomially *subquadratic space* and polynomially *sublinear query time* for GAPPED STRING INDEXING.

Our main technical contribution is a data structure for GAPPED SET INTERSECTION W. REPORTING, which is a generalization of a problem related to 3SUM INDEXING. We then show that our improved result for GAPPED STRING INDEXING W. REPORTING follows via new connections to GAPPED SET INTERSECTION W. REPORTING.

We show that the GAPPED SET INTERSECTION and 3SUM INDEXING problems (and their reporting variants) are equivalent, but with an increase in universe size in the reduction to 3SUM INDEXING. Thus, as a result, we obtain a new data structure for the indexing variant of the 3SUM problem [27], which not only outputs an arbitrary certificate, but it outputs *all certificates*. This is an interesting contribution on its own right and also has applications e.g., to the JUMBLED INDEXING problem [14].

1.2 Overview of Techniques and Paper Organization

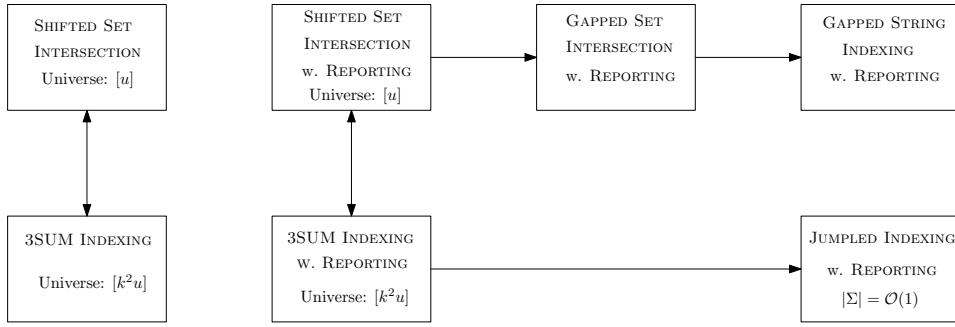
As a new fundamental tool towards obtaining our main result for GAPPED STRING INDEXING, we introduce the following problem, which we call SHIFTED SET INTERSECTION (see Figure 1).

SHIFTED SET INTERSECTION

Preprocess: A collection of k sets S_1, \dots, S_k of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \dots, u\}$.

Query: Given i, j, s , output YES if and only if there exist $a \in S_i$ and $b \in S_j$ such that $a + s = b$.

We start by showing that the SHIFTED SET INTERSECTION problem is equivalent to the indexing variant of 3SUM. We formally define the 3SUM INDEXING problem next.



■ **Figure 1** We show that SHIFTED SET INTERSECTION and 3SUM INDEXING are equivalent, and our reduction also transfers to the reporting variants of the two problems. In fact, we solve a problem that is even harder than the SHIFTED SET INTERSECTION problem; namely, the GAPPED SET INTERSECTION problem, which leads to a solution to the GAPPED STRING INDEXING problem.

3SUM INDEXING
Preprocess: A set A of N integers from a universe $U = \{1, 2, \dots, u\}$.
Query: Given $c \in U$ output YES if and only if there exist $a, b \in A$ such that $a + b = c$.

The following breakthrough result is known for 3SUM INDEXING; see also [36].

► **Theorem 2** (Golovnev et al. [27]). *For every $0 \leq \delta \leq 1$, there is a data structure for 3SUM INDEXING with space $\tilde{O}(N^{2-\delta})$ and query time $\tilde{O}(N^{3\delta})$. In particular, this data structure returns a certificate, i.e., for a query c , such that the answer is YES, it can output a pair $a, b \in A$ such that $a + b = c$ in $\tilde{O}(N^{3\delta})$ time.*

In Section 2, we show a two-way linear-time reduction between SHIFTED SET INTERSECTION and 3SUM INDEXING. In particular, this tells us that the two problems admit the same space-query time trade-offs. While the direction from 3SUM INDEXING to SHIFTED SET INTERSECTION is immediate, the opposite direction requires some careful manipulation of the input collection based on the underlying universe. Furthermore, in the same section, we give a different trade-off for SHIFTED SET INTERSECTION in the case where the set of possible distances is small, which is based on tabulating large input sets.

The main advantage of the SHIFTED SET INTERSECTION formulation is that it gives extra flexibility which we can exploit to solve several generalizations of the problem. In particular, in Section 3, we show that we can augment any SHIFTED SET INTERSECTION instance of k sets and total size N by $\mathcal{O}(k \log N)$ extra sets, such that we can represent any subset of consecutive elements in a set of the original instance by at most $\mathcal{O}(\log N)$ sets in the augmented instance. We use this to solve the reporting variant of the SHIFTED SET INTERSECTION problem, called SHIFTED SET INTERSECTION w. REPORTING, where instead of deciding whether two elements of a given shift exist, we report all such elements. The idea is to first locate one output pair using the solution for the existence variant, then split the sets into elements smaller or bigger than the output element, and recurse accordingly.

SHIFTED SET INTERSECTION w. REPORTING
Preprocess: A collection of k sets S_1, \dots, S_k of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \dots, u\}$.
Query: Given i, j, s , output all pairs (a, b) such that $a \in S_i$ and $b \in S_j$ and $a + s = b$.

We show the following result.

► **Theorem 3.** *For every $0 \leq \delta \leq 1$, there is a data structure for SHIFTED SET INTERSECTION W. REPORTING with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta \cdot (\text{occ} + 1))$ query time, where occ is the size of the output.*

As a consequence, we also give a reporting data structure for 3SUM INDEXING, which may be of independent interest. Let us first formalize the problem.

3SUM INDEXING W. REPORTING

Preprocess: A set A of N integers from a universe $U = \{1, 2, \dots, u\}$.

Query: Given $c \in U$ output all pairs $(a, b) \in A$ such that $a + b = c$.

► **Corollary 4.** *For every $0 \leq \delta \leq 1$, there is a data structure for 3SUM INDEXING W. REPORTING with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta \cdot (\text{occ} + 1))$ query time, where occ is the size of the output.*

Furthermore, in Section 4, we show that we can augment any SHIFTED SET INTERSECTION instance by $\mathcal{O}(k \log u)$ sets to solve the more general problem of GAPPED SET INTERSECTION, where instead of a *single shift*, the query asks for an interval of allowed shifts.

GAPPED SET INTERSECTION

Preprocess: A collection of k sets S_1, \dots, S_k of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \dots, u\}$.

Query: Given i, j and an integer interval $[\alpha, \beta]$, output YES if and only if there exist $a \in S_i$, $b \in S_j$ and $s \in [\alpha, \beta]$ such that $a + s = b$.

We solve this problem by first considering an approximate variant of the problem, where the interval length is a power of two plus one, and we allow false positives in an interval of roughly twice the size. Then we carefully cover the query interval by $\mathcal{O}(\log u)$ approximate queries to obtain Theorem 5.

► **Theorem 5.** *For every $0 \leq \delta \leq 1$, there is a data structure for GAPPED SET INTERSECTION with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta)$ query time.*

We combine the reduction underlying Theorem 5 with the reporting solution underlying Theorem 3 to obtain a solution for the reporting variant of GAPPED SET INTERSECTION.

GAPPED SET INTERSECTION W. REPORTING

Preprocess: A collection of k sets S_1, \dots, S_k of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \dots, u\}$.

Query: Given i, j and an integer interval $[\alpha, \beta]$, output all pairs (a, b) such that $a \in S_i$, $b \in S_j$ and there is an $s \in [\alpha, \beta]$ such that $a + s = b$.

► **Theorem 6.** *For every $0 \leq \delta \leq 1$, there is a data structure for GAPPED SET INTERSECTION W. REPORTING with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta \cdot (\text{occ} + 1))$ query time, where occ is the size of the output.*

In Section 5, we finally use all of these acquired tools to solve the GAPPED STRING INDEXING problem. In particular, we cover the suffix array of string S in dyadic subintervals, which we then preprocess into the data structure from Theorem 6. Note that the total size of these sets is $\mathcal{O}(n \log n)$, where n is the length of S . Moreover, any interval of consecutive elements in the suffix array can be covered by $\mathcal{O}(\log n)$ sets in the instance. Putting everything together we obtain our main result (Theorem 1).

16:6 Gapped String Indexing in Subquadratic Space and Sublinear Query Time

In Section 6, we show a reduction from 3SUM INDEXING to JUMBLED INDEXING, giving a data structure for JUMBLED INDEXING which improves on the state-of-the-art solution for constant alphabet sizes $\sigma > 5$. We give new upper bounds for the existence and the reporting variants of JUMBLED INDEXING. In order to define JUMBLED INDEXING, we need the following notion of a *histrogram* of a string:

► **Definition 7.** Given a string S over an alphabet Σ , a histogram h of S is a vector of dimension $|\Sigma|$ where every entry is the number of times the corresponding letter occurs in S .

JUMBLED INDEXING W. REPORTING

Preprocess: A string S of length n over an alphabet Σ .

Query: For any pattern histogram $P \in (\mathbb{N}_0)^{|\Sigma|}$, return all substrings of S whose histogram is P .

We call a substring of S whose histogram is P an *occurrence of P in S* . In the *existence variant* of the JUMBLED INDEXING problem, the query answer is YES or NO depending on whether there is at least one occurrence of P in S . We show the following result:

► **Corollary 8.** Given a string S of length n over an alphabet of size $\sigma = \mathcal{O}(1)$, for every $0 \leq \delta \leq 1$, there is a data structure for JUMBLED INDEXING W. REPORTING with $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{3\delta} \cdot (\text{occ} + 1))$ query time, where occ is the size of the output. The bounds are $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{3\delta})$ query time for the existence variant.

The best-known previous bounds for the existence variant of JUMBLED INDEXING use $\mathcal{O}(n^{2-\delta})$ space and $\mathcal{O}(m^{(2\sigma-1)\delta})$ query time [34], where m is the norm of the queried pattern (i.e., the sum of histogram entries), or $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{\delta(\sigma+1)/2})$ query time [14]. Interestingly, the reporting variant of JUMBLED INDEXING is, in general, significantly harder than the existence variant: There is a recent (unconditional) lower bound stating that any data structure for JUMBLED INDEXING W. REPORTING with $\mathcal{O}(n^{0.5-o(1)} + \text{occ})$ query time needs $\Omega(n^{2-o(1)})$ space, and this holds even for a binary alphabet [1]. By our reduction, this in particular implies that a data structure with $\tilde{\mathcal{O}}(N^{2-\delta})$ space and $\tilde{\mathcal{O}}(N^{3\delta} + \text{occ})$ query time is not possible for 3SUM INDEXING W. REPORTING. For a more complete overview, see Section 6. We conclude this paper in Section 7 with some future proposals.

In Appendix C, we show a better trade-off for the related SMALLEST SHIFT problem.

SMALLEST SHIFT

Preprocess: A collection of k sets S_1, \dots, S_k of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \dots, u\}$.

Query: Given i, j , output the smallest s such that there exists $a \in S_i$ and $b \in S_j$ with $a + s = b$.

► **Proposition 9.** There is a data structure for SMALLEST SHIFT with $\mathcal{O}(N)$ space and $\tilde{\mathcal{O}}(\sqrt{N})$ query time. Moreover, the data structure can be constructed in $\mathcal{O}(N\sqrt{N})$ time.

GAPPED SET INTERSECTION reduces to SMALLEST SHIFT in the special case where we only allow query intervals of the form $[0, \beta]$. Together with our other results, this reduction yields a $\tilde{\mathcal{O}}(N)$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + \sqrt{N} \cdot (\text{occ} + 1))$ query time trade-off for GAPPED STRING INDEXING if the query intervals are restricted to $[0, \beta]$. However, let us remark that for this restricted version of the problem, a $\tilde{\mathcal{O}}(N)$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + \sqrt{N} \cdot (\text{occ} + 1) + \text{occ})$ query time trade-off already follows from [7].

Subsequent work. Since the first publication of this paper [6], Aronov et al. [3] proposed a combination of range searching and the Fiat-Naor inversion scheme [21] (which has already been used to prove Theorem 2), recovering our main result (Theorem 1) as a corollary.

2 3SUM Indexing is Equivalent to Shifted Set Intersection

Here we show a two-way linear-time reduction between 3SUM INDEXING and SHIFTED SET INTERSECTION. We thus obtain the same space-time bounds for the two problems.

2.1 From 3SUM Indexing to Shifted Set Intersection

► **Fact 10.** *Any instance of 3SUM INDEXING of input size N and universe size u can be reduced to a SHIFTED SET INTERSECTION instance of total size $\mathcal{O}(N)$ with universe size $\mathcal{O}(u)$ in time $\mathcal{O}(N)$.*

Proof. We denote the 3SUM INDEXING instance by $A = \{a_1, a_2, \dots, a_N\}$ (the input set). We construct the SHIFTED SET INTERSECTION instance: $S_1 = A$ and $S_2 = \{-a_1, -a_2, \dots, -a_N\}$ in $\mathcal{O}(N)$ time. For any 3SUM INDEXING query c , we construct the SHIFTED SET INTERSECTION query (S_2, S_1, c) ¹ in $\mathcal{O}(1)$ time, and observe that: $a_i + a_j = c \iff c - a_i = a_j$. Thus the answer to the 3SUM INDEXING query is YES if and only if the answer to the SHIFTED SET INTERSECTION query is YES. ◀

2.2 From Shifted Set Intersection to 3SUM Indexing

► **Theorem 11.** *Any instance of SHIFTED SET INTERSECTION with $\sum_{i=1}^k |S_i| = N$ and universe size u can be reduced to a 3SUM INDEXING instance of input size $\mathcal{O}(N)$ and universe size $\mathcal{O}(k^2u)$ in time $\mathcal{O}(N)$.*

Proof. Let us start by considering an alternative yet equivalent formulation of 3SUM INDEXING. Preprocess two sets A and B from a universe $U' = \{1, 2, \dots, u'\}$ to answer queries of the following form: Given an element $c \in U'$, output YES if and only if there exist $(a, b) \in A \times B$ such that $a + b = c$. To see why it is equivalent, notice that the formulation with one set reduces trivially to this formulation by setting $A = B$. For the other direction, given A and B from universe U' , define $A' = A \cup \{b + 2u' \mid b \in B\}$. We verify that querying $c + 2u'$ on A' for any $c \in [2, \dots, 2u']$ in the 3SUM INDEXING formulation with one set is equivalent to querying c on A and B in the 3SUM INDEXING formulation with two sets. We now reduce SHIFTED SET INTERSECTION to the 3SUM INDEXING formulation with two sets.

We denote the SHIFTED SET INTERSECTION instance by S_1, \dots, S_k (the input collection over universe $U = \{1, 2, \dots, u\}$) and $\sum_{i \in [k]} |S_i|$ by N . We construct the following instance of the above 3SUM INDEXING reformulation in $\mathcal{O}(N + k) = \mathcal{O}(N)$ time:

$$A = \{e + j \cdot (k + 1) \cdot 2u \mid e \in S_j, 1 \leq j \leq k\}, B = \{-e + i \cdot 2u \mid e \in S_i, 1 \leq i \leq k\}.$$

The 3SUM INDEXING instance has input size $2N = \Theta(N)$ and universe size $U' = \mathcal{O}(k^2 \cdot u)$.

Let us denote a query of SHIFTED SET INTERSECTION by $Q(S_i, S_j, s)$. Now, we construct the 3SUM INDEXING query $Q_{3\text{SUM}}(s + (j \cdot (k + 1) + i) \cdot 2u)$ in $\mathcal{O}(1)$ time. The following claim concludes the proof.

¹ We may write (S_i, S_j, c) for a SHIFTED SET INTERSECTION query instead of (i, j, c) for clarity.

16:8 Gapped String Indexing in Subquadratic Space and Sublinear Query Time

▷ **Claim 12.** $Q(S_i, S_j, s)$ outputs YES if and only if $Q_{3SUM}(s + (j \cdot (k+1) + i) \cdot 2u)$ outputs YES.

Proof. (\Rightarrow): Let $e_1 \in S_i$, $e_2 \in S_j$ such that $e_1 + s = e_2$. Then $a = e_2 + j \cdot (k+1) \cdot 2u \in A$, $b = -e_1 + i \cdot 2u \in B$, and $a + b = s + (j \cdot (k+1) + i) \cdot 2u$. Thus $Q_{3SUM}(s + (j \cdot (k+1) + i) \cdot 2u)$ outputs YES.

(\Leftarrow): Assume there is $a \in A$ and $b \in B$ such that $a + b = s + (j \cdot (k+1) + i) \cdot 2u$. By definition of A and B , there exist i' , j' , and $e_2 \in S_{j'}$, $e_1 \in S_{i'}$ such that $a = e_2 + j' \cdot (k+1) \cdot 2u$ and $b = -e_1 + i' \cdot 2u$, thus $s + (j \cdot (k+1) + i) \cdot 2u = (e_2 - e_1) + (j' \cdot (k+1) + i') \cdot 2u$. Since $-u < s < u$ and $-u < (e_2 - e_1) < u$, we have that $j \cdot (k+1) + i = j' \cdot (k+1) + i'$ and $e_2 - e_1 = s$. Since $i \leq k$ and $i' \leq k$, we have $j = j'$ and $i = i'$. Thus $Q(S_i, S_j, s)$ outputs YES. ◀

By employing Theorem 2 we obtain the following corollary:

► **Corollary 13.** *For every $0 \leq \delta \leq 1$, there is a data structure for SHIFTED SET INTERSECTION with space $\tilde{O}(N^{2-\delta})$ and query time $\tilde{O}(N^{3\delta})$.*

We also show how we can obtain a different trade-off in the case where the number of possible shifts is bounded by some Δ (this gives us a better trade-off for some δ in the application of GAPPED STRING INDEXING). In this case, call all sets of size at most N^δ *small*, and other sets *large*. Note that there are at most $N^{1-\delta}$ large sets. For every pair of large sets and any possible shift, we precompute the answer, using space $\mathcal{O}(\Delta N^{2-2\delta})$. Additionally, we store a dictionary on every set using space $\mathcal{O}(N)$ (using e.g., perfect hashing [23]). For a query $Q(S_i, S_j, s)$, if both sets S_i and S_j are large, we look up the precomputed answer. If one set is small, wlog S_i , we check if $a + s \in S_j$ for every $a \in S_i$ using the dictionary. Note that in particular, $\Delta < u$. This gives the following lemma:

► **Lemma 14.** *For every $0 \leq \delta \leq 1$, there is a data structure for SHIFTED SET INTERSECTION with space $\mathcal{O}(u \cdot N^{2-2\delta})$ and query time $\mathcal{O}(N^\delta)$.*

3 Reporting: 3SUM Indexing and Shifted Set Intersection

In this section, we explain how we can answer reporting queries for both SHIFTED SET INTERSECTION and 3SUM INDEXING, as defined in Section 1. The following fact is trivial.

► **Fact 15.** *There is a data structure for SHIFTED SET INTERSECTION w. REPORTING with $\mathcal{O}(N)$ space and $\mathcal{O}(|S_i| + |S_j|)$ query time.*

The other extreme trade-off is also straightforward.

► **Lemma 16.** *There is a data structure for SHIFTED SET INTERSECTION w. REPORTING with $\mathcal{O}(N^2)$ space and $\mathcal{O}(\text{occ})$ query time, where occ is the size of the output.*

Proof. For every existing shift s , we save all the pairs of sets (A, B) for which $Q(A, B, s) = \text{YES}$ using perfect hashing [23]. For every such pair, we save all pairs $(a, b) \in A \times B$, such that $a + s = b$, in a list. There are $\mathcal{O}(N^2)$ pairs (a, b) in total. The space is thus $\mathcal{O}(N^2)$. ◀

We next prove the main result (Theorem 3) of this section.

► **Theorem 3.** *For every $0 \leq \delta \leq 1$, there is a data structure for SHIFTED SET INTERSECTION W. REPORTING with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta \cdot (\text{occ} + 1))$ query time, where occ is the size of the output.*

► **Corollary 4.** *For every $0 \leq \delta \leq 1$, there is a data structure for 3SUM INDEXING W. REPORTING with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta \cdot (\text{occ} + 1))$ query time, where occ is the size of the output.*

Proof of Theorem 3. The main idea behind the proof is the following: We use the fact that the data structure for 3SUM INDEXING from Theorem 2 returns a *certificate*, i.e., for a query $c \in U$ such that the answer to the query is YES it can additionally output a pair (a, b) satisfying $a + b = c$. By our reduction from SHIFTED SET INTERSECTION, when we query for S_i, S_j and s we can return $a \in S_i$ and $b \in S_j$ such that $a + s = b$. We then conceptually split S_i and S_j into two sets each, those elements in S_i which are smaller than a , those which are bigger, and similarly for S_j and b . We then want to use the fact that if $a' + s = b'$ and $a' < a$, then $b' < b$ and vice versa, to recurse on the smaller subsets. However, we cannot afford to preprocess all subsets of any set S_i into the SHIFTED SET INTERSECTION data structure. To solve this issue, we partition each set into the subsets which correspond to the dyadic intervals on the rank of the elements in sorted order, and preprocess them into our SHIFTED SET INTERSECTION data structure.

In detail, our data structure is defined as follows: Let S_1, \dots, S_k be the input to SHIFTED SET INTERSECTION W. REPORTING. We build the data structure for SHIFTED SET INTERSECTION from Corollary 13 containing, additionally to S_1, \dots, S_k , the following subsets: Let $S = \{s_1, s_2, \dots, s_m\}$ be a set in our SHIFTED SET INTERSECTION W. REPORTING instance, where $s_1 < s_2 < \dots < s_m$.

We partition S into subsets which correspond to the dyadic intervals on the rank. That is, for $j = 0, \dots, \lfloor \log m \rfloor$, we partition S into $\{s_{i+1}, s_{i+2}, \dots, s_{i+2^j}\}$, for all $i = \kappa \cdot 2^j$ and $0 \leq \kappa \leq \lfloor m/2^j \rfloor - 1$. We call these sets the *dyadic subsets* of S . Note that for a fixed j , the union of these sets is a subset of S and thus has size at most m . Hence, the total size of all dyadic subsets is $\mathcal{O}(m \log m)$.

► **Observation 17.** *Any subset of S of the form $\{x \in S \mid a \leq x \leq b\}$ is the union of at most $2 \log |S|$ dyadic subsets of S .*

The data structure for SHIFTED SET INTERSECTION W. REPORTING consists of the SHIFTED SET INTERSECTION data structure on all sets S_1, \dots, S_k and their dyadic subsets. Further, for each dyadic subset, we store its minimum and its maximum element as auxiliary information. The total size of all dyadic subsets is $\mathcal{O}(N \log N)$. The space is thus $\tilde{O}(N^{2-\delta/3})$.

To perform a query on S_i, S_j, s , we first perform a query on the SHIFTED SET INTERSECTION data structure for the same inputs. If it returns NO, we are done. If it returns YES, let a and b be the certificate pair such that $a + s = b$. Define $A_1 = \{a' \in S_i \mid a' < a\}$ and $A_2 = \{a' \in S_i \mid a' > a\}$, and similarly $B_1 = \{b' \in S_j \mid b' < b\}$ and $B_2 = \{b' \in S_j \mid b' > b\}$. Then any additional solution pair (a', b') has to either satisfy $a' \in A_1$ and $b' \in B_1$ or $a' \in A_2$ and $b' \in B_2$. Further, by Observation 17, we can decompose $A_1, A_2, B_1,$ and B_2 into $\mathcal{O}(\log N)$ dyadic subsets in $\mathcal{O}(\log N)$ time. Call these decompositions $\mathcal{A}_1, \mathcal{A}_2, \mathcal{B}_1, \mathcal{B}_2$. We could now recurse on any (A, B) where $A \in \mathcal{A}_1$ and $B \in \mathcal{B}_1$ or $A \in \mathcal{A}_2$ and $B \in \mathcal{B}_2$, dividing into $\mathcal{O}(\log^2 N)$ subproblems. Since every time before we recurse we obtain a new certificate, this brings the query time to $\tilde{O}(N^\delta \cdot (\text{occ} + 1))$, which proves the theorem. We nevertheless show next (Lemma 18) how we can reduce the recursion pairs to $\mathcal{O}(\log N)$.

16:10 Gapped String Indexing in Subquadratic Space and Sublinear Query Time

Fix $i \in \{1, 2\}$. For $A' \in \mathcal{A}_i$, let a_{\min} be its minimum element and a_{\max} its maximum element. For $B' \in \mathcal{B}_i$ define b_{\min} and b_{\max} analogously. We say A' and B' *match* if $[a_{\min} + s, a_{\max} + s]$ and $[b_{\min}, b_{\max}]$ have a non-empty intersection.

► **Lemma 18.** *There are $\mathcal{O}(\log N)$ matching pairs (A', B') , $A' \in \mathcal{A}_i$, $B' \in \mathcal{B}_i$, and $i \in \{1, 2\}$.*

Proof. For a fixed $A' \in \mathcal{A}_i$, consider all B' such that A' matches $B' \in \mathcal{B}_i$. Since the intervals $[b_{\min}, b_{\max}]$ are disjoint, all except at most two B' that match A' have the property that the corresponding interval $[b_{\min}, b_{\max}]$ is fully contained in $[a_{\min} + s, a_{\max} + s]$. Hence, those B' match *only* A' . Let $\mathcal{A}_i = \{A_i^1, \dots, A_i^{k_1}\}$ and $|\mathcal{B}_i| = k_2$ with $k_1, k_2 = \mathcal{O}(\log N)$. The number of matching pairs is given by

$$\sum_{j=1}^{k_1} (\text{Number of } B' \text{ matching } A_i^j) \leq 2k_1 + \sum_{j=1}^{k_1} (\text{Number of } B' \text{ matching only } A_i^j) \leq 2k_1 + k_2.$$

The last inequality follows since the sets $\{B' \text{ matching only } A_i^j\}$ form disjoint subsets of \mathcal{B}_i . ◀

For one A' , we can identify all matching B' in $\mathcal{O}(\log N)$ time using the precomputed information (i.e., searching for the predecessor of $a_{\min} + s$ in the minima of B_i and the successor of $a_{\max} + s$ in the maxima of B_i). In total all matching pairs can be identified in time $\mathcal{O}(\log^2 N)$. Thus, instead of recursing on all pairs (A', B') , where $A' \in \mathcal{A}_i$ and $B' \in \mathcal{B}_i$, we can recurse only on the matching pairs, which saves a $\log N$ factor in the query time. ◀

4 Gapped Set Intersection

Here we show how to solve the more general problem of GAPPED SET INTERSECTION, where we allow any shift within a given interval. Recall that the problem is defined in Section 1.

The main idea is to use the solution from Corollary 13 on $\mathcal{O}(\log u)$ carefully constructed SHIFTED SET INTERSECTION instances. To do this, we first show how we can approximately answer GAPPED SET INTERSECTION queries for query intervals whose length is a power of two plus one (approximately in the sense that we allow false positives within a larger interval), then use $\mathcal{O}(\log u)$ such queries to “cover” $[\alpha, \beta]$.

We formally define *approximate* GAPPED SET INTERSECTION.

<p>APPROXIMATE GAPPED SET INTERSECTION</p> <p>Preprocess: A collection of k sets S_1, \dots, S_k of total size $\sum_i S_i = N$ of integers from a universe $U = \{1, 2, \dots, u\}$ and a <i>level</i> l with $1 \leq l \leq \log u$.</p> <p>Query: Given i, j and a <i>center distance</i> $d = \kappa \cdot 2^l$, for some positive integer κ, output YES or NO such that:</p> <ul style="list-style-type: none"> ■ The output is YES if there exists a pair $a \in S_i, b \in S_j$ such that $b - a \in [d - 2^{l-1}, d + 2^{l-1}]$; ■ The output is NO if there exists no pair $a \in S_i, b \in S_j$ such that $b - a \in (d - 2^l, d + 2^l)$.

Note that by the definition of the APPROXIMATE GAPPED SET INTERSECTION problem, if for a query i, j and $d = \kappa \cdot 2^l$ there exists a pair $a \in S_i, b \in S_j$ such that $b - a \in (d - 2^l, d + 2^l)$, but no pair $a \in S_i, b \in S_j$ such that $b - a \in [d - 2^{l-1}, d + 2^{l-1}]$, we may answer either YES or NO.

► **Lemma 19.** *Assume there is a data structure for SHIFTED SET INTERSECTION with s space and t query time. Then there is a data structure for APPROXIMATE GAPPED SET INTERSECTION with $\mathcal{O}(s)$ space and $\mathcal{O}(t)$ query time.*

Proof. For any set S_i in the collection, define $S'_i = \{\lfloor a/2^{l-1} \rfloor, a \in S_i\}$ and build the assumed data structure for SHIFTED SET INTERSECTION on sets S'_1, \dots, S'_k . To answer a query $(i, j, d = \kappa \cdot 2^l)$ on APPROXIMATE GAPPED SET INTERSECTION, make the queries $(i, j, 2\kappa - 1)$, $(i, j, 2\kappa)$ and $(i, j, 2\kappa + 1)$ on the SHIFTED SET INTERSECTION instance. Return YES if and only if one of the queries to the SHIFTED SET INTERSECTION instance returns YES.

To show correctness, first notice that if we return YES, then there exist $a \in S_i$ and $b \in S_j$ satisfying $2\kappa - 1 \leq \lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor \leq 2\kappa + 1$. This implies $2\kappa - 2 \leq \lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor - 1 < \lfloor b/2^{l-1} \rfloor - a/2^{l-1} \leq b/2^{l-1} - a/2^{l-1}$ and $b/2^{l-1} - a/2^{l-1} \leq b/2^{l-1} - \lfloor a/2^{l-1} \rfloor < \lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor + 1 \leq 2\kappa + 2$. Thus, $b - a \in (\kappa \cdot 2^l - 2^l, \kappa \cdot 2^l + 2^l) = (d - 2^l, d + 2^l)$.

Next, assume there is a pair $a \in S_i$ and $b \in S_j$ such that $b - a \in [d - 2^{l-1}, d + 2^{l-1}] = [\kappa \cdot 2^l - 2^{l-1}, \kappa \cdot 2^l + 2^{l-1}]$. Then clearly $b/2^{l-1} - a/2^{l-1} \in [2\kappa - 1, 2\kappa + 1]$. Similar to above, we have $\lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor \leq b/2^{l-1} - \lfloor a/2^{l-1} \rfloor < b/2^{l-1} - a/2^{l-1} + 1 \leq 2\kappa + 2$ and $2\kappa - 2 \leq b/2^{l-1} - a/2^{l-1} - 1 < \lfloor b/2^{l-1} \rfloor - a/2^{l-1} \leq \lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor$. Thus, $\lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor \in (2\kappa - 2, 2\kappa + 2)$ and, because $\lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor$ is an integer, $\lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor \in [2\kappa - 1, 2\kappa + 1]$. Thus we answer YES in this case. \blacktriangleleft

► **Corollary 20.** *For every $0 \leq \delta \leq 1$, there is a data structure for APPROXIMATE GAPPED SET INTERSECTION with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta)$ query time.*

We now show how to reduce GAPPED SET INTERSECTION to $\mathcal{O}(\log u)$ APPROXIMATE GAPPED SET INTERSECTION instances, giving the following theorem:

► **Theorem 5.** *For every $0 \leq \delta \leq 1$, there is a data structure for GAPPED SET INTERSECTION with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta)$ query time.*

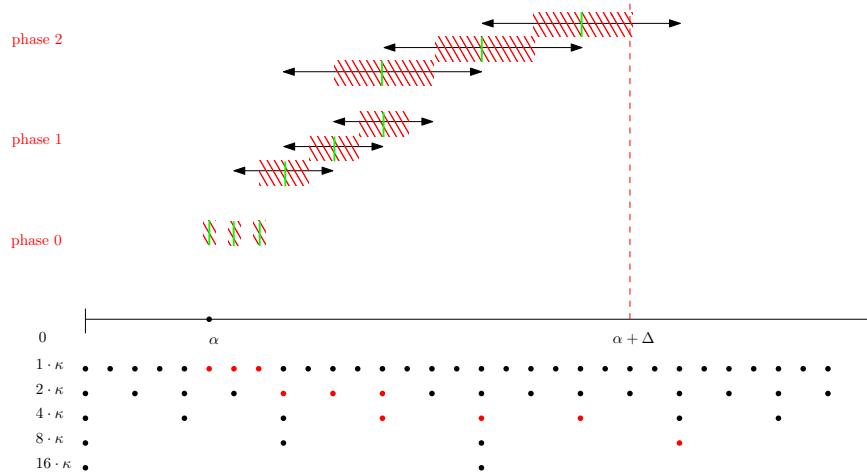
Proof. Given S_1, \dots, S_k , we build a SHIFTED SET INTERSECTION data structure and an APPROXIMATE GAPPED SET INTERSECTION data structure for S_1, \dots, S_k and every level l with $1 \leq l \leq \log u$. Assume we want to answer a query for S_i, S_j and $[\alpha, \beta]$. We show how to answer the query using $\mathcal{O}(\log(\beta - \alpha))$ APPROXIMATE GAPPED SET INTERSECTION queries. We query APPROXIMATE GAPPED SET INTERSECTION for i, j and different choices of l and d . We call a query i, j and d to APPROXIMATE GAPPED SET INTERSECTION of level l a 2^l -approximate query centered at d . We say the query covers the interval $[d - 2^{l-1}, d + 2^{l-1}]$. We call all elements in interval $(d - 2^l, d + 2^l)$ uncertain. Now, for the interval $[\alpha, \beta]$, we want to find $\mathcal{O}(\log(\beta - \alpha)) = \mathcal{O}(\log u)$ queries which together cover $[\alpha, \beta]$ such that there are no uncertain elements outside of $[\alpha, \beta]$. In detail:

We show how to cover a continuous interval $[\alpha, \alpha + \Delta]$ for growing Δ over several phases (inspect Figure 2). In the l th phase, we use a constant number of 2^l -approximate queries.

- If at any point we cover $[\alpha, \alpha + \Delta]$ such that $\Delta \geq (\beta - \alpha)/2$, we stop.
- If we never cover past $\alpha + \Delta$ with $\Delta \geq (\beta - \alpha)/2$ in phase l , we do exactly three 2^l -approximate queries in that phase:
 - In phase 0, we do regular SHIFTED SET INTERSECTION queries for $\alpha, \alpha + 1, \alpha + 2$.
 - In phase l , if we have covered up until $\alpha + \Delta$ in phase $l - 1$, we center the first 2^l -approximate query at the largest $\kappa \cdot 2^l$ such that $\kappa \cdot 2^l - 2^{l-1} \leq \alpha + \Delta$. Then we center the next two queries at $(\kappa + 1) \cdot 2^l$ and $(\kappa + 2) \cdot 2^l$.
- When we stop, we do a symmetric process starting from β .

▷ **Claim 21.** We stop after $\mathcal{O}(\log(\beta - \alpha))$ phases, after which we will have covered the full interval $[\alpha, \alpha + \Delta]$ for $\Delta \geq (\beta - \alpha)/2$.

16:12 Gapped String Indexing in Subquadratic Space and Sublinear Query Time



■ **Figure 2** Illustration of the phases from the proof of Theorem 5: At the l th phase of the algorithm, we use at most three 2^l -approximate queries to cover at least $2 \cdot 2^l$ elements in $[\alpha, \beta]$ which were not covered by previous phases. The centers of the queries are shown by short vertical line segments. The covered elements for each query are shown dashed, and the uncertain elements are shown by the black horizontal arrows. The dots at the bottom show all potential centers for the approximate queries; the ones our algorithm uses are marked in red.

Proof. By the choice of the first query in each phase, the interval we cover with that query starts at some position $\kappa \cdot 2^l - 2^{l-1} \leq \alpha + \Delta$. Thus, there are no gaps. The first interval covers up until $\kappa \cdot 2^l + 2^{l-1} = (\kappa + 1) \cdot 2^l - 2^{l-1} > \alpha + \Delta$. Thus, only the first query of a phase covers part of an interval that was covered in a previous phase, while the next two queries cover $2 \cdot 2^l$ elements which were not covered in a previous phase. Thus, after $\mathcal{O}(\log(\beta - \alpha))$ phases, we have covered at least until $\alpha + (\beta - \alpha)/2$. \triangleleft

▷ **Claim 22.** All uncertain elements introduced during the algorithm are included in $[\alpha, \beta]$.

Proof. By the argument before, if we enter phase l and have not stopped, we have covered at least $2 \sum_{j=0}^{l-1} 2^j = 2(2^l - 1) = 2^{l+1} - 2$ elements, i.e., $\Delta \geq 2^{l+1} - 2$. Further, $\Delta < (\beta - \alpha)/2$. The first query in phase l is centered at $\kappa \cdot 2^l$ with $\kappa \cdot 2^l - 2^{l-1} \leq \alpha + \Delta < \kappa \cdot 2^l + 2^{l-1}$. Thus, $\kappa \cdot 2^l \leq \alpha + \Delta + 2^{l-1}$ and $\kappa \cdot 2^l \geq \alpha + \Delta - 2^{l-1} + 1$ and the uncertain interval $(\kappa \cdot 2^l - 2^l, \kappa \cdot 2^l + 2^l)$ is contained in $[\alpha + \Delta - 2^{l-1} + 2 - 2^l, \alpha + \Delta + 2^{l-1} + 2^l - 1]$. Since $\Delta \geq 2^{l+1} - 2 \geq 2^l + 2^{l-1} - 2$, we have that $\alpha + \Delta - 2^{l-1} - 2^l + 2 \geq \alpha$ and thus there are no uncertain elements smaller than α . Since $\Delta + 2^l + 2^{l-1} - 1 \leq 2\Delta + 1 \leq \beta - \alpha$, we have that $\alpha + \Delta + 2^l + 2^{l-1} - 1 \leq \beta$ and thus there are no uncertain elements bigger than β .

If there are subsequent queries in phase l , then before the query we cover up to $\alpha + \Delta$ for $\Delta < (\beta - \alpha)/2$. The next query is centered at $\kappa \cdot 2^l = \alpha + \Delta + 2^{l-1}$, thus the argument as to why we do not introduce uncertain elements past β is analogous. Since $\Delta \geq 2^l$ and $\kappa \cdot 2^l > \alpha + \Delta$, the uncertain interval centered at $\kappa \cdot 2^l$ cannot include elements smaller than α . \triangleleft

Combining Claim 21 and Claim 22, we obtain the theorem. \blacktriangleleft

Next, we show how we can solve GAPPED SET INTERSECTION w. REPORTING, the reporting variant of the GAPPED SET INTERSECTION problem. The reduction is basically the same, but using a solution to SHIFTED SET INTERSECTION w. REPORTING instead of a solution to SHIFTED SET INTERSECTION. Again, we use an approximate variant of the problem, now defined as follows:

APPROXIMATE GAPPED SET INTERSECTION W. REPORTING

Preprocess: A collection of k sets S_1, \dots, S_k of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \dots, u\}$ and a level l with $1 \leq l \leq \log u$.

Query: Given i, j and a center distance $d = \kappa \cdot 2^l$, for some positive integer κ , output a set P of pairs (a, b) such that $a \in S_i$ and $b \in S_j$ and

- P contains all pairs (a, b) , $a \in S_i$, $b \in S_j$ such that $b - a \in [d - 2^{l-1}, d + 2^{l-1}]$;
- P contains no pair (a, b) such that $b - a \notin (d - 2^l, d + 2^l)$.

► **Corollary 23.** For every $0 \leq \delta \leq 1$, there is a data structure for APPROXIMATE GAPPED SET INTERSECTION W. REPORTING with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta \cdot |P|)$ query time.

Proof. The reduction is essentially the same as Lemma 19, just that instead of the SHIFTED SET INTERSECTION data structure, we use the SHIFTED SET INTERSECTION W. REPORTING data structure from Theorem 3. That is, we construct a SHIFTED SET INTERSECTION W. REPORTING data structure for sets $S'_i = \{\lfloor a/2^{l-1} \rfloor, a \in S_i\}$. Additionally, for any $a' \in S'_i$, we store a list $L_{a'}$ of all values $a \in S_i$ such that $\lfloor a/2^{l-1} \rfloor = a'$. To answer a query $(i, j, d = \kappa \cdot 2^l)$, we query the SHIFTED SET INTERSECTION W. REPORTING data structure for $(i, j, 2\kappa - 1)$, $(i, j, 2\kappa)$ and $(i, j, 2\kappa + 1)$. Whenever one of the queries returns a pair (a', b') , we return all pairs (a, b) for $a \in L_{a'}$ and $b \in L_{b'}$. By the same arguments as in the proof of Lemma 19, we never return a pair with $b - a \notin (d - 2^l, d + 2^l)$, and we return all pairs $b - a \in [d - 2^{l-1}, d + 2^{l-1}]$. ◀

► **Theorem 6.** For every $0 \leq \delta \leq 1$, there is a data structure for GAPPED SET INTERSECTION W. REPORTING with $\tilde{O}(N^{2-\delta/3})$ space and $\tilde{O}(N^\delta \cdot (\text{occ} + 1))$ query time, where occ is the size of the output.

Proof. We use the same reduction as in Theorem 5, only using the data structures for APPROXIMATE GAPPED SET INTERSECTION W. REPORTING from Corollary 23 instead of the data structures for APPROXIMATE GAPPED SET INTERSECTION. We perform exactly the same queries. By the same arguments as in the proof of Theorem 5, we find all pairs (a, b) with $a \in S_i$ and $b \in S_j$ and $b - a \in [\alpha, \beta]$ and only those. However, we might report the same pair many times, so we need to argue about the total size of the output. Note that any single query to APPROXIMATE GAPPED SET INTERSECTION W. REPORTING reports any pair at most once, thus, any pair is reported $\mathcal{O}(\log u)$ times and the total size of the output is $\mathcal{O}(\text{occ} \cdot \log u)$. To avoid multiple outputs we can collect all pairs, sort them, and delete duplicates, before outputting. ◀

5 Gapped String Indexing

Let us recall some basic definitions and notation on strings. An *alphabet* Σ is a finite set of elements called *letters*. A *string* $S = S[1..n]$ is a sequence of letters over some alphabet Σ ; we denote the length of S by $|S| = n$. The fragment $S[i..j]$ of S is an *occurrence* of the underlying *substring* $P = S[i] \dots S[j]$. We also write that P occurs at *position* i in S when $P = S[i] \dots S[j]$. A *prefix* of S is a fragment of S of the form $S[1..j]$ and a *suffix* of S is a fragment of S of the form $S[i..n]$.

For a string S of length n over an ordered alphabet of size σ , the *suffix array* $\text{SA}[1..n]$ stores the permutation of $\{1, \dots, n\}$ such that $\text{SA}[i]$ is the starting position of the i th lexicographically smallest suffix of S . The standard SA application is as a text index, in which S is the *text*: given any string $P[1..m]$, known as the *pattern*, the suffix array of S allows us to report all occ occurrences of P in S using only $\mathcal{O}(m \log n + \text{occ})$ operations [41].

We do a binary search in SA, which results in an interval $[s, e)$ of suffixes of S having P as a prefix. Then, $\text{SA}[s \dots e - 1]$ contains the starting positions of all occurrences of P in S . The SA is often augmented with the LCP array [41] storing the length of longest common prefixes of lexicographically adjacent suffixes. In this case, reporting all occ occurrences of P in S can be done in $\mathcal{O}(m + \log n + \text{occ})$ time [41] (see [18, 22, 47] for subsequent improvements). The suffix array can be constructed in $\mathcal{O}(n)$ time for an integer alphabet of size $\sigma = n^{\mathcal{O}(1)}$ [20]. Given the suffix array of S , we can compute the LCP array of S in $\mathcal{O}(n)$ time [32].

The *suffix tree* of S , which we denote by $\text{ST}(S)$, is the compacted trie of all the suffixes of S [52]. Assuming S ends with a unique terminating symbol, every suffix $S[i \dots n]$ of S is represented by a leaf node that we decorate with i . We refer to the set of indices stored at the leaf nodes in the subtree rooted at node v as the *leaf-list* of v , and we denote it by $LL(v)$. Each edge in $\text{ST}(S)$ is labeled with a substring of S such that the path from the root to the leaf annotated with index i spells the suffix $S[i \dots n]$. We refer to the substring of S spelled by the path from the root to node v as the *path-label* of v and denote it by $L(v)$. Given any pattern $P[1 \dots m]$, the suffix tree of S allows us to report all occ occurrences of P in S using only $\mathcal{O}(m \log \sigma + \text{occ})$ operations. We spell P from the root of $\text{ST}(S)$ (to access edges by the first letter of their label, we use binary search) until we arrive (if possible) at the first node v such that P is a prefix of $L(v)$. Then all occ occurrences of P in S are precisely $LL(v)$. The suffix tree can be constructed in $\mathcal{O}(n)$ time for an integer alphabet of size $\sigma = n^{\mathcal{O}(1)}$ [20]. To improve the query time to $\mathcal{O}(m + \text{occ})$ we use randomization to construct a perfect hash table [23] accessing edges by the first letter of their label in $\mathcal{O}(1)$ time.

We next show how to reduce GAPPED STRING INDEXING w. REPORTING to GAPPED SET INTERSECTION w. REPORTING. A query for P_1, P_2 and $[\alpha, \beta]$ corresponds to a GAPPED SET INTERSECTION w. REPORTING query for S_1, S_2 and $[\alpha, \beta]$, where S_1 is the set of occurrences of P_1 in S and S_2 is the set of occurrences of P_2 in S . An obvious strategy would be to preprocess all sets which correspond to leaves below a node in the suffix tree into a GAPPED SET INTERSECTION w. REPORTING data structure. The issue is that the total size of these sets can be $\Omega(n^2)$. However, any such set corresponds to a consecutive interval within the suffix array. Thus, the strategy is as follows: We store the suffix tree and the suffix array for S . We cover the suffix array in dyadic intervals and preprocess the resulting subarrays into the GAPPED SET INTERSECTION w. REPORTING data structure. In detail, let D be the set of dyadic intervals covering $[1, n]$. That is, D includes all intervals of the form $[1 + \kappa \cdot 2^j, (\kappa + 1) \cdot 2^j]$ for all $0 \leq \kappa \leq \lfloor \frac{n}{2^j} \rfloor - 1$ and $0 \leq j \leq \lfloor \log n \rfloor$. For any interval $[\gamma_1, \gamma_2] \in D$, we define a set containing the elements in $\text{SA}[\gamma_1 \dots \gamma_2]$. We preprocess all of these sets into the GAPPED SET INTERSECTION w. REPORTING data structure. The total size of the sets in the data structure is $\mathcal{O}(n \log n)$. For a query, we find the suffix array intervals (I_1, I_2) for P_1 and P_2 , respectively, in $\mathcal{O}(|P_1| + |P_2|)$ time, using standard pattern matching in the suffix tree. Now, let \mathcal{A} be the collection of sets corresponding to dyadic intervals covering I_1 and \mathcal{B} the collection of sets corresponding to dyadic intervals covering I_2 . We can find all pairs (i, j) of occurrences of P_1 and P_2 satisfying $j - i \in [\alpha, \beta]$ by querying GAPPED SET INTERSECTION w. REPORTING for $(A, B, [\alpha, \beta])$ for all $A \in \mathcal{A}$ and $B \in \mathcal{B}$.

In conclusion, we have shown the following:

► **Theorem 24.** *Assume there is a data structure for SHIFTED SET INTERSECTION with $s(N)$ space and $t(N)$ query time, where N is the input size, and which outputs a witness pair $(a, b) \in S_i \times S_j$ satisfying $a + s = b$ for a query (i, j, s) . Then there is a data structure for GAPPED STRING INDEXING w. REPORTING with $\tilde{\mathcal{O}}(n + s(n))$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + t(n) \cdot (\text{occ} + 1))$ query time, where n is the length of the input string and occ is the size of the output.*

The following two corollaries follow from Theorem 24 together with Theorem 6 and Lemma 14, respectively.

► **Corollary 25.** *For every $0 \leq \delta \leq 1$, there is a data structure for GAPPED STRING INDEXING W. REPORTING with $\tilde{\mathcal{O}}(n^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + n^\delta \cdot (\text{occ} + 1))$ query time, where occ is the size of the output.*

► **Corollary 26.** *For every $0 \leq \delta \leq 1$, there is a data structure for GAPPED STRING INDEXING W. REPORTING with $\tilde{\mathcal{O}}(n^{3-2\delta})$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + n^\delta \cdot (\text{occ} + 1))$ query time, where occ is the size of the output.*

Note that $n^{3-2\delta}$ is smaller than $n^{2-\delta/3}$ for $\delta > 3/5$. We have arrived at Theorem 1.

6 Jumbled Indexing

A solution to 3SUM INDEXING W. REPORTING implies solutions to other problems. As proof of concept, we show here the implications to JUMBLED INDEXING W. REPORTING: preprocess a string S of length n over an alphabet Σ into a compact data structure that facilitates efficient search for substrings of S whose characters have a specified histogram. For instance, $P = \text{acaacabd}$ has the histogram $h(P) = (4, 1, 2, 1)$: $h(P)[a] = 4$ because a occurs 4 times in P , $h(P)[b] = 1$, etc. The existence variant, JUMBLED INDEXING, answers the question of whether such a substring occurs in S or not. We show the following result:

► **Corollary 8.** *Given a string S of length n over an alphabet of size $\sigma = \mathcal{O}(1)$, for every $0 \leq \delta \leq 1$, there is a data structure for JUMBLED INDEXING W. REPORTING with $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{3\delta} \cdot (\text{occ} + 1))$ query time, where occ is the size of the output. The bounds are $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{3\delta})$ query time for the existence variant.*

Before proving the above statement, we will briefly overview the related literature.

Related work. Previous work on JUMBLED INDEXING has achieved the bounds of $\mathcal{O}(n^{2-\delta})$ space and $\mathcal{O}(m^{(2\sigma-1)\delta})$ query time [34], where m is the norm of the pattern and $\sigma = |\Sigma|$. Later, a data structure with $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{\delta(\sigma+1)/2})$ query time was presented [14].

For the special case of a binary alphabet, more efficient algorithms exist; namely, $\mathcal{O}(n)$ space and $\mathcal{O}(1)$ query time [16]. The data structure in [16] has a preprocessing time of $\mathcal{O}(n^2)$, which can be improved to $\tilde{\mathcal{O}}(n^{1.5})$ using the connection to min-plus-convolution [14, 15].

JUMBLED INDEXING has also seen interest from the lower-bound side, where [1] shows that if a data structure can report all the occ matches to a histogram query in $\mathcal{O}(n^{0.5-o(1)} + \text{occ})$ time, then it needs to use $\Omega(n^{2-o(1)})$ space, even for the special case of a binary alphabet. In fine-grained complexity, the problem has also received attention; [2] shows that under a 3SUM hardness assumption, for alphabets of size $\omega(1)$, we cannot get $\mathcal{O}(n^{2-\epsilon})$ preprocessing time and $\mathcal{O}(n^{1-\delta})$ query time for any $\epsilon, \delta > 0$. Furthermore, under the now refuted Strong 3SUM INDEXING conjecture, [26] gives conditional lower bounds, that are contradicted by our results. In particular, their conditional lower bound (conditioned on the now refuted Strong 3SUM INDEXING conjecture) argues against a $\mathcal{O}(n^{2-\frac{2(1-\alpha)}{\sigma-1-\alpha}-\Omega(1)})$ space and $\mathcal{O}(n^{1-\frac{1+\alpha(\sigma-3)}{\sigma-1-\alpha}-\Omega(1)})$ query time solution for any $0 \leq \alpha \leq 1$. Setting $\alpha = 0$ and $\sigma = 9$, this would imply that there does not exist a $\mathcal{O}(n^{2-\frac{1}{4}-\Omega(1)})$ space and $\mathcal{O}(n^{1-\frac{1}{8}-\Omega(1)})$ query time solution. However, setting $\delta = 1/4 + \epsilon$ for $\epsilon = 1/48$, we obtain an $\tilde{\mathcal{O}}(n^{2-\frac{1}{4}-\epsilon})$ space and $\tilde{\mathcal{O}}(n^{\frac{3}{4}+3\epsilon}) = \tilde{\mathcal{O}}(n^{\frac{6}{8}+\frac{1}{16}}) = \tilde{\mathcal{O}}(n^{1-\frac{1}{8}-\frac{1}{16}})$ query time, a contradiction. We now return back to the proof of Corollary 8:

Proof of Corollary 8. Let us first remind the reader of the well-known reduction to d -dimensional 3SUM INDEXING, as introduced by Chan and Lewenstein in [14]. The d -dimensional 3SUM INDEXING problem is defined as follows: Preprocess two sets A and B of N vectors from $[0, \dots, u-1]^d$ each such that we can efficiently answer queries of the following form: for some $c \in [0, \dots, u-1]^d$, decide if there exists $a \in A$ and $b \in B$ such that $c = a + b$.

We reduce JUMBLED INDEXING to σ -dimensional 3SUM INDEXING as follows. Let $\sigma = |\Sigma|$. Let A be the set of all histograms of prefixes of the input string S , and similarly, let B be the set of all histograms of suffixes of S . We thus have that the cardinality of these two sets is the length of the string S , i.e., $|A| = |B| = |S| = n$, and may thus build a σ -dimensional 3SUM INDEXING data structure for A and B . Additionally, we store the histogram $h(S)$ of S . For a query histogram P , compute $c = h(S) - P$, and query the σ -dimensional 3SUM INDEXING data structure for c . Any match a, b returned by the data structure corresponds to a histogram of the complement of some substring p whose histogram is P : a corresponds to the prefix of everything before p , and b corresponds to the suffix of everything after p .

Now, we note that there is a reduction from d -dimensional 3SUM INDEXING over a universe of size u to 3SUM INDEXING over a universe of size $\mathcal{O}(u^d)$. Our reduction works by doing the following transformation on sets A and B : define $A' = \{a_1 + a_2 \cdot u + a_3 \cdot u^2 + \dots + a_d \cdot u^{d-1} : a \in A\}$; and define B' in the same way. For a query c , define the query $c' = c_1 + c_2 \cdot u + c_3 \cdot u^2 \dots + c_d \cdot u^{d-1}$. Thus, by spacing out using u -factors, we ensure that any match to a query c to the sets A and B corresponds exactly to a match to the corresponding query c' to the sets A' and B' . With this reduction, the set size remains unchanged, that is, $|A| = |A'|$ and $|B| = |B'|$, however, we are now indexing with a universe of size $u' = \mathcal{O}(u^d)$.

Thus, finally, for JUMBLED INDEXING, let n be the length of the string S . Our reductions would result in a universe of size $u' = n^\sigma$. For constant σ this value u' is polynomial in n . Therefore, by Corollary 4, we obtain a data structure for JUMBLED INDEXING over constant sized alphabets with $\tilde{O}(n^{2-\delta})$ space and $\tilde{O}(n^{3\delta} \cdot (\text{occ} + 1))$ query time. ◀

7 Final Remarks

Our solutions show new and interesting relations between GAPPED STRING INDEXING, SHIFTED SET INTERSECTION, and 3SUM INDEXING; in particular, we contribute new trade-offs for 3SUM INDEXING w. REPORTING. Chan showed that the 3SUM problem has direct applications in computational geometry [13]; it would be interesting to see if our data structure yields improved bounds for the data structure versions of these geometric problems.

References

- 1 Peyman Afshani, Ingo van Duijn, Rasmus Killmann, and Jesper Sindahl Nielsen. A lower bound for jumbled indexing. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 592–606, 2020. doi:10.1137/1.9781611975994.36.
- 2 Amihod Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 114–125, 2014. doi:10.1007/978-3-662-43948-7_10.
- 3 Boris Aronov, Jean Cardinal, Justin Dallant, and John Iacono. A general technique for searching in implicit sets via function inversion. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 215–223, 2024. doi:10.1137/1.9781611977936.20.

- 4 Johannes Bader, Simon Gog, and Matthias Petri. Practical variable length gap pattern matching. In *Experimental Algorithms - 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*, pages 1–16, 2016. doi:10.1007/978-3-319-38851-9_1.
- 5 Philip Bille and Inge Li Gørtz. Substring range reporting. *Algorithmica*, 69(2):384–396, 2014. doi:10.1007/S00453-012-9733-4.
- 6 Philip Bille, Inge Li Gørtz, Moshe Lewenstein, Solon P. Pissis, Eva Rotenberg, and Teresa Anna Steiner. Gapped string indexing in subquadratic space and sublinear query time, 2022. doi:10.48550/ARXIV.2211.16860.
- 7 Philip Bille, Inge Li Gørtz, Max Rishøj Pedersen, and Teresa Anna Steiner. Gapped indexing for consecutive occurrences. *Algorithmica*, 85(4):879–901, 2023. doi:10.1007/S00453-022-01051-6.
- 8 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. String indexing for patterns with wildcards. *Theory Comput. Syst.*, 55(1):41–60, 2014. doi:10.1007/S00224-013-9498-4.
- 9 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and David Kofoed Wind. String matching with variable length gaps. *Theor. Comput. Sci.*, 443:25–34, 2012. doi:10.1016/J.TCS.2012.03.029.
- 10 Philip Bille and Mikkel Thorup. Regular expression matching with multi-strings and intervals. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1297–1308, 2010. doi:10.1137/1.9781611973075.104.
- 11 Philipp Bucher and Amos Bairoch. A generalized profile syntax for biomolecular sequence motifs and its function in automatic sequence interpretation. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology, August 14-17, 1994, Stanford University, Stanford, California, USA*, pages 53–61, 1994. URL: <http://www.aaai.org/Library/ISMB/1994/ismb94-007.php>.
- 12 Manuel Cáceres, Simon J. Puglisi, and Bella Zhukova. Fast indexes for gapped pattern matching. In *SOFSEM 2020: Theory and Practice of Computer Science - 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20-24, 2020, Proceedings*, pages 493–504, 2020. doi:10.1007/978-3-030-38919-2_40.
- 13 Timothy M. Chan. More logarithmic-factor speedups for 3SUM, (median, +)-convolution, and some geometric 3SUM-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020. doi:10.1145/3363541.
- 14 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 31–40, 2015. doi:10.1145/2746539.2746568.
- 15 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1529–1542, 2022. doi:10.1145/3519935.3520057.
- 16 Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Searching for jumbled patterns in strings. In *Proceedings of the Prague Stringology Conference 2009, Prague, Czech Republic, August 31 - September 2, 2009*, pages 105–117, 2009. URL: <http://www.stringology.org/event/2009/p10.html>.
- 17 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100, 2004. doi:10.1145/1007352.1007374.
- 18 Richard Cole, Tsvi Kopelowitz, and Moshe Lewenstein. Suffix trays and suffix trists: Structures for faster text indexing. *Algorithmica*, 72(2):450–466, 2015. doi:10.1007/S00453-013-9860-6.
- 19 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

- 20 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102.
- 21 Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999. doi:10.1137/S0097539795280512.
- 22 Johannes Fischer and Pawel Gawrychowski. Alphabet-dependent string searching with wexponential search trees. In *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings*, pages 160–171, 2015. doi:10.1007/978-3-319-19929-0_14.
- 23 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.
- 24 Kimmo Fredriksson and Szymon Grabowski. Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance. *Inf. Retr.*, 11(4):335–357, 2008. doi:10.1007/S10791-008-9054-Z.
- 25 Pawel Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted ancestors in suffix trees. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 455–466, 2014. doi:10.1007/978-3-662-44777-2_38.
- 26 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, pages 421–436, 2017. doi:10.1007/978-3-319-62127-2_36.
- 27 Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan. Data structures meet cryptography: 3SUM with preprocessing. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 294–307, 2020. doi:10.1145/3357713.3384342.
- 28 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.
- 29 Tuukka Haapasalo, Panu Silvasti, Seppo Sippu, and Eljas Soisalon-Soininen. Online dictionary matching with variable-length gaps. In *Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings*, pages 76–87, 2011. doi:10.1007/978-3-642-20662-7_7.
- 30 Kay Hofmann, Philipp Bucher, Laurent Falquet, and Amos Bairoch. The PROSITE database, its status in 1999. *Nucleic Acids Res.*, 27(1):215–219, 1999. doi:10.1093/NAR/27.1.215.
- 31 Costas S. Iliopoulos and M. Sohel Rahman. Indexing factors with gaps. *Algorithmica*, 55(1):60–70, 2009. doi:10.1007/S00453-007-9141-3.
- 32 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial Pattern Matching, 12th Annual Symposium, CPM 2001 Jerusalem, Israel, July 1-4, 2001 Proceedings*, pages 181–192, 2001. doi:10.1007/3-540-48194-X_17.
- 33 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.
- 34 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Efficient indexes for jumbled pattern matching with constant-sized alphabet. *Algorithmica*, 77(4):1194–1215, 2017. doi:10.1007/S00453-016-0140-0.
- 35 Tsvi Kopelowitz and Robert Krauthgamer. Color-distance oracles and snippets. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, pages 24:1–24:10, 2016. doi:10.4230/LIPICS.CPM.2016.24.
- 36 Tsvi Kopelowitz and Ely Porat. The strong 3SUM-INDEXING conjecture is false. *CoRR*, abs/1907.11206, 2019. arXiv:1907.11206.
- 37 Paul R. Kroegeer. *Analyzing Grammar: An Introduction*. Cambridge University Press, Cambridge, 2005.

- 38 Moshe Lewenstein. Indexing with gaps. In *String Processing and Information Retrieval, 18th International Symposium, SPIRE 2011, Pisa, Italy, October 17-21, 2011. Proceedings*, pages 135–143, 2011. doi:10.1007/978-3-642-24583-1_14.
- 39 Moshe Lewenstein, J. Ian Munro, Venkatesh Raman, and Sharma V. Thankachan. Less space: Indexing for queries with wildcards. *Theor. Comput. Sci.*, 557:120–127, 2014. doi:10.1016/j.tcs.2014.09.003.
- 40 Moshe Lewenstein, Yakov Nekrich, and Jeffrey Scott Vitter. Space-efficient string indexing for wildcard pattern matching. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 506–517, 2014. doi:10.4230/LIPICS.STACS.2014.506.
- 41 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.
- 42 Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.
- 43 Gerhard Mehlau and Gene Myers. A system for pattern matching applications on biosequences. *Bioinformatics*, 9(3):299–314, 1993. doi:10.1093/bioinformatics/9.3.299.
- 44 Gary Miner, Dursun Delen, John Elder, Andrew Fast, Thomas Hill, and Robert A. Nisbet. *Practical Text Mining and Statistical Analysis for Non-structured Text Data Applications*. Academic Press, Boston, 2012.
- 45 Michele Morgante, Alberto Policriti, Nicola Vitacolonna, and Andrea Zuccolo. Structured motifs search. *J. Comput. Biol.*, 12(8):1065–1082, 2005. doi:10.1089/cmb.2005.12.1065.
- 46 Eugene W. Myers. Approximate matching of network expressions with spacers. *J. Comput. Biol.*, 3(1):33–51, 1996. doi:10.1089/cmb.1996.3.33.
- 47 Gonzalo Navarro and Yakov Nekrich. Time-optimal top-k document retrieval. *SIAM J. Comput.*, 46(1):80–113, 2017. doi:10.1137/140998949.
- 48 Gonzalo Navarro and Mathieu Raffinot. Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. *J. Comput. Biol.*, 10(6):903–923, 2003. doi:10.1089/106652703322756140.
- 49 Pierre Peterlongo, Julien Allali, and Marie-France Sagot. Indexing gapped-factors using a tree. *Int. J. Found. Comput. Sci.*, 19(1):71–87, 2008. doi:10.1142/S0129054108005541.
- 50 Solon P. Pissis. MoTeX-II: structured MoTif eXtraction from large-scale datasets. *BMC Bioinform.*, 15:235, 2014. doi:10.1186/1471-2105-15-235.
- 51 Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. doi:10.1145/363347.363387.
- 52 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.
- 53 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Inf. Process. Lett.*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.
- 54 Jens Willkomm, Martin Schäler, and Klemens Böhm. Accurate cardinality estimation of co-occurring words using suffix trees. In *Database Systems for Advanced Applications - 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11-14, 2021, Proceedings, Part II*, pages 721–737, 2021. doi:10.1007/978-3-030-73197-7_50.

A A Linear-Space Solution for Gapped Indexing

The linear-space solution uses a standard linear-time pattern matching algorithm (e.g., the Knuth-Morris-Pratt algorithm [33]), which, given two strings S and P , it outputs all positions in S where P occurs, in sorted order. We run this algorithm two times: on P_1 and S ; and on P_2 and S . The two sorted lists of positions are scanned from left to right using a two-finger approach: We start with the first position i_1 where P_1 occurs in S , and scan the list of P_2 occurrences until we find the first one that is at least $i_1 + \alpha$. Then we scan and output all

pairs until we arrive at a position of P_2 which is larger than $i_1 + \beta$. Then we advance to the next position i_2 of P_1 , and from the last position we considered in the list for P_2 , we scan first backward and then forward to output all positions in $[i_2 + \alpha, i_2 + \beta]$. Every position we scan is charged to an occurrence we output, thus we achieve a running time of $\mathcal{O}(n + \text{occ})$.

B A Near-Quadratic-Space Solution for Gapped Indexing

Let us start with a data structure of $\mathcal{O}(n^3)$ space and $\mathcal{O}(|P_1| + |P_2| + \text{occ})$ query time: We store, for each pair (v, w) of nodes in the suffix tree of S , the set of distances that appear between the set of occurrences defined by v and w . To answer a query, we search for P_1 and P_2 in the suffix tree to locate the corresponding pair of nodes and then report the distances within the gap range $[\alpha, \beta]$. Since the number of distinct distances is at most $n - 1$ this leads to a solution using $\mathcal{O}(n^3)$ space and $\mathcal{O}(|P_1| + |P_2| + \text{occ})$ query time, where occ is the size of the output. A more sophisticated approach can reduce the space to $\tilde{\mathcal{O}}(n^2)$ at the cost of increasing the query time by polylogarithmic factors. Namely, the idea is to consider subarrays of the suffix array of S [41] corresponding to the dyadic intervals on $[1, n]$, and store, for every possible distance d , all pairs of subarrays (A, B) containing elements at distance d . Further, we store a list of all (a, b) , $a \in A$ and $b \in B$ such that $b - a = d$. Since the total size of the subarrays is $\mathcal{O}(n \log n)$, the total number of pairs of elements is $\mathcal{O}(n^2 \log^2 n)$, thus this solution can be stored using $\tilde{\mathcal{O}}(n^2)$ space. We also store the suffix tree of S , which takes $\Theta(n)$ extra space. To answer a query, we search for P_1 and P_2 in the suffix tree to find their suffix array intervals: their starting positions in S sorted lexicographically with respect to the corresponding suffixes. We cover each interval in dyadic intervals and use the precomputed solution for any pair consisting of a subarray in the cover of P_1 's interval and a subarray in the cover of P_2 's interval. This results in query time $\mathcal{O}(|P_1| + |P_2| + \log^2 n \cdot (\text{occ} + 1))$.

C Smallest Shift

When studying the problem of deciding whether a given shift incurs an intersection between sets, a natural next question is the optimization (minimization) variant of the problem: *what is the smallest shift that yields an intersection?* We formally define this problem next.

SMALLEST SHIFT

Preprocess: A collection of k sets S_1, \dots, S_k of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \dots, u\}$.

Query: Given i, j , output the smallest s such that there exists $a \in S_i$ and $b \in S_j$ with $a + s = b$.

Note that the problem is symmetric in i and j in the sense that finding the smallest positive shift transferring S_i to intersect S_j is equivalent to finding the largest negative shift transferring S_j to intersect S_i . Note also that by constructing a predecessor/successor data structure [53] at preprocessing time, we may answer the query in $\tilde{\mathcal{O}}(|S_i|)$ time: simply perform a successor (or predecessor) query in S_j for each element $a \in S_i$, and report the smallest element-successor distance.

Thus, we are safely able to handle SMALLEST SHIFT queries in $\mathcal{O}(\sqrt{N} \log \log N)$ time in all cases where either S_i or, because of symmetry, S_j , is of size at most \sqrt{N} . Remaining is the case where both sets are of size larger than \sqrt{N} . Here, however, we note that at most $\mathcal{O}(\sqrt{N})$ such sets can exist, so we can tabulate all answers in $\mathcal{O}(\sqrt{N} \cdot \sqrt{N}) = \mathcal{O}(N)$ space.

In other words, we have the following solution:

Preprocessing. Store for each i whether $|S_i| \leq \sqrt{N}$. Let l denote the number of sets L_1, \dots, L_l of size larger than \sqrt{N} . For each set S_i with $|S_i| > \sqrt{N}$ store its index in the list of large sets L_1, \dots, L_l .

- For all i , with $|S_i| \leq \sqrt{N}$, compute and store predecessor $\text{pre}(S_i)$ and successor $\text{suc}(S_i)$ data structures for the set S_i .
- Store an $l \times l$ table in which the (i, j) th entry stores the smallest shift s such that there exists $a \in L_i$ and $b \in L_j$ with $a + s = b$.

Query i, j .

- If $|S_i| \leq \sqrt{N}$, perform $|S_i|$ successor queries in $\text{suc}(S_j)$, and return the smallest difference.
- If $|S_j| \leq \sqrt{N}$, similarly, perform $|S_i|$ predecessor queries in $\text{pre}(S_i)$.
- If $|S_i| > \sqrt{N}$ and $|S_j| > \sqrt{N}$, use the precomputed information to output the smallest shift.

► **Proposition 27.** *The above data structure for SMALLEST SHIFT uses $\mathcal{O}(N)$ space and has $\tilde{\mathcal{O}}(\sqrt{N})$ query time. Moreover, the data structure can be constructed in $\mathcal{O}(N\sqrt{N})$ time.*

Proof. We construct predecessor and successor queries for all sets in $\sum_i |S_i| \log \log |S_i| = \mathcal{O}(N \log \log N)$ total time and $\sum_i |S_i| = \mathcal{O}(N)$ space, with $\mathcal{O}(\log \log N)$ query time [53], surmounting to an $\mathcal{O}(\sqrt{N} \log \log N)$ query time when either S_i or S_j is smaller than \sqrt{N} .

For constructing the $l \times l$ table, we note that $l \leq \sqrt{N}$. Since the (i, j) th entry of the table can be computed in time $\mathcal{O}(|L_i| + |L_j|)$ by a merge-like traversal of the sorted respective sets, the total preprocessing time becomes:

$$\begin{aligned} \sum_i \sum_{j \neq i} (|L_i| + |L_j|) &\leq \sum_i (\sqrt{N}|L_i| + \sum_j |L_j|) \leq \sum_i (\sqrt{N}|L_i| + N) \\ &= \sqrt{N} \sum_i |L_i| + \sum_i N = \mathcal{O}(\sqrt{N}N). \end{aligned}$$

Here, we are using that $\sum_i |S_i| = N$ and that there are $l \leq \sqrt{N}$ large sets. This $\mathcal{O}(N\sqrt{N})$ term dominates the $\sum_i \tilde{\mathcal{O}}(|S_i|)$ terms needed to sort the sets and construct predecessor/successor data structures for each of them. ◀

Contributions to the Domino Problem: Seeding, Recurrence and Satisfiability

Nicolás Bitar   

Université Paris-Saclay, CNRS, LISN, 91190 Gif-sur-Yvette, France

Abstract

We study the seeded domino problem, the recurring domino problem and the k -SAT problem on finitely generated groups. These problems are generalization of their original versions on \mathbb{Z}^2 that were shown to be undecidable using the domino problem. We show that the seeded and recurring domino problems on a group are invariant under changes in the generating set, are many-one reduced from the respective problems on subgroups, and are positive equivalent to the problems on finite index subgroups. This leads to showing that the recurring domino problem is decidable for free groups. Coupled with the invariance properties, we conjecture that the only groups in which the seeded and recurring domino problems are decidable are virtually free groups. In the case of the k -SAT problem, we introduce a new generalization that is compatible with decision problems on finitely generated groups. We show that the subgroup membership problem many-one reduces to the 2-SAT problem, that in certain cases the k -SAT problem many one reduces to the domino problem, and finally that the domino problem reduces to 3-SAT for the class of scalable groups.

2012 ACM Subject Classification Mathematics of computing \rightarrow Combinatoric problems

Keywords and phrases Tilings, Domino problem, SAT, Computability, Finitely generated groups

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.17

Acknowledgements I want to thank Nathalie Aubrun for her helpful comments, reviews and discussions.

1 Introduction

The domino problem, as originally formulated, is the decision procedure of determining if a given finite set of unit squares with colored edges, known as Wang tiles, can tile the infinite plane while respecting an adjacency condition: two squares can be placed next to each other if their shared edge has the same color. This problem was introduced by Wang to study the decidability of the $\forall\exists\forall$ fragment of first order logic [47], who conjectured that the domino problem was decidable. It turns out that this is not the case; a now classic result by Berger states that this problem is undecidable [10]. Since then, many proofs of the undecidability of the problem have been found (see [28]).

Perhaps one of the fundamental features of the domino problem is its usefulness in proving the undecidability of many decision problems, ranging from problems in symbolic dynamics such as the infinite snake problem [1] and the injectivity and surjectivity of two-dimensional cellular automata [30, 31], to problems from other areas such as the k -SAT problem on \mathbb{Z}^2 [17], the spectral gap problem of quantum many-body systems [15] and translation monotilings [19]. In fact, the Wang tiling model can be seen as a natural model to encode computation and prove complexity lower bounds [46].

In recent years, the domino problem has found new life in the context of symbolic dynamics over finitely generated groups [2]. The aim has been to establish which algebraic conditions make the problem of deciding whether the group is tileable subject to a finite number of local constraints – i.e., the domino problem on the group – undecidable. This project has culminated in a conjecture stating that the class of groups with decidable



© Nicolás Bitar;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 17; pp. 17:1–17:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



domino problem is the class of virtually free groups. This same turn towards generalizing to groups has been made for different problems, such as aperiodic tilings [42] and domino snake problems [3]. This article follows the same path for three different decision problems: the seeded domino problem, the recurrent domino problem, and the k -SAT problem. We introduce generalizations of these problems for finitely generated groups to understand how the algebraic and computational properties of the underlying group influence the decidability and to obtain new tools to show the undecidability of problems on finitely generated groups.

Variants of Tiling Problems

Variations on the domino problem have been present since its conception. This is the case of the seeded domino problem, whose undecidability was established even before the domino problem's [29, 11]. In the years since many more variations have been introduced: the periodic domino problem [20, 25], domino snake problems [39], the recurrent domino problem [21], the aperiodic domino problem [12, 18], and even a variant where the underlying structure is fixed to be a geometric tiling of the plane [23].

In this article, we generalize the seeded and recurrent problems to the context of finitely generated groups. The seeded version asks, given an alphabet, a finite set of local rules and a target letter from the alphabet, if there exists a coloring of the group subject to the local rules where the target letter appears. The recurrent version has the same input but asks if such a configuration exists where the target letter appears infinitely often. We will begin by formally introducing these problems and establishing connections to the domino problem: the problem many-one reduces to both variants (Section 2). Next, we show that the decidability of both problems is independent of the chosen generating set for the group, that the problems many-one reduce from subgroups and are in fact equivalent in the case of finite index subgroups (Section 3). Furthermore, we show that the recurrent problem is decidable on free groups, which paired with the domino conjecture and inheritance properties, allows us to state the following extension of the domino conjecture

- **Conjecture 1.** *Let G be a finite generated group. The following are equivalent,*
- *G is virtually free,*
 - *the domino problem on G is decidable,*
 - *the seeded domino problem on G is decidable,*
 - *the recurrent domino problem on G is decidable.*

k -SAT and the Limit of Polynomial Time Problems

The k -SAT problem on groups was introduced by Freedman in [17]. The idea of the generalization was to extend the difference between 2-SAT and 3-SAT, which are in \mathbf{P} and \mathbf{NP} respectively, to an infinite context making the former problem decidable and the latter undecidable. This is inserted into the broader program outlined in [16] that searches to separate the complexity classes \mathbf{P} and \mathbf{NP} by limit processes, the idea being that limiting behaviors of polynomial time problems should be decidable.

In this article, we slightly alter the generalization proposed by Freedman to make the decision problem compatible with finitely generated groups (Section 4). Similar generalizations have been made for other classic decision problems, such as Post's correspondence problem [38, 14, 13]. We show that the subgroup membership problem of the group many-one reduces to the complement of the 2-SAT problem and that in the class of groups where the former is decidable, the k -SAT problem many-one reduces to the domino problem for all $k > 1$. In conjunction with the work of Piantadosi [41] and the domino problem's inheritance

properties, this result implies that the k -SAT problem is decidable for virtually free groups. We introduce the class of scalable groups to find an equivalence between both decision problems. A finitely generated group is scalable if it contains a proper finite index subgroup that is isomorphic to the group. We show that for this class of groups, the domino problem many-one reduces to the 3-SAT problem. The proof of this result is inspired by techniques from [17] that are adapted to better suit our definition of the decision problem.

We begin the article by introducing some preliminary notions from computability and symbolic dynamics in Section 1. Section 2 introduces the domino problem, the seeded and recurrent variants, and some basic properties. In Section 3, we establish the invariance of the decidability of the seeded and recurrent problems under changes in the generating set of the group, in addition to some inheritance properties, most notably subgroups. We also prove that the recurrent domino problem is decidable for free groups and show that the seeded and recurrent problems are subject to the same conjecture as the normal one. Section 4 is devoted to the study of the k -SAT problem on finitely generated groups. We formally introduce the decision problem as well as its connection to the subgroup membership problem and a reduction to the domino problem. We finally present the class of scalable groups and show that for them, the domino problem many-one reduces to the 3-SAT problem.

2 Preliminaries

Given a finite alphabet A , we denote the set of words of length n by A^n , the set of words of length less or equal to n by $A^{\leq n}$, and the set of all finite words over A by A^* , including the empty word ε . The length of a word w is denoted by $|w|$. We denote the free group on n generators by \mathbb{F}_n . Throughout the article, G will be an infinite group.

2.1 Computability and Group Theory

We quickly recall some notions from computability theory and combinatorial group theory that will be needed in the article. See [45] for a reference on computability and reductions, and see [35] for a reference on combinatorial group theory.

- **Definition 2.** Let $L \subseteq A^*$ and $L' \subseteq B^*$ be two languages. We say,
- L many-one reduces to L' , denoted $L \leq_m L'$, if there exists a computable function $f : A^* \rightarrow B^*$ such that $w \in L$ if and only if $f(w) \in L'$ for every w .
 - L positive-reduces to L' , denoted $L \leq_p L'$ if for any w one can compute finitely many finite sets $F_1(w), \dots, F_n(w)$ such that $w \in L$ if and only if there exists $i \in \{1, \dots, n\}$ such that $F_i(w) \subseteq L'$.

For both notions of reducibility, the induced notion of equivalence will be denoted by $L \equiv_* L'$, meaning $L \leq_* L'$ and $L' \leq_* L$.

Notice that many-one reducibility implies positive-reducibility. The complement of a decision problem D , denoted $\text{co}D$, is the set of all “no” instances of D .

Given G a finitely generated group (f.g.) and S a finite generating set, elements in the group are represented as words over the alphabet $S \cup S^{-1}$ through the evaluation function $w \mapsto \bar{w}$. Two words w and v represent the same element when $\bar{w} = \bar{v}$, and we denote it by $w =_G v$. We say a word is *reduced* if it contains no factor of the form ss^{-1} or $s^{-1}s$ with $s \in S$. The length of an element $g \in G$ with respect to S , denoted $|g|_S$, is the length of a shortest word $w \in (S \cup S^{-1})^*$ such that $g = \bar{w}$. A group is *virtually free* if it contains a finite index subgroup isomorphic to a free group.

2.2 Subshifts of Finite Type

Let A be a finite alphabet and G a finitely generated group. The *full-shift* on A is the set of configurations $A^G = \{x: G \rightarrow A\}$. This space is acted upon by G in the form of left translations: given $g \in G$ and $x \in A^G$,

$$g \cdot x(h) = x(g^{-1}h).$$

Let F be a finite subset of G . We call $p \in A^F$ a *pattern* of support F . We say a pattern p appears in a configuration $x \in A^G$ if there exists $g \in G$ such that $p(h) = x(gh)$ for all $h \in F$. The *cylinder* defined by a pattern $p \in A^F$ at $g \in G$ is given by

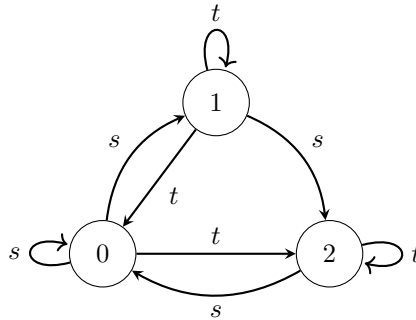
$$[p]_g = \{x \in A^G: \forall h \in F, x(gh) = p(h)\}.$$

Given a set of patterns \mathcal{F} , we define the G -*subshift* $X_{\mathcal{F}}$ as the set of configurations where no pattern from \mathcal{F} appears. That is,

$$X_{\mathcal{F}} = \{x \in A^G: \forall p \in \mathcal{F}, p \text{ does not appear in } x\} = A^G \setminus \bigcap_{g \in G, p \in \mathcal{F}} [p]_g.$$

If \mathcal{F} is finite, we say $X_{\mathcal{F}}$ is a G -*subshift of finite type* (G -SFT). We will simply write SFT when the group is clear from context.

Let S be a finite generating set for G . We say a pattern p is *nearest neighbor* if its support is given by $\{1_G, s\}$ with $s \in S$. We will denote nearest neighbor patterns through tuples (a, b, s) representing $p(1_G) = a$ and $p(s) = b$. A subshift defined by a set of nearest neighbor forbidden patterns is known as a nearest neighbor subshift. These subshifts are necessarily SFTs. Given a set of nearest neighbor patterns \mathcal{F} , we define its corresponding tileset graph, $\Gamma_{\mathcal{F}}$, by the set of vertices A , and edges given by $(a, b, s) \notin \mathcal{F}$, where a is its initial vertex, b its final vertex and s its label (see Figure 1 for an example).



■ **Figure 1** An example of a tileset graph $\Gamma_{\mathcal{F}}$ for the alphabet $\{0, 1, 2\}$ and a group with generators s and t . Edges present in the graph are exactly those that are not in \mathcal{F} , for example $(2, 1, t) \in \mathcal{F}$.

These graphs will help us in Section 4.2 when working with free groups. It is a well known fact [33] that nearest neighbor SFTs over \mathbb{Z} are characterized as the set of bi-infinite walks on their corresponding tileset graph.

3 The Domino Problem and its Variants

We begin with a formal definition of the domino problem that generalizes the original formulation with Wang tiles.

► **Definition 3.** *Let G be a finitely generated group and S a finite generating set. The domino problem on G with respect to S is the decision problem that, given an alphabet A and a finite set of nearest neighbor forbidden patterns \mathcal{F} , determines if the corresponding subshift $X_{\mathcal{F}}$ is empty. We denote this problem by $\text{DP}(G, S)$.*

It has been shown that the decidability of the problem is invariant under changes in the generating set, that is, if S_1 and S_2 are two finite generating sets for G , then $\text{DP}(G, S_1) \equiv_m \text{DP}(G, S_2)$. We can therefore talk about *the* domino problem on G , denoted $\text{DP}(G)$. The problem also satisfies many inheritance properties: both the domino problem of a finitely generated subgroup and the domino problem of a quotient by a finitely generated kernel many-one reduce to the domino problem of the group. Furthermore, the word problem of the group many-one reduces to the complement of the domino problem. Proofs of these facts can be found in [2].

A particularly important property enjoyed by the problem, is that it can be expressed in the monadic second order (MSO) logic of the group's Cayley graph [6]. Coupled with the fact that virtually free groups have decidable MSO logic [37, 32], this implies that the domino problem is decidable for virtually free groups. It is possible to go even further, this characterization of virtually free groups tells us that if a group is not virtually free its Cayley graphs contain arbitrarily large grids as minors [44]. This prompted Ballier and Stein [7] to state the following conjecture.

► **Conjecture 4 (The Domino Conjecture).** *Let G be a finitely generated group. Then $\text{DP}(G)$ is decidable if and only if G is virtually free.*

Since then, many classes of groups have been shown to satisfy the conjecture, such as Baumslag-Solitar groups [5], polycyclic groups [26], hyperbolic groups [9], Artin groups [4], direct products of two infinite groups [27], among others.

3.1 Seeded Domino Problem

Perhaps the most natural variant of the domino problem is its seeded version. In fact, it was introduced simultaneously to the original problem [47] and, as previously mentioned, was shown to be undecidable on \mathbb{Z}^2 before the domino problem [11, 29].

► **Definition 5.** *Let G be a finitely generated group and S a finite generating set. The seeded domino problem on G with respect to S is the decision problem that, given an alphabet A , a finite set of nearest neighbor forbidden patterns \mathcal{F} and a letter $a_0 \in A$, determines if there exists $x \in X_{\mathcal{F}}$ such that $x(1_G) = a_0$. We denote the decision problem by $\text{SDP}(G, S)$.*

As its definition suggests, this problem is computationally harder than the unseeded version: for a set of nearest neighbor forbidden patterns \mathcal{F} over the alphabet A , we create an instance of the seeded domino problem per letter.

► **Lemma 6.** *Let G be a finitely generated group and S a finite generating set. Then, $\text{DP}(G, S) \leq_p \text{SDP}(G, S)$.*

Just as the domino problem, there is a behavioral jump from the one-dimensional case to the two-dimensional case. Using the fact that nearest neighbor \mathbb{Z} -SFTs are defined as bi-infinite walks on a finite graph, $\text{SDP}(\mathbb{Z}, \{t\})$ is decidable. This difference in computability prompts the study of this problem on finitely generated groups. In fact, the problem can be shown to be decidable on the entire class of virtually free groups. Just as the domino problem, the seeded version can be expressed in monadic second order logic (see [8]), making the problem decidable in this class.

3.2 Recurring Domino Problem

The recurring domino problem was originally introduced by Harel as a natural decision problem that is highly undecidable, in order to find other highly undecidable problems [21]. He showed that in \mathbb{Z}^2 the problem is not only undecidable, but it is beyond the arithmetical hierarchy: it is Σ_1^1 -complete [22]. We expand the problem's definition to finitely generated groups.

► **Definition 7.** *Let G be a finitely generated group and S a finite generating set. The recurrent domino problem on G with respect to S is the decision problem that, given an alphabet A , a finite set of nearest neighbor forbidden patterns \mathcal{F} and a letter $a_0 \in A$, determines if there exists $x \in X_{\mathcal{F}}$ such that the set $\{g \in G : x(g) = a_0\}$ is infinite. We denote the decision problem by $\text{RDP}(G, S)$.*

As was the case with the seeded variant, this problem is computationally harder than the standard domino problem.

► **Lemma 8.** *Let G be a finitely generated group and S a finite generating set. Then, $\text{DP}(G, S) \leq_p \text{RDP}(G, S)$.*

Proof. Let \mathcal{F} be a set of nearest neighbor patterns for $\text{DP}(G, S)$. Notice that this is already part of the input for the recurring version, we simply create an instance for $\text{RDP}(G, S)$ for each of the letters of the alphabet. Because G is infinite, if the subshift defined by \mathcal{F} is non-empty, then at least one letter is forced to repeat itself infinitely often. ◀

Nevertheless, the behavioral jump that occurs between \mathbb{Z} and \mathbb{Z}^2 for the original problem is still present.

► **Proposition 9.** *$\text{RDP}(\mathbb{Z}, \{t\})$ is decidable.*

Proof. Let \mathcal{F} be a finite set of nearest neighbor forbidden patterns and $a_0 \in A$. Recall that we can define a graph $\Gamma_{\mathcal{F}}$, that is effectively constructible from \mathcal{F} , such that configurations on $X_{\mathcal{F}}$ correspond exactly with bi-infinite walks on $\Gamma_{\mathcal{F}}$. Therefore, to decide our problem we simply have to search for a simple cycle on $\Gamma_{\mathcal{F}}$ that is based at a_0 . If there is such a cycle, $c = a_0 a_1 a_2 \dots a_n a_0$ with $a_i \in A$, we define the periodic configuration $x = (a_0 a_1 a_2 a_3 \dots a_n)^\infty \in X_{\mathcal{F}}$. If, on the other hand, there exists a configuration $y \in X_{\mathcal{F}}$ on which a_0 appears infinitely often; take two consecutive occurrences of a_0 , say $y(k) = y(k') = a_0$ with $k < k'$. Then, because configurations correspond to bi-infinite walks, there is a cycle on $\Gamma_{\mathcal{F}}$ given by $c' = a_0 y(k+1) y(k+2) \dots y(k'-1) a_0$. As searching for simple cycles on a finite graph is computable, our problem is decidable. ◀

4 Properties for Seeded and Recurring Variants

4.1 General Inheritance Properties

Let us try and recover some inheritance properties enjoyed by the standard domino problem for the two variants, starting by the invariance under changing generating sets. We use strategies and procedures used to prove the corresponding results for the normal problem, as done in [2]. We begin by making use of pattern codings, which are a computationally tractable way of defining forbidden patterns whose support is not $\{1_G, s\}$.

► **Definition 10.** *Let G be a f.g. group, S a finite set of generators and A a finite alphabet. A pattern coding c is a finite set of tuples $c = \{(w_i, a_i)\}_{i \in I}$, where $w_i \in (S \cup S^{-1})^*$ and $a_i \in A$.*

Given a set of pattern codings \mathcal{C} , we define its corresponding subshift as:

$$X_{\mathcal{C}} = A^G \setminus \bigcup_{g \in G} \bigcap_{\substack{c \in \mathcal{C} \\ (w,a) \in c}} [a]_{gw}.$$

► **Definition 11.** *The seeded (recurrent) emptiness problem on G with respect to S asks if, given \mathcal{C} a set of pattern codings and $a_0 \in A$, there exists $x \in X_{\mathcal{C}}$ such that $x(1_G) = a_0$ (resp. a_0 appears infinitely often in x , that is, $|\{g \in G : x(g) = a_0\}|$ is infinite).*

Let us denote this problem by $\text{SEP}(G, S)$ (resp. $\text{REP}(G, S)$).

► **Lemma 12.** *Let G be a f.g. group along with two finite generating sets S_1 and S_2 . Then,*

- $\text{SDP}(G, S_1) \equiv_p \text{SDP}(G, S_2)$,
- $\text{RDP}(G, S_1) \equiv_p \text{RDP}(G, S_2)$.

Proof. We begin by noticing that we can re-write a pattern coding into any other generating set. This means that $\text{SEP}(G, S_1) \equiv_m \text{SEP}(G, S_2)$. We therefore just need to show that $\text{SDP}(G, S) \equiv_p \text{SEP}(G, S)$.

It is straight forward to re-write nearest neighbor patterns as pattern codings: each pattern (a, b, s) becomes $c = \{(\varepsilon, a), (s, b)\}$. Thus, we have the reduction $\text{SDP}(G, S) \leq_m \text{SEP}(G, S)$. Let us now focus on proving that $\text{SEP}(G, S)$ positive-reduces to $\text{SDP}(G, S)$. Given a set of pattern codings \mathcal{C} and a letter a_0 , we can compute both

$$N = \max_{c \in \mathcal{C}} \max_{(w,a) \in c} |w|,$$

and a new alphabet, \hat{A} , consisting of colorings of words of length at most N with no pattern from \mathcal{C} :

$$\hat{A} = \{\phi : S^{\leq N} \rightarrow A : \forall c \in \mathcal{C}, \exists (w, a) \in c, \phi(w) \neq a\}.$$

In addition, we are able to compute a set of forbidden patterns over \hat{A} denoted by \mathcal{F}' such that:

$$q \in \mathcal{F}' \iff q \in \hat{A}^{\{1,s\}} : \exists w \in S^{\leq N-1}, q_1(sw) \neq q_s(w).$$

Finally, we compute the set of all functions $\phi \in \hat{A}$ such that $\phi(\varepsilon) = a_0$, and denote this set by \mathcal{A} . We create $|\mathcal{A}|$ sets of inputs for $\text{SDP}(G, S)$ given by the forbidden patterns \mathcal{F}' and a target letter $\phi \in \mathcal{A}$.

If there exists a configuration $x \in X_{\mathcal{C}}$ such that $x(1_G) = a_0$, we define $y \in X_{\mathcal{F}'} \subseteq \hat{A}^G$ by $y(g)(w) = x(g\bar{w})$. This way, y contains no pattern from \mathcal{F}' , as x does not contain a pattern coding from \mathcal{C} , and $y(1_G)(\varepsilon) = x(1_G) = a_0$. Conversely, if there is function $\phi \in \mathcal{A}$ and a configuration $y \in X_{\mathcal{F}'}$ such that $y(1_G) = \phi$, we construct $x \in X_{\mathcal{C}}$ by $x(g) = y(g)(\varepsilon)$. From the definition of \mathcal{F}' , if we take $g \in G$ with $|g|_S \leq N$ and a word $w \in S^{\leq N}$ such that $\bar{w} = g$, we have that $y(1_G)(w) = y(g)(\varepsilon)$. Therefore, x is well-defined and contains no pattern codings from \mathcal{C} . Furthermore, $x(1_G) = y(1_G)(\varepsilon) = a_0$.

All the previous arguments are analogous for the case of $\text{RDP}(G, S)$ and $\text{REP}(G, S)$. ◀

This Lemma allows us to talk about *the seeded domino problem on G , $\text{SDP}(G)$, and the recurring domino problem on G , $\text{RDP}(G)$.*

► **Lemma 13.** *Let G be a f.g. group along with a finitely generated subgroup H . Then,*

- $\text{SDP}(H) \leq_m \text{SDP}(G)$,
- $\text{RDP}(H) \leq_m \text{RDP}(G)$.

Proof. Let S_H and S_G be finite sets of generators for H and G respectively. We will work with the seeded version, as the recurring one is analogous. Notice that an instance, (\mathcal{F}, a_0) , of $\text{SDP}(H, S_H)$ is also an instance of $\text{SDP}(G, S_G \cup S_H)$.

Now, if there exists $x \in X_{\mathcal{F}} \subseteq A^G$ with $x(1_G) = a_0$, then the configuration $y = x|_H \in A^H$ contains no patterns from \mathcal{F} and verifies $y(1_H) = a_0$. On the other hand, if there exists $y \in X_{\mathcal{F}} \subseteq A^H$; let L be a set of left representatives for G/H . We define $x \in A^G$ as $x(lh) = y(h)$ for all $l \in L$ and all $h \in H$. Because the forbidden patterns are supported on S_H , we have that $x \in X_{\mathcal{F}} \subseteq A^G$. ◀

► **Lemma 14.** *Let G be a f.g. group along with a subgroup H such that $[G : H] < \infty$. Then,*

- $\text{SDP}(G) \equiv_p \text{SDP}(H)$,
- $\text{RDP}(G) \equiv_p \text{RDP}(H)$.

Proof. Because finite index subgroups of finitely generated groups are finitely generated, $\text{SDP}(H) \leq_m \text{SDP}(G)$ by Lemma 13. We now prove that $\text{SDP}(H) \leq_p \text{SDP}(G)$. Without loss of generality, we may assume $H \trianglelefteq G$: every finite index subgroup H contains a normal finite index subgroup N , then if we prove $\text{SDP}(G)$ reduces to $\text{SDP}(N)$, we can conclude it reduces to $\text{SDP}(H)$ by Lemma 13.

Let $X \subseteq A^G$ be a subshift, and R a set of right co-set representatives for G/H , containing the identity 1_G . We define what is known as the R -higher power shift of X as:

$$X^{[R]} = \{y \in (A^R)^H : \exists x \in X : \forall (h, r) \in H \times R, y(h)(r) = x(hr)\}.$$

It is clear that $X^{[R]}$ is an H -subshift, and we will show that if X is a G -SFT, then $X^{[R]}$ is a H -SFT. Let S_H be a finite set of generators for H . We define the sets $D = S_H \cup (RRR^{-1} \cap H)$ and $T = RDR^{-1}$. Because $1_G \in R$ and H is a normal subgroup, $H = \langle T \rangle$.

We will positive-reduce $\text{SDP}(G, S_H \cup R)$ to $\text{SDP}(H, T)$. Let (\mathcal{F}, a_0) be an instance of $\text{SDP}(G, S_H \cup R)$. Let us construct a set \mathcal{F}' of forbidden patterns over the alphabet A^R , such that $X_{\mathcal{F}'} = X_{\mathcal{F}}^{[R]}$. We begin by defining the set of R -patterns containing a_0 :

$$\mathcal{A} = \{p \in A^R : p(1_G) = a_0\}.$$

Now, take $(a, b, s) \in \mathcal{F}$. We will add patterns to \mathcal{F}' depending on where s belongs.

- If $s \in S_H$, we add for each $r \in R$ all patterns q of support $\{1_H, rsr^{-1}\}$ such that $q(1_H)(r) = a$ and $q(rsr^{-1})(r) = b$.
- If $s \in R$, notice that for any $r \in R$, we have $rs = hr'$ where $r' \in R$ and $h \in RRR^{-1} \cap H$, as R is a set of right coset representatives. Therefore, for each $r \in R$, h and r' as before, we all patterns q of support $\{1_H, h\}$ such that $q(1_H)(r) = a$ and $q(h)(r') = b$.

A straightforward computation shows $X_{\mathcal{F}'} = X_{\mathcal{F}}^{[R]}$. Finally, we create $|\mathcal{A}|$ inputs for $\text{SDP}(H, T)$ given by \mathcal{F}' and a letter from \mathcal{A} . Suppose there exists $x \in X_{\mathcal{F}}$ such that $x(1_G) = a_0$. Define $y \in X_{\mathcal{F}'}^{[R]}$ as $y(h)(r) = x(hr)$ for all $h \in H$, $r \in R$, which implies $y(1_H) = x|_R \in \mathcal{A}$. Conversely, if there exists $y \in X_{\mathcal{F}'}^{[R]}$ such that $y(1_H) \in \mathcal{A}$, define $x \in X_{\mathcal{F}}$ by $x(hr) = y(h)(r)$ for all $h \in H$ and $r \in R$. Thus, $x(1_G) = y(1_H)(1_G) = a_0$.

Because $|R| < +\infty$, the case for RDP is analogous. ◀

4.2 Recurring Domino Problem on Free Groups

In this section we prove the following result:

► **Theorem 15.** *$\text{RDP}(\mathbb{F}_n)$ is decidable.*

Proof. Suppose we have a simple balloon $B = a_0 s_1 a_1 \dots s_{n-1} a_{n-1} s_n a_0$ in $\Gamma_{\mathcal{F}}$ based at a_0 with $a_i \in \mathcal{C}(A)$ and label $w = uvu^{-1}$ where $u = s_1 \dots s_k$ with $k \leq \lceil \frac{n}{2} \rceil - 1$. The condition over k implies that $v \neq \varepsilon$. We define a configuration $x \in X_{\mathcal{F}}$ as follows: for every $t \in \mathbb{N}$, $x(\overline{w^t}) = a_0$ and $x(\overline{w^t s_1 \dots s_i}) = a_i$ with $i \in \{1, \dots, n-1\}$ (see Figure 2). Because B is a balloon, x is well defined, as the balloon's definition guarantees

$$x(\overline{w^t u v s_k^{-1} \dots s_i^{-1}}) = a_{n-i} = a_i = x(\overline{w^{t+1} s_1 \dots s_i}).$$

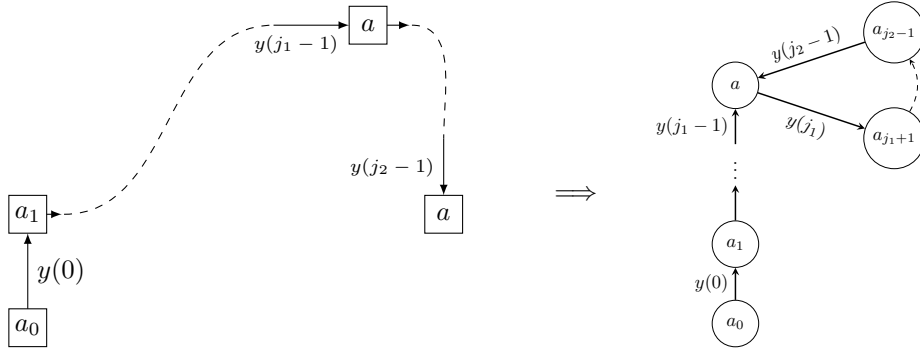
Finally, because every letter belongs to $\mathcal{C}(A)$, the rest of the configuration can be completed without forbidden patterns. Therefore, $x \in X_{\mathcal{F}}$.

Conversely, suppose there exists $x \in X_{\mathcal{F}}$ where a_0 occurs infinitely often. Without loss of generality we can assume $x(1_{\mathbb{F}_n}) = a_0$. Recall that $x(\mathbb{F}_n) \subseteq \mathcal{C}(A)$. Let us denote the set of elements $w \in \mathbb{F}_n$ where $x(w) = a_0$ by \mathcal{O} . Because \mathcal{O} is infinite, there exists $s_0 \in S \cup S^{-1}$ such that infinitely many words in \mathcal{O} begin with s_0 . Furthermore, there exists $s_1 \in S \cup S^{-1}$ with $s_1 \neq s_0^{-1}$ such that infinitely many words in \mathcal{O} begin with $s_0 s_1$. By iterating this argument, we obtain a one-way infinite sequence $y \in (S \cup S^{-1})^{\mathbb{N}}$ such that $y(i) \neq y(i+1)^{-1}$ for all $i \in \mathbb{N}$. Let $\omega(i) = y(0) \dots y(i-1) \in \mathbb{F}_n$. By definition, for every $i \in \mathbb{N}$ there are infinitely many words in \mathcal{O} that begin with $\omega(i)$. We will say $w \in \mathcal{O}$ is *rooted* at $i \in \mathbb{N}$ if $w = \omega(i)v$ for some $v \in (S \cup S^{-1})^*$, and such that the concatenation is reduced. Because there are infinitely many words rooted along some point of y , and $\mathcal{C}(A)$ is finite, there exist $j_1 < i < j_2$ and $w \in \mathcal{O}$ such that $x(\omega(j_1)) = x(\omega(j_2))$ and w is rooted at i . Using this fact, we will create a balloon depending on two cases.

1. If $y(j_1 - 1) \neq y(j_2 - 1)$, and calling $a_j = x(\omega(j))$, define the balloon that represents going from $x(1_G)$ to $x(\omega(j_2))$ by the path $\omega(j_2)$ and then return via the path $\omega(j_1)$ (see Figure 3). Formally,

$$B = a_0 y(0) a_1 \dots y(j_1 - 1) a_{j_1} \dots y(j_2 - 1) a_{j_2} y(j_1 - 1)^{-1} a_{j_1 - 1} \dots y(0)^{-1} a_0,$$

which is labeled by $\omega(j_2)\omega(j_1)^{-1}$, a reduced word.

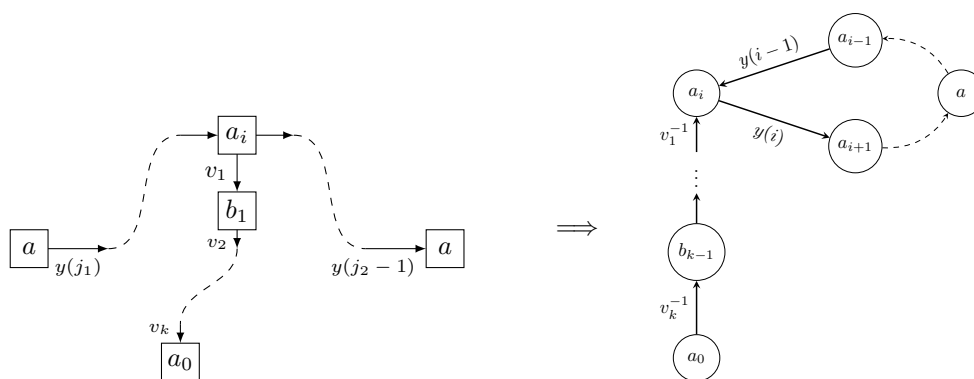


■ **Figure 3** On the left, the path defined by y in the configuration. This is an example of the first case, where $y(j_1 - 1) \neq y(j_2 - 1)$ and the repeated letter is $a = x(\omega(j_1)) = x(\omega(j_2))$. On the right, the corresponding balloon within the tileset graph $\Gamma_{\mathcal{F}}$.

2. If $y(j_1 - 1) = y(j_2 - 1)$, then $y(j_1) \neq y(j_2 - 1)^{-1}$. Let $v \in (S \cup S^{-1})^k$ such that $w = \omega(i)v$. Once again, calling $a_j = x(\omega(j))$ and $b_j = x(\overline{wv_1 \dots v_j})$, we define the balloon

$$B = a_0 v_k^{-1} b_{k-1} v_{k-1}^{-1} \dots v_1^{-1} a_i y(i) \dots y(j_2 - 1) a_{j_2} y(j_1) \dots y(i - 1) a_i v_1 b_1 \dots v_k a_0,$$

which is labelled by $vy(i) \dots y(j_2 - 1)y(j_1) \dots y(i - 1)v^{-1}$, a reduced word (see Figure 4).



■ **Figure 4** On the left, the path defined by y in the configuration as well as the path leading from the root $x(\omega(i))$ to w . This is an example of the second case, where $y(j_1 - 1) = y(j_2 - 1)$ and the repeated letter is $a = x(\omega(j_1)) = x(\omega(j_2))$. On the right, the corresponding balloon within the tileset graph $\Gamma_{\mathcal{F}}$.

Finally, if B contains a repeated letter/generator pair, we can simply cut the portion between them while preserving all other balloon conditions. This guarantees that B will be a simple balloon in $\Gamma_{\mathcal{F}}$ based at a_0 , with all its vertices in $\mathcal{C}(A)$. ◀

Proof of Theorem 15. Given a finite set of nearest neighbor forbidden patterns \mathcal{F} and a letter a_0 , by Lemma 18, it suffices to search for simple balloons in $\Gamma_{\mathcal{F}}$ whose vertices are contained in $\mathcal{C}(A)$. A simple balloon passes through each vertex-label pair at most once; and there are a finite number of such paths starting and ending in a_0 , so we can check whether they satisfy the simple balloon conditions. Therefore, we can effectively decide whether the conditions of Lemma 18 are met, making the recurrence problem decidable. ◀

4.3 Consequences and Conjectures

As previously stated, we are interested in understanding the class of groups that have decidable seeded domino problem, and the class of groups that have decidable recurring domino problem.

▶ **Theorem 19.** *Let G be a virtually free group. Then, both $\text{SDP}(G)$ and $\text{RDP}(G)$ are decidable.*

Proof. By Theorem 15 we know the recurring domino problem is decidable on free groups. Adding Lemma 14, we have that it is decidable for virtually free groups. For the seeded version, as we mentioned earlier, we have that the problem is expressible in MSO logic and is therefore decidable for virtually free groups. ◀

Are these the only groups where each individual problem is decidable? The combination of Conjecture 4 and Lemmas 6 and 8 suggest so.

▶ **Corollary 20.** *If the Domino Conjecture is true the following are equivalent:*

- G is virtually free,
- $\text{DP}(G)$ is decidable,
- $\text{SDP}(G)$ is decidable,
- $\text{RDP}(G)$ is decidable.

Nevertheless, virtually free groups being the only groups where the seeded domino problem or the recurring domino problem is decidable does not directly imply Conjecture 4. We can nonetheless state a conjecture for the seeded domino problem due to the fact it can be expressed in MSO logic.

► **Conjecture 21.** *Let G be a finitely generated group. Then, $\text{SDP}(G)$ is decidable if and only if G is virtually free.*

5 The k -SAT Problem on Groups

As mentioned in the introduction, we define a generalized version of the k -SAT problem for finitely generated groups. This version is slightly different from the one introduced by Freedman [17] in order to correctly capture the structure of finitely generated groups.

Let G be a finitely generated group and $H \leq G$ a finitely generated subgroup. As variables for our formulas we use elements of G . For $g \in G$, we denote its negation by $\neg g$ and we use the ambiguous notation g' to refer to either g or $\neg g$ depending on the formula. We denote the set of formulas over G containing k literals as N_k , that is, $\phi \in N_k$ if

$$\phi = \bigwedge_{i=1}^m ((g_{i1})' \vee \dots \vee (g_{ik})').$$

Next, we define the set of formulas HN_k as all the formulas of the form:

$$\bigwedge_{h \in H} \bigwedge_{i=1}^m ((hg_{i1})' \vee \dots \vee (hg_{ik})')$$

We use $\phi(h)$ to denote the formula ϕ with each literal left-multiplied by h .

► **Definition 22.** *We say a formula $\phi \in HN_k$ is satisfiable, if there exists an assignment of truth values $\alpha : G \rightarrow \{0, 1\}$ such that:*

$$\bigwedge_{h \in H} \bigwedge_{i=1}^m (\alpha(hg_{i1})' \vee \dots \vee \alpha(hg_{ik})') = 1.$$

Let S be a finite generating set for G . To arrive at a valid decision problem, we will specify a function by a set of words over $S \cup S^{-1}$ that will evaluate to the literals of the function, and a list of words, also over $S \cup S^{-1}$, that will specify a generating set for a subgroup. Formally, an *input formula* is a formula of the form

$$\phi = \bigwedge_{i=1}^m (v'_{i1} \vee \dots \vee v'_{ik}),$$

where $v_{ij} \in (S \cup S^{-1})^*$ for all $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, k\}$, such that its evaluated version

$$\bar{\phi} = \bigwedge_{i=1}^m (\bar{v}'_{i1} \vee \dots \vee \bar{v}'_{ik})$$

belongs to N_k .

► **Definition 23.** *Let G be a finitely generated group, S a finite generating set and $k > 1$. The k -SAT problem over G is the decision problem that given an input formula ϕ and $\{w_i\}_{i=1}^n$ determines if the formula $\bigwedge_{h \in H} \bar{\phi}(h)$ is satisfiable, where $H = \langle w_1, \dots, w_n \rangle$.*

Notice that the decidability of this problem does not depend on the chosen generating set, as we can re-write any input into any other generating set. We therefore denote this problem by $k\text{-SAT}(G)$.

The first observation to make is that this problem depends on the computational structure of the group.

► **Lemma 24.** *The subgroup membership problem of G many-one reduces to $\text{co2-SAT}(G)$.*

The subgroup membership problem of a f.g. group G is the decision problem that takes as input words $u, \{w_i\}_{i=1}^n$ over $S \cup S^{-1}$ and determines if $\bar{u} \in \langle w_1, \dots, w_n \rangle$.

Proof. Let $u, \{w_i\}_{i=1}^n \in (S \cup S^{-1})^*$ be an instance of the subgroup membership problem. We define the formula

$$\psi = (\neg \varepsilon \vee s) \wedge (u \vee s) \wedge (\neg \varepsilon \vee \neg s) \wedge (u \vee \neg s),$$

for some fixed $s \in S$, which along with the words w_i is an input to $2\text{-SAT}(G)$. Notice that ψ is equivalent to the formula $\neg \varepsilon \wedge u$. Let us denote $H = \langle w_1, \dots, w_n \rangle$ and $\Psi = \bigwedge_{h \in H} \bar{\psi}(h)$.

Suppose $\bar{u} \in H$. Then, we have that $\bar{\psi}(1_G) \wedge \bar{\psi}(\bar{u}) = (\neg 1_G \wedge \bar{u}) \wedge (\neg \bar{u} \wedge \bar{u}^2)$ is never satisfiable, and thus Ψ is not satisfiable. On the other hand, if $\bar{u} \notin H$, we can define the assignment $\alpha: G \rightarrow \{0, 1\}$ by $\alpha(h) = 0$ and $\alpha(hu) = 1$ for all $h \in H$, and $\alpha(g) = 0$ for all other $g \in G \setminus H$. This way $\neg \alpha(h) \wedge \alpha(hu) = 1$ for all h , and therefore Ψ is satisfied. ◀

Examples of groups with undecidable subgroup membership problems are $\mathbb{F}_n \times \mathbb{F}_n$ [36], some hyperbolic groups [43], as well as groups with undecidable word problem.

► **Lemma 25.** *Let G be a finitely generated group with decidable subgroup membership problem. Then, for every $k \geq 2$ we have that $k\text{-SAT}(G) \leq_m \text{DP}(G)$.*

Proof. Let S be a finite generating set for G , ϕ an input formula and $\{w_i\}_i$ words over $S \cup S^{-1}$ that form an instance of $k\text{-SAT}(G)$ such that

$$\phi = \bigwedge_{i=1}^m (v'_{i1} \vee \dots \vee v'_{ik}),$$

with $v_{ij} \in (S \cup S^{-1})^*$. Let us once again denote $H = \langle w_1, \dots, w_n \rangle$. Our alphabet, A , consists of 0-1 matrices of size $m \times k$ that satisfy ϕ , that is, all matrices $M \in \{0, 1\}^{m \times k}$ such that

$$\bigwedge_{i=1}^m ((M_{i1})' \vee \dots \vee (M_{ik})') = 1.$$

To obtain this alphabet we must solve the standard $k\text{-SAT}$ problem, which is computable.

For convenience, let us denote the finite subset of words involved in the formula by $L = \{v_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq k\}$. In addition, we define the set H_L as the set of all $h_{abcd} \in H \cap LL^{-1}$, where

$$h_{abcd} = \begin{cases} v_{ab}v_{cd}^{-1} & \text{if } v_{ab}v_{cd}^{-1} \in H, \\ 1_H & \text{otherwise.} \end{cases}$$

Notice that $|H_L| \leq |L|^2 = m^2k^2$, and that this set is computable as G has decidable subgroup membership problem. Let us proceed by specifying a set of nearest neighbor forbidden rules, \mathcal{F} , with respect to the generating set $S \cup H_L$. Given a configuration $x \in X_{\mathcal{F}}$

17:14 Contributions to the Domino Problem: Seeding, Recurrence and Satisfiability

the idea is that, for $h \in H$, the matrix $x(h)$ will stock the values assigned to the elements of hL . For each $h_{abcd} \in H_L$, we forbid patterns q of support $\{1_H, h_{abcd}\}$, such that if $q(1_H) = M$ and $q(h_{abcd}) = \hat{M}$,

$$M_{ab} \neq \hat{M}_{cd}.$$

Suppose $X_{\mathcal{F}}$ contains a configuration x . The assignment of truth values $\alpha : G \rightarrow \{0, 1\}$ is defined by

$$\alpha(g) = \begin{cases} 0 & \text{if } g \notin H \cdot L, \\ x(h)_{ab} & \text{if } g = h\bar{v}_{ab} \end{cases}.$$

It follows that α is well defined; if $g = h_1\bar{v}_{ab} = h_2\bar{v}_{cd}$, then $h_2 = h_1h_{abcd}$, and by the forbidden patterns we know $(x_{h_1})_{ab} = (x_{h_2})_{cd}$. In addition, because $x \in A^G$, for all $h \in H$,

$$\bigwedge_{i=1}^m (\alpha(h\bar{v}_{i1})' \vee \dots \vee \alpha(h\bar{v}_{ik})') = \bigwedge_{i=1}^m (x(h)'_{i1} \vee \dots \vee x(h)'_{ik}) = 1.$$

This means that the assignation α satisfies $\bigwedge_{h \in H} \bar{\phi}(h)$.

Finally, suppose we have an assignation of truth values $\beta : G \rightarrow \{0, 1\}$ that satisfies $\bigwedge_{h \in H} \bar{\phi}(h)$. Given a set of right coset representatives R containing 1_G , we define $z \in \{0, 1\}^{m \times k}$ by $z(hr)_{ab} = \beta(hg_{ab})$, for all $h \in H$ and $r \in R$. Because β satisfies $\bigwedge_{h \in H} \bar{\phi}(h)$, for all $h \in H$

$$\bigwedge_{i=1}^m (z(h)'_{i1} \vee \dots \vee z(h)'_{ik}) = \bigwedge_{i=1}^m (\beta(h\bar{v}_{i1})' \vee \dots \vee \beta(h\bar{v}_{ik})') = 1.$$

Therefore, $z \in A^G$. For $h_{abcd} \in H_L$, $h_1 \in H$ and $h_2 = h_1h_{abcd}$ we have that

$$z(h_1)_{ab} = \beta(h_1\bar{v}_{ab}) = \beta(h_1h_{abcd}\bar{v}_{cd}) = \beta(h_2\bar{v}_{cd}) = z(h_2)_{cd}$$

Therefore z satisfies the local rules and is thus in $X_{\mathcal{F}}$. This concludes our reduction. \blacktriangleleft

Virtually free groups not only have decidable domino problem, as previously mentioned, but also have decidable subgroup membership problem (see [34]).

► **Corollary 26.** *For G a virtually free group, k -SAT(G) is decidable for all $k > 1$.*

To determine when the converse reduction is true, we introduce a new class of groups that has the required properties.

► **Definition 27.** *Let G be a finitely generated group. We say G is scalable if there exists a proper finite index subgroup $H \triangleleft G$ that is isomorphic to G .*

Examples of such groups are finitely generated abelian groups, the Heisenberg group, solvable Baumslag-Solitar groups $BS(1, n)$, Lamplighter groups $F \wr \mathbb{Z}$ with F a finite abelian group, the affine group $\mathbb{Z}^d \rtimes GL(d, \mathbb{Z})$ for $d \geq 2$ [40], among others. Examples of non-scalable groups are finitely generated free groups.

► **Theorem 28.** *For G a scalable group, $DP(G) \leq_m 3\text{-SAT}(G)$.*

Proof. Let A be a finite alphabet of size n , and \mathcal{F} a finite set of nearest neighbor forbidden patterns for G with generating set S . As G is scalable, there exists H a proper subgroup of finite index as well as an isomorphism $F: G \rightarrow H$. Let $f: S \rightarrow S^*$ the function that is extended to the isomorphism F , that is, $\{f(s)\}_{s \in S}$ is a generating set for H . Fix $R \subseteq (S \cup S^{-1})^*$ a set of words representing a finite set of right coset representatives for H that includes 1_G . Notice that for every $m \in \mathbb{N}$ the subgroup $H_m = F^m(G)$ is isomorphic to G with $[G : H_m] = [G : H]^m \geq m$. In addition, a simple computation shows that $R_m = F^{m-1}(R) \dots F(R)R$ defines a set of right coset representatives for H_m .

The idea of the reduction is to represent each letter of the alphabet by a unique code on the left coset representatives and then create a formula that assigns a letter to each element of H_m . The index of the subgroup, m , will be chosen so there is enough room to code the alphabet and write our formula in the required form.

First off, take as a preliminary estimate $m \geq \lceil \log_2(n) \rceil$, and denote $f_m = f^m$. Let $\{\phi_a\}_{a \in A}$ be the set of formulas that code the elements of the alphabet A using the words in R_m as variables. This way $\phi_a(h) \equiv 1$ means we place the letter a at $g = (F^m)^{-1}(h)$, and the variables are contained in hR_m . Our formula is given by,

$$\varphi = \left(\bigvee_{a \in A} \phi_a(1_G) \right) \wedge \left(\bigwedge_{(a,b,s) \in \mathcal{F}} \neg \phi_a(1_G) \vee \neg \phi_b(f_m(s)) \right),$$

which represents the fact that we place one letter at the given point (1_G in this case) and that there are no forbidden patterns in its neighborhood. If modified to be in CNF form, φ is a conjunction of $|\mathcal{F}| + \lceil \log_2(n) \rceil^n$ clauses of $\leq n$ literals (the clauses coding the forbidden patterns contain $2\lceil \log_2(n) \rceil$ literals). By adding $(|\mathcal{F}| + \lceil \log_2(n) \rceil^n)n$ dummy variables we can transform φ into an equivalent formula φ' whose clauses contain exactly 3 literals.

Therefore, take $m \geq (|\mathcal{F}| + \lceil \log_2(n) \rceil^n)n + \lceil \log_2(n) \rceil$, which gives us enough space in the set of left coset representatives to code the elements of the alphabet and the dummy variables. Furthermore, φ' is computable from A and \mathcal{F} , and $\Phi' = \bigwedge_{h \in H} \bar{\varphi}'(h) \in H_m \mathcal{N}_3$.

Let us prove the reduction. If there exists $x \in X_{\mathcal{F}} \subseteq A^G$, we create an assignment such that for all $g \in G$, the variables in $F^m(g)R_m$ are given values so as to satisfy the code for $\phi_{x(g)}(F^m(g)) \equiv 1$. Because x contains no patterns from \mathcal{F} , $\bigwedge_{h \in H} \bar{\varphi}'(h)$ will be satisfied. We finish by filling out the rest of the variables so that $\Phi' \equiv 1$.

Now, if Φ' is satisfied so is $\bigwedge_{h \in H} \bar{\varphi}'(h)$. Let $y \in A^G$ be the configuration defined by $y(g) = a$ if $\phi_a(F^m(g)) \equiv 1$. Because the codes used make sure that the values in $F^m(g)R_m$ code a unique letter, for each $g \in G$ a unique $\phi_a(F^m(g))$ is satisfied. Thus y is well defined. Finally, $y \in X_{\mathcal{F}}$ because if there was $g \in G$ such that $y(g) = a$ and $y(gs) = b$ with $(a, b, s) \in \mathcal{F}$ we would have that $\phi_a(F^m(g)) \wedge \phi_b(F^m(g)f_m(s))$ is true. This shows $\text{DP}(G) \leq_m \text{3-SAT}(G)$. \blacktriangleleft

► Corollary 29. *3-SAT(G) is undecidable for finitely generated abelian groups, solvable Baumslag-Solitar groups, the Heisenberg group and affine groups.*

References

- 1 Leonard Adleman, Jarkko Kari, Lila Kari, and Dustin Reishus. The undecidability of the infinite ribbon problem: implications for computing by self-assembly. *SIAM J. Comput.*, 38(6):2356–2381, 2009. doi:10.1137/080723971.
- 2 Nathalie Aubrun, Sebastián Barbieri, and Emmanuel Jeandel. About the domino problem for subshifts on groups. In *Sequences, groups, and number theory*, Trends Math., pages 331–389. Birkhäuser/Springer, Cham, 2018. doi:10.1007/978-3-319-69152-7_9.
- 3 Nathalie Aubrun and Nicolás Bitar. Domino snake problems on groups. In *International Symposium on Fundamentals of Computation Theory*, pages 46–59. Springer Nature Switzerland, 2023. doi:10.1007/978-3-031-43587-4_4.
- 4 Nathalie Aubrun, Nicolás Bitar, and Sacha Huriot-Tattegrain. Strongly aperiodic SFTs on generalized Baumslag–Solitar groups. *Ergodic Theory and Dynamical Systems*, pages 1–30, 2023. doi:10.1017/etds.2023.44.
- 5 Nathalie Aubrun and Jarkko Kari. Tiling problems on Baumslag-Solitar groups. In *Proceedings: Machines, Computations and Universality 2013*, volume 128 of *Electron. Proc. Theor. Comput. Sci. (EPTCS)*, pages 35–46. EPTCS, 2013. doi:10.4204/EPTCS.128.12.
- 6 Alexis Ballier and Emmanuel Jeandel. Tilings and model theory. In *First Symposium on Cellular Automata "Journées Automates Cellulaires" (JAC 2008), Uzès, France, April 21-25, 2008. Proceedings*, pages 29–39. MCCME Publishing House, Moscow, 2008.
- 7 Alexis Ballier and Maya Stein. The domino problem on groups of polynomial growth. *Groups Geom. Dyn.*, 12(1):93–105, 2018. doi:10.4171/GGD/439.
- 8 Laurent Bartholdi. Monadic second-order logic and the domino problem on self-similar graphs. *Groups Geom. Dyn.*, 16(4):1423–1459, 2022. doi:10.4171/ggd/689.
- 9 Laurent Bartholdi. The domino problem for hyperbolic groups. *arXiv preprint arXiv:2305.06952*, 2023.
- 10 Robert Berger. The undecidability of the domino problem. *Mem. Amer. Math. Soc.*, 66:72, 1966.
- 11 J. Richard Büchi. Turing-machines and the Entscheidungsproblem. *Math. Ann.*, 148:201–213, 1962. doi:10.1007/BF01470748.
- 12 Antonin Callard and Benjamin Hellouin de Menibus. The aperiodic Domino problem in higher dimension. In *39th International Symposium on Theoretical Aspects of Computer Science*, volume 219 of *LIPICs. Leibniz Int. Proc. Inform.*, pages Art. No. 19, 15. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2022.
- 13 Laura Ciobanu, Alex Levine, and Alan D Logan. Post’s correspondence problem for hyperbolic and virtually nilpotent groups. *arXiv preprint arXiv:2211.12158*, 2022.
- 14 Laura Ciobanu and Alan D Logan. Variations on the Post correspondence problem for free groups. In *International Conference on Developments in Language Theory*, pages 90–102. Springer, 2021.
- 15 Toby Cubitt, David Perez-Garcia, and Michael M. Wolf. Undecidability of the spectral gap. *Forum Math. Pi*, 10:Paper No. e14, 102, 2022. doi:10.1017/fmp.2021.15.
- 16 Michael H. Freedman. Limit, logic, and computation. *Proc. Natl. Acad. Sci. USA*, 95(1):95–97, 1998. doi:10.1073/pnas.95.1.95.
- 17 Michael H. Freedman. k -SAT on groups and undecidability. In *STOC '98 (Dallas, TX)*, pages 572–576. ACM, New York, 1999.
- 18 Anael Grandjean, Benjamin Hellouin de Menibus, and Pascal Vanier. Aperiodic point in \mathbb{Z}^2 -subshifts. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPICs. Leibniz Int. Proc. Inform.*, pages Art. No. 128, 13. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.
- 19 Rachel Greenfeld and Terence Tao. Undecidability of translational monotilings. *arXiv preprint arXiv:2309.09504*, 2023.
- 20 Ju. Š. Gurevič and I. O. Korjakov. A remark on R. Berger’s work on the domino problem. *Sibirsk. Mat. Ž.*, 13:459–463, 1972.

- 21 David Harel. Recurring dominoes: making the highly undecidable highly understandable. In *Topics in the theory of computation (Borgholm, 1983)*, volume 102 of *North-Holland Math. Stud.*, pages 51–71. North-Holland, Amsterdam, 1985.
- 22 David Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *J. Assoc. Comput. Mach.*, 33(1):224–248, 1986. doi:10.1145/4904.4993.
- 23 Benjamin Hellouin de Menibus, Victor H. Lutfalla, and Camille Noûs. The Domino problem is undecidable on every rhombus subshift. In *Developments in language theory*, volume 13911 of *Lecture Notes in Comput. Sci.*, pages 100–112. Springer, Cham, [2023] ©2023. doi:10.1007/978-3-031-33264-7_9.
- 24 Benjamin Hellouin de Menibus and Hugo Maturana Cornejo. Necessary conditions for tiling finitely generated amenable groups. *Discrete Contin. Dyn. Syst.*, 40(4):2335–2346, 2020. doi:10.3934/dcds.2020116.
- 25 Emmanuel Jeandel. The periodic domino problem revisited. *Theoret. Comput. Sci.*, 411(44-46):4010–4016, 2010. doi:10.1016/j.tcs.2010.08.017.
- 26 Emmanuel Jeandel. Aperiodic subshifts on polycyclic groups. *arXiv preprint arXiv:1510.02360*, 2015.
- 27 Emmanuel Jeandel. Translation-like actions and aperiodic subshifts on groups. *arXiv preprint arXiv:1508.06419*, 2015.
- 28 Emmanuel Jeandel and Pascal Vanier. The undecidability of the Domino Problem. In *Substitution and tiling dynamics: introduction to self-inducing structures*, volume 2273 of *Lecture Notes in Math.*, pages 293–357. Springer, Cham, [2020] ©2020. doi:10.1007/978-3-030-57666-0_6.
- 29 A. S. Kahr, Edward F. Moore, and Hao Wang. Entscheidungsproblem reduced to the $\forall\exists\forall$ case. *Proc. Nat. Acad. Sci. U.S.A.*, 48:365–377, 1962. doi:10.1073/pnas.48.3.365.
- 30 Jarkko Kari. Reversibility of 2d cellular automata is undecidable. *Physica D: Nonlinear Phenomena*, 45(1-3):379–385, 1990. doi:10.1016/0167-2789(90)90195-U.
- 31 Jarkko Kari. Reversibility and surjectivity problems of cellular automata. *J. Comput. System Sci.*, 48(1):149–182, 1994. doi:10.1016/S0022-0000(05)80025-X.
- 32 Dietrich Kuske and Markus Lohrey. Logical aspects of Cayley-graphs: the group case. *Ann. Pure Appl. Logic*, 131(1-3):263–286, 2005. doi:10.1016/j.apal.2004.06.002.
- 33 Douglas Lind and Brian Marcus. *An introduction to symbolic dynamics and coding*. Cambridge Mathematical Library. Cambridge University Press, Cambridge, second edition, 2021. doi:10.1017/9781108899727.
- 34 Markus Lohrey. Subgroup membership in $GL(2, \mathbb{Z})$. *Theory of Computing Systems*, pages 1–26, 2023.
- 35 Roger C. Lyndon and Paul E. Schupp. *Combinatorial group theory*, volume Band 89 of *Ergebnisse der Mathematik und ihrer Grenzgebiete [Results in Mathematics and Related Areas]*. Springer-Verlag, Berlin-New York, 1977.
- 36 K. A. Mihaïlova. The occurrence problem for free products of groups. *Mat. Sb. (N.S.)*, 75(117):199–210, 1968.
- 37 David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoret. Comput. Sci.*, 37(1):51–75, 1985. doi:10.1016/0304-3975(85)90087-8.
- 38 Alexei Myasnikov, Andrey Nikolaev, and Alexander Ushakov. The Post correspondence problem in groups. *Journal of Group Theory*, 17(6):991–1008, 2014.
- 39 D Myers. Decidability of the tiling connectivity problem. abstract 79t-e42. *Notices Amer. Math. Soc.*, 195(26):177–209, 1979.
- 40 Volodymyr Nekrashevych and Gábor Pete. Scale-invariant groups. *Groups Geom. Dyn.*, 5(1):139–167, 2011. doi:10.4171/GGD/119.
- 41 Steven T. Piantadosi. Symbolic dynamics on free groups. *Discrete Contin. Dyn. Syst.*, 20(3):725–738, 2008. doi:10.3934/dcds.2008.20.725.
- 42 Yo’av Rieck. Strongly aperiodic SFTs on hyperbolic groups: where to find them and why we love them. *arXiv preprint arXiv:2202.00212*, 2022.

17:18 Contributions to the Domino Problem: Seeding, Recurrence and Satisfiability

- 43 E. Rips. Subgroups of small cancellation groups. *Bull. London Math. Soc.*, 14(1):45–47, 1982. doi:10.1112/blms/14.1.45.
- 44 Neil Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *J. Combin. Theory Ser. B*, 41(1):92–114, 1986. doi:10.1016/0095-8956(86)90030-4.
- 45 Hartley Rogers, Jr. *Theory of recursive functions and effective computability*. McGraw-Hill Book Co., New York-Toronto-London, 1967.
- 46 Peter van Emde Boas. The convenience of tilings. In *Complexity, logic, and recursion theory*, volume 187 of *Lecture Notes in Pure and Appl. Math.*, pages 331–363. Dekker, New York, 1997.
- 47 Hao Wang. Proving theorems by pattern recognition—II. *Bell system technical journal*, 40(1):1–41, 1961.

Removable Online Knapsack and Advice

Hans-Joachim Böckenhauer  

Department of Computer Science, ETH Zürich, Switzerland

Fabian Frei  

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Peter Rossmanith  

Department of Computer Science, RWTH Aachen, Germany

Abstract

In the *proportional knapsack* problem, we are given a knapsack of some capacity and a set of variably sized items. The goal is to pack a selection of these items that fills the knapsack as much as possible. The *online* version of this problem reveals the items and their sizes not all at once but one by one. For each item, the algorithm has to decide immediately whether to pack it or not. We consider a natural variant of this online knapsack problem, which has been coined *removable knapsack*. It differs from the classical variant by allowing the removal of any packed item from the knapsack. Repacking is impossible, however: Once an item is removed, it is gone for good.

We analyze the *advice complexity* of this problem. It measures how many *advice bits* an omniscient oracle needs to provide for an online algorithm to reach any given *competitive ratio*, which is – understood in its strict sense – just the algorithm’s approximation factor. The online knapsack problem is known for its peculiar advice behavior involving three jumps in competitiveness. We show that the advice complexity of the version with removability is quite different but just as interesting: The competitiveness starts from the golden ratio when no advice is given. It then drops down to $1 + \epsilon$ for a constant amount of advice already, which requires logarithmic advice in the classical version. Removability comes as no relief to the perfectionist, however: Optimality still requires linear advice as before. These results are particularly noteworthy from a structural viewpoint for the exceptionally slow transition from near-optimality to optimality.

Our most important and demanding result shows that the *general knapsack* problem, which allows an item’s value to differ from its size, exhibits a similar behavior for removability, but with an even more pronounced jump from an unbounded competitive ratio to near-optimality within just constantly many advice bits. This is a unique behavior among the problems considered in the literature so far.

An advice analysis is interesting in its own right, as it allows us to measure the information content of a problem and leads to structural insights. But it also provides insurmountable lower bounds, applicable to any kind of additional information about the instances, including predictions provided by machine-learning algorithms and artificial intelligence. Unexpectedly, advice algorithms are useful in various real-life situations, too. For example, they provide smart strategies for cooperation in winner-take-all competitions, where several participants pool together to implement different strategies and share the obtained prize. Further illustrating the versatility of our advice-complexity bounds, our results automatically improve some of the best known lower bounds on the competitive ratio for removable knapsack with randomization. The presented advice algorithms also automatically yield deterministic algorithms for established deterministic models such as knapsack with a resource buffer and various problems with more than one knapsack. In their seminal paper introducing removability to the knapsack problem, Iwama and Taketomi have indeed proposed a multiple knapsack problem for which we can establish a one-to-one correspondence with the advice model; this paper therefore even provides a comprehensive analysis for this up until now neglected problem.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Removable Online Knapsack, Multiple Knapsack, Advice Analysis, Advice Applications, Machine Learning and AI

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.18

Related Version *Full Version:* <https://arxiv.org/abs/2005.01867>

Funding *Fabian Frei:* Work done in part while at ETH Zürich.



© Hans-Joachim Böckenhauer, Fabian Frei, and Peter Rossmanith;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;
Article No. 18; pp. 18:1–18:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

In this first section, we briefly summarize what online algorithms and advice are, then informally present the problem whose advice complexity we will be analyzing, and finally describe several applications of such advice complexity results.

1.1 Online Algorithms and Advice Complexity

Online algorithms receive their input piece by piece and have to determine parts of the solution before knowing the entire instance. This often leaves them unable to compete with offline algorithms, which know the entire input in advance, in a meaningful way. In the *advice* model, we assume an omniscient oracle that provides the online algorithm with some information on how to solve the upcoming instance best. If the oracle can communicate to the algorithm an unlimited amount of such advice, it will of course be able to lead the algorithm to an optimal solution for every instance. The *advice complexity* measures the minimum amount of information necessary for the online algorithm to achieve any given approximation ratio, which is commonly called strict *competitive ratio* or *competitiveness* in this context. Advice complexity is a well-established tool to gauge the information content of an online problem [3, 9, 15]. For a detailed and careful introduction to the theory, we refer to the textbook by Komm [20]. Another classical textbook on online problems is written by Borodin and Yaniv [5]. The trade-off between a low number of transmitted advice bits on the one hand and achieving a good competitive ratio on the other hand has been examined for a wealth of problems – see the survey by Boyar et al. [6] – but one stands out for its peculiar behavior: the knapsack problem.

1.2 Knapsack and Removability

A knapsack instance presents the online algorithm with a sequence of items of different sizes. Upon the arrival of each item, the algorithm has to decide whether to pack it into a knapsack or discard it. The goal is to fill the knapsack as much as possible without ever exceeding the knapsack's given capacity. This problem is sometimes also referred to as the *proportional* or *simple* knapsack problem, as opposed to the *general* knapsack problem, in which every item has not only a size but also a value.¹ In the generalized version, the goal is to maximize the total value of all packed items. With no further specification given, we are always referring to the proportional case.

A variant of the knapsack problem has been proposed by Iwama and Taketomi [17] under the name of *removable* knapsack. In this model, we can discard an item not only when it is first presented to us; we may also remove a packed item from the knapsack at any point. This is possible only once for each item, however; once removed, an item cannot be repacked. As for the classical problem without removability, the capacity of the knapsack may not be exceeded at any point in time. Recently, Rossmanith has introduced a similar relaxed online setting for graph problems where decisions are taken only when constraints make it inevitable [23].

¹ It is also quite common for the proportional and general knapsack problem to be called unweighted and weighted, respectively. The notion *weight* is ambiguous, however, as some authors [4] use it for what is called size here, while others [18] use it for what is called the value here or profit elsewhere. For the sake of clarity, we are well advised to avoid the term weight altogether.

This model is arguably just as natural a way to translate the knapsack problem into the online setting as the more well-examined variant without removability. In many cases, it will not be hard to discard items at regular intervals, only the chance of obtaining specific objects is subject to special circumstances. For a practical example, consider a storage room in which you can store all kinds of objects that you come across over time. In the beginning you can just keep collecting everything, but by doing so you inevitably run out of space before too long. Then you will have to start disposing of some of your possessions to make room for new, potentially more interesting acquisitions. Your goal is to end up with a selection of just the most meaningful and useful items that you could have. This paper analyzes the advice complexity of both the proportional and general removable knapsack problem. It is telling you how much information about upcoming opportunities you need to ensure an outcome that is either optimal or off by at most a given factor.

1.3 Advice Applications

Besides inherently interesting insights into the information complexity of the knapsack problem, our advice algorithms also offer more concrete applications. Any algorithm reading a bounded number of advice bits can be implemented by running a bounded number of deterministic algorithms in parallel and selecting the best result. An advice analysis thus tells us, for example, how to optimally organize a betting pool in a winner-take-all scenario. Our main result in particular provides a smart selection of strategies to be assigned to a mere constant number of actors such that one is guaranteed to be as close to optimality as we desire, no matter how difficult the instances of the general knapsack problem with removability may become.

A further advantage of analyzing the advice complexity of a problem is that the resulting bounds are very versatile. The lower bounds are particularly strong. They show that a certain competitiveness cannot be achieved with a given amount of additional information, regardless of the form this advice may take. The oracle is indeed able to convey to the algorithm all kinds of structural information about the adversarial instance; for example, in the case of our knapsack problem, whether items smaller than a given threshold should or must not be ignored, whether replacing packed items by later ones will ever be beneficial, whether the values span more than a certain range, whether an optimal solution fills the knapsack completely, whether there are multiple optimal solutions, and so on. Lower bounds on the competitiveness of advice algorithms imply lower bounds for randomized algorithms, and our results indeed improve upon the best bounds known for randomization; Theorem 12 even completely closes the remaining gap in the analysis of barely random algorithms for the general knapsack problem.

There are also interesting implications for deterministic algorithms. Consider the multiple knapsack problem in which every item is either rejected or packed into one of $k > 1$ knapsacks; the goal is for the algorithm to have in the end one knapsack that is as full as possible. This problem has been analyzed with removability by Iwama and Taketomi in the proportional case. In the conclusion of their paper [17], they pose it as an open problem to analyze this model if we are allowed to copy items and pack them into arbitrarily many of the available knapsacks. It turns out that deterministic algorithms for this problem with different k s are equivalent to advice algorithms: An advice algorithm restricted to $\log k$ advice bits can read up to k different advice strings. Even if the algorithm reads the entire advice string right at the beginning, before taking any decision, it will thus implement one of k deterministic strategies. Having k knapsacks and being able to pack each item into several of them at the same time means that we can just simulate each possible strategy in one of the knapsacks

and see which one leads to the best result in the end. Conversely, the oracle in our model already knows the optimal choice and can communicate to an advice algorithm which of the knapsacks it should be simulating. Knowing the advice complexity of our problem with only one knapsack therefore automatically yields a comprehensive competitive analysis for this deterministic problem for any $k > 1$. All of this remains true for the general knapsack problem; thus our results provide a comprehensive picture for the proposed model in both the proportional and non-proportional case. We remark that algorithms for the resource buffer model are generally not applicable here. Algorithm 2 by Han et al. [13, Thm. 9], for example, keeps regrouping items in every step and thus crucially relies on having a single large buffer instead of multiple standard-sized knapsacks without an option to shuffle items between them.

2 Preliminaries

Throughout this paper, \log denotes the binary logarithm. We formally define the removable knapsack problem as follows.

► **Definition 1** (Removable Knapsack Problem). *REMKNAP is an online maximization problem. An instance I is a sequence $(s_1, v_1), \dots, (s_n, v_n)$ of n items, each of which is a pair of some real positive size s_i and value v_i . Where useful, we denote size and value of an item i functionally by $s(i) = s_i$ and $v(i) = v_i$. The domain of this function naturally extends to arbitrary subsets $T \subseteq \{1, \dots, n\}$ of items by defining $s(T) = \sum_{i \in T} s(i)$ and $v(T) = \sum_{i \in T} v(i)$. The knapsack has a maximum size capacity, which we normalize to be 1.*

The instance is presented item by item to an online algorithm \mathcal{A} that has to maintain a packing, a set of packed items. We call the total size of the currently packed items the current filling of the knapsack. The algorithm starts out with an empty knapsack, represented by the empty set $T_0 = \emptyset$. When presented with item i , the algorithm may first remove any of the items T_{i-1} packed so far; then it may pack the new item if this does not exceed the knapsack capacity. In other words, the algorithm selects a subset $T_i \subseteq T_{i-1} \cup \{i\}$ with $s(T_i) \leq 1$ in step i . The algorithm learns the size of item i only once it is presented and only learns the total number n of items after selecting T_n . The final packing computed by \mathcal{A} is denoted by $T = T_n$. The gain that we aim to maximize is the total value $v(T)$ of the final packing.

The proportional variant PROPREMKNAP additionally satisfies $s_i = v_i$ for each item i .

► **Definition 2** (Competitive Ratio). *Let an online maximization problem with instance set \mathcal{I} be given and let \mathcal{A} be an online algorithm solving it. For any instance $I \in \mathcal{I}$, denote by $\text{alg}(I)$ the gain that \mathcal{A} achieves on I and by $\text{opt}(I)$ the gain of an optimal solution to I computed offline. The competitive performance of \mathcal{A} on an instance $I \in \mathcal{I}$ is $\text{opt}(I)/\text{alg}(I)$. For any $\rho \in \mathbf{R}$, the algorithm \mathcal{A} is called strictly ρ -competitive if it performs ρ -competitively across all instances, that is, if $\forall I \in \mathcal{I}: \text{opt}(I)/\text{alg}(I) \leq \rho$. The infimal competitiveness $\inf\{\rho \in \mathbf{R} \mid \mathcal{A} \text{ is strictly } \rho\text{-competitive}\}$ is called strict competitive ratio of \mathcal{A} . We can weaken the defining inequality above so that it only needs to hold asymptotically in the sense of $\exists \alpha \in \mathbf{R}_+: \forall I \in \mathcal{I}: \text{opt}(I) \leq \rho \cdot \text{alg}(I) + \alpha$. If this condition is met, we call \mathcal{A} nonstrictly ρ -competitive.*

Note that strict ρ -competitiveness implies nonstrict ρ -competitiveness but not vice versa, making it harder to prove lower bounds for nonstrict competitiveness. For the knapsack problem, however, it makes sense to always analyze competitiveness in the strict sense: On the one hand, we obtain a nonstrict lower bound from a strict one by scaling up the knapsack capacity and all item sizes in a hard instance set such that the smallest item is strictly larger than α .

If, on the other hand, scaling is impossible due to the problem being defined with a fixed knapsack capacity of 1, for example, then choosing $\alpha = 1$ shows any online algorithm is 1-competitive in the nonstrict sense.

3 Related Work

Knapsack is one of the 21 NP-complete decision problem in Karp's famous list [19]. An algorithm based on dynamic programming solves both the proportional and the general version in pseudo-polynomial time; see Bellman [1, Section 1.4] for the general technique and Dantzig [8, p. 275] for a concrete description of its application to the knapsack problem. The pseudo-polynomial time algorithm can be adapted to the optimization version, yielding a fully polynomial-time approximation scheme [16]. In the following two subsections, we list the known results on the advice complexity of the proportional knapsack problem, first for the classical version and then for the variant allowing the removal of packed items.

3.1 Knapsack without Removability

Marchetti-Spaccamela and Vercellis were the first to consider the classical online version of the knapsack problem in 1995. They called it the $\{0, 1\}$ *knapsack problem* to distinguish it from the *fractional* knapsack problem, which allows for packing items partially. They proved that both versions have an unbounded competitive ratio if items are allowed to have sizes different from their values [21, Thm. 2.1]. We denote the classical problem with neither fractional items nor removability by KNAP and its proportional variant by PROPKNAP.

The concept of advice emerged much later. When it did, KNAP quickly became one of the prime examples of a problem with an interesting advice complexity.

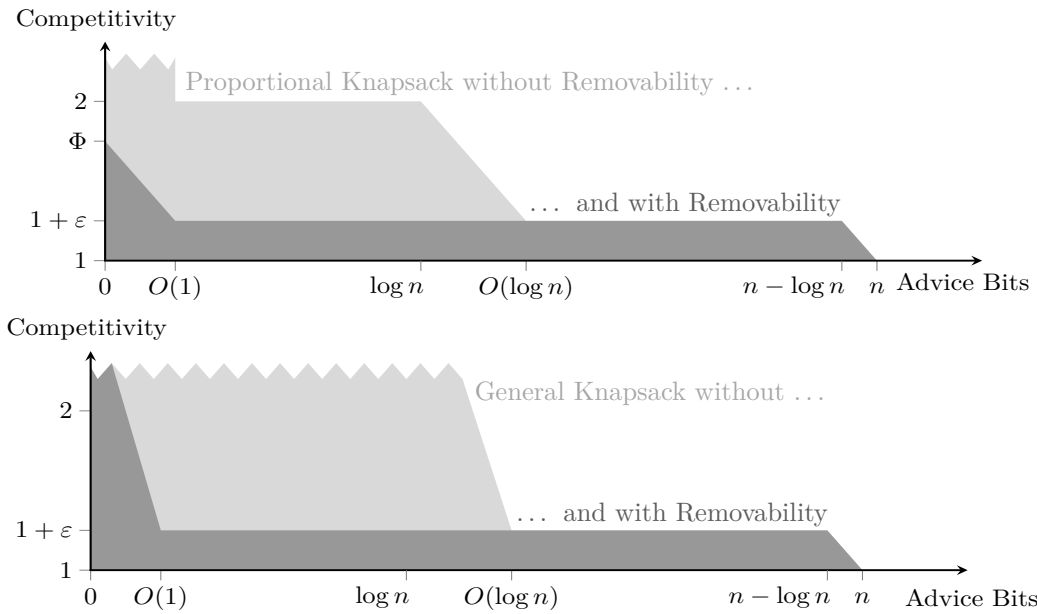
First, just a single advice bit brings with it a jump from non-competitiveness to a 2-competitive algorithm [4, Thm. 4]. More advice bits do not help however, as long as the number stays below $\lfloor \log(n - 1) \rfloor$ [4, Thm. 5]. Once this threshold is surpassed, logarithmic advice allows for a competitive ratio that is arbitrarily close to 1 [4, Thm. 6]. Achieving optimality, finally, requires at least $n - 1$ advice bits [4, Thm. 3].

The situation for the general variant is simpler: Any algorithm reading less than $\log n$ advice bits has an unbounded competitive ratio, but $\mathcal{O}(\log n)$ advice suffices for a near-optimal solution [4, Thms. 11 and 12]. A schematic plot of the advice complexity behaviors just described can be found in Figure 1 in light gray.

3.2 Online Knapsack Variants

Iwama and Taketomi [17] proposed the online knapsack model with removability as it is examined in the present paper. They proved that the competitive ratio for the proportional variant of this problem, which we denote by PROPREMKNAP, is exactly the golden ratio. Iwama and Zhang later considered the problem with resource augmentation, that is, for online algorithms that may use a larger knapsack than the offline algorithm [18].

Later still, Han et al. [12] proved an upper bound of $5/3$ on the competitive ratio for a variant of PROPREMKNAP where the value v of an item is not necessarily proportional to its size s but not arbitrary either; instead, the value is given by a convex function $v = f(s)$ known to the algorithm. They also proved the golden ratio to be optimal if f has some further technical properties. Han et al. [10] considered online knapsack with removal costs, a variant of PROPREMKNAP where items can be removed, but not for free.



■ **Figure 1** A schematic plot of the advice complexity behavior of the classical online knapsack problem in light gray and the relaxed variant with removability in dark gray. For the proportional version without removability there are two large plateaus; removability collapses to a single vast expanse. For the general version, in which an item's value may differ from its size, there is only one but a more extreme jump directly from an unbounded competitive ratio to near optimality; with removability, this jump is occurring earlier and even steeper.

Noga and Sarbua [22] considered a knapsack variant, where it is possible to split each arriving item in two parts of not necessarily equal size, and combine this with resource augmentation. Han and Makino [14] considered another partially fractional variant of PROPREMKNAP where each item can be split a constant number of times at any time. Most importantly in our context, Han et al. [11] examined randomized algorithms for PROPREMKNAP, proving an upper bound of $10/7$ and a lower bound of $5/4$ on the expected competitiveness. Cygan et al. [7] extended the study of randomization for PROPREMKNAP to a variant with multiple knapsacks. Recently, Böckenhauer et al. [2] have introduced a new model for the online proportional knapsack problem in which items can be stored outside of the knapsack until the instance ends after paying a reservation fee that is a fixed fraction α of the item's value.

4 Results for Proportional Removable Knapsack

In Section 4.1, we consider how much – or rather, how little – removability helps when trying to obtain an optimal solution. In Section 4.2, we prove bounds on what is possible with a single advice bit. Finally, we prove in Section 4.3 that a constant amount of advice is sufficient to achieve a competitive ratio of $1 + \epsilon$, for an arbitrary $\epsilon > 0$, and a constant depending on ϵ . See Figure 1 for a rough representation of these results in dark gray.

4.1 Achieving Optimality

We begin by briefly considering PROPKNAP, the classical proportional knapsack problem without removability. Solving it optimally is trivial with n advice bits: The algorithm reads one bit per item, telling it whether to accept or reject. Theorem 3 proves this to be tight by lifting the best known lower bound from $n - 1$ advice bits [4, Thm. 3] to n advice bits. Proofs immediately follow the theorems or are deferred to the full version of the paper.

► **Theorem 3.** *Any algorithm for PROPKNAP reading less than n advice bits is suboptimal.*

Having determined PROPKNAP's advice complexity for optimality, we now do the same for PROPREMKNAP, the variant with removability. It turns out that the option to remove items hardly helps at all in achieving optimality. We begin the upper bound, which is simple but instructive as to what is possible with removability.

► **Theorem 4.** *There is an optimal algorithm for PROPREMKNAP reading $n - 1$ advice bits.*

Proof. Consider an algorithm that packs the first item without reading any advice bits. For each subsequent item, it reads one advice bit, telling it whether the new item is part of a fixed optimal solution. If so, then the new item is packed; otherwise, it is rejected. The first item, which has been packed without advice, is kept in the knapsack as long as there is enough room for it. If the first item is part of the fixed optimal solution, then it will always fit in beside the other items being packed; otherwise, it will be discarded at some point. Thus the algorithm is able to reproduce the fixed optimal solution exactly. ◀

► **Theorem 5.** *Solving PROPREMKNAP optimally requires more than $n - \log n$ advice bits.*

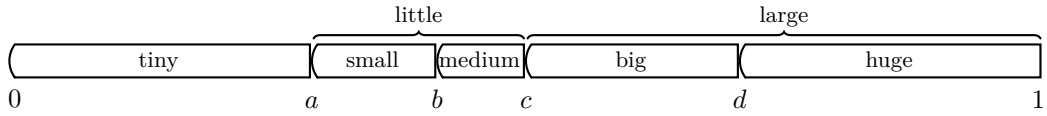
4.2 A Single Advice Bit

The previous section covered the upper end of the advice spectrum, showing that, asymptotically, reading one advice bit for each item in the instance is necessary and sufficient for ensuring an optimal solution. We now turn to the other extreme and ask what can be done with the least nonzero amount of advice, one single bit for the entire instance.

First, we describe a very simple $3/2$ -competitive advice algorithm where a single advice bit indicates whether there is an optimal solution containing more than one item from the interval $[1/3, 2/3]$: If the answer is yes, the algorithm maintains the smallest item in this interval until a second item fits in, while ignoring all items outside of the interval. As soon as a second item fits, it is packed and all remaining items are rejected. If the answer is no, the algorithm maintains in the knapsack the largest item of size at least $1/3$ seen so far while packing all items smaller than $1/3$ as long as they fit. If the knapsack capacity is never exceeded, the solution is optimal. If the knapsack capacity is exceeded at some point, then there are items that are all smaller than $1/3$ but have a total size of more than $1/3$. Discard these items one by one, in arbitrary order, until we are within the capacity of the knapsack again. The remaining gap is at most $1/3$.

Han et al. [11, Thm. 6] have presented a randomized algorithm that relies on a partition of the items into six size classes. It is rather involved and hard to analyze, yet yields an expected competitive ratio of $10/7 \approx 1.428571$. Because it uses only a single random bit, it provides an upper bound for our case of one advice bit as well. In Theorem 6, we undercut this bound with a more manageable $\sqrt{2}$ -competitive algorithm that needs only five classes. We then complement this with a lower bound of $(1 + \sqrt{17})/4 = 4/(\sqrt{17} - 1) \approx 1.2808$.

► **Theorem 6.** *There is a $\sqrt{2}$ -competitive algorithm for PROPREMKNAP reading only one advice bit.*



■ **Figure 2** The partition of the interval $(0, 1]$ of possible sizes into the five subintervals used in the proof of Theorem 6 – namely $(0, a]$, $(a, b]$, $(b, c]$, $(c, d]$, and $(d, 1]$ – plus the corresponding class names. The values are $a = 1 - 1/\sqrt{2} \approx 0.293$, and $b = \sqrt{2} - 1 \approx 0.414$, and $c = 1/2$, and $d = 1/\sqrt{2} \approx 0.707$.

Proof. We split the interval $(0, 1]$ of possible sizes into subintervals at four points $a < b < c < d$. We will call the items with sizes in one of these five intervals *tiny*, *small*, *medium*, *big*, and *huge*, respectively. Formally, we partition the items into the five classes

$$P_{\text{tiny}} = \{i \mid 0 < s(i) \leq a\}, \quad P_{\text{small}} = \{i \mid a < s(i) \leq b\}, \quad P_{\text{medium}} = \{i \mid b < s(i) \leq c\}, \\ P_{\text{big}} = \{i \mid c < s(i) \leq d\}, \quad P_{\text{huge}} = \{i \mid d < s(i) \leq 1\},$$

where $a = 1 - 1/\sqrt{2} \approx 0.29289$, $b = \sqrt{2} - 1 \approx 0.41421$, $c = 1/2$, and $d = 1/\sqrt{2} \approx 0.70711$.

We will call the small and medium items the *little* ones collectively and refer to the big and huge items as the *large* ones. Accordingly, we let $P_{\text{little}} = P_{\text{small}} \cup P_{\text{medium}}$ and $P_{\text{large}} = P_{\text{big}} \cup P_{\text{huge}}$. See Figure 2 for an illustration of the subintervals and class names.

The oracle uses the one available advice bit to tell the algorithm which of the two strategies described below to apply. For the decision, the oracle picks an arbitrary optimal solution S to the given instance. If S contains a large item, the first strategy will be chosen, with one exception: If the instance contains no huge item but a little and a big item that fit into the knapsack together, then the first strategy is chosen only if a minimal big item appears in the instance before a minimal small item. In all other cases, the second strategy is implemented.

Strategy One If at any point a huge item appears, the algorithm packs it and keeps it until the end, discarding everything else. Otherwise, the algorithm operates with two slots, a primary and a secondary one. In the primary slot, it maintains the minimal big item and in the secondary slot it maintains the minimal little item. The primary slot takes precedence; that is, in case of a conflict where a new minimal item for one slot is presented that does not fit with the minimal item in the other slot, we discard the little item.

While maintaining the slot contents, tiny items are always packed greedily. If at any point a presented tiny item does not fit, the current contents of the knapsack are frozen and kept as they are until the instance has ended. The same happens after a step in which only tiny items have been discarded.

Strategy Two This strategy manages not only two but three slots, all of which maintain minimal items of some class. In order of precedence, the primary slot maintains two medium items, the secondary slot up to three small items, and the tertiary one big one. As an exception, if at any point a big item appears that can be packed alongside a currently packed small item by discarding everything else, then this is done and these two items are kept till the end. The tiny items are handled as before: They are packed greedily and if either a presented tiny item does not fit or only tiny items have been discarded in one step, then the current knapsack configuration is kept up to the very end.

We now need to carefully work through a case distinction according to the conditions listed in Table 1 and show that the algorithm's competitiveness is indeed bounded from above by $\max\{1/d, d/c, 1/2b, 1/(a+b), 1/(1-a), b/a\} = \sqrt{2}$. For the details, we refer to the full version of the paper. ◀

■ **Table 1** The mutually exclusive cases considered in Theorem 6.

Case	Strategy	Competitiveness	Case Conditions			
A	One	$1/d$	$ P_{\text{huge}} > 0$			
B	One/Two	d/c	$ P_{\text{huge}} = 0$	$ S \cap P_{\text{big}} > 0$	$ P_{\text{medium}} \leq 1$	
C	Two	$1/2b$	$ P_{\text{huge}} = 0$	$ S \cap P_{\text{big}} \geq 0$	$ P_{\text{medium}} > 1$	
D	Two	$1/(a+b)$	$ P_{\text{huge}} = 0$	$ S \cap P_{\text{big}} = 0$	$ P_{\text{medium}} = 1$	$ P_{\text{small}} > 0$
E	Two	b/a	$ P_{\text{huge}} = 0$	$ S \cap P_{\text{big}} = 0$	$ P_{\text{medium}} = 0$	$ P_{\text{small}} > 0$
F	Two	$1/(1-a)$	$ P_{\text{huge}} = 0$	$ S \cap P_{\text{big}} = 0$	$ P_{\text{medium}} \leq 1$	$ P_{\text{small}} = 0$

► **Theorem 7.** *No algorithm for PROPREMKNAP reading only a single advice bit can have a better competitive ratio than $(1 + \sqrt{17})/4$.*

Note again that an advice bit is at least as powerful as a random bit, hence Theorem 7 also improves the best known lower bound of $5/4$ for one random bit due to Han et al. [11, Thm. 8].

4.3 Near Optimality with Constant Advice

Having seen how much advice is necessary for optimality and what the effect of a single advice bit can be, we now address the entire range in between. For this, we prove the following generalization of Theorem 7.

► **Theorem 8.** *Let an arbitrary integer $k > 1$ be given. No algorithm for PROPREMKNAP reading at most $\log k$ advice bits can achieve a better competitive ratio than $4/(3 - 2k + \sqrt{4k(k+1) - 7})$.*

We remark that Theorem 8 and its analogue for REMKNAP instead of PROPREMKNAP, Theorem 13, improve upon the best known lower bounds implied by Han et al.’s results on the resource buffer model [13, Thms. 17 and 6]. In this model, the online algorithm may use a knapsack of some increased capacity $R > 1$, but only until the instance ends, at which point it has to choose from the reserved items a selection that fits a knapsack of capacity one. A resource buffer of some natural size R allows us to simulate any algorithm using up to $\log R$ advice bits: We think of the resource buffer as split into R knapsacks of capacity 1, allowing us to accommodate the items stored by the advice algorithm for every possible advice string simultaneously.

Clearly, the lower bound of Theorem 8 tends to 1 for increasingly large but still constant advice. With our most surprising result for the proportional knapsack problem, Theorem 10, we will prove that the true competitive ratio displays the same general behavior as the lower bound of Theorem 8: For any given $\varepsilon > 0$, we can guarantee a competitive ratio of $1 + \varepsilon$ with a constant number of advice bits. It is of course also possible to derive more specific upper bounds for very few advice bits such as the following one.

► **Theorem 9.** *There is a $4/3$ -competitive algorithm for PROPREMKNAP reading two advice bits.*

We now turn to our main result for the proportional knapsack problem, which complements Theorem 8 with an upper bound. Theorem 14 will generalize this result to the general version where an item’s size may differ from its value, albeit with a far more complicated proof. To

18:10 Removable Online Knapsack and Advice

make it as easily understandable as possible, we first present here the proof for the simple variant, which introduces the idea of slots that are reserved for items with certain properties. This will serve as a useful foundation for the proof of the general variant, which is also making use of such a slot system, although as merely one besides many more components.

► **Theorem 10.** *For any $\varepsilon > 0$, there is a strictly $(1 + \varepsilon)$ -competitive algorithm for PROPPACK reading a constant number of advice bits.*

Proof. We describe such an algorithm called PROPPACK; see the full version of the paper for a pseudo-code implementation. We begin by describing the advice communicated to PROPPACK with a constant number of bits, then explain how the algorithm operates on this advice, prove that it is correct and terminates, and finally analyze its competitive ratio.

Notions and Notation. Without loss of generality, we assume that all items have size at most 1 and that $\varepsilon \leq 1/2$. We define the constant $K = \lceil \log_{1-\varepsilon/2} \varepsilon/2 \rceil$.

Let an instance with n items be given. Denote the items in the order of their appearance in the instance by $1, 2, \dots, n$ and denote the size of item i by $s(i)$. We divide the n items into *small* and *big* ones, with $\delta = (1 - \varepsilon/2)^K$ serving as the dividing line: $C_{\text{small}} = \{i \mid s(i) \leq \delta\}$ and $C_{\text{big}} = \{i \mid \delta < s(i)\}$. We further partition the big items into the subclasses $C_k = \{i \mid (1 - \varepsilon/2)^k < s(i) \leq (1 - \varepsilon/2)^{k-1}\}$ for $k \in \{1, \dots, K\}$. To alleviate the notation, we will often refer to C_k as class k and to C_{small} as class 0. We also use this convention when writing $C(i)$ to indicate the class to which item i belongs: We have $C(i) \in \{0, \dots, K\}$, with $C(i) = 0$ meaning that $i \in C_{\text{small}}$ and $C(i) = k \neq 0$ meaning that $i \in C_k$.

The oracle chooses an arbitrary but fixed optimal solution $S \subseteq \{1, \dots, n\}$. We denote the partition classes that are naturally induced by this solution by $S_{\text{small}} = S \cap C_{\text{small}}$, $S_{\text{big}} = S \cap C_{\text{big}}$, and $S_k = S \cap C_k$ for $k \in \{1, \dots, K\}$. Let $m = |S_{\text{big}}|$ be the number of big items in the optimal solution and denote them by $i_1 < \dots < i_m$ in order of appearance.

Constant Advice. The oracle communicates to the algorithm a tuple (b_1, \dots, b_m) with the classes of the big items in the chosen optimal solution in order of appearance; that is, we have $b_j = C(i_j)$ for each $j \in \{1, \dots, m\}$. We remark that this tuple needs to be encoded in a self-delimiting way. A constant number of bits suffices for this because b_j is bounded by the constant K for every $j \in \{1, \dots, m\}$ and m is bounded by the constant $1/\delta$. The latter bound is an immediate consequence of the fact that $s(S_{\text{big}}) \leq 1$ and that any big item has a size larger than δ .

Algorithm Description. The algorithm PROPPACK proceeds in m phases as follows. In every phase, the algorithm opens a new virtual *slot* within the knapsack that can store exactly one item at a time; multiple items in succession are allowed, however. The slot opened in phase i will accommodate items belonging to class b_i exclusively; we say that items from this class *match* slot i . Slots are never closed, thus there are exactly m of them in the end. Small items are generally packed in a greedy manner and discarded one by one whenever necessary to pack a big item.

In the first phase, the algorithm rejects all big items until one of class b_1 appears. As soon as this is the case, said item is packed into the first slot, ending the first phase.

In the second phase, the algorithm opens the second slot to pack a matching item, that is, one of class b_2 . It waits for the first item from this class that fits into the knapsack alongside the item in the first slot. As soon as such an item appears, it is packed and the

phase ends. In the meantime, whenever an item of class b_1 appears during the second round, the algorithm substitutes it for the one stored in the first slot if and only if this reduces the size of the stored item.

In general, phase i begins with the opening of slot i , which is reserved for items of class b_i . The phase continues until an item appears that both matches the newly opened slot and fits in beside the items currently stored in the previously opened and filled slots without exceeding the capacity. Then this item is packed into the new slot, which ends the phase. During the entire phase, the algorithm maintains in all filled slots the smallest matching items seen so far: Whenever the algorithm is presented with a big item that either belongs to a class other than b_i or does not fit in alongside the items in the previously opened slots, then the new item replaces a largest item in the matching open slots, unless the new item itself is even larger.

The entire time, even after the last phase has terminated, small items are packed greedily and discarded one by one whenever this is necessary to make room for a big item according to the description above. Moreover, we may assume that, whenever a new item has been packed into the knapsack, the algorithm sorts the items in the matching open slots in increasing order. This sorting is not necessary for the algorithm to fulfill its duty, but it facilitates the proof by induction below.

Termination of All Phases. We need to show that PROPPACK does in fact finish all m phases; that is, all m slots will be filled with a matching item without ever exceeding the knapsack capacity. Consider the big items of the optimal solution, which we denote by $u_1 < \dots < u_m$ in their order of appearance. To ensure the termination of all phases, we prove by induction over $i \leq m$ that, after processing item u_i , the first i slots store items with a total size of $s(u_1) + \dots + s(u_i)$ or less.

We may start from $i = 0$ as the trivial, if degenerate, base case. For the induction step, assume the hypothesis for $i < m$ and observe that no item in a slot is ever replaced by a larger one. Therefore, the items in the first i slots still have a total size of at most $s(u_1) + \dots + s(u_i)$ when u_{i+1} is presented.

There are now three possibilities. If slot $i + 1$ has remained closed up to this point, it is now opened and filled with u_{i+1} , which fits in because $s(u_1) + \dots + s(u_{i+1}) \leq s(S_{\text{big}}) \leq 1$. Otherwise, slot $i + 1$ is already storing an item: If said item is larger than $s(u_{i+1})$, then u_{i+1} replaces either this item or one that is at least as large. During the subsequent sorting, u_{i+1} is then moved to slot $i + 1$ or one of the slots from 1 to i , which may force some items from slots 1 through i into higher slots but never beyond slot $i + 1$. The third possibility is that slot $i + 1$ contains an item of size at most $s(u_{i+1})$ already. We immediately obtain the induction claim for $i + 1$ in all three cases.

Competitive Analysis. We still denote by S the optimal solution that served as the basis for the given advice, by T the final output of the online algorithm, and the respective partition classes by S_{small} , S_{big} , S_k and T_{small} , T_{big} , and T_k .

Since PROPPACK opens one slot for each big item in the optimal solution T and fills it with an item from the same subclass, as proved above, we have $|S_k| = |T_k|$ for every $k \in \{1, \dots, K\}$. Moreover, the sizes within a subclass C_k vary by a factor of at most $1 - \varepsilon/2$; this means that we can bound both $s(S_{\text{big}})$ and $s(T_{\text{big}})$ from below by $L = \sum_{k=1}^K |S_k|(1 - \varepsilon/2)^k$ and from above by $L/(1 - \varepsilon/2)$. We conclude $s(T_{\text{big}}) \geq s(S_{\text{big}}) \cdot (1 - \varepsilon/2)$.

Furthermore, since small items are packed greedily and only discarded one by one whenever necessary to make room for the big items, we will not lose much from their side either. If the presented small items have a total size of at most $1 - L/(1 - \varepsilon/2)$, none is ever discarded.

18:12 Removable Online Knapsack and Advice

In this case, we have $s(T_{\text{small}}) \geq s(S_{\text{small}})$ and thus immediately $s(T) \geq s(S) \cdot (1 - \varepsilon/2)$. If small items are discarded, however, the worst case is the following type of instance: It starts with only small items of the largest possible size δ , some of which are then discarded to accommodate big items with sizes right at the upper limit for the classes indicated by the advice, leaving a gap of almost δ , follows up with slightly smaller big items that are in the optimal solution and would not have lead to any discarded small items, and finally presents big items at the lower end of the size span, replacing all previously packed big items.

Even in this worst case, the algorithm remains $(1 - \varepsilon/2)$ -competitive on the big items and detracting the largest possible loss of δ on the small items yields $s(T) \geq s(S) \cdot (1 - \varepsilon/2) - \delta$. By the definition of δ and K and due to the simple fact that $s(S)$ is at most 1, we have $\delta = (1 - \varepsilon/2)^K \leq \varepsilon/2 \leq s(S) \cdot \varepsilon/2$. This implies $s(T)/s(S) \geq 1 - \varepsilon$, as desired. ◀

5 Results for General Removable Knapsack

First, we note that all lower bounds for the proportional removable knapsack problem carry over to the general removable knapsack problem, in particular Theorem 5.

Iwama and Zhang [18] have shown that the competitive ratio of REMKNAP is unbounded without advice. This can be seen using an interactive instance that starts with an item $(1, 1)$ and then presents items $(\varepsilon^2, \varepsilon)$ repeatedly, up to $1/\varepsilon^2$ times, until one is packed, at which point the instance ends.

The following two theorems show REMKNAP's competitiveness for one advice bit to be exactly 2.

► **Theorem 11.** *There is a 2-competitive algorithm for REMKNAP reading only a single advice bit.*

► **Theorem 12.** *No algorithm for REMKNAP reading only a single advice bit can have a competitive ratio better than 2.*

The existence of a 2-competitive algorithm already follows from a result by Han et al. [11, Thm. 9], who proved the statement even for a single random bit instead of an advice bit. We can prove Theorem 11 by describing a concrete advice algorithm. Advice being more powerful than randomness, Theorem 12 also closes the remaining gap for barely random algorithms by lifting the previously best known lower bound by Han et al. [11, Thm. 12] from $1 + 1/e \approx 1.367$ to 2.

The analysis of the hard instance presented in the proof of Theorem 12 could be adapted to the case of more than one advice bit. Two advice bits would mean that the oracle can provide to the algorithm one out of four advice strings instead of the two advice strings possible with one bit. We may also consider an intermediate advice algorithm limited to three advice strings, which corresponds to $\log 3$ advice bits. Such an algorithm cannot achieve a competitive ratio better than $2/\Phi = 4/(1 + \sqrt{5}) \approx 1.2361$ since it is forced to use one of the three advice strings to keep the most valuable item x_0 , one to pack the items y_1, \dots, y_k yield-greedily, and one to keep the x_j with the value $\Phi \approx 1.618$, resulting in a competitive ratio of $2/\Phi = 2\Phi/(1 + \Phi)$.

This approach deteriorates too quickly, however. Adapting Theorem 8 to the case of REMKNAP is the better choice; this results in Theorem 13, which already yields a better bound in the case of three advice strings, that is, for $k = 3$.

► **Theorem 13.** *Let an arbitrary integer $k > 1$ be given. No algorithm for REMKNAP reading at most k advice bits can achieve a better competitive ratio than $1/2 + \sqrt{1/4 + 1/k}$.*

We point out again that Theorem 13 slightly improves over the lower bounds known from the resource buffer model by Han et al. [13, Thm. 6], as illustrated after the proof of Theorem 13 in the full version of the paper.

We now move on to the core result of this paper, proving that a constant amount of advice bits is sufficient to reach a near-optimal competitive ratio not only for the proportional but even for the general removable knapsack problem. This will complete the picture of the global advice behavior of the online knapsack problem with removability outlined in Figure 1.

We first point out that algorithm PROPPACK generally does not work on instances where the value of an item can vary independently of its size, as seen by the following counterexample: Assume that $1/2$ lies in the interior of some size class and choose an $\varepsilon > 0$ such that $1/2 + \varepsilon$ and $1/2 - \varepsilon$ are still in the same class. Present two items $(1/2 + \varepsilon, 2)$ and $(1/2, 1)$. If the algorithm picks the first one, $(1/2, 2)$ is presented as the last item; otherwise, the instance ends with the item $(1/2 - \varepsilon, 1)$. In both cases, the algorithm achieves a total value of 2, whereas the optimum is 3. The advice does not help us to distinguish the two cases, it only tells us that the optimal solution contains two items from the class. Clearly, we have to adapt the algorithm to take the value of the items into account somehow. A major obstacle is that the online algorithm has no bound on the values of appearing items, thus the algorithm has no way of reconstructing the constantly many value classes used by the oracle just from the parameter ε .

Moreover, the proof of Theorem 10 cannot be adapted for the general case in any simple way. Using only classes based on size, it is impossible for the algorithm to know, when maintaining an item in a slot, how to balance minimizing the size against maximizing the value. On the one hand, if the size is not minimized, then the excess size may prevent other slots from being filled. On the other hand, not maximizing the values, the algorithm may incur an arbitrarily high loss because the potential values of items cannot be bounded. This is also the reason why simple value classes do not work either. The algorithm does not know the maximal value occurring in the instance until it has ended and can therefore not use it as a reference point, in contrast to the size classes that can be chosen relative to the known knapsack capacity.

A first step toward solving these issues is the definition of dynamic value classes that are anchored to both the value of the first item appearing in the instance and to the optimal solution value. The latter is of course also unknown to the algorithm until the instance ends. However, we are able to define our classes with some additional properties that enable our algorithm to compute at any point useful provisional bounds on the optimal solution value. These bounds will either turn out to be valid or the algorithm is able to notice that they are off just in time to adjust and take a fresh start before having lost too much due to bad decisions. The adversary may foil the algorithm over and over, forcing it to abandon its plans and adjust the bounds arbitrarily often.

To properly deal with these repeated resets, we develop a *level system*. One major challenge is to square the level system with some sort of *slot system* as used by the algorithm for the proportional case. We manage to do this by introducing the concept of a *virtual algorithm*, which has the special, even though only imagined, capability of keeping one item in a *splitting slot* and use arbitrary fractions of the item stored in it. We then describe an actual algorithm that tries to equal the idealized performance of the virtual algorithm without making use of the splitting slot. While it cannot quite achieve this, it will fare well enough in the end. Having one algorithm emulating another, we are going to prove the claimed competitiveness in two stages, first for the virtual version and then for the actual algorithm.

This split analysis presents several further challenges, for example, a desynchronization of the current phase and the number of slots filled by the algorithm, which coincided in the proportional case. The necessary adaptations entail a number of further challenges, for example a judicious handling of the *paltry* items, which are worth almost nothing individually, yet may be too numerous to neglect. In fact, the algorithm will need to partition the items not only by their value but simultaneously by their size as well. Overcoming these and a few other obstacles, we are able to prove in the full version of the paper our final theorem.

► **Theorem 14.** *For any $\varepsilon > 0$, there is a strictly $(1+\varepsilon)$ -competitive algorithm for REMKNAP reading a constant number of advice bits.*

Proof. Proving this theorem requires far more effort than what was necessary for its proportional counterpart Theorem 10, where an item's value is always identical to its size. Having explained already why any straightforward adaption of the substantially simpler approach for the proportional variant is impossible, we now provide a high-level outline of the proof.

High-Level Outline. As announced, we first provide a high-level outline of the workings of the advice algorithm.

The oracle chooses an arbitrary optimal solution S and, based on its value $v(S)$, partitions the items of the instance into *precious* and *paltry* ones; the paltry ones are those worth less than $\varepsilon_{\text{paltry}} \cdot v(S)$ for a suitable $\varepsilon_{\text{paltry}} > 0$. The precious items are further split into finitely many classes such that the item values within any class are at most some factor $1 - \varepsilon_{\text{spread}}$ apart from each other.

The advice will encode exactly the classes of the precious items from the optimal solution S in their order of appearance; the goal is to ensure that the algorithm packs just as many precious items from each class as the optimal solution does, thus achieving a competitive factor of $1/(1 - \varepsilon_{\text{spread}})$ on these items. Assuming that the algorithm knows the exact value ranges of each class, it can achieve this using a system of *slots*, which will be filled with precious items in two stages: first just *virtually* – assuming the algorithm were able to do certain things that are in fact impossible – and then also *actually* at some point. Each virtual filling of a slot starts a new *phase* of the algorithm.

However, the algorithm knows only the target value but nothing about the size of the items belonging into each slot; it is necessary to prove that despite this, the algorithm is able to fill the slots in the right order without blocking important items yet to come.

And there is another problem: It is even impossible for the oracle to communicate to the algorithm the exact value range of each class with a constant amount of advice since there is no bound on the potential values occurring in an instance. Instead, the value ranges will be described in relation to the value of the first item of the instance, and merely modulo some constant factor. The algorithm will then operate under the assumption that the first item is a precious one, in which case all of the above will work out. Since the algorithm cannot know for sure which items are precious, it divides them into *presumably precious* and *provenly paltry* ones according to some computations based on the advice and the instance seen so far. If the algorithm's assumption is mistaken, it is able to recognize this just in time by continually comparing the best solution realizable with the already presented items – whether they have been accepted or not – to a rather intricate estimate for the optimal solution value $v(S)$. Once the algorithm discovers its mistake, it resets with a revised set of assumptions on the value ranges; we say that the algorithm *levels up*. This is done such that the algorithm can go through arbitrarily many levels, resetting and taking a fresh start as often as necessary without incurring more than a negligible value loss.

The algorithm also needs to take care of the paltry items, which might constitute a considerable part of the optimal solution if there are sufficiently many of them. The algorithm is packing the paltry items in a somewhat inhibited greedy manner that optimizes the value-to-size ratio, which we also call *yield*. The volume taken up by the paltry items will be restricted sufficiently to guarantee that the precious items can always fill their slots, but not as severely as to lose too much value on the paltry items. The right volume restrictions in each phase of the algorithm are communicated via a constant amount of advice as well. Again, this cannot happen directly, since the right volume range might be infinitesimally small. Instead, the volume is controlled indirectly, via bounds not on some size but instead on the value that is provided by the paltry items packed before filling the current slot with a precious item.

Communicating the necessary volumes in this way is possible only up to some precision $\varepsilon_{\text{round}}$, and an overestimation could mean the loss of a crucial precious item. If we always round down, then this cannot happen, but the algorithm might reject some paltry item it should have kept for a selection of maximum yield. This is negligible if it happens only once, but we cannot tolerate taking such a loss in every phase, with every new volume bound. The solution is to analyze the situation using a special *splitting slot*, which can accommodate one paltry item at the time outside of the knapsack. We imagine the splitting slot lending us from the item stored in it any desired fraction at any time – just always the same fraction of the value and size. We refer to the item currently stored in the splitting slot as the *split item*. We will consider an algorithm that maintains a split item of highest yield after the remaining paltry items kept in the knapsack. This imaginary algorithm is an antecedent to our real advice algorithm, which builds on it but cannot actually split any items of course; we refer to the purely hypothetical precursor as the *virtual version* of our *actual algorithm*.

The actual algorithm will mimic the virtual version as closely as possible and deliver a result that is only marginally worse. Whenever the virtual version splits an item, the actual algorithm needs to decide whether to discard this item or store it completely. The challenge is to take the right decisions to allow for all slots to be filled in time and also avoid an undue accumulation of losses by passing on too many split items.

The advice helps the actual algorithm by indicating for every phase whether an item stored in the splitting slot by the virtual version is to be packed or discarded. This is done in such a way that in the end, the actual algorithm will have relinquished only the value of a single paltry item, namely the one kept in the splitting slot when the instance ends.

Packing entire items from the splitting slot comes with problems on its own; these paltry items might block for the actual algorithm some precious items that are packed by the virtual version. This problem is addressed by further advice to the algorithm on how to prioritize the packing of precious versus paltry items in each phase. This advice, telling the algorithm when to *actualize* a virtual packing of an item, is based on the solution that the virtual version would eventually produce if it existed. The virtual version does not depend on the actual algorithm and has no need for the part of the advice on actualization, which avoids any circular reasoning.

There are a few more technical issues to be dealt with, for example the special case that the precious items contribute only marginally to the value of the optimal solution. This undermines the estimates for the optimal solution value, is thus flagged by a dedicated advice bit b_{small} , and handled by switching from the elaborate value-based limits that dampen the general yield-greedy strategy to a simpler size-sensitive strategy. ◀

References

- 1 Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.
- 2 Hans-Joachim Böckenhauer, Elisabet Burjons, Juraj Hromkovič, Henri Lotze, and Peter Rossmanith. Online simple knapsack with reservation costs. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16–19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPICs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 3 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Kráľovič, Richard Kráľovič, and Tobias Mömke. On the advice complexity of online problems. In *ISAAC 2009*, number 5878 in LNCS, pages 331–340, 2009.
- 4 Hans-Joachim Böckenhauer, Dennis Komm, Richard Kráľovič, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theoretical Computer Science*, 527:61–72, 2014.
- 5 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
- 6 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2):19:1–19:34, 2017.
- 7 Marek Cygan, Łukasz Jeż, and Jiří Sgall. Online knapsack revisited. *Theory Comput. Syst.*, 58(1):153–190, 2016.
- 8 George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–277, 1957.
- 9 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.
- 10 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Online unweighted knapsack problem with removal cost. *Algorithmica*, 70(1):76–91, 2014.
- 11 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Randomized algorithms for online knapsack problems. *Theoretical Computer Science*, 562:395–405, 2015.
- 12 Xin Han, Yasushi Kawase, Kazuhisa Makino, and He Guo. Online removable knapsack problem under convex function. *Theoretical Computer Science*, 540:62–69, 2014.
- 13 Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku. Online knapsack problems with a resource buffer. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8–11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPICs*, pages 28:1–28:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 14 Xin Han and Kazuhisa Makino. Online removable knapsack with limited cuts. *Theoretical Computer Science*, 411(44-46):3956–3964, 2010.
- 15 Juraj Hromkovič, Rastislav Kráľovič, and Richard Kráľovič. Information complexity of online problems. In *MFCS 2010*, number 6281 in LNCS, pages 24–36. Springer, 2010.
- 16 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4), 1975.
- 17 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *ICALP 2002*, number 2380 in LNCS, pages 293–305. Springer, 2002.
- 18 Kazuo Iwama and Guochuan Zhang. Online knapsack with resource augmentation. *Information Processing Letters*, 110(22):1016–1020, 2010.
- 19 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum, 1972.
- 20 Dennis Komm. *An Introduction to Online Computation – Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.
- 21 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68:73–104, 1995.

- 22 John Noga and Veerawan Sarbua. An online partially fractional knapsack problem. In *8th International Symposium on Parallel Architectures, Algorithms, and Networks, ISPAN 2005, December 7-9, 2005, Las Vegas, Nevada, USA*, pages 108–112. IEEE Computer Society, 2005.
- 23 Peter Rossmanith. On the advice complexity of online edge- and node-deletion problems. In *Adventures Between Lower Bounds and Higher Altitudes – Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, number 11011 in LNCS, pages 449–462. Springer, 2018.

The Complexity of Homomorphism Reconstructibility

Jan Böker  

RWTH Aachen University, Germany

Louis Härtel  

RWTH Aachen University, Germany

Nina Runde  

RWTH Aachen University, Germany

Tim Seppelt  

RWTH Aachen University, Germany

Christoph Standke  

RWTH Aachen University, Germany

Abstract

Representing graphs by their homomorphism counts has led to the beautiful theory of homomorphism indistinguishability in recent years. Moreover, homomorphism counts have promising applications in database theory and machine learning, where one would like to answer queries or classify graphs solely based on the representation of a graph G as a finite vector of homomorphism counts from some fixed finite set of graphs to G . We study the computational complexity of the arguably most fundamental computational problem associated to these representations, the *homomorphism reconstructibility problem*: given a finite sequence of graphs and a corresponding vector of natural numbers, decide whether there exists a graph G that realises the given vector as the homomorphism counts from the given graphs.

We show that this problem yields a natural example of an $\text{NP}^{\#P}$ -hard problem, which still can be NP-hard when restricted to a fixed number of input graphs of bounded treewidth and a fixed input vector of natural numbers, or alternatively, when restricted to a finite input set of graphs. We further show that, when restricted to a finite input set of graphs and given an upper bound on the order of the graph G as additional input, the problem cannot be NP-hard unless $\text{P} = \text{NP}$. For this regime, we obtain partial positive results. We also investigate the problem's parameterised complexity and provide fpt-algorithms for the case that a single graph is given and that multiple graphs of the same order with subgraph instead of homomorphism counts are given.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Parameterized complexity and exact algorithms; Mathematics of computing \rightarrow Graph theory

Keywords and phrases graph homomorphism, counting complexity, parameterised complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.19

Related Version *Full Version*: <https://arxiv.org/abs/2310.09009> [4]

Funding *Jan Böker*: European Union (ERC, SymSim, 101054974).

Nina Runde: German Research Foundation (DFG) – 453349072.

Tim Seppelt: European Union (ERC, SymSim, 101054974), German Research Foundation (DFG) – GRK 2236/2 (UnRAVeL).

Christoph Standke: German Research Foundation (DFG) – GR 1492/16-1, GRK 2236/2 (UnRAVeL).

Acknowledgements We would like to thank Martin Grohe, Athena Riazsadri, and Jan Tönshoff for fruitful discussions.



© Jan Böker, Louis Härtel, Nina Runde, Tim Seppelt, and Christoph Standke; licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 19; pp. 19:1–19:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

Representing a graph in terms of homomorphism counts has proven to be fruitful in theory and applications. Many graph properties studied in logic [16, 21], algebraic graph theory [15], quantum information theory [40], and convex optimisation [51, 23] can be expressed as homomorphism counts from some family of graphs. Homomorphism counts provide a basis for other counting problems [10] and have been studied extensively using diverse tools ranging from algorithmics [15] to algebra [23, 40], from combinatorics [50, 53] to category theory [14, 1]. In database theory, they correspond to evaluations of Boolean conjunctive queries under bag semantics [7, 34]. In graph learning, representations of graphs as vectors of homomorphism counts yield embeddings into a continuous latent space and underpin theoretically meaningful and successfully field-tested machine learning architectures [45, 6, 58].

In this work, we consider representations of a graph G as a finite vector of homomorphism counts $\text{hom}(\mathcal{I}, G) \in \mathbb{N}^{\mathcal{I}}$ for some finite set of graphs \mathcal{I} , which we call a *homomorphism embedding*. The rich theory of homomorphism counts calls for algorithmic applications of homomorphism embeddings: in database theory, one would ideally like to decide properties of the graph G having access to the vector $\text{hom}(\mathcal{I}, G)$ only [22, 8]. In graph learning, certain entries of the homomorphism embedding might be associated with desirable properties of the graph being embedded, and one would like to be able to synthesise a graph having these desirable properties from a vector in the latent space [5, 24]. Despite its ubiquity in the contexts described above, homomorphism embeddings have not undergone a systematic complexity-theoretic analysis yet. The arguably most fundamental computational problem associated to them is to decide whether a vector $h \in \mathbb{N}^{\mathcal{I}}$ actually represents a graph, i.e. it is of the form $h = \text{hom}(\mathcal{I}, G)$ for some graph G . Therefore, we consider the following *homomorphism reconstructibility problem* for graph classes \mathcal{F} and \mathcal{G} :

HOMREC(\mathcal{F}, \mathcal{G})

Input Pairs $(F_1, h_1), \dots, (F_m, h_m) \in \mathcal{F} \times \mathbb{N}$ where h_1, \dots, h_m are given in binary.

Question Is there a graph $G \in \mathcal{G}$ such that $\text{hom}(F_i, G) = h_i$ for every $i \in [m]$?

We simply write **HOMREC**(\mathcal{F}) if \mathcal{G} is the class of all graphs. While the problem has a clean-cut motivation in practical applications, one quickly encounters surprising connections to deep theoretical results and long-standing open questions. One does not only have to carefully keep distance to the notorious graph reconstruction conjecture of Ulam [46], but also be aware that various decision problems involving the set $R(\mathcal{I})$ of all vectors $\text{hom}(\mathcal{I}, G) \in \mathbb{N}^{\mathcal{I}}$ where G ranges over all graphs have received much attention recently, e.g. the homomorphism domination problem [32], whose decidability is open, the homomorphism determinacy problem [34], or the undecidable problem of determining whether inequalities of homomorphism counts hold [28, 25]. Beyond computational concerns, an abstract characterisation of the set $R(\mathcal{I})$ akin to [18, 37] is desirable, yet elusive [3].

In this paper, we establish a firm grasp on the computational complexity of the homomorphism reconstructibility problem **HOMREC**(\mathcal{F}, \mathcal{G}) by exploring from which of its aspects computational hardness arises and then finding restrictions for which efficient algorithms can be found. Despite the interest in the problem, surprisingly little progress on this question has been made. The only result related to our work is a theorem from [31], which asserts that a variant of **HOMREC**(\mathcal{F}, \mathcal{G}) with Boolean subgraph constraints instead of homomorphism counts is NP^{NP} -complete. In particular, a formal definition of **HOMREC**(\mathcal{F}, \mathcal{G}) has not been made before, and we would like to remark on a curious peculiarity of the definition we have chosen: a polynomial-time algorithm for **HOMREC**(\mathcal{F}, \mathcal{G}) may exploit algebraic properties of

homomorphism numbers and deduce via arithmetic operations on the given numbers whether these can be realised by a graph G or not; let us call an algorithm of this type *arithmetic*. However, it is also conceivable that an algorithm for $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ would operate by explicitly constructing the graph G , for example, in a dynamic-programming fashion; let us call such an algorithm *constructive*. A constructive algorithm seems just as reasonable as an arithmetic one but may not be a polynomial-time algorithm for $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ since the order of G does not have to be polynomial in the length of the binary encoding of the given homomorphism numbers, even if one of the constraint graphs is $F_i = \bullet$. Hence, it also seems reasonable to define the *bounded homomorphism reconstructability problem* $\text{BHOMREC}(\mathcal{F}, \mathcal{G})$ where an additional input $n \in \mathbb{N}$ given in unary imposes a bound on the number of vertices in G that is linear in the input encoding. This bound, however, poses an additional constraint to the graph G , which may make the design of arithmetic algorithms more difficult or impossible. Hence, both $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ and $\text{BHOMREC}(\mathcal{F}, \mathcal{G})$ are arguably reasonable definitions of the reconstructability problem, each in their own right, and we consider both.

The Ocean of Hardness

Let C=P denote the class of all decision problems L for which there is a function $f \in \#\text{P}$ and a polynomial-time computable function g such that, for every instance x of L , we have $x \in L \iff f(x) = g(x)$ [54, 57, 26]. We first show that both the unbounded and the bounded reconstructability problem are $\text{NP}^{\text{C=P}}$ -hard when not restricted in any way. Note that $\text{NP}^{\text{C=P}} = \text{NP}^{\#\text{P}}$ since, whenever we issue a call to the $\#\text{P}$ -oracle, we may guess the output using nondeterminism and verify it using a C=P -oracle instead [56, 9]; we refer to the full version [4] for more details about counting classes.

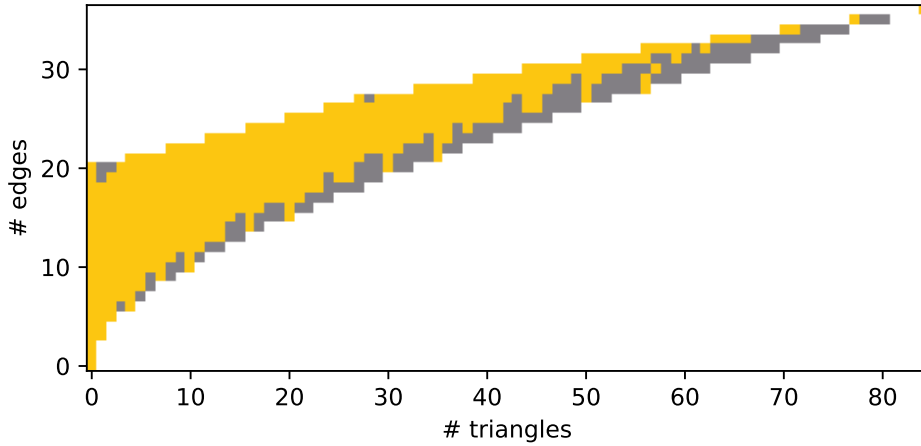
► **Theorem 1.** *Let \mathcal{F} denote the class of all graphs. Then, $\text{HOMREC}(\mathcal{F})$ is in NEXP and $\text{BHOMREC}(\mathcal{F})$ is in $\text{NP}^{\text{C=P}}$. Moreover, both problems are $\text{NP}^{\text{C=P}}$ -hard. Hence, they are not contained in the polynomial hierarchy unless it collapses.*

This result illustrates two intertwined sources of hardness in the reconstructability problem: first, the *reconstruction hardness* of finding a graph G , which corresponds to the class NP , and secondly, the *counting hardness* of verifying that G actually satisfies the given constraints, which corresponds to the C=P -oracle. The reconstruction hardness is what we are interested in since the counting hardness simply reflects the hardness of counting homomorphisms, which is well understood: the problem $\#\text{HOM}$ of counting the number of homomorphisms from a graph F to a graph G is $\#\text{P}$ -complete and, under the complexity-theoretic assumption that $\text{FPT} \neq \#\text{W}[1]$, becomes tractable if and only if one restricts the graphs F to come from a (recursively enumerable) class of bounded treewidth [12].

To isolate the reconstruction hardness, we restrict \mathcal{F} to be a class of bounded treewidth, which allows us to verify in polynomial time that a graph G satisfies all given constraints. In particular, the bounded problem is then in NP since one can guess a graph G of the given size and verify that it satisfies all constraints in polynomial time. We show that the intuition conveyed by Theorem 1 is correct: $\text{HOMREC}(\mathcal{F})$ is still NP -hard if \mathcal{F} has bounded treewidth and even remains NP -hard when only a constant number of constraints is allowed to appear in the input.

► **Theorem 2.** *There is a class \mathcal{F} of graphs of bounded treewidth such that $\text{HOMREC}(\mathcal{F})$ and $\text{BHOMREC}(\mathcal{F})$ are NP -hard.*

The reduction used to prove Theorem 2 further demonstrates that both problems remain NP -hard when only allowing some fixed number of constraints: it produces a family of instances with graphs from \mathcal{F} where the number of constraints, all homomorphism numbers,



■ **Figure 1** Reconstructable \triangle and $\bullet\text{---}\bullet$ subgraph counts (yellow) for graphs on nine vertices. The grey area depicts the values which are not ruled out by the Kruskal–Katona bound [33, 30], cf. [36, 13.31b], or the Razborov bound [49], cf. [38, Theorem 16.14]. The \triangle -counts realisable by graphs on nine vertices correspond to columns with at least one yellow box.

and all but one constraint graph are fixed. This raises the question what happens if we fix all input graphs and allow the homomorphism numbers to vary instead, i.e. does $\text{HOMREC}(\mathcal{F})$ become tractable if \mathcal{F} is *finite*? Even though the input to $\text{HOMREC}(\mathcal{F})$ for finite \mathcal{F} essentially consists only of natural numbers encoded in binary, we are still able to show that $\text{HOMREC}(\mathcal{F})$ is also NP-hard in this case. In contrast, however, $\text{BHOMREC}(\mathcal{F})$ is *sparse* for finite \mathcal{F} , i.e. it only has polynomial number of yes-instances, and hence, unlikely to be NP-hard [39].

► **Theorem 3.** *There is a finite set \mathcal{F} of graphs such that $\text{HOMREC}(\mathcal{F})$ is NP-hard. If $\text{BHOMREC}(\mathcal{F})$ is NP-complete for a finite set \mathcal{F} of graphs, then $\text{P} = \text{NP}$.*

Hence, the complexity of the reconstructability problem becomes much more nuanced for finite \mathcal{F} , and in order to design efficient algorithms for it, we seemingly have to focus on $\text{BHOMREC}(\mathcal{F})$ for finite \mathcal{F} . The fact that there are only polynomially many feasible combinations of homomorphism numbers in this case might be somehow exploitable, e.g. by a dynamic-programming algorithm operating on a table indexed by them.

Islands of Tractability

The first tractable instance of $\text{HOMREC}(\mathcal{F})$ that comes to mind is given by $F_1 = \bullet$ and $F_2 = \bullet\text{---}\bullet$ and $h_1, h_2 \in \mathbb{N}$. In this case, we need to decide whether $h_2 \leq h_1(h_1 - 1)$ and h_2 is even. Although this is fairly trivial, we encounter severe combinatorial difficulties when attempting to generalise this even to $F_1 = \bullet$ and $F_2 = \triangle$. Figure 1 shows that the set of reconstructible vectors has a non-trivial shape. In particular, while highly engineered results from extremal combinatorics [38, 49, 27, 20] provide insights in the (asymptotic) behaviour of the upper and lower boundary of that set, we are unable to characterise the seemingly erratic gaps and spikes depicted in Figure 1. On the positive side, we are able to map out large regions of reconstructible vectors using number-theoretic insights:

► **Theorem 4.** *There exists a function $\gamma: \mathbb{N} \rightarrow \mathbb{N}$ such that for every $k \geq 2$, $n \geq 1$, $h \leq \binom{n}{k}$, there exists a graph G on $n + \gamma(k - 1) - 1$ vertices such that $\text{sub}(K_k, G) = h$.*

Theorem 4 allows for the construction of graphs with almost all possible numbers of clique subgraphs. Indeed, the proportion of covered values is $\binom{n}{k} / \binom{n+\gamma(k-1)-1}{k} = 1 - o(1)$ for $n \rightarrow \infty$ and fixed k . In other words, all sensible values can be realised by only slightly deviating from the stipulated size constraint.

The problem arising when dealing with the remaining admissible parameters, i.e. $\binom{n}{k} < h \leq \binom{n+\gamma(k-1)-1}{k}$, seems to be that the constraints on the number of vertices and cliques interact in an elusive fashion. Although understanding such interactions better remains a direction for future investigations, we are able to identify certain combinatorial conditions under which the constraints are somewhat independent. This yields fixed-parameter algorithms for variants of $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ contrasting Theorem 3. Here, Proviso 20 stipulates mild constraints on the graph classes \mathcal{F} and \mathcal{G} . For example, \mathcal{G} can be taken to be the class of all graphs and \mathcal{F} to be the class of all connected graphs.

► **Theorem 5.** *For graph classes \mathcal{F}, \mathcal{G} as in Proviso 20, the following problem is in FPT:*

<p>p-SINGLEHOMREC(\mathcal{F}, \mathcal{G})</p> <p>Input a graph $F \in \mathcal{F}$, an integer $h \in \mathbb{N}$ given in binary</p> <p>Parameter $V(F)$</p> <p>Question Does there exist a graph $G \in \mathcal{G}$ such that $\text{hom}(F, G) = h$?</p>

Curiously, any fpt-algorithm for p -SINGLEHOMREC(\mathcal{F}, \mathcal{G}) has to be arithmetic: In FPT, one can neither construct the graph G nor count homomorphisms from F to G [12]. Indeed, our algorithm essentially only operates with integers and exploits number-theoretic properties of the set of reconstructible numbers. In Theorem 21, we apply similar ideas to derive an fpt-algorithm for a version of $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ with multiple equi-sized subgraph constraints.

2 Preliminaries and Conventions

Write $\mathbb{N} = \{0, 1, 2, \dots\}$ for the set of natural numbers. $A \leq_p B$ denotes that a decision problem A is polynomial-time many-one reducible to the decision problem B . A *graph* is a pair $G = (V, E)$ of a set of *vertices* V and a set of *edges* $E \subseteq \binom{V}{2}$. We usually write $V(G)$ and $E(G)$ for V and E , respectively, and use n to denote the *order* $n := |V(G)|$ of G . For ease of notation, we denote an edge $\{u, v\}$ by uv or vu . A *homomorphism* from a graph F to a graph G is a mapping $h: V(F) \rightarrow V(G)$ such that $h(uv) \in E(G)$ for every $uv \in E(F)$. A (\mathcal{C} -*vertex-*)*coloured graph* is a triple $G = (V, E, c)$ where (V, E) is a graph, the *underlying graph*, and $c: V(G) \rightarrow \mathcal{C}$ a function assigning a *colour* from a set \mathcal{C} to every vertex of G . An (\mathcal{L} -)*labelled graph* is defined analogously with a function $\ell: \mathcal{L} \rightarrow V(G)$ assigning a vertex of G to every *label* from a set of labels \mathcal{L} instead. Homomorphisms between coloured graphs and between labelled graphs are then defined as homomorphisms of the underlying graphs that respect colours and labels, respectively.

A graph G' is a *subgraph* of a graph G , written $G' \subseteq G$, if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The *subgraph induced by a set* $U \subseteq V(G)$, written $G[U]$, is the subgraph of G with vertices U and edges $E(G) \cap \binom{U}{2}$. We write $\text{hom}(F, G)$ for the number of homomorphisms from F to G , $\text{sub}(F, G)$ for the number of subgraphs $G' \subseteq G$ such that $G' \cong F$, and $\text{indsub}(F, G)$ for the number of subsets $U \subseteq V(G)$ such that $G[U] \cong F$. This notation generalises to coloured and labelled graphs in the straightforward way.

The definition of $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ from the introduction directly generalises to classes \mathcal{F} and \mathcal{G} of relational structures over the same signature, and in particular, classes of labelled and coloured graphs. We call a pair (F, h) of a structure F and a number $h \in \mathbb{N}$ a *constraint*

and also denote the pair by $\text{hom}(F) = h$, or for example in the context of reconstructibility of subgraph counts, by $\text{sub}(F) = h$. As stipulated in the introduction, if \mathcal{F} is a class of graphs, we abbreviate $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ for the class of all graphs \mathcal{G} to $\text{HOMREC}(\mathcal{F})$. We use the abbreviation $\text{BHOMREC}(\mathcal{F})$ in the same way, and for a class of labelled or coloured graphs \mathcal{F} , we analogously abbreviate the problem name if \mathcal{G} is the class of all labelled or all coloured graphs, respectively. We define the problems $\text{SUBREC}(\mathcal{F}, \mathcal{G})$ and $\text{BSUBREC}(\mathcal{F}, \mathcal{G})$ analogously to $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ and $\text{BHOMREC}(\mathcal{F}, \mathcal{G})$, respectively, with subgraph counts instead of homomorphism counts and also follow the conventions agreed upon above. See the full version [4] for details.

3 Decidability

The problem $\text{BHOMREC}(\mathcal{F}, \mathcal{G})$ is trivially decidable if membership in \mathcal{G} is decidable since it is possible to perform a brute-force search for G by testing all graphs up to order n . For $\text{HOMREC}(\mathcal{F}, \mathcal{G})$, decidability is implied by the following lemma:

► **Lemma 6.** *Let \mathcal{F} and \mathcal{G} be classes of structures over the same signature. Suppose that \mathcal{G} is closed under taking induced substructures. Let $(F_1, h_1), \dots, (F_m, h_m) \in \mathcal{F} \times \mathbb{N}$. Let $G \in \mathcal{G}$ be such that $\text{hom}(F_i, G) = h_i$ for all $i \in [m]$. Then there exists $H \in \mathcal{G}$ such that $|H| \leq \sum_{i=1}^m h_i |F_i|$ and $\text{hom}(F_i, H) = h_i$ for all $i \in [m]$.*

Proof. Let U denote the union over all images of homomorphisms $F_i \rightarrow G$, $i \in [m]$. Clearly, $|U| \leq \sum_{i=1}^m \text{hom}(F_i, G) |F_i|$. Let $H := G[U] \in \mathcal{G}$. For all $i \in [m]$, $\text{hom}(F_i, H) = \text{hom}(F_i, G)$. Indeed, every homomorphism $F_i \rightarrow H$ gives rise to a homomorphism $F_i \rightarrow G$ by composition with the embedding $H \hookrightarrow G$. Conversely, observe that every homomorphism $F_i \rightarrow G$ is in fact a homomorphism $F_i \rightarrow H$ since its image is contained in U . It remains to observe that this correspondence establishes a bijection. ◀

Under the assumptions of the previous lemma, $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ can be decided via a brute-force search on all graphs in \mathcal{G} up to order $\sum_{i=1}^m h_i |F_i|$. Together with our insights on $\text{BHOMREC}(\mathcal{F}, \mathcal{G})$, this shows the following theorem.

► **Theorem 7.** *Let \mathcal{F} and \mathcal{G} be classes of structures over the same signature. Suppose that membership in \mathcal{G} is decidable. Then $\text{BHOMREC}(\mathcal{F}, \mathcal{G})$ is decidable. If \mathcal{G} is closed under taking induced substructures, then also $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ is decidable.*

4 Hardness

In this section, we prove the hardness results presented in the introduction. First, we remark that, for the class \mathcal{F} of all graphs, $\text{HOMREC}(\mathcal{F})$ is in NEXP since we can non-deterministically guess a graph G of exponential size by Lemma 6 and then count homomorphisms to G in exponential time by simply going through all mappings from the given constraint graphs to G . For the bounded problem, we are given a size bound on G as part of the input, which means that $\text{BHOMREC}(\mathcal{F}, \mathcal{F})$ is in $\text{NP}^{\text{C=P}}$ since we can non-deterministically guess a graph G of linear size and then verify that G satisfies all constraints by using the C=P -oracle.

► **Theorem 8.** *Let \mathcal{F} denote the class of all graphs. Then, $\text{HOMREC}(\mathcal{F})$ is in NEXP and $\text{BHOMREC}(\mathcal{F})$ is in $\text{NP}^{\text{C=P}}$.*

Let \mathcal{F} denote the class of all graphs. The fact that $\text{BHOMREC}(\mathcal{F}) \in \text{NP}^{\text{C=P}}$ reflects the intuition on the hardness on $\text{BHOMREC}(\mathcal{F})$ that we gave in the introduction, i.e. that there are two intertwined sources of hardness, the *reconstruction hardness*, manifested as NP,

and the *counting hardness*, manifested as the $C=P$ -oracle. In Section 4.1, we show that this intuition is in fact correct and that $\text{HOMREC}(\mathcal{F})$ and $\text{BHOMREC}(\mathcal{F})$ are $\text{NP}^{C=P}$ -hard. In Section 4.2, we further reinforce this intuition by presenting a reduction from the well-known NP-complete problem SETSPLITTING to $\text{HOMREC}(\mathcal{F})$ and $\text{BHOMREC}(\mathcal{F})$ for a class of bounded treewidth \mathcal{F} that proves that these problems are NP-hard for a family of inputs where the number of constraints, all homomorphism numbers, and all graphs but one are fixed. This isolates the reconstruction hardness and supports our intuition of the reconstruction hardness being NP-hardness.

In our reduction from SETSPLITTING , the given instance to SETSPLITTING is encoded as a constraint graph that grows with the size of the given instance while the number of produced constraints and the produced homomorphism numbers remain fixed. This raises the question if we can achieve tractability by restricting the order of the constraint graphs instead, i.e. if $\text{HOMREC}(\mathcal{F})$ and $\text{BHOMREC}(\mathcal{F})$ become tractable if \mathcal{F} is *finite*. In Section 4.3, we show that there is a finite class \mathcal{F} for which $\text{HOMREC}(\mathcal{F})$ is NP-hard by reducing from the NP-complete problem QPOLY of solving an equation involving a quadratic polynomial. We further show that this reduction cannot be adapted to work for $\text{BHOMREC}(\mathcal{F}, \mathcal{G})$ and then prove that $\text{BHOMREC}(\mathcal{F}, \mathcal{G})$ is sparse, i.e. it only has polynomial number of yes-instances, which means that it cannot be NP-hard under the assumption that $P \neq NP$. In Section 4.4, we briefly discuss which of our hardness results also hold for the subgraph reconstructability problem.

For the sake of presentability, we only provide the reductions to the reconstructability problem for labelled or coloured graphs in the main body of this paper. In the full version [4], we show that all these reductions be adapted to (unlabelled and uncoloured) graphs via gadget constructions that employ *Kneser graphs*, cf. the full version [4].

4.1 $\text{NP}^{C=P}$ -Hardness

We first show that both problems $\text{HOMREC}(\mathcal{F})$ and $\text{BHOMREC}(\mathcal{F})$ are $\text{NP}^{C=P}$ -hard. We reduce from the following $\text{NP}^{C=P}$ -complete variant of 3-COLOURING, cf. the full version [4].

EC-3-COLOURING

Input A graph G , a subset of vertices $S \subseteq V(G)$, and a $k \in \mathbb{N}$ given in binary.

Question Is there a homomorphism $c: G[S] \rightarrow \mathfrak{K}_3$ such that there are exactly k homomorphisms $\hat{c}: G \rightarrow \mathfrak{K}_3$ such that $\hat{c}|_S = c$?

The idea is that the number of homomorphisms from a graph F to the complete graph on three vertices \mathfrak{K}_3 is precisely the number of 3-colourings of F . Then, one can formulate constraints that can only be satisfied by \mathfrak{K}_3 , and by adding an additional constraint $\text{hom}(F) = h$, one obtains a yes-instance to the reconstructability problem if and only if the number of 3-colourings of F is exactly h . By additionally employing labels on F and \mathfrak{K}_3 , we obtain a reduction from EC-3-COLOURING.

► **Theorem 9.** *Let \mathcal{LF} denote the class of all labelled graphs. Then, $\text{EC-3-COLOURING} \leq_p \text{HOMREC}(\mathcal{LF})$ and $\text{EC-3-COLOURING} \leq_p \text{BHOMREC}(\mathcal{LF})$.*

Proof. Given an instance (F, S, k) of EC-3-COLOURING, let $m := |S|$. Fix an arbitrary linear order on S , i.e. $S = \{s_1, \dots, s_m\}$. We use the labels ℓ_1, \dots, ℓ_m to classify vertices of F as members of S : let F' be the labelled graph obtained from F and S by assigning label ℓ_i to the vertex $s_i \in S$ for every $i \in [m]$. The reduction then produces the following constraints, where for $\text{BHOMREC}(\mathcal{LF})$, we set the additional size constraint to three:

19:8 The Complexity of Homomorphism Reconstructibility

- (a) $\text{hom}(F') = k$,
- (b) $\text{hom}(\bullet) = 3$,
- (c) $\text{hom}(\bullet \rightarrow \bullet) = 6$, and
- (d) $\text{hom}(\bullet \ell_i) = 1$ for every $i \in [m]$.

Note that \mathfrak{A} is the unique graph that satisfies $\text{hom}(\bullet) = 3$ and $\text{hom}(\bullet \rightarrow \bullet) = 6$. The remaining constraints enforce that each label from $\{\ell_1, \dots, \ell_m\}$ appears exactly once in G . For a partition $\mathcal{L} = \{L_1, L_2, L_3\}$ of these labels into at most three parts, let $G_{\mathcal{L}}$ denote the \mathfrak{A} with its three vertices labelled by the labels in L_1 , L_2 , and L_3 , respectively. Then, (F, S, k) is in EC-3-COLOURING if and only if there is a partition \mathcal{L} as above such that $\text{hom}(F', G_{\mathcal{L}}) = k$, which again is the case if and only if there is a labelled graph G that satisfies the constraints produced by the reduction. \blacktriangleleft

By encoding vertex labels by gadgets consisting of Kneser graphs, we can also obtain a reduction for uncoloured graphs. Since the number of labels used in the reduction above depends on the input instance, we can view every label as indexed by a binary number and construct a gadget from distinct Kneser graphs for 0 and 1 to encode its index. This allows us to only use a constant and finite set of Kneser graphs, which means that we do not have to worry about their size, and guarantees that the resulting reduction still runs in polynomial time. The proof can be found in the full version [4].

► **Theorem 10.** *Let \mathcal{F} denote the class of all graphs. Then, $\text{EC-3-COLOURING} \leq_p \text{HOMREC}(\mathcal{F})$ and $\text{EC-3-COLOURING} \leq_p \text{BHOMREC}(\mathcal{F})$.*

Together with the following observation, this then finishes the proof of Theorem 1.

► **Corollary 11.** *Let \mathcal{F} denote the class of all graphs. Then, $\text{HOMREC}(\mathcal{F})$ and $\text{BHOMREC}(\mathcal{F})$ are not in PH unless PH collapses.*

Proof. This follows from Toda's Theorem [55] since every oracle query in the computation of a $\text{P}^{\#\text{P}}$ -machine can be simulated by a nondeterministic polynomial-time machine guessing the answer and then verifying it with an oracle query to a problem in $\text{C}=\text{P}$. \blacktriangleleft

4.2 NP-Hardness for Constraints of Bounded Treewidth

The reduction used to prove $\text{NP}^{\text{C}=\text{P}}$ -hardness in the previous section uses the graph given as input for EC-3-COLOURING as a constraint, which has the side effect that the treewidth of the produced instances is not bounded. Moreover, the same reduction cannot be easily adapted to prove NP-hardness of $\text{HOMREC}(\mathcal{F})$ for a class of graphs \mathcal{F} of bounded treewidth by simply considering input graphs of bounded treewidth: the NP-complete problem 3-COLOURING [19] restricted to graphs of bounded treewidth is polynomial-time solvable [2]. Hence, we need a different approach to such a reduction. We reduce from the following problem SETSPLITTING, which is well-known to be NP-complete [35].

SETSPLITTING
Input A collection \mathcal{C} of subsets of a finite set S .
Question Is there a partition of S into two subsets S_1 and S_2 such that no subset in \mathcal{C} is entirely contained in either S_1 or S_2 ?

The idea is that we represent every element i of S by a vertex of colour i . A set $T \in \mathcal{C}$ is encoded by a star that has a leaf for every element of T and a root vertex of some fixed colour that is distinct from the colours used for elements of S . Intuitively, the graph G

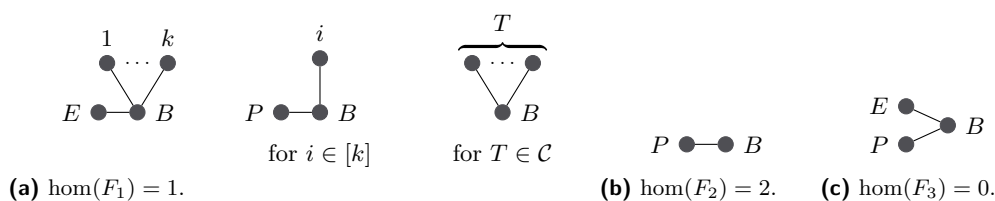


Figure 2 The three constraints produced by the reduction of Theorem 12.

then consists of two stars that encode the two sets S_1 and S_2 . To ensure that no subset in \mathcal{C} is entirely contained in S_1 or S_2 , we use the constraints to require that there are no homomorphisms from our constraint graphs to G .

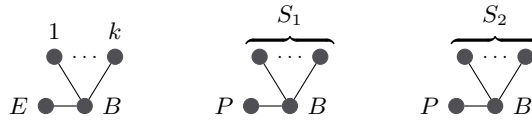
The idea sketched above produces a single constraint for every set $T \in \mathcal{C}$. We can use the following trick to combine these constraints into a single one: a graph F consisting of several connected components has exactly one homomorphism to G if and only if all its connected components have exactly one homomorphism to G . Hence, if we choose G to consist of two stars that encode S_1 and S_2 and also add a star that has all elements of S as its leaves, we can instead require to have exactly one homomorphism from all our constraint graphs to G and, thus, combine all these constraint graphs into a single (disconnected) graph from which we require to have exactly one homomorphism.

► **Theorem 12.** *Let \mathcal{CS} denote the class of all disjoint unions of coloured stars. Then, $\text{SETSPLITTING} \leq_p \text{HOMREC}(\mathcal{CS})$ and $\text{SETSPLITTING} \leq_p \text{BHOMREC}(\mathcal{CS})$, where the reduction only produces three constraints $\text{hom}(F_1) = 1$, $\text{hom}(F_2) = 2$, and $\text{hom}(F_3) = 0$.*

Proof. Given a collection \mathcal{C} of subsets of a finite set S , we may assume that $S = [k]$ by re-labelling the elements of S . In the construction of the graphs F_1, F_2, F_3 , we use the colours $1, \dots, k$ and also B (“black”), E (“everything”), and P (“partition”). We construct these graphs as shown in Figure 2 and add the constraints $\text{hom}(F_1) = 1$, $\text{hom}(F_2) = 2$, $\text{hom}(F_3) = 0$. Note that the constraint $\text{hom}(F_1) = 1$ is equivalent to $\text{hom}(F) = 1$ for every connected component F of F_1 . For $\text{BHOMREC}(\mathcal{CS})$, we set the size bound to $2k + 6$.

Given a partition of S into sets S_1 and S_2 such that no subset in \mathcal{C} is entirely contained in either S_1 or S_2 , the graph G_{S_1, S_2} in Figure 3 satisfies all constraints. Conversely, let G be a coloured graph that satisfies all constraints. By the constraint $\text{hom}(F_2) = 2$, the graph F_2 occurs exactly twice as a subgraph in G ; call these occurrences G_1 and G_2 . Let $S_1 \subseteq [k]$ be the set of all $i \in [k]$ such that a vertex of colour i is connected to the B -vertex of G_1 . If the B -vertex in $G_2 \subseteq G$ is distinct from the B -vertex in $G_1 \subseteq G$, then let $S_2 \subseteq [k]$ be the set of all $i \in [k]$ such that a vertex of colour i is connected to the B -vertex of G_2 ; otherwise, let $S_2 := \emptyset$. The first constraint yields that the i - B - P -graph occurs exactly once as a subgraph of G for every $i \in [k]$. By the second constraint, every i - B - P graph has to be a supergraph of one of the two occurrences G_1 and G_2 of F_2 . Hence, the sets S_1 and S_2 cover S . Moreover, as every i - B - P graph occurs exactly once as a subgraph of G , the sets S_1 and S_2 have to be a partition of S . Finally, by the first constraint, the 1 - k - B - E -graph is a subgraph of G . Observe that, for every $T \in \mathcal{C}$, the set T cannot be a subset of either S_1 or S_2 as the T - B -graph occurs exactly once in G by the first constraint and it already is a subgraph of the 1 - k - B - E -graph whose B -vertex has to be distinct from the B -vertices of the occurrences of F_2 by the third constraint. ◀

19:10 The Complexity of Homomorphism Reconstructibility



■ **Figure 3** The graph G_{S_1, S_2} constructed from S_1 and S_2 in the proof of Theorem 12.

Note the following curiosity of Theorem 12: the numbers in the constraints produced by the reduction are constant, i.e. they do not depend on the specific problem instance given as an input. Hence, the hardness solely lies in the graphs and not the homomorphism numbers; more specifically, there is only a single constraint graph that depends on the input instance and is the cause of hardness.

We again use Kneser graphs to turn this reduction into one for uncoloured graphs. We view the colours as numbers and use gadgets of Kneser graphs that encode these numbers in binary. This allows us to argue that the reduction is still correct, where in particular, we have to argue that, if there is a graph G satisfying all constraints, then we can extract a solution to the given SETSPLITTING instance from it; this is not straightforward since such a graph G does not have to adhere to our encoding of coloured graphs. This then yields the following theorem, which implies Theorem 2. The proof can be found in the full version [4].

► **Theorem 13.** *There is a class of graphs \mathcal{F} of bounded treewidth such that $\text{SETSPLITTING} \leq_p \text{HOMREC}(\mathcal{F})$ and $\text{SETSPLITTING} \leq_p \text{BHOMREC}(\mathcal{F})$, where the number of constraints, the homomorphism numbers, and all constraint graphs but one are constant.*

4.3 NP-Hardness for a Finite Set of Graphs

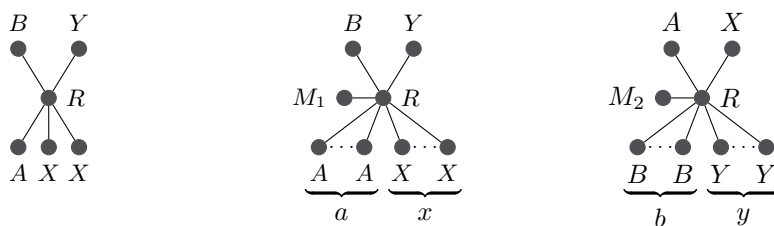
The previous sections shows that restricting the constraint graphs to be from a class \mathcal{F} of bounded treewidth and the number of constraints to a constant is not enough to achieve tractability: both $\text{HOMREC}(\mathcal{F})$ and $\text{BHOMREC}(\mathcal{F})$ remain NP-hard. What happens if we go even further and consider a finite class \mathcal{F} ? Then, the treewidth of \mathcal{F} is trivially bounded and so is the number of constraints. At first glance, hardness in this case seems unlikely since the reductions presented in the previous sections rely heavily on encoding the input instance as the constraint graphs and only used small constants for the homomorphism numbers. Now, the input essentially consists just of homomorphism numbers encoded in binary and, for $\text{BHOMREC}(\mathcal{F})$, also the size of the desired graph encoded in unary. This makes it all the more surprising that we can actually prove the NP-hardness of $\text{HOMREC}(\mathcal{F})$. We reduce from the following decision problem, which only takes three natural numbers in binary encoding as input and is NP-complete [41, Theorem 1]:

QPOLY

Input Natural numbers a , b , and c in binary encoding.

Question Are there natural numbers x and y such that $ax^2 + by = c$?

The idea of the reduction is simple: we encode the polynomial $ax^2 + by$ as a coloured star F from which we require exactly c homomorphisms. This star has a leaf of colour A and two leaves of colour X to encode the monomial ax^2 . Furthermore, it has a leaf of colour B and of colour Y to encode the monomial by . Then, the sum $ax^2 + by$ is realised by encoding x and y as two separate components of G – additional constraints are used to enforce that G has precisely two components, that the first component has exactly a leaves of colour A , and that the second component has exactly b leaves of colour B .



(a) The graph F_{poly} . (b) The graph $G_{a,b,x,y}$ constructed from $a, b, x, y \in \mathbb{N}$.

■ **Figure 4** The most important graphs used in the reduction of Theorem 14.

► **Theorem 14.** *There is a finite set \mathcal{F} of coloured stars such that $\text{QPOLY} \leq_p \text{HOMREC}(\mathcal{F})$.*

Proof. We use the colours R, A, X, B, Y, M_1 and M_2 . The main observation is that, for the graphs F_{poly} and $G_{a,b,x,y}$ from Figure 4, we have $\text{hom}(F_{\text{poly}}, G_{a,b,x,y}) = ax^2 + by$. Note that, however, the size of $G_{a,b,x,y}$ is not polynomial in $\log a + \log b + \log c$, i.e. in the size of an instance (a, b, c) of QPOLY. Given an instance (a, b, c) of QPOLY, we produce the following constraints and denote the set of all coloured graphs used in these constraints by \mathcal{F} ; note that \mathcal{F} is independent of the instance (a, b, c) :

- (a) $\text{hom}(A \bullet \bullet R) = a + 1$,
- (b) $\text{hom}(B \bullet \bullet R) = b + 1$,
- (c) $\text{hom}(F_{\text{poly}}) = c$,
- (d) $\text{hom}(\bullet R) = 2$,
- (e) $\text{hom}(M_1 \bullet \bullet R) = 1$,
- (f) $\text{hom}(M_2 \bullet \bullet R) = 1$,
- (g) $\text{hom}\left(\begin{smallmatrix} M_1 \\ M_2 \end{smallmatrix} \bullet \bullet R\right) = 0$,
- (h) $\text{hom}\left(\begin{smallmatrix} B & Y \\ M_1 & R \end{smallmatrix} \bullet \bullet R\right) = 1$,
- (i) $\text{hom}\left(\begin{smallmatrix} A & X \\ M_2 & R \end{smallmatrix} \bullet \bullet R\right) = 1$.

If (a, b, c) is an instance of QPOLY with $x, y \in \mathbb{N}$ such that $ax^2 + by = c$, then $G_{a,b,x,y}$ from Figure 4b satisfies all these constraints. Conversely, if there is a coloured graph G that satisfies all constraints, we know by (d)–(g) that there are two R -coloured vertices v_1 and v_2 such that v_1 is connected to an M_1 -coloured vertex but not an M_2 -coloured vertex and v_2 is connected to an M_2 -coloured vertex but not an M_1 -coloured vertex. By (h) and (i), v_1 has exactly one B -coloured neighbor and exactly one Y -coloured neighbor and v_2 has exactly one A -coloured neighbor and exactly one X -coloured neighbor. Hence, by (a) and (b), v_1 has exactly a neighbors of colour A and v_2 has exactly b neighbors of colour B . Let x be the number of X -coloured neighbors of v_1 and y be the number of Y -coloured neighbors of v_2 . Then, we have $c = \text{hom}(F_{\text{poly}}, G) = ax^2 + by$. ◀

We can again use Kneser graphs to obtain a reduction for uncoloured graphs, which implies the hardness stated in Theorem 3. The proof can be found in the full version [4].

► **Theorem 15.** *There is a finite set \mathcal{F} of graphs such that $\text{QPOLY} \leq_p \text{HOMREC}(\mathcal{F})$.*

Can such a reduction also be used to prove NP-hardness of $\text{BHOMREC}(\mathcal{F})$ for a finite class \mathcal{F} ? This is not possible, since for a fixed graph F , the number of homomorphisms from F to a graph G is polynomial in the order of G . Hence, with a finite set \mathcal{F} of graphs and a

19:12 The Complexity of Homomorphism Reconstructibility

graph G of order up to n , we can only realise polynomially many distinct homomorphism numbers, while the hardness of solving the equation $ax^2 + by = c$ stems from the fact that there is an exponential number of solution candidates. This implies that $\text{BHOMREC}(\mathcal{F})$ is sparse for every finite \mathcal{F} , which means that it cannot be NP-hard unless $\text{P} = \text{NP}$ [39].

► **Theorem 16.** *If $\text{BHOMREC}(\mathcal{F})$ is NP-hard for a finite set of graphs \mathcal{F} , then $\text{P} = \text{NP}$.*

4.4 Subgraph Counts

While subgraphs counts can be expressed as linear combinations of homomorphism numbers via injective homomorphism numbers, cf. [10], adapting our reductions to subgraph counts is not as straightforward. There is no obvious way of adapting the reduction of Theorem 9 since non-injectivity is crucial to encode colourability of graphs. The reduction of Theorem 12 can partially be salvaged by producing individual constraints instead of only three constraints. Then, all constraint graphs are *colourful*, which means that their subgraph counts are homomorphism counts, which can be computed efficiently. However, the gadget construction used in Theorem 13 then produces constraint graphs of unbounded vertex-cover number. Hence, this reduction is uninteresting since the determining factor for tractability of subgraphs counts is the vertex-cover number [11].

The results for finite \mathcal{F} transfer to subgraphs and are meaningful since subgraph counts can trivially be computed in polynomial time in this case. However, for subgraph counts, the reduction of Theorem 3 encodes the binomial equation $a\binom{x}{2} + by = c$ instead of $ax^2 + by = c$. Luckily, the problem BPOLY of solving this equation is still NP-complete, cf. the full version [4]. Moreover, $\text{BSUBREC}(\mathcal{F})$ is still sparse for every finite class of graphs \mathcal{F} .

► **Theorem 17.** *There is a finite set \mathcal{F} of graphs such that $\text{BPOLY} \leq_p \text{SUBREC}(\mathcal{F})$. If $\text{BSUBREC}(\mathcal{F})$ is NP-hard for a finite set of graphs \mathcal{F} , then $\text{P} = \text{NP}$.*

5 Towards Tractability: Reconstructing Clique Counts

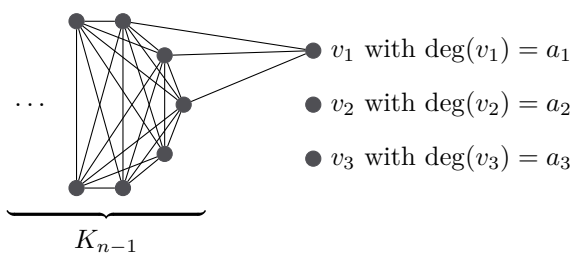
In this section, we show that $\text{BHOMREC}(\mathcal{F})$ is tractable when the only constraint graph is a clique and the constraints are in a certain range. The proofs rely on number-theoretic insights and tailored combinatorial constructions. Using a number-theoretic result by Kamke [29], we show that for almost all sensible $n, h, k \in \mathbb{N}$ there exists a graph G on slightly more than n vertices containing h copies of the k -vertex clique K_k as a subgraph.

► **Theorem 18** (Kamke [29], cf. [42, Theorem 11.10]). *There exists a function $\gamma: \mathbb{N} \rightarrow \mathbb{N}$ such that for every $k \geq 1$ and $n \geq 1$ there exist $a_1, \dots, a_{\gamma(k)} \in \mathbb{N}$ such that $n = \sum_{i=1}^{\gamma(k)} \binom{a_i}{k}$.*

Nečaev [43] showed that $\gamma(k)$ can be chosen to be of order at most $O(k \log k)$ and gave a similar lower bound in [44]. Specific values of γ include $\gamma(1) = 1$, Gauß' Eureka Theorem, cf. [47], stating $\gamma(2) = 3$ and the unproven Tetrahedral Numbers Conjecture of Pollock [48] asserting $\gamma(3) = 5$.

► **Theorem 4.** *There exists a function $\gamma: \mathbb{N} \rightarrow \mathbb{N}$ such that for every $k \geq 2$, $n \geq 1$, $h \leq \binom{n}{k}$, there exists a graph G on $n + \gamma(k - 1) - 1$ vertices such that $\text{sub}(K_k, G) = h$.*

Proof. Let γ denote the function from Theorem 18. For every fixed k , the proof is by induction on n . For $n \leq k$, the claim is trivial. Suppose subsequently that $n > k$. Inductively, we may suppose that there exists a graph G on $n - 1 + \gamma(k - 1) - 1$ vertices with $h \leq \binom{n-1}{k}$ copies of K_k . One may add an isolated vertex to obtain a graph on $n + \gamma(k - 1) - 1$ vertices with h copies of K_k , as desired.



■ **Figure 5** Example for $k = 3$, building a graph G with $n - 1 + \gamma(k - 1)$ vertices and $\text{sub}(\triangle, G) = h > \binom{n-1}{3}$. Since $\gamma(2) = 3$, there are a_1, a_2, a_3 such that $h - \binom{n-1}{3} = \binom{a_1}{2} + \binom{a_2}{2} + \binom{a_3}{2}$. Connecting a fresh vertex v_i with a_i vertices from the K_{n-1} , adds $\binom{a_i}{2}$ subgraphs \triangle to G .

Thus, it remains to construct a graph with $\binom{n-1}{k} < h \leq \binom{n}{k}$ copies of K_k and $n + \gamma(k - 1) - 1$ vertices. Write $h' := h - \binom{n-1}{k} \leq \binom{n-1}{k-1}$. By Theorem 18, there exist non-negative integers $a_1, \dots, a_{\gamma(k-1)}$ such that $h' = \sum_{i=1}^{\gamma(k-1)} \binom{a_i}{k-1}$. It can be easily seen that $\binom{h}{k} > \binom{n}{k}$ for all integers $h > n \geq k \geq 1$. Hence, $\binom{a_i}{k-1} \leq h' \leq \binom{n-1}{k-1}$ implies that $a_i \leq n - 1$ for all $1 \leq i \leq \gamma(k - 1)$.

Define the graph G by taking the disjoint union of a clique K_{n-1} and fresh vertices $v_1, \dots, v_{\gamma(k-1)}$. For $1 \leq i \leq \gamma(k - 1)$, the vertex v_i is connected to an arbitrary selection of a_i many vertices of the clique. Note that this adds $\binom{a_i}{k-1}$ copies of K_k to the graph. The resulting graph on $n - 1 + \gamma(k - 1)$ vertices satisfies $\text{sub}(K_k, G) = \binom{n-1}{k} + \sum_{i=1}^{\gamma(k-1)} \binom{a_i}{k-1} = h$. For an example with $k = 3$, see Figure 5. ◀

For the special case of $k = 3$, i.e. \triangle , we can do slightly better than in Theorem 4. While Theorem 4 requires $\gamma(2) - 1 = 2$ extra vertices to realise any sensible h , we show in Theorem 19 that for large n one additional vertex suffices. While Theorem 4 builds on an $(n - 1)$ -vertex clique to which new vertices and edges are added, for Theorem 19 we start with an $(n + 1)$ -vertex clique and remove edges to realise the precise subgraph count. The proof of Theorem 19 can be found in the full version [4].

► **Theorem 19.** *For every $n \geq 130$ and $h \leq \binom{n}{3}$, there is a graph G on $n + 1$ vertices such that $\text{sub}(\triangle, G) = h$.*

6 Parametrised Complexity

For graph classes \mathcal{F} and \mathcal{G} , we consider the parametrised version $p\text{-HOMREC}(\mathcal{F}, \mathcal{G})$ of the homomorphism reconstructability problem. For an instance $(F_1, h_1), \dots, (F_m, h_m)$, the parameter is $k := \sum_{i=1}^m |V(F_i)|$. We aim for an fpt-algorithm, i.e. an algorithm that runs in $f(k) \text{polylog}(h_1, \dots, h_m)$ for some computable function f . By Theorem 15, $p\text{-HOMREC}(\mathcal{F}, \mathcal{G})$ is para-NP-hard and thus we cannot expect to obtain an fpt-algorithm unless $\text{P} = \text{NP}$, cf. [17, Corollary 2.13], which means that we have to restrict the problem in some way. Surprisingly, it turns out that a certain restriction of $p\text{-HOMREC}(\mathcal{F}, \mathcal{G})$ and of the analogous $p\text{-SUBREC}(\mathcal{F}, \mathcal{G})$ are in FPT. Curiously, in FPT, one cannot even count homomorphisms or subgraphs from arbitrary graphs [12, 11]. Our algorithm has to make do with the integers from the input and cannot construct the graph G explicitly.

Given constraint graphs $\mathcal{I} \subseteq \mathcal{F}$, the overall strategy is to compute in time only depending on \mathcal{I} a data structure representing the (infinite) set of all reconstructable vectors of homomorphism counts

$$R(\mathcal{I}) := \{ \text{hom}(\mathcal{I}, G) \in \mathbb{N}^{\mathcal{I}} \mid G \in \mathcal{G} \} \tag{1}$$

19:14 The Complexity of Homomorphism Reconstructibility

or analogously of subgraph counts. This data structure is required to admit a polynomial-time procedure for testing whether a given vector $h \in \mathbb{N}^{\mathcal{I}}$ is in this set. We identify various combinatorial conditions sufficient for guaranteeing the feasibility of this approach. To start with, we impose the following conditions on the graph classes \mathcal{F} and \mathcal{G} :

► **Proviso 20.**

- (i) membership in \mathcal{G} is decidable,
- (ii) \mathcal{G} is closed under taking induced subgraphs,
- (iii) \mathcal{G} is closed under disjoint unions,
- (iv) all $F \in \mathcal{F}$ are connected.

Items (iii) and (iv) imply that the set $R(\mathcal{I})$ of all reconstructable vectors is closed under addition. Indeed, for all connected graphs F and graphs G and H it holds that $\text{hom}(F, G + H) = \text{hom}(F, G) + \text{hom}(F, H)$ and $\text{sub}(F, G + H) = \text{sub}(F, G) + \text{sub}(F, H)$. Thus, we can use the vectors realised by small graphs, i.e. those which can be inspected in FPT time, to construct vectors realised by bigger graphs. More formally, writing

$$S(\mathcal{I}) := \left\{ \text{hom}(\mathcal{I}, G) \in \mathbb{N}^{\mathcal{I}} \mid G \in \mathcal{G} \text{ with } |V(G)| \leq \max_{I \in \mathcal{I}} |V(I)| \right\}, \quad (2)$$

it holds that the set of all finite linear combinations of elements in $S(\mathcal{I})$ with coefficients from \mathbb{N} is contained in $R(\mathcal{I})$, i.e. $\text{NS}(\mathcal{I}) \subseteq R(\mathcal{I})$. The challenge is to ensure that all reconstructable vectors can be constructed in this way.

We require item (ii) to relate vectors realised by large graphs to those realised by small graphs, cf. Lemmas 22 and 24. However, this assumption is not sufficient to yield an fpt-algorithm. To that end, we make further assumptions which ensure that the set of realised vectors is in a sense linear and thus admits the aforementioned desired data structure. Combinatorially, these assumptions mean that the various constraints may not interact non-trivially.

6.1 Equi-Size Subgraph Constraints

The restriction we impose on $p\text{-SUBREC}(\mathcal{F}, \mathcal{G})$ to put it in FPT is that all constraint graphs are of the same size:

► **Theorem 21.** *For graph classes \mathcal{F}, \mathcal{G} as in Proviso 20, the following problem is in FPT:*

$p\text{-EQUIIZESUBREC}(\mathcal{F}, \mathcal{G})$
Input Pairs $(F_1, h_1), \dots, (F_m, h_m) \in \mathcal{F} \times \mathbb{N}$ with $|V(F_1)| = \dots = |V(F_m)| =: k$
Parameter km
Question Is there a $G \in \mathcal{G}$ such that $\text{sub}(F_i, G) = h_i$ for every $i \in [m]$?

Given an instance $\mathcal{I} \subseteq \mathcal{F}$, define $R(\mathcal{I})$ and $S(\mathcal{I})$ as in Equations (1) and (2) but with sub instead of hom . As argued in the previous section, we have $\text{NS}(\mathcal{I}) \subseteq R(\mathcal{I})$. In the context of $p\text{-EQUIIZESUBREC}(\mathcal{F}, \mathcal{G})$, the following Lemma 22 yields that $\text{NS}(\mathcal{I}) = R(\mathcal{I})$.

► **Lemma 22.** *Let F and G be graphs. Then*

$$\text{sub}(F, G) = \sum_{H \text{ s.t. } |V(H)|=|V(F)|} \text{sub}(F, H) \text{indsub}(H, G)$$

where the sum ranges over all isomorphism types of graphs H on $|V(F)|$ vertices.

Indeed, by Lemma 22, every vector $\text{sub}(\mathcal{I}, G) \in R(\mathcal{I})$ is an \mathbb{N} -linear combination of the vectors $\text{sub}(\mathcal{I}, H)$ where H has exactly k vertices. It is crucial that all graphs in \mathcal{I} have the same number of vertices since otherwise any statement akin to Lemma 22 would involve negative coefficients stemming from Inclusion–Exclusion. In virtue of Lemma 22, testing membership in $R(\mathcal{I})$ reduces to solving a system of linear equations over \mathbb{N} .

► **Example 23.** Let $p, c \geq 0$ be integers. There exists a graph G with $\text{sub}(\mathcal{L}, G) = p$ and $\text{sub}(\mathcal{A}, G) = c$ if and only if $p \geq 3c$.

Proof. By Lemma 22, there exists a graph G with $\text{sub}(\mathcal{L}, G) = p$ and $\text{sub}(\mathcal{A}, G) = c$ if and only if the system $\begin{pmatrix} p \\ c \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ has solutions $x, y \in \mathbb{N}$, i.e. $\text{sub}(\mathcal{L}, y\mathcal{A} + x\mathcal{L}) = x + 3y$ and $\text{sub}(\mathcal{A}, y\mathcal{A} + x\mathcal{L}) = y$. The columns of this matrix correspond to the two graphs on three vertices which have a subgraph \mathcal{L} or \mathcal{A} , namely \mathcal{L} and \mathcal{A} . Solving this system yields that $y = c$ and $x = p - 3c$, as desired. ◀

The matrix constructed in Example 23 via Lemma 22 can clearly be computed in FPT. It remains to solve its system of linear equations, which is also possible in FPT [13].

Proof of Theorem 21. Let $\mathcal{I} \subseteq \mathcal{F}$ denote the instance. As observed above, $\text{NS}(\mathcal{I}) = R(\mathcal{I})$. Write \mathcal{H} for the set of all isomorphism types of graphs on k vertices. Write $A \in \mathbb{N}^{\mathcal{I} \times \mathcal{H}}$ for the matrix with entries $\text{sub}(F, H)$ for $(F, H) \in \mathcal{I} \times \mathcal{H}$. This matrix can be computed in FPT. Then $b = (h_1, \dots, h_m) \in R(\mathcal{I})$ if and only if $Ax = b$ has a solution over the non-negative integers. Testing the latter condition can be done in FPT by [13]. ◀

6.2 A Single Homomorphism Constraint

For p -HOMREC(\mathcal{F}, \mathcal{G}), the restriction to ensure fixed parameter-tractability is that there is only one (connected) constraint graph.

► **Theorem 5.** For graph classes \mathcal{F}, \mathcal{G} as in Proviso 20, the following problem is in FPT:

p -SINGLEHOMREC(\mathcal{F}, \mathcal{G})
Input a graph $F \in \mathcal{F}$, an integer $h \in \mathbb{N}$ given in binary
Parameter $|V(F)|$
Question Does there exist a graph $G \in \mathcal{G}$ such that $\text{hom}(F, G) = h$?

Define $R(F)$ and $S(F)$ as in Equations (1) and (2) replacing \mathcal{I} by the singleton set $\{F\}$. As before, $\text{NS}(F) \subseteq R(F)$. Dealing with p -SINGLEHOMREC(\mathcal{F}, \mathcal{G}) is more complicated than tackling p -EQUIIZESUBREC(\mathcal{F}, \mathcal{G}) in the sense that we will not be able to prove that $\text{NS}(F) = R(F)$. In fact, these sets only coincide for large enough numbers. The key combinatorial identity is the following, whose proof is deferred to the full version [4]:

► **Lemma 24.** Let F be a graph on k vertices. Then for all graphs G on more than k vertices,

$$\text{hom}(F, G) = \sum_{H \text{ s.t. } |V(H)| \leq k} \text{hom}(F, H) \text{indsub}(H, G) (-1)^{k-|V(H)|} \binom{|V(G)| - |V(H)| - 1}{k - |V(H)|},$$

where the sum ranges over all isomorphism types of graphs H on at most k vertices.

Lemma 24 yields that $R(F) \subseteq \mathbb{Z}S(F)$, i.e. every realised number is a linear coefficient of numbers in $S(F)$ with (not necessarily non-negative) integer coefficients. What allows us to obtain Theorem 5 is the purely number-theoretic observation that $\text{NS}(F)$ and $\mathbb{Z}S(F)$ coincide on sufficiently large numbers. Lemma 25 is based on Bézout’s identity and proven in the full version [4].

19:16 The Complexity of Homomorphism Reconstructibility

► **Lemma 25.** *Given $y_1, \dots, y_n \in \mathbb{N}$, one can compute integers y and N such that*

$$X \cap \{N, N + 1, \dots\} = y\mathbb{N} \cap \{N, N + 1, \dots\}$$

where $X := \mathbb{N}\{y_1, \dots, y_n\}$.

► **Example 26.** Let $y_1 = 6$ and $y_2 = 16$. Their greatest common divisor is 2. The set of their \mathbb{N} -linear combinations is $X = \mathbb{N}\{6, 16\} = 2\mathbb{N} \setminus \{2, 4, 8, 10, 14, 20\}$, i.e. the set of all even numbers except 2, 4, 8, 10, 14, 20.

This concludes the preparations for the proof of Theorem 5:

Proof of Theorem 5. The algorithm operates as follows: Compute $S(F)$ in time only depending on $|V(F)|$. Let y denote the greatest common divisor of the numbers in $S(F)$. By Lemmas 24 and 25, there exists a number N only depending on $|V(F)|$ such that for every $h \geq N$ there exists a graph G with $\text{hom}(F, G) = h$ if and only if h is a multiple of y . This settles the question for all $h \geq N$. It remains to consider the case $h < N$. By Lemma 6, it suffices to consider graphs G of size bounded in $|V(F)|$ to conclude. ◀

To illustrate our algorithm, we include an example:

► **Example 27.** Consider the constraint graph $F = \mathfrak{N}$. Enumerating all graphs on at most 4 vertices yields that $S(\mathfrak{N}) = \{0, 6, 16, 48\}$. For example, $\text{hom}(\mathfrak{N}, \mathfrak{A}) = 6$, $\text{hom}(\mathfrak{N}, \mathfrak{B}) = 16$, and $\text{hom}(\mathfrak{N}, \mathfrak{C}) = 48$. Hence, $R(\mathfrak{N})$ is a subset of the set in Example 26. It remains to check the finitely many exceptions 2, 4, 8, 10, 14, 20. By Lemma 6, this can be done by inspecting graphs on at most $20 \cdot 4 = 80$ vertices.

7 Conclusion

This paper provides the first systematic study of the homomorphism reconstructibility problem. Our results show that this deceptively simple-to-state problem generally is hard – not only in terms of its computational complexity but also in terms of finding efficient algorithms for the simplest of cases, being subject to intricate phenomena from combinatorics and number theory. The following questions remain open and warrant further investigation:

- Is $\text{BHOMREC}(\mathcal{F})$ $\text{NP}^{\#\text{P}}$ -complete for every class of unbounded treewidth \mathcal{F} , analogous to the $\#\text{W}[1]$ -completeness of $\#\text{HOM}(\mathcal{F})$ [12]? Is $\text{HOMREC}(\mathcal{F})$ NEXP -complete for the class of all graphs \mathcal{F} ? Is there a graph class \mathcal{F} for which $\text{HOMREC}(\mathcal{F})$ is not only NP-hard but actually NP-complete?
- While p - $\text{SINGLEHOMREC}(\mathcal{F}, \mathcal{G})$ is fpt, Theorem 3 implies that there exists a constant C such that p - $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ restricted to instances with $\leq C$ constraints is para-NP-hard. What is the minimal such C ? Is p - $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ with two connected constraints fpt? Is there a sharp threshold from fixed-parameter tractability to para-NP-hardness?
- The proof of Theorem 12 suggests that in some cases the number of constraints can be decreased by increasing the number of connected components of the constraint graphs. Notably, a crucial ingredient in Proviso 20 is that the constraint graphs are connected. How do the parameters number of constraints and number of connected components affect the complexity of $\text{HOMREC}(\mathcal{F}, \mathcal{G})$? What does the complexity hierarchy under these two parameters look like?
- In [18, 37], the functions $f: \mathcal{G} \rightarrow \mathbb{N}$ which are of the form $f = \text{hom}(-, G)$ for some graph G were characterised. Here, \mathcal{G} denotes the class of all graphs. For which finite graph classes \mathcal{I} does a characterisation of functions $f: \mathcal{I} \rightarrow \mathbb{N}$ of this form exist? Our Theorem 3 implies that in some cases deciding whether a given f is of this form is NP-hard.

- Are there non-trivial examples of combinations of constraint graphs for which reconstructability is tractable? Is there an effective description of the yellow area in Figure 1?
- Is $\text{HOMREC}(\mathcal{F}, \mathcal{G})$ self-reducible [52]? That is, can we efficiently construct a graph G that realises the given constraints if we have access to an oracle for $\text{HOMREC}(\mathcal{F}, \mathcal{G})$?
- What is the computational complexity of deciding whether homomorphism constraints are *approximately* reconstructable?
- More generally, one could study the complexity of questions about the set of graphs satisfying a given set of homomorphism constraints – such as computing its cardinality.
- How can one sample graphs satisfying homomorphism constraints uniformly at random?

References

- 1 Samson Abramsky, Tomáš Jakl, and Thomas Paine. Discrete Density Comonads and Graph Parameters. In Helle Hvid Hansen and Fabio Zanasi, editors, *Coalgebraic Methods in Computer Science*, pages 23–44, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-10736-8_2.
- 2 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, April 1989. doi:10.1016/0166-218X(89)90031-0.
- 3 Albert Atserias, Phokion G. Kolaitis, and Wei-Lin Wu. On the Expressive Power of Homomorphism Counts. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470543.
- 4 Jan Böker, Louis Härtel, Nina Runde, Tim Seppelt, and Christoph Standke. The complexity of homomorphism reconstructibility. *CoRR*, abs/2310.09009, 2023. doi:10.48550/ARXIV.2310.09009.
- 5 Angela Bonifati, Irena Holubová, Arnau Prat-Pérez, and Sherif Sakr. Graph Generators: State of the Art and Open Challenges. *ACM Computing Surveys*, 53(2):1–30, March 2021. doi:10.1145/3379445.
- 6 Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668, 2023. doi:10.1109/TPAMI.2022.3154319.
- 7 Surajit Chaudhuri and Moshe Y. Vardi. Optimization of Real Conjunctive Queries. In Catriel Beeri, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC, USA*, pages 59–70. ACM Press, 1993. doi:10.1145/153850.153856.
- 8 Yijia Chen, Jörg Flum, Mingjun Liu, and Zhiyang Xun. On Algorithms Based on Finitely Many Homomorphism Counts. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2022.32.
- 9 Radu Curticapean. Parity Separation: A Scientifically Proven Method for Permanent Weight Loss. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2016.47.

- 10 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms Are a Good Basis for Counting Small Subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 210–223. Association for Computing Machinery, 2017. event-place: Montreal, Canada. doi:10.1145/3055399.3055502.
- 11 Radu Curticapean and Dániel Marx. Complexity of Counting Subgraphs: Only the Boundedness of the Vertex-Cover Number Counts. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS '14, pages 130–139, USA, October 2014. IEEE Computer Society. doi:10.1109/FOCS.2014.22.
- 12 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1):315–323, December 2004. doi:10.1016/j.tcs.2004.08.008.
- 13 Peter Damaschke. Sparse solutions of sparse linear systems: Fixed-parameter tractability and an application of complex group testing. *Theoretical Computer Science*, 511:137–146, 2013. Exact and Parameterized Computation. doi:10.1016/j.tcs.2012.07.001.
- 14 Anuj Dawar, Tomáš Jakl, and Luca Reggio. Lovász-Type Theorems and Game Comonads. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470609.
- 15 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász Meets Weisfeiler and Leman. *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, pages 40:1–40:14, 2018. doi:10.4230/LIPICs.ICALP.2018.40.
- 16 Zdeněk Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, August 2010. doi:10.1002/jgt.20461.
- 17 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/3-540-29953-X.
- 18 Michael Freedman, László Lovász, and Alexander Schrijver. Reflection positivity, rank connectivity, and homomorphism of graphs. *Journal of the American Mathematical Society*, 20:37–51, 2007. doi:10.1090/S0894-0347-06-00529-7.
- 19 M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, February 1976. doi:10.1016/0304-3975(76)90059-1.
- 20 Roman Glebov, Andrzej Grzesik, Ping Hu, Tamás Hubai, Daniel Král', and Jan Volec. Densities of 3-vertex graphs, April 2017. URL: <http://arxiv.org/abs/1610.02446>.
- 21 Martin Grohe. Counting Bounded Tree Depth Homomorphisms. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 507–520, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394739.
- 22 Martin Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 1–16. ACM, 2020. doi:10.1145/3375395.3387641.
- 23 Martin Grohe, Gaurav Rattan, and Tim Seppelt. Homomorphism Tensors and Linear Equations. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 70:1–70:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPICs.ICALP.2022.70.
- 24 Xiaojie Guo and Liang Zhao. A Systematic Survey on Deep Generative Models for Graph Generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–20, 2022. doi:10.1109/TPAMI.2022.3214832.
- 25 Hamed Hatami and Serguei Norine. Undecidability of linear inequalities in graph homomorphism densities. *Journal of the American Mathematical Society*, 24(2):547–547, May 2011. doi:10.1090/S0894-0347-2010-00687-X.

- 26 Lane A. Hemaspaandra and Heribert Vollmer. The satanic notations: Counting classes beyond $\#P$ and other definitional adventures. *ACM SIGACT News*, 26(1):2–13, March 1995. doi:10.1145/203610.203611.
- 27 Hao Huang, Nati Linial, Humberto Naves, Yuval Peled, and Benny Sudakov. On the 3-Local Profiles of Graphs: On the 3-Local Profiles of Graphs. *Journal of Graph Theory*, 76(3):236–248, July 2014. doi:10.1002/jgt.21762.
- 28 Yannis E. Ioannidis and Raghu Ramakrishnan. Containment of Conjunctive Queries: Beyond Relations as Sets. *ACM Trans. Database Syst.*, 20(3):288–324, September 1995. Place: New York, NY, USA Publisher: Association for Computing Machinery. doi:10.1145/211414.211419.
- 29 E. Kamke. Verallgemeinerungen des Waring-Hilbertschen Satzes. *Mathematische Annalen*, 83(1-2):85–112, March 1921. doi:10.1007/BF01464230.
- 30 G. Katona. A theorem of finite sets. *Theory of Graphs, Proc. Colloquium Tihany, Hungary 1966*, 187–207, 1968.
- 31 Ker-I Ko and Wen-Guey Tzeng. Three Σ_2^P -complete problems in computational learning theory. *computational complexity*, 1(3):269–310, September 1991. doi:10.1007/BF01200064.
- 32 Swastik Kopparty and Benjamin Rossman. The homomorphism domination exponent. *European Journal of Combinatorics*, 32(7):1097–1114, 2011. Homomorphisms and Limits. doi:10.1016/j.ejc.2011.03.009.
- 33 Joseph B. Kruskal. The Number of Simplices in a Complex. In Richard Bellman, editor, *Mathematical Optimization Techniques*, pages 251–278. University of California Press, December 1963. doi:10.1525/9780520319875-014.
- 34 Jarosław Kwiecień, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. Determinacy of Real Conjunctive Queries. The Boolean Case. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 347–358. Association for Computing Machinery, 2022. doi:10.1145/3517804.3524168.
- 35 László Lovász. Coverings and colorings of hypergraphs. *Proc. 4th Southeastern Conference of Combinatorics, Graph Theory, and Computing*, pages 3–12, 1973. URL: <https://zbmath.org/0322.05114>.
- 36 László Lovász. *Combinatorial problems and exercises*. Akadémiai Kiado Elsevier Pub. Co, Budapest Amsterdam London, 2nd éd edition, 1993.
- 37 László Lovász and Alexander Schrijver. Semidefinite Functions on Categories. *Electron. J. Comb.*, 16(2), 2009. doi:10.37236/80.
- 38 László Lovász. *Large networks and graph limits*. Number volume 60 in American Mathematical Society colloquium publications. American Mathematical Society, Providence, Rhode Island, 2012. doi:10.1090/coll/060.
- 39 Stephen R. Mahaney. Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, October 1982. doi:10.1016/0022-0000(82)90002-2.
- 40 Laura Mančinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–672, 2020. doi:10.1109/FOCS46700.2020.00067.
- 41 Kenneth Manders and Leonard Adleman. NP-complete decision problems for quadratic polynomials. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing, STOC '76*, pages 23–29, New York, NY, USA, May 1976. Association for Computing Machinery. doi:10.1145/800113.803627.
- 42 Melvyn B Nathanson. *Elementary Methods in Number Theory*, volume 195 of *Graduate Texts in Mathematics*. Springer New York, New York, NY, 2000. doi:10.1007/b98870.
- 43 V. I. Nečaev. On the representation of natural numbers as a sum of terms of the form $x(x+1)\cdots(x+n-1)/n!$. *Izvestiya Akad. Nauk SSSR. Ser. Mat.*, pages 485–498, 1953. URL: <https://www.mathnet.ru/eng/im3481>.

- 44 V. I. Nečaev. On the question of representing natural numbers by a sum of terms of the form $x(x+1)\dots(x+n-1)/n!$. *Trudy Mat. Inst. Steklov.*, 142:195–197, 270, 1976. Number theory, mathematical analysis and their applications. URL: <https://www.mathnet.ru/eng/tm2571>.
- 45 Hoang Nguyen and Takanori Maehara. Graph homomorphism convolution. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7306–7316. PMLR, 2020. URL: <http://proceedings.mlr.press/v119/nguyen20c.html>.
- 46 P. V. O’Neil. Ulam’s conjecture and graph reconstructions. *The American Mathematical Monthly*, 77(1):35–43, 1970. doi:10.2307/2316851.
- 47 Ken Ono, Sinai Robins, and Patrick T. Wahl. On the representation of integers as sums of triangular numbers. *aequationes mathematicae*, 50(1):73–94, August 1995. doi:10.1007/BF01831114.
- 48 Jonathan Frederick Pollock. On the extension of the principle of Fermat’s theorem of the polygonal numbers to the higher orders of series whose ultimate differences are constant. With a new theorem proposed, applicable to all the orders. *Abstracts of the Papers Communicated to the Royal Society of London*, 5:922–924, December 1851. doi:10.1098/rspl.1843.0223.
- 49 Alexander A. Razborov. On the Minimal Density of Triangles in Graphs. *Combinatorics, Probability and Computing*, 17(4):603–618, July 2008. doi:10.1017/S0963548308009085.
- 50 David E. Roberson. Oddomorphisms and homomorphism indistinguishability over graphs of bounded degree, June 2022. URL: <http://arxiv.org/abs/2206.10321>.
- 51 David E. Roberson and Tim Seppelt. Lasserre Hierarchy for Graph Isomorphism and Homomorphism Indistinguishability. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 101:1–101:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.ICALP.2023.101.
- 52 Claus-Peter Schnorr. Optimal algorithms for self-reducible problems. In S. Michaelson and Robin Milner, editors, *Third International Colloquium on Automata, Languages and Programming, University of Edinburgh, UK, July 20-23, 1976*, pages 322–337. Edinburgh University Press, 1976.
- 53 Tim Seppelt. Logical Equivalences, Homomorphism Indistinguishability, and Forbidden Minors. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 82:1–82:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2023.82.
- 54 Janos Simon. *On Some Central Problems in Computational Complexity*. PhD thesis, Cornell University, USA, 1975.
- 55 Seinosuke Toda. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20(5):865–877, October 1991. doi:10.1137/0220053.
- 56 Jacobo Torán. Complexity classes defined by counting quantifiers. *Journal of the ACM*, 38(3):752–773, July 1991. doi:10.1145/116825.116858.
- 57 Klaus W. Wagner. Some observations on the connection between counting and recursion. *Theoretical Computer Science*, 47:131–147, January 1986. doi:10.1016/0304-3975(86)90141-6.
- 58 Hinrikus Wolf, Luca Oeljeklaus, Pascal Kühner, and Martin Grohe. Structural Node Embeddings with Homomorphism Counts, August 2023. doi:10.48550/arXiv.2308.15283.

Solving Discontinuous Initial Value Problems with Unique Solutions Is Equivalent to Computing over the Transfinite

Olivier Bournez  

École polytechnique, LIX, Paris, France

Riccardo Gozzi  

École polytechnique, LIX, Paris, France

University Paris-Est Créteil Val de Marne, LACL, Paris, France

Abstract

We study a precise class of dynamical systems that we call *solvable ordinary differential equations*. We prove that analog systems mathematically ruled by solvable ordinary differential equations can be used for transfinite computation, solving tasks such as the halting problem for Turing machines and any Turing jump of the halting problem in the hyperarithmetical hierarchy. We prove that the computational power of such analog systems is exactly the one of transfinite computations of the hyperarithmetical hierarchy.

It has been proved recently that polynomial ordinary differential equations correspond unexpectedly naturally to Turing machines. Our results show that the more general exhibited class of solvable ordinary differential equations corresponds, even unexpectedly, naturally to transfinite computations. From a wide philosophical point of view, our results contribute to state that the question of whether such analog systems can be used to solve untractable problems (both for complexity for polynomial systems and for computability for solvable systems) is provably related to the question of the relations between mathematical models, models of physics and our real world.

More technically, we study a precise class of dynamical systems: bounded initial value problems involving ordinary differential equations with a unique solution. We show that the solution of these systems can still be obtained analytically even in the presence of discontinuous dynamics once we carefully select the conditions that describe how discontinuities are distributed in the domain. We call the class of right-hand terms respecting these natural and simple conditions the class of *solvable* ordinary differential equations. We prove that there is a method for obtaining the solution of such systems based on transfinite recursion and taking at most a countable number of steps. We explain the relevance of these systems by providing several natural examples and showcasing the fact that these solutions can be used to perform limit computations and solve tasks such as the halting problem for Turing machines and any Turing jump of the halting problem in the hyperarithmetical hierarchy.

2012 ACM Subject Classification Mathematics of computing → Ordinary differential equations; Computer systems organization → Analog computers; Theory of computation; Theory of computation → Models of computation; Theory of computation → Computability

Keywords and phrases Analog models, computability, transfinite computations, dynamical systems

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.20

Funding *Olivier Bournez*: Partially supported by ANR Project *DIFFERENCE*.

Riccardo Gozzi: Partially supported by ANR Project *DIFFERENCE*.



© Olivier Bournez and Riccardo Gozzi;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 20; pp. 20:1–20:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

It has been understood quite recently that it is possible to program with Ordinary Differential Equations (ODEs): for any discrete model such as Turing machines, it is possible to build a polynomial ODE that simulates its evolution. This possibility of programming with ODEs has already been exploited to obtain various results and solve various open problems. Examples include: characterization of computability and complexity classes using ODEs [22, 21, 30, 17]; proof of the existence of a universal (in the sense of Rubel) ODE [6]; proof of the strong Turing completeness of biochemical reactions [15], or more generally various statements about the completeness of reachability problems (e.g. PTIME-completeness of bounded reachability) for ODEs [29].

Most of these studies and conclusions originated as a side effect of attempts to relate the computational power of analog computational models to classical computability. Indeed all these results relate classical computations, such as computation with Turing machines, to polynomial ordinary differential equations. One fascinating question is to understand whether it could be possible to formulate stronger models than Turing machines using classes of ODEs that are more general than polynomial ODEs. The question is interesting both from a computability point of view (can we compute more?) and a complexity point of view (can we compute faster?). In general, this investigation has pertinence in the broader context of analog computation, or computation by various alternative models, and is not limited to the framework of ODEs.

► **Remark 1 (A parallel that may help).** If this helps, our discussions can be put in parallel with the context of quantum computations, often better known than the context of analog computations. Quantum computations are mathematically based on computation over the continuum using complex numbers. Quantum models can solve some problems faster than digital models: Grover's algorithm is corroborative evidence for such an argument. This seems to suggest that models of computations based on the continuum might have additional power compared to discrete ones. Is it true for the case of models based on polynomial ODEs? The answer is clearly no with polynomial ODEs if the point of view is computability. However, from the results of [30, 29], this is more subtle and related to lengths of solutions when considering time complexity. Is it true for the case of ODEs that are more general than polynomial ones? In this article, we prove that it is possible to solve undecidable problems with discontinuous (hence non-polynomial) ODEs: we prove that it is possible to simulate transfinite computations with some ODEs.

► **Remark 2 (Is this "realistic"? Can considered results be used "in practice"?).** It is important to realise that this relates to deep philosophical questions about the relations between mathematical models, physics and our real world. For example, are mathematical models of ODEs capturing the dynamics of our physical world? Are models of physics related to our physical world? We do not aim to discuss this. However, we point out that these questions are already present for any alternative model of computation. Using the above parallel, we mean: the statement about Grover's algorithm above is that the mathematical model, considered by Grover and others, based on quantum postulates, can solve a problem faster than by digital means. But then, today's question of constructing a quantum computer can be seen as whether this mathematical model can be implemented in our real world and hence whether this mathematical model is relevant. We try to avoid these questions as much as possible in the current article, and we stay at a mathematical model level. We however think that our statements help to discuss these issues and how various models relate to our physical world.

► Remark 3 (So what is our positioning). The point of the current article is to consider a mathematically well-founded natural notion of ordinary differential equation in a context where there is a (unique, hence unambiguous) solution. We then explore its computational power and relate it to models from computability. From a mathematical point of view, we prove that it corresponds exactly to transfinite computations.

Indeed, we identify such a robust class of ODEs as the class whose dynamic is ruled by functions that we call *solvable functions*. We prove that for such a class, the solution concept is well defined, and a transfinite procedure can solve these systems. To demonstrate that, we describe the transfinite procedure in detail revealing that the maximum number of transfinite steps needed is countable. This result is expressed clearly by our main result in Section 5. Moreover, for each countable ordinal $\alpha < \omega_1$ (or $\alpha < \omega_1^{CK}$ if effectiveness is involved, where ω_1^{CK} is the first non-recursive ordinal), we show that it is possible to construct examples of discontinuous IVPs with unique solution whose solution can be obtained only after α steps of our procedure. This suggests that, according to the spirit of the approach of [3], this class of IVPs can be used in order to simulate oracle machines deciding the α -jump of the empty set, fully populating the hierarchy of hyperarithmetical reals.

More on some historical accounts and related work

About Denjoy’s totalization method for integration. It is clear that ODE solving and integrations are related since integrating is a particularly simple (restricted) case of ODE solving where the derivative is given explicitly. The question we solve has great similarities with a historical question in the context of mathematics about antidifferentiation and integration: does a method exist that can reconstruct a function f from its derivative f' in the most general setting? Unfortunately, two of the most well-known integration methods, the Riemann and Lebesgue integrals, are insufficient since they both require specific conditions on the derivative to work. Historically, Denjoy was the first to propose a concept of integral that extends the two and that is sufficiently general to solve the problem: starting from f' , using some transfinite process, one can find back f for any derivative f' . He called the method describing his integral the *totalization* method [12]. The method was purely rooted on analysis and made use of transfinite iterations of operations such as taking limits and repeated Lebesgue integrations. Starting from the derivative f' , the method can retrieve f within a maximum number of countably many transfinite steps.

Our method for solving ODEs can be related to the ideas of Denjoy, and our class of solvable ODEs have similar properties: solving such systems of ODEs can always be done at the price of a transfinite computational process. And as such ODEs can simulate any transfinite computation, they capture transfinite computations and relate transfinite computations over digital models to computations with analog models that use solvable dynamics.

ODEs as analog model of computation. From Shannon’s model to (polynomial) ODEs.

The idea of using ordinary differential equations (ODEs) as a computational model dates back to the original work of Claude Shannon. Shannon’s theoretical interest was focused on defining a general model of computation that could describe the behaviour of integrator devices. He called the model GPAC, for general purpose analog computer. The key element immediately evident to Shannon in designing his model was that every function that can be produced as output of these machines is differential-algebraic [36]. It was therefore the first stone that led years later to interpret systems of polynomial ODEs as a proper analog

computational model. The details of this evolution from algebraic differential equations to polynomial ODEs are articulate and technical: in [31], it has been demonstrated that Shannon's model was lacking of completeness and formality and hence required modifications. The authors of [20] solved these problems by restricting the connections allowed within the circuits described in the model. This modification naturally produced the interesting phenomenon of restricting the class of considered GPAC models, and the proof that its dynamics correspond precisely to solutions of polynomial ODEs.

What is known about polynomial ODEs: lower bounds. This was later proved to be equivalent to computable analysis, generating all computable functions over the reals [3]. This result was important to establish a practical bridge that could be crossed to pass from an analog model such as the GPAC to a discrete model such as the model of Turing machines. Specifically, the proofs included in [3] were based on the idea of simulating Turing machine computations by only using initial value problems (IVPs) constructed with polynomial ODEs. These particular results opened the doors for further investigations into the complexity of the GPAC model. It was subsequently discovered that the length of the solution of the ODEs involved was the right parameter to consider to measure complexity [4].

This correspondence between length and complexity can be effectively used to define a zoo of different complexity classes within the model, capturing for each of them a natural equivalence with discrete time-complexity classes such as FP or FEXP [4], [18]. The introduction of proper robustness conditions for the dynamical systems utilized to simulate Turing machines was then the last missing ingredient, which proved to be enough also to capture an equivalence with the polynomial-space-complexity class FPSPACE, as shown in [5]. The collection of all these results forms wide and substantial evidence of the fact that polynomial ODEs represent a valid paradigm for analog computation as well as showcasing their Turing completeness.

What is known about ODEs: upper bounds. Computing properties and solutions of ODEs.

On the other side of the spectrum, various investigations have been conducted to outline computability and complexity properties of the more general operation of ODEs solving. The approach from this line of work is conceptually different from what we have discussed so far. Instead of producing a continuous version of an originally discrete computation, it wonders which classes of ODEs solutions can be solved algorithmically. Undecidability of related problems is quick to arise in this context even in the presence of computable data. For example, the authors of [19] proved that the boundedness of the domain of definition is undecidable even when only polynomial ODEs are considered. For the class of polynomial-time computable, Lipschitz continuous ODEs, it is known that the solution is computable since [28]. In this specific realm, a careful analysis of the complexity of this operation has been conducted in [25], where it is proved that the solution of such problems is indeed PSPACE-complete. Following the clue provided by the Lipschitz condition, it is reasonable to assume that the uniqueness of the solution of a given IVP is a necessary prerequisite for hoping to be able to compute it. It is not a sufficient condition, as there are cases of IVPs with a unique solution and computable data for which the unique solution is not computable, such as the one in [32]. The authors have further investigated the gap between necessary and sufficient conditions for these systems in [9], where they show that solutions of continuous ODEs with unique solutions are always computable. The algorithm formulated in [9] has been called *Ten Thousand Monkeys algorithm*, since it relies on a search method on the whole solution space by listing all finite sequences of open rational boxes.

Notations

We start the coming technical part by introducing the notation and the main definitions that will be used throughout this work. We use the standard notation \mathbb{N} , \mathbb{R} and \mathbb{Q} for the set of natural, real and rational numbers respectively while \mathbb{R}^+ and \mathbb{Q}^+ represent the positive reals and the positive rationals. When making use of the norm operator, we always consider Euclidean norms. We refer to a compact domain of an Euclidean space as a nonempty, bounded, connected, closed subset of such space. Given a metric space X we indicate with the notation d_X the distance function in such space and with the notation $B_X(x, \delta)$ the open ball centred in $x \in X$ with radius $\delta > 0$. By default, we describe as open *rational ball* or as open *rational box* an open ball or box with rational parameters. Precisely, an open rational box B is a set of the form $(a_1, b_1) \times \dots \times (a_r, b_r) \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$ where $a_i, b_i \in \mathbb{Q}$ for $i = 1, \dots, r$. We indicate with the notations $\text{diam}(B)$ and $\text{rad}(B)$ its diameter and its radius. Moreover, given a function $f : [a, b] \rightarrow \mathbb{R}^r$ for some $a, b \in \mathbb{R}, a < b$ and some $r \in \mathbb{N}$ we indicate with the notation $f' : [a, b] \rightarrow \mathbb{R}^r$ the derivative of such function, where the derivative on the extremes a and b is defined as the limit on the left and on the right respectively. Given a function $f : X \rightarrow Y$ and a set $K \subseteq X$ we indicate with the notation $f \upharpoonright_K$ the restriction of the function f to the set K , i.e. $f \upharpoonright_K$ is the function from K to Y defined as $f \upharpoonright_K(x) = f(x)$. Given two topological spaces X and Y and a function $f : X \rightarrow Y$ we indicate with the notation D_f the set of discontinuity points of f on X . If A and B are two sets, we refer to the set difference operation using the symbol $A \setminus B$ and indicate with the notation $A + B$ the Minkowski sum of set A with set B . The expression $\text{cl}(A)$ indicates the closure of A , \emptyset stands for the empty set, while the notation ω_1 stands for the first uncountable ordinal number. Given a property of a function $f : X \rightarrow Y$, we say that this property is satisfied *almost everywhere* if the property is satisfied on $X \setminus D$, where D is a set with Lebesgue measure equal to zero.

2 IVPs with discontinuous ODEs

In this section we formally present the class of IVPs and dynamical systems considered, providing examples and motivations leading to the definition of our hypothesis over the right-hand term of the ODEs involved.

First, we recall the classical settings of initial value problems (IVPs) and ordinary differential equations (ODEs): Consider an interval $[a, b] \subset \mathbb{R}$, a compact domain $E \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$, a point $y_0 \in E$ and a function $f : E \rightarrow E$ such that the dynamical system:

$$\begin{cases} y'(t) = f(y(t)) \\ y(a) = y_0 \end{cases} \quad (1)$$

has one unique solution $y : [a, b] \rightarrow \mathbb{R}^r$ with $y([a, b]) \subset E$. Given y_0 and f , obtaining the solution in such a setting is called an *initial value problem*. The condition $y(a) = y_0$ (or, in short, just the point y_0) is referred to as the initial condition of the problem and function f is referred to as the right-hand term of the problem. Since the solution is uniquely defined in this case, we refer to function $y : [a, b] \rightarrow E$ satisfying Equation (1) as the solution of the problem.

Then, we discuss methods to solve initial value problems: In this particular setting, different ways exist to obtain the solution analytically when the right-hand term is continuous. Many of these methods, such as building Tonelli sequences, are often introduced for proving Peano's theorem related to the existence of the solution for IVPs with continuous

right-hand terms and are based on the concept of defining sequences of continuous functions eventually converging to the solution. Once it is known that the solution is unique, every sequence considered in each of these methods can be shown to converge to the unique solution. Their argument generally relies on fixed point theorems on the function space; therefore, it is not constructive. An analysis based on these methods can also achieve computability for the solution, where computability has to be intended in the sense of computable analysis, as proved by the authors of [9] in the description of their (so-called) *ten thousand monkeys algorithm*. The idea of this algorithm is to exploit the hypothesis of unicity for enclosing the solution into covers of arbitrarily close rational boxes in E . Nonetheless, all these methods are correctly functioning as long as the right-hand term of the IVP is **continuous**.

It is very natural to relax continuity for the right-hand term of the IVP. This is motivated by the observation that we can easily have a discontinuous IVP with a unique solution as in the coming example.

► **Example 4 (A discontinuous IVP with a unique solution).** As the simplest case of discontinuous IVP, we consider the following example: let $E = [-5, 5] \times [-15, 15]$ and define the function $f : E \rightarrow E$ as $f(x, z) = (1, 2x \sin \frac{1}{x} - \cos \frac{1}{x})$ if $x \neq 0$ and $f(0, z) = (1, 0)$. It is easy to see that f is a function of class Baire one, i.e. it is the pointwise limit of a sequence of continuous functions. Note also that in this case the set of discontinuity points of function f on E is the closed set $D_f = \{(0, z) \text{ for } z \in [-15, 15]\}$. Then consider the following IVP, with $y : [-2, 2] \rightarrow \mathbb{R}^2$ and $y_0 = (-3, 9 \sin(-\frac{1}{3}))$:

$$\begin{cases} y'(t) = f(y(t)) \\ y(-2) = y_0 \end{cases} \tag{2}$$

It is easy to verify that the solution of such a system is unique, and it is for the first component: $y_1(t) = t - 1$, and for the second component $y_2(t) = (t - 1)^2 \sin(\frac{1}{t-1})$ for $t \neq 1$ and $y_2(1) = 0$. Therefore the solution $y : [-2, 2] \rightarrow \mathbb{R}^2$ is differentiable and can be expressed as the unique solution of the IVP above with right-hand side f discontinuous on E . Note that the only discontinuity of f encountered by the solution is the point $(0, 0)$, i.e. $D_f \cap y([-2, 2]) = (0, 0)$.

► **Remark 5 (Such an ODE cannot be solved using numerical methods from literature).** It is very important to stress that neither the ten thousand monkeys algorithm nor any (at least that we know) of the general well-known methods used in analysis for obtaining the solution of IVPs we know works when applied to the above example. The reason for it being the discontinuity of the right-hand term on a straight line in the domain and the fact that these methods assume continuity.

However, one would expect to be able to solve such an ODE, as its solution is very clear, and solving such an ODE is a very classical mathematical exercise or example found in most of the books about ODEs: see similar examples in [23].

The construction of this simple example (and of the solution of this classical exercise) is based on the well-known fact that the real function $f(x) = x^2 \sin(\frac{1}{x})$ if $x \neq 0$ and $f(0) = 0$ is differentiable over $[0, 1]$ and its derivative is bounded and discontinuous in 0. Moreover, we avoided some problems that arise for mono-dimensional ODEs with null derivative by introducing a *time* variable y_1 whose role is to prevent the system from stalling and ensure the unicity of the solution.

► **Remark 6 (About literature & discontinuous ODEs).** Several mathematical theories exist for discussing discontinuous ODEs: see for example [16, 1, 11]. It is important to realize that the concept of solution differs from one theory to the other (there is not a unique theory for

discontinuous ODEs) and that the existence of a solution is often a non-trivial problem in all these theories. We are here, and in all the examples, in the case where we know that there is a solution and a unique solution, so in a case where there is no ambiguity and agreement about the solution concept. Furthermore, all the theories we know consider that equality almost everywhere in (1) is sufficient, mostly to be able to use Lebesgue integration. We indeed consider the notion of solution above: this must hold for all points. This is different in spirit to all these theories¹.

The concept behind such an example can be easily generalized. The following example provides an intuitive tool that can be deployed to construct a huge class of discontinuous IVPs with unique solutions.

► **Example 7** (Converting complex derivatives into complex IVPs). Whenever we consider a differentiable function $g : [a, b] \rightarrow \mathbb{R}$ such that $g(a) = g_0$ and with derivative $g' : [a, b] \rightarrow \mathbb{R}$ we can obtain such function as a solution of an IVP of the type of (1) by constructing a system as the following:

$$\begin{cases} y_1'(t) = 1 \\ y_2'(t) = g'(y_1(t)) \end{cases} \quad \begin{cases} y_1(a) = a \\ y_2(a) = g_0 \end{cases} \quad (3)$$

Getting to more and more complex examples. This consideration allows us to construct examples for which the set of discontinuity points of the right-hand term on the domain is more and more sophisticated.

For instance, we might consider a function whose set of discontinuity points is uncountable and nowhere dense. This case is considerably more complex to construct compared to the previous one, from a technical standpoint, but it is theoretically based on the same concept. Indeed, the idea is to use the discontinuous derivative seen in the previous case and copy it inside the Cantor set. This is done similarly to what happens when defining Volterra's function [7]. We first make use of the following statement.

► **Lemma 8.** *There exists a function $g : [0, 1] \rightarrow \mathbb{R}$ such that g is differentiable and its derivative g' is bounded and discontinuous on the Cantor set C .*

At this point, by using function g' just defined as the derivative involved with the construction of an IVP of the type of (3), we can construct an IVP with a right-hand term f with set of discontinuity points homeomorphic to the Cantor set.

More generally, the above technique makes it possible to introduce solutions that are more and more complicated depending on how discontinuous is the derivative used as function g' in (3). There are at least two possible directions in which the latter could be done.

First, since every uncountable closed subset of the Cantor set is homeomorphic to the Cantor set, it is possible to construct the differentiable function g in such a way that the restriction of its derivative to the Cantor set, i.e. $g' \upharpoonright_C$ is discontinuous on an uncountable closed subset of C . This sets the basis for iterating the procedure any infinitely countable number of times due to homeomorphism.

Second, we can construct examples using the known possible complexity of a differentiable function. Several *differentiability ranks* for measuring descriptive complexity of differentiable functions have been introduced in the literature. These ranks can be used for the purpose of

¹ And this also explains that we are closer to the question of Denjoy, which was asking about antidifferentiation of a derivative, observing that restricting to Lebesgue's integration was not solving the problem in the general case. Notice that ODE solving is a more general problem than integration, so we are also not exactly in the framework of Denjoy.

providing in a structured way more and more complex functions to play the role of function g in (3): indeed, several different differentiable ranks have been proposed in literature, such as the Kechris-Woodin rank or the Zalcwasser rank, which can be found in [26] and [38] respectively. The relations between all the existing notions is not yet fully clear, and [34] and [27] detail interesting comparisons. However, independently of which of these two routes is taken, the key concept behind the creation of highly elaborate examples is the same: one elementary discontinuous derivative (such as the one in Example 4) is used as a building block that is then rescaled and concatenated into smaller and smaller intervals converging to a new discontinuity point. It is then clear that the nature of the discontinuity on this point of the new function obtained this way would be strictly more complex than the nature of the discontinuities featured in the example used as an elementary block. Then, by using the function obtained this way as a new elementary block to rescale and concatenate, the process can continue in a fractal-like iterative fashion.

Therefore, our study establishes the premises for ranking discontinuous IVPs depending on the complexity required to solve them and foretells the development of a related hierarchy. Up to that point, all examples were mostly obtained from integrating various derivatives. However, ODE solving is a more general problem than integration, and more complicated examples can be constructed. We now go in particular to constructing an ODE that solves the halting problem of Turing machines².

3 Undecidability: solving the halting problem with discontinuous IVPs

We now show how these dynamical systems can be used for the purpose of obtaining precise undecidability results: given a Gödel enumeration of Turing machines, we define the halting problem as the problem of deciding the halting set $H = \{e : M_e(e) \downarrow\}$ where $M_e(e) \downarrow$ means that machine represented by natural e halts on input e . We consider a one-to-one total computable function over the naturals $h : \mathbb{N} \rightarrow \mathbb{N}$ that enumerates such a set. It is known that any such function enumerating a noncomputable set naturally generates a noncomputable real number [33]. The following definition expresses this:

► **Proposition 9.** *Let $h : \mathbb{N} \rightarrow \mathbb{N}$ be a one-to-one computable function such that $h(i) > 0$ for all $i \in \mathbb{N}$ and such that it enumerates a non-computable set A . Then the real number μ defined as:*

$$\mu = \sum_{i=0}^{\infty} 2^{-h(i)} \quad (4)$$

is noncomputable.

Note that in this way we always have $0 < \mu < 1$. We now describe a bidimensional dynamical system that generates the real number μ associated in the sense of the definition above to function h enumerating the halting problem.

The example mentioned above of the IVP that can assume value μ is illustrated by the following theorem:

► **Theorem 10.** *Let $E = [0, 5] \times [0, 5]$. There exists an IVP with unique solution $y : [0, 5] \rightarrow E$, rational initial condition and right-hand term computable everywhere on E except a straight-line, with $y_2(5) = \mu$.*

² We believe this is not feasible using integration only.

► **Remark 11.** The spirit of the construction of such an example is inspired by the technique used in [19], where the solution of the IVP considered is stretched in a controlled manner so that it grows infinitely approaching a fixed noncomputable time. In our case instead, with the above example, we are replacing their indefinite growth with a damped oscillation whose frequency increases as we approach the noncomputable target but whose absolute value decreases accordingly, yielding a finite convergence for the solution. This introduces many complications, and the fact that we want to guarantee the derivability of the solution is a true difficulty.

The sketch of the proof of the theorem above is the following. We first discretize time by introducing specific time slots in which both components of the solution, y_1 and y_2 , have a well-defined behaviour. Specifically, we require the first component, which is negative, to increase by a factor of 2 in each of these time slots, converging to zero. Instead, the second component, which is positive, is required to incrementally converge to the real μ by adding to itself the quantity $2^{-h(i)}$ on the i -th time step. We need two components because we want the right-hand term to be computable outside its set of discontinuity points. This is achievable in this way since indeed it is possible to implement the correct derivative for y_1 in each time slot by only going through the enumeration described by function h while looking at the value of the second component y_2 . Then, the existence and continuity of the solution is granted by designing an infinitely countable sequence of time slots that converges suitably.

We first define the function that represents the discretized time evolution of the dynamical system:

► **Definition 12.** Let $h : \mathbb{N} \rightarrow \mathbb{N}$ be a one-to-one computable function such that $h(i) > 0$ for all $i \in \mathbb{N}$ and such that it enumerates the halting set H . Define the function $\tau : \mathbb{N} \rightarrow \mathbb{Q}$ to be the total computable function such that:

$$\tau(i) = \begin{cases} 2^{-\frac{h(i)}{2}} & \text{if } h(i) < i \\ 2^{-\frac{i}{2}} & \text{if } h(i) \geq i \end{cases} \quad (5)$$

That means that $\tau^* = \sum_{i=0}^{\infty} \tau(i)$ is finite and $\tau^* < \mu + 2 + \sqrt{2} < 5$. This quantity τ^* represents the time required for the solution to reach the noncomputable value μ .

► **Remark 13.** The reason for measuring time steps with Definition 12, instead of directly exploiting the construction of μ via Proposition 9, is technical. The intuition behind it is that we want time to evolve slowly enough when compared to the increasing rate of the solution. This consideration takes care of the construction's main difficulty: the solution's differentiability at time τ^* , when the derivative is discontinuous.

Let us now proceed to analyze the behaviour of the solution y . For the first component y_1 we have a dynamic given by a function f_1 such that, for all $i \in \mathbb{N}$, if we have:

$$\begin{cases} y_1'(t) = f_1(y_1(t)) & \forall t \in [0, \tau(i)] \\ y_1(0) = -2^{-i} \end{cases} \quad (6)$$

then we have $y_1(\tau(i)) = -2^{-(i+1)}$. In other words, we require y_1 to be an increasing function such that at every time step $\tau(i)$ its value increases by a factor of 2, converging then to 0 as time converges to τ^* . Therefore, to make it continuous, we require $y_1(\tau^*) = 0$. Moreover, we define $y_1'(\tau^*) = 0$. Note also that to achieve this goal we require f_1 to be autonomous, with no explicit dependence on time. It is clear that if we construct y_1 this way, then its derivative will be discontinuous in τ^* .

20:10 Solving Discontinuous IVPs with Unique Solutions = Computing over the Transfinite

For the second component y_2 we have a dynamic given by a function f_2 such that, for all $i \in \mathbb{N}$, if we have:

$$\begin{cases} y_2'(t) = f_2(y_1(t)) & \forall t \in [0, \tau(i)] \\ y_1(t) \in [-2^{-i}, -2^{-(i+1)}] & \forall t \in [0, \tau(i)] \\ y_2(0) = \sum_{m=0}^i 2^{-h(m)} \end{cases} \quad (7)$$

then we have $y_2(\tau(i)) = \sum_{m=0}^{i+1} 2^{-h(m)}$ (condition *). In other words, we require y_2 to be an increasing function such that at every time step $\tau(i)$ its value increases of the quantity $2^{-h(i+1)}$, converging then to μ as time converges to τ^* . Once again, we require $y_2(\tau^*) = 0$ and $y_2'(\tau^*) = 0$ and it is clear that if we construct y_2 this way then its derivative will be discontinuous in τ^* .

Such a solution y is indeed differentiable at time τ^* and the derivative of both components exists and equals zero at such time. Moreover, we have that $y_1(\tau^*) = 0$ and $y_2(\tau^*) = \mu$. At this point, forcing the dynamic to remain constant for the remaining time to obtain $y_2(5) = \mu$ is sufficient. This yields the desired outcome since the IVP has reached, at a computable time, a noncomputable value that encodes the halting problem. This ends the sketch of the proof of Theorem 10.

► **Remark 14.** Even if this is not fully formal: it is important to observe that the obtained function f remains very “simple” from a (possibly effective) descriptive point of view. Outside its straight-line of discontinuity, it is computable, and from the fact that Turing machines computations can be done using polynomial ODEs, we could even assume that it has a very simple form outside this straight-line: we basically only need to get condition (*).

We mean, it is important to realize that the fact that the solution is not computable does not come from the intrinsic uncomputability of the function f , nor from the form of the subset of discontinuities, but actually from the fact that it has a discontinuity and that the whole process and above construction is intrinsically forcing the solution to compute a limit.

Iterating limits. For a set A , let us call real number $\sum_{i \in A} 2^{-i}$ the real encoding of A . We just described a dynamics that maps some rational initial condition to the real μ encoding the halting set H . Writing A' for the jump of set A , H corresponds to \emptyset' .

It is possible to extend the previous construction to make it work for any set A (not only the empty set): starting from some initial condition corresponding to the real encoding of A , it eventually reaches the real encoding of A' .

We can climb the arithmetical hierarchy by iterating finitely many times this technique. Indeed, by repeating the ODE twice, we get a way to map a real encoding of A to a real encoding of A'' , then A''' , and so on.

We can even go up to higher levels. Indeed, for example, we can go up to A^ω : A^ω is the set of the pairs (n, w) such that word w is in the n th jump of A . Iterating the trick, we can reach the upper levels of the hyperarithmetical hierarchy. Given any recursive ordinal $\alpha < \omega_{CK}^1$, this provides a technique to map some real encoding of A in the initial condition to the encoding of A^α , where A^α is the α th jump: see [35] for the concepts from computability theory involved. Doing it in a recursive manner requires technically to deal with the encoding of ordinals, and in particular to deal with fixed point constructions as in [13, 37].

4 Transfinite analysis of complexity

4.1 The concept of solvable system

We propose the following definition.

► **Definition 15** (Solvable function). *Let $E \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$ and $f : E \rightarrow \mathbb{R}^r$. We say that f is solvable if it is a function of class Baire one such that for every closed set $K \subseteq E$ the set of discontinuity points of the restriction $f \upharpoonright_K$ is a closed set.*

It is important first to make the following observation:

► **Remark 16.** All the examples of dynamical systems discussed in Section 2 as well as the IVP introduced by Theorem 10 have a right-hand term that is a solvable function.

We say that a dynamical system or some ordinary differential equation is *solvable* when this holds: the right-hand term of the ordinary differential equation is a solvable function.

It is easy to see that the remarks still apply to the case of any of the more complicated examples that can be constructed based on the previous ones and on Theorem 10, once the techniques for building such examples are the ones mentioned at the end of Section 2.

The choice of the terminology *solvable* for these right-hand terms is made more clear by the coming Theorem 22. It says basically that solutions of such initial value problems can be solved, through a transfinite recursion process.

4.2 A ranking for solvable systems

Before getting to this, we introduce a ranking that allows us to quantify involved levels of discontinuities.

We have seen in previous sections that we can build examples of unique solutions of IVPs that are extremely complicated and that even simple examples can be used to obtain noncomputable reals. We now want to produce a more precise quantification of these statements.

► **Remark 17** (Using differentiability ranks?). Following previous arguments, the most intuitive direction seems to derive such quantification directly from the examples and the differentiability ranks. Nonetheless, this approach does not suit our purpose, since it only characterizes a limited subclass of systems, i.e. the systems yielded by application of the trick introduced with (3). Despite being a relevant and insightful subclass, this approach “from below” fails to exhaust the generality of the problem. We instead propose an approach “from above” which builds from the commonalities of those examples and extends beyond them to a more complete analysis that is not just tailored on derivatives defined over the reals.

As a first step, we prepare the right setting for a transfinite classification of the right-hand terms of our systems. As it is clear by the examples illustrated in previous sections, such stratification should be based upon the degree of discontinuity for the right-hand term of the system. We can quantify this precisely by introducing the following definition.

► **Definition 18** (Sequence of f -removed sets on E). *Consider a compact domain $E \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$ and a function $f : E \rightarrow \mathbb{R}^r$. Let $\{E_\alpha\}_{\alpha < \omega_1}$ be a transfinite sequence of sets and $\{f_\alpha\}_{\alpha < \omega_1}$ a transfinite sequence of functions such that $f_\alpha = f \upharpoonright_{E_\alpha} : E_\alpha \rightarrow \mathbb{R}^r$ defined as following:*

- Let $E_0 = E$
- For every $\alpha = \beta + 1$, let $E_\alpha = D_{f_\beta}$
- For every α limit ordinal, let $E_\alpha = \bigcap_{\beta < \alpha} E_\beta$ with $\beta < \alpha$

we call the sequence $\{E_\alpha\}_{\alpha < \omega_1}$ the sequence of f -removed sets on E .

We remark that since functions in the sequence $\{f_\alpha\}_{\alpha < \omega_1}$ above are allowed to be defined over disconnected sets, the notion of continuity in the above definition has to be intended with respect to the induced topology relative to E_α as a subset of \mathbb{R}^r . Moreover, note that it follows from the definition of the sequence that such sequence is decreasing, meaning that $E_\delta \subseteq E_\gamma$ if $\delta > \gamma$ for every E_δ and E_γ in the sequence.

► **Remark 19** (Similar ranking for measuring discontinuities in literature). The definition of this sequence, or of slight variations of the same sequence, has already been considered in the literature. For instance, the author of [24] selects a version of this sequence where the closure of the sets is taken at each level and relates such sequence of functions with a bound for the topological complexity of any algorithm that computes them while using only comparisons and continuous arithmetic (and information) operations. Similarly, starting from the same transfinite sequence of functions applied to countably based Kolmogorov spaces, it is shown in [10] that a given function is at the α level of the hierarchy if and only if it is realizable through the α -jump of a representation.

Since computing the unique solution of continuous IVP is always possible, from the arguments of [9], being able to obtain analytically the unique solution of any given discontinuous IVP should be directly related to the amount of discontinuity for the right-hand term f , and consequently to the ordinal number of nonempty levels of the above sequence of f -removed sets. Moreover, we would like to obtain the solution within a countable number of steps. Hence, we want to pinpoint some sufficient conditions on f that permit us to restrict our attention to these well-behaved classes of discontinuous systems.

This ranking turns out to provide a way to rank the concept of *solvable* systems. Using the Cantor-Baire stationary principle, we can prove:

► **Theorem 20.** *Consider a closed domain $E \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$ and a function $f : E \rightarrow \mathbb{R}^r$. If f is solvable, then there exists an ordinal $\alpha < \omega_1$ such that $E_\alpha = \emptyset$.*

Once we have singled out which conditions we must require for the right-hand term f , we can present the main tool used to converge to the solution of the IVP. In a similar fashion to the method designed by Denjoy, where the tool to be repeatedly applied was Lebesgue integration, we need to be able to apply such tool for each considered level of the sequence of f -removed sets on E until we finally reach the empty set. This is why we created a tool that can be defined for any countable ordinal in a uniform manner. We call the tool (α)Monkeys approach in honor of the ten thousand monkey algorithm from [9], since such an algorithm inspires the definition.

► **Definition 21** ((α)Monkeys approach). *Consider an interval $[a, b] \subset \mathbb{R}$, a domain $E \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$ and a right-hand term $f : E \rightarrow \mathbb{R}^r$ for an ODE of the form of (1) with initial condition y_0 . Let $\{E_\gamma\}_{\gamma < \omega_1}$ be the sequence of f -removed sets on E and let E_α be one set in the sequence for some $\alpha < \omega_1$. We call the (α)Monkeys approach for (f, y_0) the following method: consider all tuples of the form $(X_{i,\beta,j}, h_{i,\beta,j}, B_{i,\beta,j}, C_{i,\beta,j}, Y_{i,\beta,j})$ for $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$, where $h_{i,\beta,j} \in \mathbb{Q}^+$, $l, m_i \in \mathbb{N}$ and $X_{i,\beta,j}$, $B_{i,\beta,j}$, $C_{i,\beta,j}$ and $Y_{i,\beta,j}$ are open rational boxes in E . A tuple is said to be valid if $y_0 \in \bigcup_{\beta,j} X_{0,\beta,j}$ and for all $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$ we have:*

1. Either $(B_{i,\beta,j} = \emptyset)$ or $(\text{cl}(B_{i,\beta,j}) \cap E_\beta \neq \emptyset \text{ and } \text{cl}(B_{i,\beta,j}) \cap E_{\beta+1} = \emptyset)$
2. $f \upharpoonright_{E_\beta} (\text{cl}(B_{i,\beta,j})) \subset C_{i,\beta,j}$;
3. $X_{i,\beta,j} \cup Y_{i,\beta,j} \subset B_{i,\beta,j}$;
4. $X_{i,\beta,j} + h_{i,\beta,j} C_{i,\beta,j} \subset Y_{i,\beta,j}$;
5. $\bigcup_{\beta,j} Y_{i,\beta,j} \subset \bigcup_{\beta,j} X_{i+1,\beta,j}$;

We spend a few words explaining the rationale behind such a definition. Similarly to the case of the ten thousand monkey algorithm, this definition defines a search method within the space E where the solution of the IVP lives. The search is performed by considering tuples of the form $(X_{i,\beta,j}, h_{i,\beta,j}, B_{i,\beta,j}, C_{i,\beta,j}, Y_{i,\beta,j})$ which describe finite sequences of l open sets. Each of these tuples should be thought as an expression of its related finite sequence of l open sets that is $\{\bigcup_{\beta,j} X_{i,\beta,j}\}_{i=0,2,\dots,l}$. Each of these open sets is the union of a transfinite collection of open rational boxes. Stating that one of these tuples is valid means two things: one, that the related sequence starts from a set that contains the initial condition, and two, that the sets in the sequence are concatenated correctly according to the five rules above. These rules are chosen so that the concatenation between the sets is dictated by the action of f , but only in a controlled fashion, i.e. in a manner that takes care of the portions of the domain where each restriction f_β is continuous, for all $\beta < \alpha$. This is clarified by the first item in the above list, whose direct consequence is that f_β is continuous on all rational boxes $B_{i,\beta,j}$. This consideration allows to interpret the sequences expressed by valid tuples as good candidates for possibly containing the solution. This sets the premises for the next section, where the method to obtain the solution of the IVP is finally presented.

5 Obtaining the solution: an analytical method

We now have all the elements needed to describe the transfinite method that obtains analytically the solution of the IVP considered and hence proves our main result.

The idea is, as in [9], to use a search method based on boxes covering the domain. By considering smaller and smaller radii for these boxes, we can derive a sequence of continuous piecewise linear functions that eventually converge to a solution. As we know that the solution is unique, it must converge to the solution.

However, compared to the authors of [9], we have to deal with possibly transfinitely many boxes, unlike their framework where everything remains finite. This requires some modifications (e.g. the domain must be bounded in the reasoning), and more technical care (e.g. for concatenating the boxes).

► **Theorem 22.** *Consider a closed interval, a compact domain $E \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$ and a function $f : E \rightarrow E$ such that, given an initial condition, the IVP of the form of (1) with right-hand term f has a unique solution on the interval. If f is solvable, then we can obtain the solution analytically via transfinite recursion up to an ordinal α such that $\alpha < \omega_1$.*

Proof. Let $[a, b]$ be the closed interval such that $y : [a, b] \rightarrow E$ is the unique solution of the IVP with right-hand term f and initial condition $y_0 = y(a)$. Let $\{E_\gamma\}_{\gamma < \omega_1}$ be the sequence of f -removed sets on E . Since f is solvable, by means of Theorem 20 we know that there exists an $\alpha < \omega_1$ such that $E_\alpha = \emptyset$ and $E_\beta = \emptyset$ for all $\beta \geq \alpha$. Therefore by transfinite recursion up to α based on repeated application of f we can consider the whole sequence of f -removed sets on E . We now show how to obtain the solution $y([a, b])$. We first pick a $n \in \mathbb{N}$ and consider a valid tuple of the (α) Monkeys approach for (f, y_0) for this value of n . We consider a valid tuple with $(X_{i,\beta,j}, h_{i,\beta,j}, C_{i,\beta,j}, Y_{i,\beta,j})$ for $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$ dependent on this fixed n . To do so, consider a set $\bigcup_{\beta,j} X_{0,\beta,j}$ such that $y_0 \in \bigcup_{\beta,j} X_{0,\beta,j}$. Then, for all sets $\bigcup_{\beta,j} X_{i,\beta,j}$ for all $i = 0, \dots, l-1$ we can select each open rational box $X_{i,\beta,j}$ so that it satisfies either $(X_{i,\beta,j} = \emptyset)$ or $(X_{i,\beta,j} \cap E_\beta \neq \emptyset \text{ and } X_{i,\beta,j} \cap E_{\beta+1} = \emptyset)$ for all $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$. Moreover, because every continuous function on a closed domain is uniformly continuous, we can define for all $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$ the function $\delta_{i,\beta,j} : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ to be a modulus of continuity of f_β on $\text{cl}(X_{i,\beta,j}) \cap E_\beta$, i.e. a

function such that $\|f_\beta(x) - f_\beta(z)\| < \delta_{i,\beta,j}(\|x - z\|)$ for all $x, z \in \text{cl}(X_{i,\beta,j}) \cap E_\beta$, for all $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$. By convention, for all $\epsilon > 0$, if there are no points $x, z \in \text{cl}(X_{i,\beta,j}) \cap E_\beta$ such that $\|x - z\| < \epsilon$ then we define $\delta_{i,\beta,j}(\epsilon) = \epsilon$. Let us now call $K \in \mathbb{Q}^+$ a rational such that $\max_{x \in E} \|x\| < K$. At this point, by taking the partition sufficiently small, we can make sure to take each nonempty open rational box $X_{i,\beta,j}$ and rational $h_{i,\beta,j}$ such that $0 < \text{rad}(X_{i,\beta,j}) < \delta_{i,\beta,j}(\frac{1}{2n}) - Kh_{i,\beta,j}$ and such that its $Kh_{i,\beta,j}$ neighborhood has no intersection with $E_{\beta+1}$ and has the same modulus of continuity $\delta_{i,\beta,j}$. Take then each one of these neighborhoods as the set $B_{i,\beta,j}$ for all $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$. It follows that $\text{rad}(B_{i,\beta,j}) < \text{rad}(X_{i,\beta,j}) + Kh_{i,\beta,j} < \delta_{i,\beta,j}(\frac{1}{2n})$. We then choose open rational boxes $C_{i,\beta,j}$ such that they satisfy $f_\beta(\text{cl}(B_{i,\beta,j})) \subset C_{i,\beta,j}$. Note that we can make the choice in a way that ensures $\text{rad}(C_{i,\beta,j}) < \frac{1}{2n}$ by definition of the moduli of continuity. From these choices it follows that we can pick open rational boxes $Y_{i,\beta,j}$ satisfying $X_{i,\beta,j} + h_{i,\beta,j}C_{i,\beta,j} \subset Y_{i,\beta,j}$ and $\text{rad}(Y_{i,\beta,j}) < \delta_{i,\beta,j}(\frac{1}{2n})$ for all $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$. Finally, we consider as the set $\bigcup_{\beta,j} X_{i+1,\beta,j}$ a set such that $\bigcup_{\beta,j} Y_{i,\beta,j} \subset \bigcup_{\beta,j} X_{i+1,\beta,j}$ for all $i = 0, \dots, l-1$. It is clear that the tuple described this way is a valid tuple of the (α) Monkeys approach for (f, y_0) .

Let us now define two sequences $\{h_{i,\beta(i),j(i)}\}_{i=0,\dots,l-1}$ and $\{t_i\}_{i=0,\dots,l}$ where $t_0 = a$ and $t_i = a + \sum_{k=0}^{i-1} h_{k,\beta(k),j(k)}$ for all $i = 1, \dots, l$ and a piecewise linear function $\eta_n : [a, t_i] \rightarrow E$ such that $\eta_n(a) = y_0$ and such that for all $i = 0, \dots, l-1$ we have $\eta_n(t_i) \in X_{i,\beta(i),j(i)}$ and $\eta_n(t) = \eta_n(t_i) + (t - t_i)c_{i,\beta(i),j(i)}$ for all $t_i < t \leq t_{i+1}$, for some $c_{i,\beta(i),j(i)} \in C_{i,\beta(i),j(i)}$. Note that this function is well defined because $|t_{i+1} - t_i| = h_{i,\beta(i),j(i)}$ for all $i = 0, \dots, l-1$ and so it follows that $\eta_n(t) \in Y_{i,\beta(i),j(i)} \subset \bigcup_{\beta,j} X_{i+1,\beta,j}$ for all $t_i < t \leq t_{i+1}$. In other words, we can always choose the sequences in such a way that function η_n is well defined. Moreover, note that $\eta'_n(t) = c_{i,\beta(i),j(i)} \in C_{i,\beta(i),j(i)}$ for all $t_i < t < t_{i+1}$, for all $i = 0, \dots, l-1$; note also that since $\eta_n(t) \in B_{i,\beta(i),j(i)}$ we have $f_{\beta(i)}(\eta_n(t)) \in C_{i,\beta(i),j(i)}$ for all $t_i < t < t_{i+1}$, for all $i = 0, \dots, l-1$. Therefore we have $\|\eta'_n(t) - f_{\beta(i)}(\eta_n(t))\| \leq \text{diam}(C_{i,\beta,j}) < \frac{1}{n}$ for all $t_i < t < t_{i+1}$ such that $\eta_n(t) \in E_{\beta(i)}$, for all $i = 0, \dots, l-1$.

Suppose we have just considered a valid tuple of the (α) Monkeys approach for (f, y_0) for a fixed value $\bar{n} \in \mathbb{N}$ following the above procedure and we have defined function $\eta_{\bar{n}} : [0, t_i] \rightarrow E$ in the way described. Let us indicate this t_l with the symbol T . It is clear that we can consider a new valid tuple and a new function $\eta_n : [0, T] \rightarrow E$ for each value of $n > \bar{n}$ while maintaining the same domain for each function. We can then consider a sequence of the functions $\{\eta_n\}_{n > \bar{n}}$ as defined above, where each function in the sequence is defined based on the valid tuple $(X_{i,\beta,j}, h_{i,\beta,j}, C_{i,\beta,j}, Y_{i,\beta,j})$ with $\text{rad}(X_{i,\beta,j}) < \delta_{i,\beta,j}(\frac{1}{2n})$, $\text{rad}(B_{i,\beta,j}) < \delta_{i,\beta,j}(\frac{1}{2n})$ and $\text{rad}(C_{i,\beta,j}) < \frac{1}{2n}$ for all $n > \bar{n}$, $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$ as described above. We want to show that such sequence $\{\eta_n\}_{n > \bar{n}}$ is uniformly bounded and equicontinuous. To prove that it is uniformly bounded we need to prove that there exists a constant $R \in \mathbb{R}^+$ such that $\|\eta_n(t)\| \leq R$ for all $n > \bar{n}$, for all $a \leq t \leq T$. This is indeed trivial since $\eta_n(t) \in \bigcup_{i,\beta,j} B_{i,\beta,j}$ for all $n > \bar{n}$, for all $a \leq t \leq T$ and each open rational box $B_{i,\beta,j} \subset E$ for all $n > \bar{n}$, $i = 0, \dots, l-1$, $\beta < \alpha$, $1 \leq j \leq m_{i,\beta}$. For equicontinuity it is enough to prove that there exists a constant $M \in \mathbb{R}^+$ such that $\|\eta_n(\tilde{t}) - \eta_n(t)\| \leq M|t - \tilde{t}|$ for all $n > \bar{n}$, for all $a \leq t, \tilde{t} \leq T$. The existence of M follows from the fact that the sequence is uniformly bounded together with the fact that $\|\eta'_n(t)\| < K$ for all $n > \bar{n}$, for almost all $a \leq t \leq T$. Therefore, since the sequence is uniformly bounded and equicontinuous, we can apply a well-known theorem in analysis (Ascoli's theorem, Theorem 28) in order to conclude that the sequence $\{\eta_n\}_{n > \bar{n}}$ has a subsequence $\{\eta_{n(u)}\}_{u > \bar{n}}$ that converges uniformly on $[a, T]$ to a function $\eta : [a, T] \rightarrow E$. Moreover, another known result for the differentiability of the limit of sequences (Theorem 29) tells us that function η is differentiable almost everywhere on

$[a, T]$. Note that by taking limit of $n \rightarrow \infty$ we have $l \rightarrow \infty$ and $\text{rad}(B_{i,\beta,j}), \text{rad}(C_{i,\beta,j}) \rightarrow 0$ and $h_{i,\beta,j} \rightarrow 0$ for all $i = 0, \dots, l-1$, $\beta < \alpha$, $j = 1, \dots, m_{i,\beta}$. Therefore the inequality $\|\eta'_n(t) - f_\beta(\eta_n(t))\| \leq \text{diam}(C_{i,\beta,j}) < \frac{1}{n}$ for all $t_i < t < t_{i+1}$ such that $\eta_n(t) \in E_{\beta(i)}$, for all $i = 0, \dots, l-1$, leads to the equation $\eta'(t) = f(\eta(t))$ for almost all $t \in [a, T]$. Since $\eta(a) = y_0$, continuity and unicity of the solution of the IVP imply $\eta(t) = y(t)$ for all $t \in [a, T]$. Specifically, this means that any convergent subsequence converges to the same function, which is precisely the solution y of the IVP.

Finally, to obtain the solution over the whole domain $[a, b]$, it is sufficient to consider the initial valid tuple of the (α)Monkeys approach for (f, y_0) as described above but in such a way that, when defining the sequence of times $\{t_i\}_{i=0, \dots, l}$ we have $t_l \geq b$ and then take $T = b$. This is always possible due to the definition of a valid tuple and the fact that f is bounded within E . ◀

The above statement, combined with the constructions from Section 3, hence proves that the computational power of solvable ODEs is the one of transfinite computations, up to some limit ordinal. We now discuss what this ordinal is, according to the adopted viewpoint (the issue is about which functions are considered definable in the above reasonings, and appears only for ordinals that would be countable but non-recursive, i.e. non-countable in the considered model of set theory).

► **Remark 23 (On ω_1 vs ω_1^{CK} , Boldface view).** The statement of Theorem 22 is formulated in an approach based on descriptive set theory, using the approach of the so-called boldface hierarchies. The description of the examples in Section 3 follows an approach that is closer to a computability theoretic point of view, that is to say using the approach of the so-called lightface hierarchies. From a boldface point of view, what the constructions of Section 3 say is that it is possible to reach the level of Baire's hierarchy using limits constructed in the spirit of the examples of this section. This can be done up to level ω_1 (non-included), the first uncountable ordinal. Combined with Theorem 22, the computational power of solvable ODEs corresponds to transfinite iterations of limits up to any ordinal less than ω_1 .

► **Remark 24 (On ω_1 vs ω_1^{CK} , Lightface view).** From a lightface point of view, it makes sense to replace the hypothesis “of Class Baire one” (it is the pointwise limit of a sequence of continuous functions) in the definition of Solvable function (Definition 15) by the fact that it is the pointwise limit of a computable sequence of computable functions. All solvable examples that we considered are solvable in this new sense. Hyperarithmetical sets are known to correspond to sets that can be defined using transfinite induction up to ω_1^{CK} , which is the first non-recursive ordinal [2]. The above reasoning of the proof of Theorem 22 provides a way to obtain (define) the solution in the hyperarithmetical hierarchy via a transfinite recursion up to an ordinal α such that $\alpha < \omega_1^{CK}$: we are using arguments similar to the ones of [13]. We hence obtain that there is a precise correspondence between computations by the class of solvable systems and the hyperarithmetical hierarchy, as the hyperarithmetical hierarchy is known to correspond to transfinite recursion up to ordinal ω_1^{CK} : see [35, 2].

6 Conclusions and future work

We have discussed the properties of IVPs involving discontinuous ODEs that have a unique solution. This study has led us to the identification of a robust class of these systems, which we called *solvable*, for which the solution can always be obtained analytically by means of transfinite recursion up to a countable number of maximum steps. We have presented several examples of such systems and illustrated a technique that constructs examples of

ever-increasing complexity. We have established that the solutions of solvable systems can be used, even in the simple case of a basic set of discontinuity points, to yield noncomputable values and solve the halting problem.

Due to the similarity of our approach to the method proposed by Denjoy for the problem of antidifferentiation, and in light of results and constructions illustrated in papers such as [13] and [37], we believe in having set the tables for a more in-depth light-face and bold-face analysis of the class of systems at hand, possibly leading to a rank and a hierarchy of these systems, as well to a classification of hyperarithmetical real numbers as targets reachable by solutions of discontinuous IVPs. More precisely: the integrability rank that inspired the modus operandi of our ranking is the Denjoy rank. The creation of such rank is directly based on a transfinite method.

- An analysis has been done for the Denjoy rank: an unpublished theorem from Ajtai, whose proof is included in [13], demonstrates that once a code for a derivative f is given as a computable sequence of computable functions converging pointwise to f , then the antiderivative F of f is Π_1^1 relative to f . This fact has direct implications related to the hierarchy of hyperarithmetical reals. The hyperarithmetical reals, or Δ_1^1 , are defined as the reals x for which the set $\{r \in \mathbb{Q} : r < x\}$ is Δ_1^1 . The implication mentioned above is expressed formally by a Theorem from [13], which proves that the hyperarithmetical reals are exactly those reals x such that $x = \int_0^1 f$ for some derivative f of which we know the code of.
- An alternative computability theoretic analysis of Denjoy's rank has been done in [37], relating levels of the hierarchy to levels of the arithmetical hierarchy in some precise manner, using a slightly alternative setting, on the way objects are encoded.

We believe that adapting similar analysis to the framework of solvable systems can be done using both views, and could lead to similar statements, with a more precise analysis of involved rankings, and of involved ordinals, given some class of functions or dynamics.

The latter, combined with papers such as [3] and [4] that describe simulations of discrete models of computations by analog models based on systems of ODEs, open the doors for identifying the model of solvable IVPs as an analog model for simulating transfinite computation, or as an alternative approach for presenting transfinite computations.

Notice that our discussions also pointed out classes of ordinary differential equations with solutions with levels of complications that we did not see discussed in any books about ordinary differential equations, in addition to many already existing counterexamples in literature. In particular, from arguments similar to [13], it follows that our results show that the totality of countable ordinals is necessary in any constructive process for solving an ODE in the general case and that for any countable ordinal, we can construct an example of solvable ODE of that difficulty.

References

- 1 Jean-Pierre Aubin and Arrigo Cellina. *Differential inclusions: set-valued maps and viability theory*, volume 264. Springer Science & Business Media, 2012.
- 2 Jon Barwise. *Admissible Sets and Structures*. Perspectives in mathematical logic. Cambridge University Press, 2016.
- 3 O. Bournez, M. L. Campagnolo, D. S. Graça, and E. Hainry. The General Purpose Analog Computer and Computable Analysis are two equivalent paradigms of analog computation. In J.-Y. Cai, S. B. Cooper, and A. Li, editors, *Proc. Theory and Applications of Models of Computation (TAMC 06)*, volume 3959 of *Lecture Notes in Computer Science*, pages 631–643. Springer, 2006.

- 4 O. Bournez, D. S. Graça, and A. Pouly. Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length. *Journal of the ACM*, 64(6):38:1–38:76, 2017.
- 5 Olivier Bournez, Riccardo Gozzi, Daniel S Graça, and Amaury Pouly. A continuous characterization of pspace using polynomial ordinary differential equations. *Journal of Complexity*, 77:101755, 2023.
- 6 Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. *Logical Methods in Computer Science*, 16(1), 2020. [arXiv:1702.08328](https://arxiv.org/abs/1702.08328).
- 7 David M Bressoud. *A radical approach to Lebesgue's theory of integration*. Cambridge University Press, 2008.
- 8 E. A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations*. McGraw-Hill, 1955.
- 9 P. Collins and D. S. Graça. Effective computability of solutions of differential inclusions — the ten thousand monkeys approach. *Journal of Universal Computer Science*, 15(6):1162–1185, 2009.
- 10 Matthew de Brecht. Levels of discontinuity, limit-computability, and jump operators¹. *Logic, computation, hierarchies*, 4:79, 2014.
- 11 Klaus Deimling. *Multivalued differential equations*, volume 1. Walter de Gruyter, 2011.
- 12 Arnaud Denjoy. Une extension de l'intégrale de m. lebesgue. *CR Acad. Sci. Paris*, 154:859–862, 1912.
- 13 Randall Dougherty and Alexander S Kechris. The complexity of antidifferentiation. *Advances in Mathematics*, 88(2):145–169, 1991.
- 14 R. Estrada and J. Vindas. On romanovski's lemma. *Real Analysis Exchange*, 35:431–444, 2010.
- 15 Francois Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In *Computational Methods in Systems Biology-CMSB 2017*, 2017.
- 16 A. Filippov. *Differential equations with discontinuous right-hand sides*. Kluwer Academic Publishers, 1988.
- 17 Riccardo Gozzi. *Analog Characterization of Complexity Classes*. PhD thesis, Instituto Superior Técnico, Lisbon, Portugal and University of Algarve, Faro, Portugal, 2022.
- 18 Riccardo Gozzi and Daniel Graça. Characterizing time computational complexity classes with polynomial differential equations. *Computability*, 12(1):23–57, 2023.
- 19 D. S. Graça, J. Buescu, and M. L. Campagnolo. Boundedness of the domain of definition is undecidable for polynomial ODEs. In R. Dillhage, T. Grubba, A. Sorbi, K. Weihrauch, and N. Zhong, editors, *Proc. 4th International Conference on Computability and Complexity in Analysis (CCA 2007)*, volume 202 of *Electronic Notes in Theoretical Computer Science*, pages 49–57. Elsevier, 2007.
- 20 D. S. Graça and J. F. Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19(5):644–664, 2003.
- 21 Daniel S. Graça. *Computability with Polynomial Differential Equations*. PhD thesis, Instituto Superior Técnico, 2007.
- 22 Emmanuel Hainry. *Modèles de calculs sur les réels. Résultats de Comparaisons*. PhD thesis, LORIA, 7 Décembre 2006.
- 23 P. Hartman. *Ordinary Differential Equations*. Birkhäuser, 2nd edition, 1982.
- 24 Peter Hertling. Topological complexity with continuous operations. *Journal of Complexity*, 12(4):315–338, 1996.
- 25 A. Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. *Computational Complexity*, 19(2):305–332, 2010.
- 26 Alexander S Kechris and W Hugh Woodin. Ranks of differentiable functions. *Mathematika*, 33(2):252–278, 1986.
- 27 Haseo Ki. On the denjoy rank, the kechris-woodin rank and the zalcwasser rank. *Transactions of the American Mathematical Society*, 349(7):2845–2870, 1997.

- 28 K.-I Ko. On the computational complexity of ordinary differential equations. *Information and control*, 58:157–194, 1983.
- 29 Daniel S. Graça Olivier Bournez and Amaury Pouly. Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. *Journal of the ACM*, 64(6):38:1–38:76, 2017. doi:10.1145/3127496.
- 30 Amaury Pouly. *Continuous models of computation: from computability to complexity*. PhD thesis, Ecole Polytechnique and Unidersidade Do Algarve, 2015. Defended on july 6, 2015, Prix de Thèse de l’Ecole Polytechnique 2016, Ackermann Award 2017. URL: <https://pastel.archives-ouvertes.fr/tel-01223284>.
- 31 M. B. Pour-El. Abstract computability and its relations to the general purpose analog computer. *Transactions of the American Mathematical Society*, 199:1–28, 1974.
- 32 M. B. Pour-El and J. I. Richards. The wave equation with computable initial data such that its unique solution is not computable. *Advances in Mathematics*, 39:215–239, 1981.
- 33 Marian B Pour-El and J Ian Richards. *Computability in analysis and physics*, volume 1. Cambridge University Press, 2017.
- 34 TI Ramsamujh. Three ordinal ranks for the set of differentiable functions. *Journal of Mathematical Analysis and Applications*, 158(2):539–555, 1991.
- 35 Hartley Rogers Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, April 1987.
- 36 C. E. Shannon. Mathematical theory of the differential analyzer. *Journal of Mathematics and Physics*, 20:337–354, 1941.
- 37 Linda Westrick. An effective analysis of the denjoy rank. *Notre Dame Journal of Formal Logic*, 61(2), 2020.
- 38 Zygmunt Zalcwasser. Sur une propriete du champ des fonctions continues. *Studia Mathematica*, 2(1):63–67, 1930.
- 39 Dongsheng Zhao. Functions whose composition with baire class one functions are baire class one. *Soochow Journal of Mathematics*, 33(4):543, 2007.

A Appendix: Useful definitions and results

We include in this appendix some definitions and theorems that can be used to integrate the main document for a deeper understanding of its arguments.

A.1 Ordinal numbers

We describe the process of *transfinite recursion* as the process that for each ordinal α associates with α an object that is described in terms of objects already associated with ordinals $\beta < \alpha$. Moreover, we use the expression *transfinite recursion up to α* if the process associates an object for all ordinals $\beta < \alpha$. We present the Cantor-Baire stationary principle [14], as expressed by the following theorem:

► **Theorem 25** (Cantor-Baire stationary principle). *Let $\{E_\gamma\}_{\gamma < \omega_1}$ be a transfinite sequence of closed subsets of \mathbb{R}^r for some $r \in \mathbb{N}$. Suppose $\{E_\gamma\}_{\gamma < \omega_1}$ is decreasing; i.e., $E_\gamma \subseteq E_\beta$ if $\gamma \geq \beta$. Then there exists $\alpha < \omega_1$ such that $E_\beta = E_\alpha$ for all $\beta \geq \alpha$.*

A.2 Sequences

Given a set X of elements and an index set Y we indicate sequences of elements from X with the notation $\{x_n\}_{n \in Y}$ where $x_n \in X$ for each $n \in Y$. If the index set is the set of natural numbers, we simply write $\{x_n\}_n$. Instead, if the index set is some ordinal number, we talk about *transfinite sequences*. Given a sequence $\{x_n\}_n$ for some set of elements X we indicate

a subsequence of such sequence with the notation $\{x_{n(u)}\}_u$ where $n : \mathbb{N} \rightarrow \mathbb{N}$ is the function determining the elements of the subsequence considered. We now define *uniformly bounded* sequences of functions:

► **Definition 26** (Uniformly bounded). *Let $I \subset \mathbb{R}$, $E \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$ and let $\{g_n\}_n : I \rightarrow E$ be a sequence of functions. We say that the sequence is uniformly bounded if there exists a constant $K > 0$ such that $\|g_n(t)\| \leq K$ for all $g_n \in \{g_n\}_n$ and for all $t \in I$.*

We then define *equicontinuous* sequences of functions:

► **Definition 27** (Equicontinuous). *Let $I \subset \mathbb{R}$, $E \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$ and let $\{g_n\}_n : I \rightarrow E$ be a sequence of functions. We say that the sequence is equicontinuous if for any $\epsilon > 0$ there exists a $\delta_\epsilon > 0$ such that $\|g_n(t) - g_n(\tilde{t})\| \leq \epsilon$ whenever $|t - \tilde{t}| \leq \delta_\epsilon$ for all $g_n \in \{g_n\}_n$ and for all $t, \tilde{t} \in I$.*

For infinite, uniformly bounded, equicontinuous sequence of functions over the reals there exists a famous result due to Ascoli [8]:

► **Theorem 28** (Ascoli). *Let $I \subset \mathbb{R}$ be a bounded interval, $E \subset \mathbb{R}^r$ for some $r \in \mathbb{N}$ and let $\{g_n\}_n : I \rightarrow E$ be an infinite, uniformly bounded, equicontinuous sequence of functions. Then the sequence $\{g_n\}_n$ has a subsequence $\{g_{n(u)}\}_u$ that converges uniformly on I .*

It follows a theorem concerning differentiability of limits of converging sequences of functions:

► **Theorem 29**. *Let $\{f_n\}_n$ be sequence of functions from the closed interval $[a, b] \subset \mathbb{R}$ to \mathbb{R}^r for some $r \in \mathbb{N}$ and pointwise converging to function f . Let $M > 0$ be such that $\|f_n(\tilde{t}) - f_n(t)\| \leq M |t - \tilde{t}|$ for all $n \in \mathbb{N}$, for all $a \leq t, \tilde{t} \leq b$. Then f is differentiable almost everywhere and $f(x) = \int_a^x f'(t)dt$ for all $x \in [a, b]$.*

A.3 Functions of class Baire one

We define the *set of discontinuity points* of a given function:

► **Definition 30** (Set of discontinuity points). *Let f be a function $f : X \rightarrow Y$ where X and Y are two complete metric spaces. We define the set of discontinuity points (of f on X) as the the set:*

$$D_f = \{x \in X : \exists \epsilon > 0 : \forall \delta > 0 \exists y, z \in B_X(x, \delta) : d_Y(f(y), f(z)) > \epsilon\}$$

We define what it means for a given function to be *of class Baire one*:

► **Definition 31** (Baire one). *Let X, Y be two separable, complete metric spaces. A function $f : X \rightarrow Y$ is of class Baire one if it is a pointwise limit of a sequence of continuous functions, i.e. if there exists a sequence of continuous functions from X to Y , $\{f_m\}_m$, such that $\lim_{m \rightarrow \infty} f_m(x) = f(x)$ for all $x \in X$.*

An important property of functions of class Baire one is that the composition of a function of class Baire one with a continuous functions yields a function of class Baire one [39]. We now refresh a well known topological concept:

► **Definition 32** (Nowhere dense set). *Let X be a topological space and let S be a subset of X . We say that S is nowhere dense (in X) if its closure has empty interior.*

Local Certification of Local Properties: Tight Bounds, Trade-Offs and New Parameters

Nicolas Bousquet ✉ 

Univ Lyon, CNRS, INSA Lyon, UCBL, LIRIS, UMR5205 F-69622 Villeurbanne, France

Laurent Feuilloley ✉ 

Univ Lyon, CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69622 Villeurbanne, France

Sébastien Zeitoun ✉ 

Univ Lyon, CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69622 Villeurbanne, France

Abstract

Local certification is a distributed mechanism enabling the nodes of a network to check the correctness of the current configuration, thanks to small pieces of information called certificates. For many classic global properties, like checking the acyclicity of the network, the optimal size of the certificates depends on the size of the network, n . In this paper, we focus on properties for which the size of the certificates does not depend on n but on other parameters.

We focus on three such important properties and prove tight bounds for all of them. Namely, we prove that the optimal certification size is: $\Theta(\log k)$ for k -colorability (and even exactly $\lceil \log k \rceil$ bits in the anonymous model while previous works had only proved a 2-bit lower bound); $(1/2) \log t + o(\log t)$ for dominating sets at distance t (an unexpected and tighter-than-usual bound); and $\Theta(\log \Delta)$ for perfect matching in graphs of maximum degree Δ (the first non-trivial bound parameterized by Δ). We also prove some surprising upper bounds, for example, certifying the existence of a perfect matching in a planar graph can be done with only two bits. In addition, we explore various specific cases for these properties, in particular improving our understanding of the trade-off between locality of the verification and certificate size.

2012 ACM Subject Classification Theory of computation \rightarrow Distributed algorithms

Keywords and phrases Local certification, local properties, proof-labeling schemes, locally checkable proofs, optimal certification size, colorability, dominating set, perfect matching, fault-tolerance, graph structure

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.21

Related Version *Full Version*: <https://arxiv.org/abs/2312.13702> [5]

Funding This work is supported by the ANR grant GrR (ANR-18-CE40-0032).

Acknowledgements The authors thank anonymous reviewers for useful comments.

1 Introduction

Local certification. Local certification is a topic at the intersection of locality and fault-tolerance in distributed computing. Very roughly, the main concern is to measure how much information the nodes of a network need to know in order to verify that the network satisfies a given property. Local certification originates from self-stabilization and is tightly related to the minimal memory needed to be sure that a self-stabilizing algorithm has reached a correct configuration locally. The topic is now studied independently, and the area has been very active during the last decade. We refer to the survey [14] for an introduction to the topic.

The standard model for local certification is the following. The nodes are first assigned labels, called *certificates*, and then every node looks at its certificate and the certificates of its neighbors, and decides to accept or reject. A certification scheme for a given property is correct if, for any network, the property is satisfied if and only if there exists a certificate



© Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 21; pp. 21:1–21:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



assignment such that all the nodes accept. (We discuss variations later, and give proper definitions in Section 3.) The usual measure of performance of a certification scheme is the maximum certificate size over all nodes, and all networks of a given size.

One of the fundamental results in local certification is that any property can be certified if the network is equipped with unique identifiers [21, 20], but this is at the expense of huge certificates. Indeed, the scheme consists in giving to every node the full map of the graph, which takes $\Theta(n^2)$ bits in n -node graphs. The question then is: when can we do better? There exist three typical certificate sizes. For some properties, *e.g.* related to graph isomorphism, $\Theta(n^2)$ bits is the best we can do [20]. For many natural properties, the optimal certificate size is $\Theta(\log n)$; for example most properties related to trees (acyclicity, spanning tree, BFS, and minimum spanning tree for small edge weights [21]). A recent research direction tries to capture precisely which properties have such *compact certification* (see [4, 19, 18]). Finally, some properties are *local* from the certification point of view, in the sense that the optimal certification size *does not depend on n* . This third type of property is the topic of this paper.

Local certification of local properties

Until recently, studying local properties has not been the focus of the community, since the usual parameter for measuring complexity is the network size. A recent paper by Ardévol Martínez, Caoduro, Feuilloley, Narboni, Pournajafi and Raymond [22] is the first to target this regime. We refer to [22] for the full list of motivations to study this topic, and we just highlight a few points here.

First, it will appear in this paper that the size of the certificates for local properties is often expressed as functions of parameters different from the number of nodes. Therefore, one should not (always) see these are constants. It has been highlighted before (see discussion before Open problem 4 in [14]) that we have basically no understanding of certification size expressed by other parameters than the number of nodes.

Second, *locally checkable languages* (LCL) are at the core of the study of the LOCAL and CONGEST models. These are basically the properties that are local from a certification point of view: the output can be checked by looking at all the balls of some constant radius. Certification is a way to question the encoding of LCLs. A typical example is coloring, for which one uses the colors as output of a construction algorithm, and as certificates for colorability certification. If there would exist a better certification, this would shed a new light on the encoding of this LCL. For example, the celebrated round elimination technique [25] is very sensitive to the problem encoding, and one could hope that feeding it with a different encoding could provide new bounds. In a more general perspective, we argue that just like understanding the complexity of LCLs, understanding certification of local properties is a fundamental topic.

In addition to these two general motivations, our work is guided by two open problems, the k -colorability question and the trade-off conjecture, that we detail now.

The k -colorability question

Let us first discuss the case of colorability, which will be central in this paper. The property we want to certify is that the graph is k -colorable, that is, one can assign colors from $\{1, \dots, k\}$ to vertices such that no two neighbors have the same color. It is straightforward to design a local certification with k certificates for this property: the certificates encode colors in $\{1, \dots, k\}$ and the nodes just have to check that there is no conflict.¹ This uses $O(\log k)$ bits. The natural open problem here is the following.

¹ A *conflict* being an edge whose two endpoints are colored the same.

► **Open problem 1** (Open problem 1 in [14]). *Is $\Theta(\log k)$ optimal for k -colorability certification?*

The first result on that question is the very recent paper [22] which establishes that one bit is not enough to certify k -colorability. This lower bound holds in the anonymous and in the proof-labeling scheme models, that we will define later. In a nutshell, the technique is an indistinguishability argument: assuming that there exists a 1-bit certification, one can argue about the number of 1s in a node neighborhood and take an accepting certification of some k -colorable graph to derive an accepting certification of a $k + 1$ -clique, which is a contradiction.

Interestingly, [22] also shows a case where the natural encoding is not the best certification. Namely, certifying a distance-2 3-coloring can be done with only 1 bit, while the obvious encoding of the colors takes 2 bits. Finally, let us mention that non- k -colorability, the complement property, is much harder to certify. Indeed, it is proved in [20] that one needs $\tilde{\Omega}(n^2)$ bits to certify non-3-colorability (where $\tilde{\Omega}$ hides inverse logarithmic factors).

Trade-off conjecture

We finish this general introduction, with yet another motivation to study local properties. The *trade-off conjecture*, first stated in [15], basically states that for any property, if the optimal certification size is s for the classic certification mechanism, where the nodes see their neighbors certificates, then it is in $O(s/d)$ if the vertices are allowed to see their whole neighborhood at distance d . This was proved to be true for many classic properties, and in many large graph classes [15, 17, 24]. Implicitly, the big-O of the conjecture refers to functions of n , but for local properties, the conjecture is interesting only if it refers to the parameters that appear in the certificate size.

► **Open problem 2** (Trade-off conjecture). *Consider a property with optimal certification size s at distance 1 (where s depends on the natural parameters of the problem). Is it true that if we allow the verification algorithm to look at distance d , then the optimal size is at most $\alpha \cdot s/d$ for some constant α ?*

The authors of [22] argue that the conjecture might actually be wrong for local properties. In other words, there might exist properties such that looking further in the graph is useless (unless you can see the whole graph), or at least not as useful as claimed in [15] (instead of being s/d the optimal size could be a less-decreasing function of d). We consider this question to be very intriguing and important to the study of locality, and we will discuss it several times in the paper.

A sample of local properties

Local properties have different behaviors, which prevented us to establish general theorems capturing all of them. Instead, we looked for a sample of properties widely studied in the distributed community having diverse behaviors, and such that many other properties would behave similarly to one of the sample. First, we chose to study the colorability question, for the reasons cited above. Second we looked at domination at distance t , where a set of nodes is selected, and we want to check that every node is at distance at most t from a selected node. This property has inputs and an external parameter, which makes it very different from colorability. Moreover, a dominating set distance t is a building block for many self-stabilizing algorithms. Third, we study the property of “having a perfect matching”. This differs from

21:4 Local Certification of Local Properties

the two first ones by being an edge-related problem instead of the node-related problem, and it appears that the key parameter there is the maximum degree, a new parameter for certification. One more motivation is that matchings are classic objects in distributed graph algorithms.

Organization of the paper

The paper is organized the following way. After this introduction, we give a detailed overview of the context, results. We also give the proof techniques of the main results. Then, after a definition section, there are technical sections that correspond to the local properties we study. For perfect matchings, the proofs of the results does not appear in this version due to the page limit. All the omitted proofs can be found in the full version [5]. The overview and the technical parts can be read independently. Readers interested in motivations, general discussions, proof ideas and comparison with previous techniques can read the first and cherry-pick specific proofs they are curious about in the second; while others will prefer to go directly to the model section and formal proofs. The overview and the technical part are organized in the same way to allow easy back-and-forth reading.

2 Overview of our results and techniques

Quick description of the models

In order to state the results, we need to informally define the different models of certification (see Section 3 for more formal definitions).

- In the anonymous model, the nodes have no identifiers/port-numbers.
- In the proof-labeling scheme model, every node has a unique identifier (encoded on $O(\log n)$ bits) but it cannot access the identifiers of the other nodes.
- In the locally checkable proof model, nodes have identifiers and can see the identifiers of the other nodes.

The anonymous model is the one for which we have the largest number of results. It is usually less considered in the literature, but argue that for local properties it is the most natural. See the discussion in Subsection 3.2.

The standard assumption is that the nodes can only see their neighbors. Since we are interested in the trade-off conjecture, we will also consider certification at distance d , where the view is the full neighborhood at distance d .

It is often handy to say that the certificates are given by a *prover*, that intuitively tries to convince the nodes that the network is correct (both on correct and incorrect instances).

Now that we are equipped with these notions, we will review our results and techniques, problem by problem.

2.1 Overview for colorability

In this subsection, we consider the *k-colorability* property, already mentioned, which states that the graph is *k*-colorable.

Two lower bounds for colorability

We have already discussed the colorability property, and cited Open problem 1. Our first result is in the anonymous model, where we strongly solve Open problem 1 by determining the *exact bound*: in the anonymous model it is necessary and sufficient to have k different certificates.

► **Theorem 3.** *For every $k \geq 2$, in the anonymous model where vertices can see at distance 1, k certificates are needed in order to certify that a graph is k -colorable. Therefore, in the anonymous model, exactly $\lceil \log(k) \rceil$ bits are needed to certify k -colorability.*

The upper bound is trivial since we can simply give the colors as certificates. The technique to establish this lower bound is a form of crossing technique. We take a large enough complete k -partite graph, thus a graph that is maximally k -colorable, in the sense that any edge added to it would make it non- k -colorable. The idea of the proof is to argue by counting, that for any certificate assignment that would make all nodes accept this graph, there must exist two edges that we can cross (that is replace $(a, b); (x, y)$ by $(a, x); (b, y)$ for example) such that all neighborhoods appearing in this instance did appear in the previous one, thus no node rejects. In addition, the graph obtained after this crossing is non- k -colorable, which is a contradiction with the correctness of the scheme.

Now in the most general model, we also give an (asymptotic) tight bound, fully answering Open question 1. We actually prove a more general lower bound parametrized by the verification distance.

► **Theorem 4.** *In the locally checkable proofs model, at least $\Omega(\log(k)/d)$ bits are needed to certify k -colorability when the vertices are allowed to see their neighborhoods at distance d .*

We describe the proof of this result in a communication complexity framework to provide more intuition, although the actual proof does not rely on any black-box result from communication complexity. The instances we use have the typical shape of communication complexity constructions: the graph has two parts that correspond to the players (left and right) and a part in the middle. The middle part has a large diameter (to be sure that left and right cannot communicate at distance d), and has two sets of k special vertices on the left (top left and bottom left), and two sets of k special vertices on the right (top right and bottom right). The part of the left (resp. right) player is an antimatching between the top left and bottom left (resp. top right and bottom right), where an antimatching is a complete bipartite graph in which a matching has been removed. The middle part has a very constrained structure whose role is to enforce that the graph is k -colorable, if and only if, the left and right antimatchings are intuitively mirrors one of the other. The idea is then that the information about the exact matchings on the left and right parts has to be transferred to some node, to be compared, and this can happen only via the certificates. There are $k!$ possible forms for the left and right antimatchings, (because there are $k!$ possible matching in a complete bipartite graph of size k basically). Thus, the information that has to be transferred from one side of the graph to the other has size $k \log k$ (via Stirling equivalent) and since our graph has cuts of order k , we get the $\Omega(\log k)$ lower bound.

Uniquely k -colorable graphs and other natural counterexample candidates

Our two lower bound constructions have in common to be very constrained, in a precise sense: for every vertex v , every ball centered in v of radius at least 2 admits a unique proper k -coloring (up to color renaming). In this case, we say that the graph is *uniquely k -colorable at distance d* (where d is the radius of the neighborhood). It is easy to see that if a graph is uniquely k -colorable at distance d , then either it has a unique k -coloring or it is not k -colorable. Intuitively, graphs with this property are hard for certification, since there is no slack in the coloring, thus the transfer of information between different parts of the graph cannot a priori be compressed. Perhaps surprisingly, we prove that for these graphs the trade-off conjecture does hold, even in the anonymous model.

► **Theorem 5.** *For every $d \leq \log k$, in the anonymous model where vertices can see at distance d , $O(\log k/d)$ bits are sufficient to certify that a uniquely k -colorable graph at distance $d - 2$ is k -colorable.*

Let us briefly explain the main ingredient of that proof. Since the coloring is locally unique, by looking far enough, a node can decide which other nodes are in the same color class in a k -coloring, if one exists globally. This is not enough to be sure that the graph is k -colorable, because the color classes might not coincide nicely. For example, every cycle is uniquely 2-colorable at distance 1, but this does not certify that the full cycle is 2-colorable. The key idea is that a node will recover the name of its color by gathering the bits of information spread on the nodes *of its own color class* at distance at most d from it. This allows to spread the information of the color classes on several vertices and then use smaller certificates. Slightly more formally, the certifier will assign to a well-chosen subset of nodes X a special certificate. The vertices of X are chosen far enough from each other so that we can store information of the different color classes on the nodes close to it. But they are also chosen not too far away from each other to be sure that all the vertices are close to a vertex of X . Now every node just have to perform the following verification: (i) if it is too far from any vertex of X , it rejects, (ii) it checks that its color is the same for all vertices of X close to it, (iii) it checks that for all the vertices of X close to it, the color associated to it is different from colors given to its neighbors.

After Theorem 5, a question is whether we can go down to a constant number of bits, or in other words, how large the constant of the big-O needs to be. We prove that at a distance $O(\log k)$, *one bit is sufficient* (in other words only two different certificates are needed). Note that it means in particular that we can avoid the special certificate of the proof of Theorem 5 by being very careful on how we represent each vertex of X and the colors classes around it (and proving that even if a node makes a mistake in the choice of X when some vertices are indistinguishable, its decision will be anyway correct).

► **Theorem 6.** *In the anonymous model where vertices can see to distance d , 2 certificates are enough to certify that a uniquely k -colorable graph at distance $d - 2$ is k -colorable, if $d \geq 9\lceil \log_2 k \rceil + 8$.*

The scheme for this is built on the same framework, but with more ideas and technicalities, and we refer to the technical part for the details.

Now that we proved that locally uniquely colorable graphs are not going to help, we wonder what would be a good candidate to disprove the trade-off conjecture (of course one might not exist, if the conjecture is true). We believe that for graphs that are *almost* locally uniquely colorable (that is, they have few correct colorings locally, up to color renaming) the proof of Theorem 5 could be adapted, with more layers of technicalities. Hence, one might go for graphs that have *many* possible colorings. This could make sense from a communication complexity point of view, because there also exist difficult problems with many correct pairs of inputs (*e.g.* the disjointness problem). What is problematic here is that unlike in communication complexity, in our case, on a k -colorable graph, the prover has the choice of the coloring, thus can choose one that is easier to encode compactly.

Since graphs that are very structured seem to admit a linear scaling, another approach could be to consider graphs with a more chaotic structure. Random graphs are natural candidates here, but even if we could come up with a satisfying definition for what it means to certify the colorability of a random graph, it is not clear that this would be hard. Indeed, there exist efficient algorithms to k -color k -colorable graphs (*e.g.* [27]) that exploit only the local structure of the graph, so one could even hope for a verification without certificates.

In the end, graphs with large girth and large chromatic number might be the right type of graphs to consider because they have both a large number of colorings locally and a rigid structure globally. Several constructions for these have been designed (see *e.g.* [13, 23, 10]), but they are randomized or complex, which makes their study rather challenging.

Certification versus output encoding

More generally, it is interesting to explore the links between (1) certifying that a given structure exists (*e.g.* a coloring here, but also a perfect matching a bit later in the paper) (2) explicitly describing it, and (3) implicitly describing it, that is giving enough information to recover the structure, but not directly (*e.g.* with fewer bits than the natural encoding). We refer to [3] for the related topic of distributed zero-knowledge proofs.

For colorability, the case of perfect graphs is a nice example of the discrepancy that can exist between these. A graph is *perfect* if its chromatic number is equal to the size of its maximum clique for every induced subgraph. Many classic graph classes are perfect, *e.g.* bipartite graphs, chordal graphs, comparability graphs (see [26] for a survey on perfect graphs). If we are promised to be in such a graph, the verification (at distance 2) is very easy: a vertex just has to check whether it belongs to a clique of size larger than k (in which case it rejects, otherwise it accepts). Thus, no certificates are needed for colorability certification, while it seems really difficult to have the nodes output a coloring without giving them quite a lot of information. Proving formally such a discrepancy would be a nice result.

Finally, let us mention yet another lower bound approach, related to problem encodings. One can note that if we have certification using s bits for a property, where the local verification algorithm runs in polynomial time in the network size, then we have a *centralized* decision algorithm for this property in time $s^n \text{poly}(n)$. Indeed, one can just enumerate all the possible certificate assignments of this size and check whether one is accepted. One could hope that having a too-good bound for certification would imply that some big conjecture (*e.g.* SETH, the strong exponential hypothesis) is wrong, and get a conditional lower bound.² Unfortunately, this does not help us much for certification, since there are known algorithms for computing the chromatic number of a graph in time $O(c^n)$ with $c < 3$ (*e.g.* in [7]).

2.2 Overview for domination

The *domination at distance t* property applies to graphs with inputs: every node should be labeled with 0 or 1, and every node should be at distance at most t from a node labeled 1. To avoid confusion, let us highlight that in the domination property, the inputs are part of the instance and are different from the certificates.

This problem is quite different from colorability, in several ways. First, it is a problem that is centered on inputs: for any graph there are inputs that are correct, so in some sense it is the inputs that are certified more than the graph itself. (This is actually closer to the original self-stabilizing motivation of certifying the output of an algorithm.) Second, the natural certification has a different flavor: we give every node its distance to the closest node of the dominating set (that is, a node labeled 1), and every node checks that these distances do make sense. One can then consider domination at distance t to be the local analogue of acyclicity, which is certified by providing the nodes of the network with the distance to the root.

² Such bounds are not common in certification, but not unseen, see [11].

Given this analogy with acyclicity, a $\Theta(\log t/d)$ optimal certificate size is expected, and indeed we prove it. But we go one step further by providing precise bounds on the number of different certificates. First, for the anonymous model at distance 1, we prove that the optimal number of different certificates is basically \sqrt{t} .

► **Theorem 7.** *Let $t \in \mathbb{N}^*$. In the anonymous model where vertices can see to distance 1, at least $\sqrt{t-1}$ different certificates are needed to certify a dominating set at distance t , even if the graphs considered are just paths and cycles.*

The lower bound is again quite expected: the proof for this kind of bound is based on arguing whether the same pair of certificates can appear several times on a path or not, thus the square root pops up naturally.

► **Theorem 8.** *In the anonymous model where vertices can see to distance 1, $3 \cdot \lceil \sqrt{t} \rceil$ certificates are sufficient to certify a dominating set at distance t .*

This upper bound is more surprising. It reveals that the natural encoding (consisting of giving the minimum distance to a labeled vertex) is not the best, and that the square root is not an artifact of the lower bound proof but is necessary. The proof of Theorem 8 is based on an elegant argument using *de Bruijn words*. Roughly, for some parameters k and n , a de Bruijn word is a (cyclic) word on an alphabet of size k , which contains all the factors (that is subwords of consecutive letters) of size n exactly once. The idea here is that instead of giving the certificate r to nodes at distance r from a node labeled 1, we will give them the r -th letter in a predefined de Bruijn word that corresponds to $n = 2$ and $k = \sqrt{t}$. Since every node will see its neighbors' certificates, thus a factor of size at least 2, it will be able to decode what is its position in the word, thus its distance to the 1-labeled node. The parametrization $k = \sqrt{t}$ ensures that the de Bruijn word of the correct length exists. Finally, we generalize these bounds to larger distances using generalizations of the techniques described above.

► **Theorem 9.** *In the anonymous model where vertices can see to distance $d < \frac{t}{2}$, at least $\frac{2^d \sqrt{t} - 2d + 1}{2}$ different certificates are needed to certify a dominating set at distance t , even if the graphs considered are just paths and cycles.*

► **Theorem 10.** *In the anonymous model where vertices are allowed to see to distance d , $O(d^{d+1} \sqrt{t})$ certificates are sufficient to certify a dominating set at distance t .*

2.3 Overview for perfect matchings

A graph has the *perfect matching property* if it has a perfect matching, that is, a set of edges such that every vertex belongs to exactly one such edge. It is yet another type of property, that differs from the others by the fact that it has no built-in parameter (no number k of colors, or distance t). As we will see, the relevant parameter here is the maximum degree of the graph.

Perfect matching certification upper bounds

The natural way to locally encode a matching in distributed computing is to make every node know which port number corresponds to a matched edge (if the vertex is matched). In the port-number model, this directly leads to a certification: give the relevant port number

to each node, and let them check the consistency. This takes $O(\log \Delta)$ bits per node, but it requires port numbers, and the ability for the nodes to know the port numbers of their neighbors.

Our first result is that we do not need port-numbers (nor any kind of initial symmetry-breaking). The strategy for the prover is the following. First, choose a perfect matching, and color the matched edges such that there are no two edges (u, v) and (w, z) of the same color with (v, w) being an edge of the graph. We call this a *matching coloring*. Then, give to every node the color of its matched edge. Then every node simply checks that it has exactly one neighbor with the same color.

► **Theorem 11.** *Let $k \in \mathbb{N}^*$. Let \mathcal{C} be a class of graphs such that, for every $G \in \mathcal{C}$, if G has a perfect matching then G has a k -matching coloring. Then, in the anonymous model at distance 1, k certificates are enough to certify the existence of a perfect matching in G for every $G \in \mathcal{C}$.*

We can easily show (see Lemma 28 in the full version [5]) that if G has a perfect matching then it admits a $(2\Delta - 1)$ -matching coloring.

► **Corollary 12.** *For every graph G , $2\Delta - 1$ certificates are enough to certify the existence of a perfect matching, in the anonymous model at distance 1.*

To better appreciate the lower bound of the next paragraph, let us mention a surprising result on the upper bound side. Note first that if we can get a matching coloring using fewer colors, then we automatically get a more compact certification. Now consider a planar graph with an arbitrary perfect matching. We claim that this perfect matching can be colored with only four colors, even though planar graphs can have an arbitrary large maximum degree. Indeed, if we contract the edges of the matching, we still have a planar graph, and we can color this graph with four colors, by the four color theorem. Now undoing the contraction, and giving to the matched edges the color of their contracted vertices, we get a proper matching coloring with only four colors. We also prove a more general result on minor-free and bounded treewidth graphs.

► **Corollary 13.** *In the anonymous model at distance 1:*

- *Only 2 bits are enough to certify the existence of a perfect matching for planar graphs.*
- *Only $O(\log k)$ bits are needed to certify the existence of a perfect matching in K_k -minor-free graphs.*
- *Only $\lceil \log_2(k + 1) \rceil$ bits are needed to certify the existence of a perfect matching in graphs of treewidth at most k .*

Note that, all our upper bounds follow from the existence of k -matching colorings. One can easily remark that, if instead of certifying a perfect matching one is simply interested in representing and certifying a matching, we can also do it with the same method by simply coloring all the unmatched vertices with an additional color. Since all the graph classes we mention in the upper bounds are closed under vertex deletion, our results ensure that the following holds: We can represent and certify matchings with 2Δ certificates for general graphs, 5 certificates for planar graphs and $t + 2$ certificates for graphs of treewidth at most t .

Perfect matching certification lower bounds

In the full version [5], we prove the following lower bound for perfect matching certification.

21:10 Local Certification of Local Properties

► **Theorem 14.** *For every $\Delta \geq 2$, in the anonymous model where vertices can see at distance 1, Δ different certificates are needed to certify the existence of a perfect matching for graphs of maximum degree Δ .*

Before we sketch the proof, let us note that there is no natural candidate for a lower bound: on the one hand, graphs that are sparse are ruled out by Corollary 13, and on the other hand many dense graph classes are known to always have perfect matchings, *e.g.* even cliques, or even random graphs with at least $n \log n$ edges [12].

The key to the proof is the notion of *half-graphs*. Half-graphs are bipartite graphs with vertices u_1, \dots, u_n and v_1, \dots, v_n such that u_i is linked to v_1, \dots, v_i . In such a graph, there exists a *unique* perfect matching, and it uses the edges (u_i, v_i) . Indeed, u_1 must be matched with v_1 , thus u_2 must be matched with v_2 etc. Now, we prove by counting argument that if such a graph is accepted with fewer than Δ certificates, we can create a graph without perfect matching that is also accepted. The idea is to take two copies of this certified graph, and then to carefully remove edges from within these graphs to add edges in between. Again, we refer to the technical section for the details.

Discussion of the parameter Δ

A remarkable feature of the perfect matching property is that the optimal certificate size is completely captured by the maximum degree (if we do not consider restricted graph classes). As far as we know, it is the first time that Δ appears as a natural parameter for local certification. This is interesting, since there are few results that use graph parameters other than the size of the graph to measure certificate size. To our knowledge, the only pure graph parameter that has been used before and that does not appear as a parameter of the problem is the girth, for approximate certification [11].³ It has been highlighted, *e.g.* in [14] (discussion before open problem 5), that developing a theory of parametrized certification is an interesting research direction.

Let us note however that it is maybe not very surprising that the maximum degree appears as a key parameter for matching-related problems, since celebrated papers have proved that it is central to the complexity of such problems in other models of computation, *e.g.* the LOCAL model [1, 6].

3 Model and definitions

3.1 Graphs

All the graphs we consider are finite, simple, and non-oriented. For completeness, let us recall the following classical graph definitions. Let $G = (V, E)$ be a graph, $u, v \in V$, $S \subseteq V$, $i \in \mathbb{N}$. The *distance between u and v* , denoted by $d(u, v)$, is the length (number of edges) of the shortest path from u to v . The *layer at distance i from u* , denoted by $N^i(u)$, is the set of vertices $v \in V$ such that $d(u, v) = i$. The *ball of radius i centered in u* , is $B(u, i) := \bigcup_{0 \leq j \leq i} N^j(u)$. The *closed (resp. open) neighborhood* of u is $N[u] := B(u, 1)$ (resp. $N^1(u)$). We can similarly define $d(u, S)$, $N(S)$ and $N[S]$ when S is a subset of vertices.

³ The maximum edge weight also appears for problems in weighted graphs, *e.g.* minimum spanning tree [21], max-weight matching [20, 8].

3.2 Certification

Let $G = (V, E)$ be a graph, and let C, I be non-empty sets. A *certificate function* of G (with certificates in C) is a mapping $c : V \rightarrow C$. An *identifier assignment* of G (with identifiers in I) is an injective mapping $Id : V \rightarrow I$.

► **Definition 15.** Let c be a certificate function of G and Id be an identifier assignment of G . Let $u \in V, d \in \mathbb{N}^*$. The view of u at distance d consists in all the information available at distance at most d from u , that is:

- the vertex u ;
- the graph with vertex set $B(u, d)$ and the edges $(v_1, v_2) \in E(G)$ such that $\{v_1, v_2\} \cap B(u, d-1) \neq \emptyset$;
- the restriction of c to $B(u, d)$;
- the restriction of Id to $B(u, d)$.

► **Remark 16.** For a vertex u , the subgraph induced by $B(u, d-1)$ is included in the view of u at distance d . However, the subgraph induced by $B(u, d)$ is *not* included in the view of u at distance d in general (because the view of u does not contain the edges between two vertices v_1 and v_2 which are both at distance exactly d from u).

A *verification algorithm (at distance d) in the locally checkable proof model* is a function which takes as input the view (at distance d) of a vertex, and outputs a decision, *accept* or *reject*. For a property \mathcal{P} on graphs, we say that there is a *certification for \mathcal{P}* using k certificates (resp. k bits) if C has size k (resp. 2^k), and if there exists a verification algorithm A such that for every graph G and every identifier assignment Id , G has property \mathcal{P} if and only if there exists a certificate function c such that A accepts for every $v \in V(G)$.

A *verification algorithm in the anonymous model* is defined in the exact same way that a verification algorithm in the locally checkable proof model, but vertices are not equipped with a unique identifier (or equivalently, the output is invariant with respect to the identifier assignment).

In this paper, we do not use the proof-labeling scheme model, where the node has access only its own ID, but it appears in a relevant previous work [22].

Discussion of the anonymous model

Note that the model we chose as our main model is the anonymous one. Indeed, all our results are for this model, except for the lower bound on colorability which we wanted to strengthen to the model with identifiers in order to fully solve Open Problem 1.

We think that the natural model for local properties is anonymous. Indeed, the main reason why identifiers are common in local certification is that they are often necessary, which is not the case for local properties. For example, certifying tree-like structures requires certifying that there is a unique connected component, and for this identifiers are needed. Note that in the LOCAL model, identifiers are also used to break symmetry, but in certification, the certificates can do this.

Moreover, the anonymous model is very common in the self-stabilizing literature (see *e.g.* [16, 2, 9]) which is the origin of local certification.

Third, in the paper, we draw a parallel between certification of local properties and LCLs, and the identifiers do not appear in the definition of LCLs.

Finally, in the cases known in the literature where anonymous and “with-ID” bounds match (e.g. acyclicity), the proofs are similar in spirit, except that the ID case involves more counting arguments (usually assuming that the ID interval is not of linear size), which implies loosing constants everywhere, getting results that are less crisp, and proofs that are more obfuscated.

4 Colorability certification

In this section, we will consider the certification of the k -colorability property. Let us recall that a graph G is said to be k -colorable if there exists *proper k -coloring* of G , that is, is a mapping $\varphi : V \rightarrow \{1, \dots, k\}$ such that for all $(u, v) \in E$, $\varphi(u) \neq \varphi(v)$.

For completeness, let us start by proving the following simple upper bound.

► **Proposition 17.** *In the anonymous model where vertices can see at distance 1, k -colorability can be certified with $\lceil \log k \rceil$ bits.*

Proof. For a graph $G = (V, E)$ which is k -colorable, the certificate function given by the prover is the following. The prover chooses a proper k -coloring φ of G , and assigns certificate $c(u) := \varphi(u)$ to every $u \in V$. The verification algorithm of every vertex u consists in checking if for every neighbor v , $c(u) \neq c(v)$. If it is the case, u accepts. Otherwise, u rejects. It is clear that the graph is accepted if and only if it is k -colorable. ◀

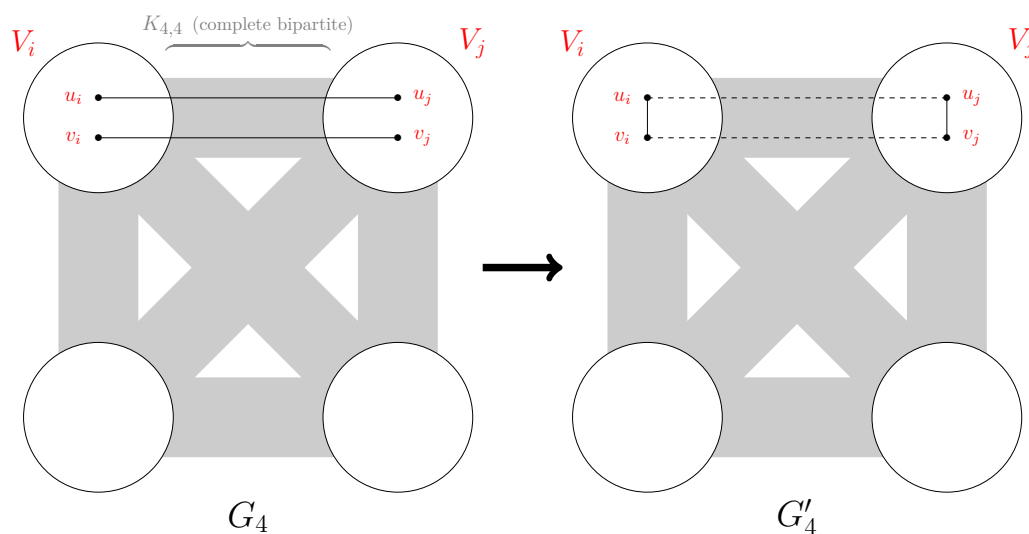
Lower bounds

► **Theorem 3.** *For every $k \geq 2$, in the anonymous model where vertices can see at distance 1, k certificates are needed in order to certify that a graph is k -colorable. Therefore, in the anonymous model, exactly $\lceil \log(k) \rceil$ bits are needed to certify k -colorability.*

Proof. Assume by contradiction that there exists a certification of k -colorability in the anonymous model using only $k - 1$ different certificates. The idea of the proof is to consider a specific k -colorable graph G_k , for which there must exist an accepting certification function. We will prove that we can flip (i.e. cross) two edges of G_k such that the resulting graph G'_k is not k -colorable and no vertex is able to detect it (meaning that the local view of each vertex will be unchanged). Thus, in the new graph G'_k , each vertex accepts, which is a contradiction since G'_k is not k -colorable.

Let us denote by G_k the complete k -partite graph, where each set has size $\max(k, 3)$. More formally, let V_1, \dots, V_k be k disjoint sets, each of size $\max(k, 3)$. Let G_k be the graph with vertex set $V = \bigcup_{i=1}^k V_i$ where (u, v) is an edge if and only if u and v do not belong to the same set V_i . Since $\{V_1, \dots, V_k\}$ is a partition of V into k independent sets, G_k is k -colorable. Thus, by hypothesis, there exists a certificate function $c : V \rightarrow \{1, \dots, k-1\}$ such that every vertex accepts. By the pigeonhole principle, for every $i \in \{1, \dots, k\}$ there exist two different vertices $u_i, v_i \in V_i$ such that $c(u_i) = c(v_i)$ (since each set has size at least k). Again, by the pigeonhole principle, there exist $i \neq j$ such that $c(u_i) = c(u_j)$. Thus, we get $c(u_i) = c(v_i) = c(u_j) = c(v_j)$, with $u_i, v_i \in V_i$ and $u_j, v_j \in V_j$. Let G'_k be the graph obtained from G_k by removing the two edges $(u_i, u_j), (v_i, v_j)$ and adding the two edges $(u_i, v_i), (u_j, v_j)$, as depicted on Figure 1.

We claim that, with the same certificate function c , all the vertices of G'_k accept. Indeed, the only vertices whose neighborhood have been modified between G_k and G'_k are u_i, v_i, u_j, v_j . So every vertex in $V \setminus \{u_i, v_i, u_j, v_j\}$ accepts. For u_i , the only difference between its



■ **Figure 1** Our constructions for the proof of Theorem 3, in the case of $k = 4$. The gray strips indicate complete bipartite graphs. The edges we are interested in appear explicitly.

neighborhood in G_k and G'_k is that u_j is replaced by v_i . Since $c(u_j) = c(v_i)$, the view of u_i is the same in G_k and G'_k , so u_i accepts in G'_k as well. Similarly, one can prove that v_i , u_j , and v_j accept in G'_k .

However, G'_k is not k -colorable. Indeed, assume by contradiction that it is, and let φ be a proper k -coloring of G'_k . For any $r \notin \{i, j\}$, let $w_r \in V_r$. Let w_j be a vertex of $V_j \setminus \{u_j, v_j\}$ which exists, since each set contains at least 3 vertices. Then, $K = \{w_1, \dots, w_{i-1}, w_{i+1}, \dots, w_k\}$ is a clique in G'_k , so the $(k - 1)$ vertices of K receive pairwise different colors in φ . Moreover, both u_i and v_i are complete to K . So if G'_k is k -colorable, then u_i and v_i have to be colored the same, which is a contradiction since $(u_i, v_i) \in E(G'_k)$. ◀

Obtaining lower bounds is usually harder in the locally checkable proofs model. In this more demanding model, we do not get the exact bound in terms of number certificates, but still we get a bound that would be considered optimal by the usual standards. Namely, we will prove that $\Omega(\log k)$ bits are needed to certify the k -colorability of a graph. More generally, we prove that, if vertices can see their neighborhoods at distance d , at least $\Omega(\log k/d)$ bits are needed to certify k -colorability. Note that the graph constructed in the proof of Theorem 3 has diameter two, thus no lower bound at distance at least 3 can be obtained from that construction. The lower bound of the following theorem is obtained with a completely different graph.

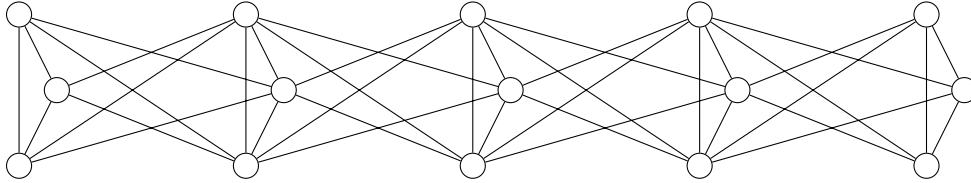
► **Theorem 4.** *In the locally checkable proofs model, at least $\Omega(\log(k)/d)$ bits are needed to certify k -colorability when the vertices are allowed to see their neighborhoods at distance d .*

Proof. For $r, s \geq 2$, let us denote by $P_{r,s}$ the graph on vertex set $V = \{1, \dots, r\} \times \{1, \dots, s\}$ with the following edges:

- for all $i \neq j \in \{1, \dots, r\}$ and all $p \in \{1, \dots, s\}$, $((i, p), (j, p)) \in E$
- for all $i \neq j \in \{1, \dots, r\}$ and all $p \in \{1, \dots, s - 1\}$, $((i, p), (j, p + 1)) \in E$

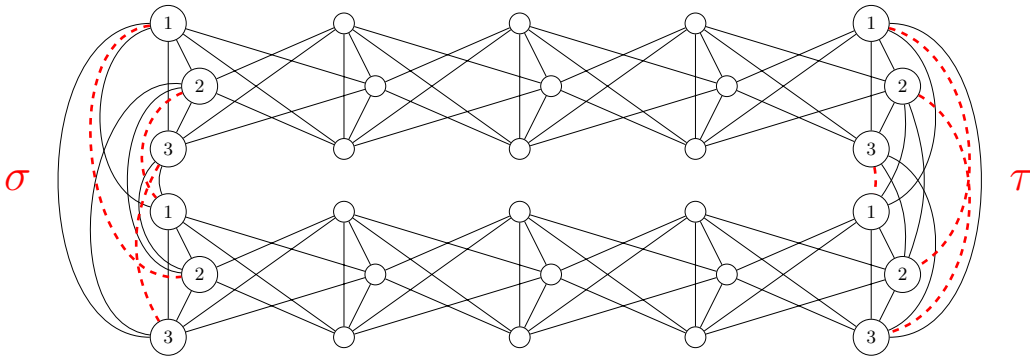
21:14 Local Certification of Local Properties

In other words, the graph $P_{r,s}$ is a sequence of s cliques $\mathcal{K}_1, \dots, \mathcal{K}_s$, each having size r , with an antimatching⁴ between \mathcal{K}_i and \mathcal{K}_{i+1} for all $i \leq s-1$. The second coordinate indicates the index of the clique the vertex belongs to, and for all (i, p) with $p \leq s-1$, (i, p) is the only vertex in \mathcal{K}_p which is not adjacent to $(i, p+1)$. For instance, the graph $P_{3,5}$ is depicted on Figure 2.



■ **Figure 2** The graph $P_{3,5}$. Here the cliques are triangles, and two consecutive cliques are linked by all possible edges except for the horizontal ones (the antimatching).

Now, for any pair of permutations σ, τ of $\{1, \dots, r\}$, let us denote by $G_{r,s}(\sigma, \tau)$ the graph obtained in the following way. We take two copies of $P_{r,s}$, and we denote their cliques by $\mathcal{K}_1, \dots, \mathcal{K}_s$ and $\mathcal{K}'_1, \dots, \mathcal{K}'_s$. We give $2rs$ different identifiers to the vertices of the two copies (the identifiers are fixed, they do not depend on σ and τ). Finally, we add the antimatching $\{(i, \sigma(i))\}_i$ between \mathcal{K}_1 and \mathcal{K}'_1 , and the antimatching $\{(i, \tau(i))\}_i$ between \mathcal{K}_s and \mathcal{K}'_s (see Figure 3 for an illustration).



■ **Figure 3** The graph $G_{3,5}(\sigma, \tau)$, where σ is the permutation $(1, 2)$ and τ the cycle $(1, 2, 3)$.

▷ **Claim 18.** The graph $G_{r,s}(\sigma, \tau)$ is r -colorable if and only if $\sigma = \tau$.

Proof. Since \mathcal{K}_1 is a clique of size r , in a proper coloring with r colors, every color appears exactly once inside \mathcal{K}_1 . Similarly, every color appears exactly once in a proper coloring of \mathcal{K}_2 . For all $i \in \{1, \dots, r\}$, the vertex $(i, 1)$ is a neighbor of every vertex in \mathcal{K}_2 except $(i, 2)$. Thus, the vertices $(i, 1)$ and $(i, 2)$ are colored the same. Hence, given a coloring of \mathcal{K}_1 , there exists a unique way to properly color \mathcal{K}_2 . This propagates along the cycle of cliques $\mathcal{K}_2, \dots, \mathcal{K}_s, \mathcal{K}'_s, \dots, \mathcal{K}'_1$, and it leads to a proper coloring of the whole graph if and only if the coloring of \mathcal{K}'_1 is compatible with the coloring of \mathcal{K}_1 , that is, if and only if $\sigma = \tau$. ◁

⁴ Remember that an antimatching between two sets of nodes is an edge set containing all the possible edges between the two sets, except for a matching.

Using Claim 18, we will deduce a lower bound on the number of bits needed to certify k -colorability, when vertices are allowed to see at distance d . We consider the construction above with parameters $r = k$ and $s = 2d$. Assume that m bits are sufficient. Then, for any permutation σ , there exists a certificate function $c_\sigma : V \rightarrow \{0, \dots, 2^m - 1\}$ such that all the vertices of $G_{k,2d}(\sigma, \sigma)$ accept. Suppose that there are two different permutations σ, τ such that the certification functions are the same: $c_\sigma = c_\tau$, that is, all vertices receive the same certificate in both instances. Then $G_{k,2d}(\sigma, \tau)$ would be accepted with this certificate function, since the view of every vertex would be identical as its view in $G_{k,2d}(\sigma, \sigma)$ (for vertices in the left part) or in $G_{k,2d}(\tau, \tau)$ (for vertices in the right part). This would be a contradiction because $G_{k,2d}(\sigma, \tau)$ is not k -colorable (by Claim 18).

Hence, all the certificate functions c_σ are different. In particular, there are no more permutations of $\{1, \dots, k\}$ than functions $V \rightarrow \{0, \dots, 2^m - 1\}$. The set V has size $4kd$ since it is made of two copies of a graph of size $k \times 2d$. Therefore, we get $k! \leq 2^{4mkd}$, leading to $m \geq \frac{\log_2(k!)}{4kd}$. Finally, since $\log_2(k!) = \Omega(k \log k)$, we get the result. \blacktriangleleft

► **Remark 19.** Two ingredients of the proof, the communication complexity insight and the antimatchings to propagate the coloring, have already been used in [20], to get a lower bound on the certification of not-3-colorable graphs.

5 Dominating sets at distance t

Most of the results concerning dominating sets at distance t can be found in the full version [5]. We simply give a proof of Theorem 8 with an elegant argument using de Bruijn words:

► **Theorem 8.** *In the anonymous model where vertices can see to distance 1, $3 \cdot \lceil \sqrt{t} \rceil$ certificates are sufficient to certify a dominating set at distance t .*

Before proving Theorem 8, we will define a few notions. Let A be an alphabet, and $\omega, \omega' \in A^*$. We say that ω is a *factor* of ω' if there exists a sequence of consecutive letters in ω' which is equal to ω . Let us now define *de Bruijn words*, whose existence is well-known.

► **Proposition 20.** *Let $k, n \in \mathbb{N}^*$, and A be an alphabet of size k . There exists a word $\omega \in A^*$ of length k^n , such that every word of A^n appears at most once as a factor of ω . Such a word ω is called a (k, n) -de Bruijn word.⁵*

Proof of Theorem 8. Let $\tau = \lceil \sqrt{t} \rceil$. Let us prove that 3τ certificates are sufficient to certify a dominating set at distance t (in the anonymous model, where vertices can see at distance 1). Let $A := \{1, \dots, \tau\}$. The certificates used in the scheme will be pairs in $C := \{0, 1, 2\} \times A$. For $(x, y) \in C$, let $\pi_1(x, y) := x$ and $\pi_2(x, y) := y$. Let $\omega' \in A^*$ be a $(\tau, 2)$ -de Bruijn word (which, by definition, has length at least t), and let us denote by $\omega = \omega_1 \dots \omega_t$ its prefix of length exactly t .

Let $G = (V, E)$ be a graph and $S \subseteq V$ be the set of labeled vertices. If S is dominating at distance t , the certificate function given by the prover is the following. The vertices of S are given an arbitrary certificate, and for every $u \in V \setminus S$ at distance i from S , the prover gives to u the certificate $c(u)$ which is $(i \bmod 3, \omega_i)$.

⁵ Usually, de Bruijn words are defined as words of length k^n such that each word of A^n appears at least once when ω is considered circularly. But due to the length of ω , each word of A^n actually appears exactly once circularly, so at most once if ω is not seen as a circular word.

21:16 Local Certification of Local Properties

The informal idea of the verification is the following one. In its certificate, every vertex u is given a letter of ω . By looking at its neighbors, u will be able to determine its position in ω (since a pair of letters defines a unique position in the de Bruijn word ω), which corresponds to its distance to S .

More formally, let c be a certificate function. Each vertex u checks the certificate as follows :

- (i) If $N[u] \cap S \neq \emptyset$, then u accepts.⁶
- (ii) Else, u checks that, for all $u', u'' \in N[u]$, if $\pi_1(c(u')) = \pi_1(c(u''))$ then $\pi_2(c(u')) = \pi_2(c(u''))$. If it is not the case, u rejects.
- (iii) Then, u checks if it has at least one neighbor v such that $\pi_1(c(v)) = \pi_1(c(u)) - 1 \pmod 3$, and if $\pi_2(c(v))\pi_2(c(u))$ is a factor of ω . If it is not the case, u rejects.
- (iv) Finally, for every neighbor w such that $\pi_1(c(w)) = \pi_1(c(u)) + 1 \pmod 3$, u checks if ω has $\pi_2(c(v))\pi_2(c(u))\pi_2(c(w))$ as a factor. If it is not the case, then u rejects.
- (v) If u did not reject at this point, it accepts.

It remains to show that there exists a certificate function such that all the vertices of G accept if and only if S is dominating at distance t . If S is a dominating set at distance t , then one can easily check that all the vertices accept with the certificates assigned by the prover as described previously. Note that with this certificate function, for step (ii), if $u', u'' \in N[u]$ satisfy $\pi_1(c(u')) = \pi_1(c(u''))$, then $d(u', S) = d(u'', S)$ so $\pi_2(c(u')) = \pi_2(c(u''))$.

For the converse, assume that G is accepted with some certificate function c . By (iii), every $u \in V$ such that no vertex is labeled in $N[u]$ should have a neighbor v such that $\pi_1(c(v)) = \pi_1(c(u)) - 1 \pmod 3$ and $\pi_2(c(v))\pi_2(c(u)) = \omega_\ell \omega_{\ell+1}$ for some ℓ . Note that since ω is a de Bruijn word, this ℓ is unique. Let us prove by induction on $\ell \in \{1, \dots, t-1\}$ that $d(u, S) \leq \ell + 1$.

- For $\ell = 1$, we have $\pi_2(c(v))\pi_2(c(u)) = \omega_1 \omega_2$. Let us prove that $d(u, S) \leq 2$. It is sufficient to prove that $d(v, S) \leq 1$. If v accepts at step (i), the conclusion holds. So we can assume that v accepts at step (v). By (iii), v has a neighbor v' such that $\pi_1(c(v')) = \pi_1(c(v)) - 1 \pmod 3$ and $\pi_2(c(v'))\pi_2(c(v))$ is a factor of ω . Since v does not reject at step (iv), $\pi_2(c(v'))\pi_2(c(v))\pi_2(c(u))$ is a factor of ω . So the letters $\omega_1 \omega_2 = \pi_2(c(v))\pi_2(c(u))$ appear at least twice as a factor of ω , which is a contradiction with the definition of de Bruijn words. Indeed, $\pi_2(c(v))\pi_2(c(u))$ are already the first two letters of ω . So if a factor $\pi_2(c(v'))\omega_1 \omega_2$ appears somewhere, $\omega_1 \omega_2$ must appear at least twice. Hence, v accepts at step (i), so $d(v, S) \leq 1$.
- Assume now that $\ell \geq 2$. To prove that $d(u, S) \leq \ell + 1$, it is sufficient to prove that $d(v, S) \leq \ell$. If v accepts at step (i) the conclusion holds. So we can assume that v accepts at step (v). By (iii), v has a neighbor v' such that $\pi_1(c(v')) = \pi_1(c(v)) - 1 \pmod 3$ and $\pi_2(c(v'))\pi_2(c(v))$ is a factor of ω . Since v does not reject at step (iv), $\pi_2(c(v'))\pi_2(c(v))\pi_2(c(u))$ is a factor of ω . Since the factor $\pi_2(c(v))\pi_2(c(u))$ appears exactly once in ω and $\pi_2(c(v))\pi_2(c(u)) = \omega_\ell \omega_{\ell+1}$, we have $\pi_2(c(v'))\pi_2(c(v)) = \omega_{\ell-1} \omega_\ell$ by (iv). By induction hypothesis, it implies that $d(v, S) \leq \ell$.

Thus, for any vertex $u \in V$, if u accepts at step (i) we have $d(u, S) \leq 1$, otherwise we have $d(u, S) \leq t$ by the previous induction. So S is indeed a dominating set at distance t . ◀

⁶ We could verify that $\pi_2(c(u))$ corresponds to a letter in the beginning of ω , but it is not necessary.

References

- 1 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *J. ACM*, 68(5):39:1–39:30, 2021. doi:10.1145/3461458.
- 2 Samuel Bernard, Stéphane Devismes, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal deterministic self-stabilizing vertex coloring in unidirectional anonymous networks. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009*, pages 1–8, 2009. doi:10.1109/IPDPS.2009.5161053.
- 3 Aviv Bick, Gillat Kol, and Rotem Oshman. Distributed zero-knowledge proofs over networks. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 2426–2458. SIAM, 2022. doi:10.1137/1.9781611977073.97.
- 4 Nicolas Bousquet, Laurent Feuilloley, and Théo Pierron. What can be certified compactly? compact local certification of MSO properties in tree-like graphs. In *PODC '22: ACM Symposium on Principles of Distributed Computing*, pages 131–140, 2022. doi:10.1145/3519270.3538416.
- 5 Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun. Local certification of local properties: tight bounds, trade-offs and new parameters, 2023. arXiv:2312.13702.
- 6 Sebastian Brandt and Dennis Olivetti. Truly tight-in- Δ bounds for bipartite maximal matching and variants. In *PODC '20: ACM Symposium on Principles of Distributed Computing*, pages 69–78, 2020. doi:10.1145/3382734.3405745.
- 7 Jesper Makhholm Byskov. Enumerating maximal independent sets with applications to graph colouring. *Oper. Res. Lett.*, 32(6):547–556, 2004. doi:10.1016/j.orl.2004.03.002.
- 8 Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. *Theor. Comput. Sci.*, 811:112–124, 2020. doi:10.1016/j.tcs.2018.08.020.
- 9 Johanne Cohen, Jonas Lefèvre, Khaled Maâmra, Laurence Pilard, and Devan Sohier. A self-stabilizing algorithm for maximal matching in anonymous networks. *Parallel Process. Lett.*, 26(4):1650016:1–1650016:17, 2016. doi:10.1142/S012962641650016X.
- 10 Amin Coja-Oghlan, Charilaos Efthymiou, and Samuel Hetterich. On the chromatic number of random regular graphs. *J. Comb. Theory, Ser. B*, 116:367–439, 2016. doi:10.1016/j.jctb.2015.09.006.
- 11 Yuval Emek and Yuval Gil. Twenty-two new approximate proof labeling schemes. In *34th International Symposium on Distributed Computing, DISC 2020*, volume 179, pages 20:1–20:14, 2020. doi:10.4230/LIPIcs.DISC.2020.20.
- 12 P Erdős and A Rényi. On the existence of a factor of degree one of a connected random graph. *Acta Mathematica Hungarica*, 17(3-4):359–368, 1966.
- 13 Paul Erdős. Graph theory and probability. *Canadian Journal of Mathematics*, 11:34–38, 1959.
- 14 Laurent Feuilloley. Introduction to local certification. *Discret. Math. Theor. Comput. Sci.*, 23(3), 2021. doi:10.46298/dmtcs.6280.
- 15 Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs. *Distributed Comput.*, 34(2):113–132, 2021. doi:10.1007/s00446-020-00386-z.
- 16 Michael J. Fischer and Hong Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In Alexander A. Shvartsman, editor, *Principles of Distributed Systems, 10th International Conference, OPODIS 2006*, volume 4305, pages 395–409, 2006. doi:10.1007/11945529_28.
- 17 Orr Fischer, Rotem Oshman, and Dana Shamir. Explicit space-time tradeoffs for proof labeling schemes in graphs with small separators. In *25th International Conference on Principles of Distributed Systems, OPODIS 2021*, volume 217 of *LIPIcs*, pages 21:1–21:22, 2021. doi:10.4230/LIPIcs.OPODIS.2021.21.
- 18 Pierre Fraigniaud, Frédéric Mazoit, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. Distributed certification for classes of dense graphs. *CoRR*, abs/2307.14292, 2023. doi:10.48550/arXiv.2307.14292.

- 19 Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. A meta-theorem for distributed certification. In *Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022*, volume 13298, pages 116–134, 2022. doi:10.1007/978-3-031-09993-9_7.
- 20 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory Comput.*, 12(1):1–33, 2016. doi:10.4086/toc.2016.v012a019.
- 21 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Comput.*, 22(4):215–233, 2010. doi:10.1007/s00446-010-0095-3.
- 22 Virginia Ardévol Martínez, Marco Caoduro, Laurent Feuilloley, Jonathan Narboni, Pegah Pournajafi, and Jean-Florent Raymond. Lower bound for constant-size local certification. In *Stabilization, Safety, and Security of Distributed Systems - 24th International Symposium, SSS 2022*, volume 13751, pages 239–253, 2022. doi:10.1007/978-3-031-21017-4_16.
- 23 Moshe Morgenstern. Existence and explicit constructions of $q + 1$ regular ramanujan graphs for every prime power q . *J. Comb. Theory, Ser. B*, 62(1):44–62, 1994. doi:10.1006/jctb.1994.1054.
- 24 Rafail Ostrovsky, Mor Perry, and Will Rosenbaum. Space-time tradeoffs for distributed verification. In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017*, volume 10641, pages 53–70, 2017. doi:10.1007/978-3-319-72050-0_4.
- 25 Jukka Suomela. Using round elimination to understand locality. *SIGACT News*, 51(3):63–81, 2020. doi:10.1145/3427361.3427374.
- 26 Nicolas Trotignon. Perfect graphs: a survey. *CoRR*, abs/1301.5149, 2013. arXiv:1301.5149.
- 27 Jonathan S. Turner. Almost all k -colorable graphs are easy to color. *J. Algorithms*, 9(1):63–82, 1988. doi:10.1016/0196-6774(88)90005-3.

Spectral Approach to the Communication Complexity of Multi-Party Key Agreement

Geoffroy Caillat-Grenier ✉ 

LIRMM, University of Montpellier, CNRS, Montpellier, France

Andrei Romashchenko ✉ 

LIRMM, University of Montpellier, CNRS, Montpellier, France

Abstract

We propose a linear algebraic method, rooted in the spectral properties of graphs, that can be used to prove lower bounds in communication complexity. Our proof technique effectively marries spectral bounds with information-theoretic inequalities. The key insight is the observation that, in specific settings, even when data sets X and Y are closely correlated and have high mutual information, the owner of X cannot convey a reasonably short message that maintains substantial mutual information with Y . In essence, from the perspective of the owner of Y , any sufficiently brief message $m = m(X)$ would appear nearly indistinguishable from a random bit sequence.

We employ this argument in several problems of communication complexity. Our main result concerns cryptographic protocols. We establish a lower bound for communication complexity of multiparty secret key agreement with unconditional, i.e., information-theoretic security. Specifically, for one-round protocols (simultaneous messages model) of secret key agreement with three participants we obtain an asymptotically tight lower bound. This bound implies optimality of the previously known *omniscience* communication protocol (this result applies to a non-interactive secret key agreement with three parties and input data sets with an arbitrary symmetric information profile).

We consider communication problems in one-shot scenarios when the parties inputs are not produced by any i.i.d. sources, and there are no ergodicity assumptions on the input data. In this setting, we found it natural to present our results using the framework of Kolmogorov complexity.

2012 ACM Subject Classification Mathematics of computing → Information theory; Theory of computation → Communication complexity; Security and privacy → Information-theoretic techniques; Theory of computation → Expander graphs and randomness extractors

Keywords and phrases communication complexity, Kolmogorov complexity, information-theoretic cryptography, multiparty secret key agreement, expander mixing lemma, information inequalities

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.22

Related Version *Full Version:* <https://arxiv.org/abs/2305.01355>

Funding supported in part by the ANR project FLITTLA ANR-21-CE48-0023.

Acknowledgements We thank the anonymous referees for valuable and helpful comments.

1 Introduction

Within computer science, a broad range of communication complexity problems has been studied in recent decades. In these problems several (two or more) agents solve together some task (compute a function, search an elements in a set, sample a distribution, and so on) when the input data are distributed among the agents. In different context we may impose different constraints on the class of admissible protocols (protocols can be deterministic or randomized, one-way or interactive, with a one shot of simultaneous messages or with several rounds, etc.). The cost of a communication protocol is the total number of bits that must be exchanged between participants, typically in the worst-case situation.



© Geoffroy Caillat-Grenier and Andrei Romashchenko;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 22; pp. 22:1–22:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



In this paper we focus on communication problems with three parties (Alice, Bob, and Charlie), though our techniques can be extended to bigger number of participants. We deal with the situation when the input data accessible to Alice, Bob, and Charlie are correlated. In a popular model *number-on-forehead*, the datasets given to Alice, Bob, and Charlie have large intersections, which is a very particular form of correlation between the data. We study a more general setting (more usual in cryptography and information theory) where the input data sets given to the parties have large mutual information, but it might be impossible to materialize this mutual information as common chunks of bits shared by several parties.

The principal communication problem under consideration is *secret key agreement*: Alice, Bob, and Charlie use the correlation between their input data sets to produce a common secret key. A special feature of this setting is the implicit presence of another participant in the game, Eve (eavesdropper/adversary). The eavesdropper can intercept all messages between Alice, Bob, and Charlie, but this should not give Eve any information about the final result of the protocol – the produced secret key. A secret key agreement (for two or many participants) is one of the basic primitives in cryptography; it can serve as a part of more sophisticated protocols (the produced secret key can be used in a one-time pad encryption or in more complicated cryptographic schemes).

In practice, the most standard and well known method of secret key agreement is the Diffie-Hellman key exchange [8, 22] and its generalizations, see [27]. The security of this protocol is based on assumptions of computational complexity. In particular, the Diffie-Hellman scheme is secure only if the eavesdropper cannot solve efficiently the problem of discrete logarithms. Such an assumption looks plausible for most practical applications. However, theoretical cryptography studies also secret key agreement in information-theoretic settings, where we impose no restrictions on the computational power of the eavesdropper. Besides a natural theoretical interest, such a scheme can be useful as a building block in more complex protocols. In particular, a protocol of information-theoretic secret key agreement (pretty conventional, involving communication and computational tools conceivable in the framework of the classical physics) is an indispensable component of the protocol of quantum key distribution ([4, 7, 16]). Besides quantum cryptography, secret-key agreement based on correlated information appears in various cryptographic schemes connected with noisy data (biometric information, observations of an inherently noisy communication channel or other physical phenomenon, see the discussions in [17, 10]), in the bounded-storage model ([9, 11]), and so on. We refer the reader to the survey [5] for a more detailed discussion.

In the Diffie-Hellman scheme, the parties may start the protocol from zero, holding initially no secret information. In contrast, a secret key agreement with information-theoretic secrecy is impossible if the parties start from scratch. To produce a key that is secret in information-theoretic sense, the participants of the protocol need to be given some input data (inaccessible to the eavesdropper). The pieces of input data provided to the parties must be correlated with each other, and the measure of this correlation determines the optimal size of the common secret key that can be produced.

So far we were very informal and did not specify the mathematical definitions behind the words *secrecy* (of the key) and *correlation* (between parties' inputs). Let us describe the settings of information-theoretic secret key agreement more precisely. This can be done in different mathematical frameworks.

Historically, information-theoretically secure protocols of secret key agreement were introduced in classical information theory, [1, 21]. In this setting, the input data of the parties are produced by correlated random variables. In the settings with two parties it is

usually assumed that there is a sequence of i.i.d. pairs of random variables with finite range, (X_i, Y_i) , $i = 1, \dots, n$, and Alice and Bob receive the values of $(X_1 \dots X_n)$ and $(Y_1 \dots Y_n)$ respectively. Then Alice and Bob run a communication protocol and try to produce a common value (secret key) W asymptotically independent of the *transcript* (the transcript consist of the messages sent by Alice and Bob to each other). Ahlswede–Csiszar [1] and Maurer [21] found a characterization of the optimal size of W in terms of Shannon’s entropy of the input data. They showed that the optimal size of the secret key is asymptotically equal to the mutual information between Alice’s and Bob’s inputs. A similar characterization of the optimal secret key is known for multi-party protocols, with $k \geq 3$ parties, [6]. The problem of secret key agreement and a related problem of *common randomness generation* were extensively studied in the information theory community and also (in somewhat different settings) in theoretical computer science, see, e.g., [28, 13] and the survey [29].

In this paper we follow the paradigm of building the foundations of cryptography in the framework of algorithmic information theory, as suggested in a general form in [2] and more specifically for secret key agreement in [24, 15]. In this approach, the information-theoretic characteristics of the data are defined not in terms of Shannon’s entropy but in terms of Kolmogorov complexity. In this setting, we can talk about properties of *individual* inputs, keys, transcripts, and not about *probability distributions*. We assume that the parties (Alice, Bob, Charlie) are given as inputs binary strings x, y, z respectively, and that the parties know the complexity profile of these strings, i.e., the optimal compression rate of these inputs (precisely or at least approximately, see below). The secrecy of the produced key means that this key must be incompressible, even conditional on the public data including the transcript of the communication protocol. In other words, the mutual information (in the sense of Kolmogorov complexity) between the key and the messages sent via the communication channel (the transcript) must be negligibly small. Practically, this property guarantees that the adversary can crack an encryption scheme based on this key only by the brute-force search, see the discussion in [15].

► **Remark 1.1.** The approach based on Kolmogorov complexity seems more general since we do not need to assume that inputs have any property of stationarity or ergodicity, we do not fix in advance the probability distribution of the pairs of inputs, we do not even assume the existence of such a distribution. However, the frameworks of Shannon and Kolmogorov for the definition of secrecy have similar practical interpretations. Indeed, a distribution W on $\{0, 1\}^n$ has a high entropy, i.e., $H(W) \approx n$, if and only if with a high probability W returns an n -bit string with Kolmogorov complexity close to n . For a more detailed discussion of the connection between Shannon’s and Kolmogorov’s formalism see [14]. The formal statements in Kolmogorov’s framework are usually stronger than their homologues in Shannon’s framework, and theorems from the former theory in most cases formally imply the corresponding results from the latter theory, see [24]. ◻

A characterization of the optimal size of the secret key in term of Kolmogorov complexity was suggested in [24]. We begin with the case of two parties, see Theorem 1.2 below. In this theorem, a communication protocol is randomized (we assume that the parties may use a public source of random bits, which is also accessible to the eavesdropper). Let x and y stand for inputs of Alice and Bob, r denote the string of bits produced by a public source of randomness (used by the parties and accessible to the eavesdropper), and t denote the transcript of the protocol.

► **Theorem 1.2** ([24]).

- (i) For any numbers $k, \ell \in \mathbb{N}$ and $\epsilon, \delta > 0$ there exist a randomized communication protocols $\pi_{k, \ell, \epsilon, \delta}$ such that on every pair of input strings (x, y) (of length at most n) satisfying¹ $C(x) \stackrel{\delta}{=} k$ and $C(x | y) \stackrel{\delta}{=} \ell$, Alice and Bob with probability $1 - \epsilon$ both obtain a result $w = w(x, y, r)$ such that

$$[\text{length of } w \text{ in bits}] = C(x) - C(x | y) - O(\delta) - o(n) \text{ and } C(w | \langle t, r \rangle) \geq |w| - o(n) \quad (1)$$

(for $n = |x| + |y|$), which means that the size of the produced secret key is asymptotically equal to the mutual information between Alice's and Bob's inputs, and the leakage of information on the key to the eavesdropper (who can access the transcript of the protocol t and public randomness r) is negligibly small.

- (ii) The size of the key in (i) is pretty much optimal: no communication protocol can produce a key w longer than $C(x) - C(x | y) + O(\delta) + o(n)$ without loosing the property of secrecy $C(w | \langle t, r \rangle) \geq [\text{length of } w \text{ in bits}] - o(n)$ (the size of a secret key cannot be made asymptotically greater than the mutual information between Alice's and Bob's inputs).

► **Remark 1.3.** In Theorem 1.2, the values of k and ℓ are embedded in the communication protocol $\pi_{k, \ell, \epsilon, \delta}$. This means that the parties in some sense “know” (at least approximately) the values of $C(x)$ and $C(x | y)$. This is similar to the settings of the classical information theory, where the parties “know” the probability distribution on random inputs and can use a suitable protocol. The theorem is nontrivial if the approximation rate $\delta = o(n)$ as $n \rightarrow \infty$.

The secrecy of the key is understood in the information-theoretic sense: the last inequality in (1) claims that complexity of the key w conditional on *all data accessible to the adversary* must be (almost) maximal. The theorem can be adapted to a non-uniform setting where the adversary is given an auxiliary inputs s_n . In this case, all terms of Kolmogorov complexity appearing in the theorem should be relativized conditional on s_n . The theorem remains meaningful if the size of s_n is $o(n)$. ┘

Theorem 1.2 can be extended to the multi-party setting, where $k > 2$ parties are given correlated data and need to agree on common secret key communicating via a public channel. Let us discuss in more detail the version with $k = 3$ participants. We assume now that three parties (Alice, Bob, and Charlie) are involved in the protocol. They are given inputs x, y, z respectively. We assume that all parties have an access to a common source of random bits (we denote by r the bits produced by this source) and exchange messages via a public channel (we use the conventional definition of a multi-party communication protocol with a public source of random bits, see [19]). It is assumed that every message sent by any party reaches every other party (and the eavesdropper). In what follows we consider only triples of inputs (x, y, z) with a “symmetric” complexity profile such that $C(x) \approx C(y) \approx C(z)$ and $C(x, y) \approx C(x, z) \approx C(y, z)$.

► **Theorem 1.4** (symmetric version of [24, Theorem 5.11]).

- (i) For any profile $(k_1, k_2, k_3) \in \mathbb{N}^3$ and $\epsilon, \delta > 0$ there exist a randomized communication protocols $\pi_{k_1, k_2, k_3, \epsilon, \delta}$ for three parties such that on every triple of binary input strings (x, y, z) (of length at most n) satisfying

$$C(x) \stackrel{\delta}{=} C(y) \stackrel{\delta}{=} C(z) \stackrel{\delta}{=} k_1, \quad C(x, y) \stackrel{\delta}{=} C(x, z) \stackrel{\delta}{=} C(y, z) \stackrel{\delta}{=} k_2, \quad C(x, y, z) \stackrel{\delta}{=} k_3 \quad (2)$$

¹ The term $C(x)$ stands for the plain Kolmogorov complexity of x (optimal compression of x), the term $C(x|y)$ stands for conditional Kolmogorov complexity of x conditional on y (optimal compression of x given advice y), and the notation $C(x) \stackrel{\delta}{=} k$ and $C(x|y) \stackrel{\delta}{=} \ell$ means that $|C(x) - k| \leq \delta$ and $|C(x|y) - \ell| \leq \delta$.

Alice, Bob, and Charlie can agree with probability $1 - \epsilon$ on a key $w = w(x, y, z, r)$ such that

$$[\text{length of } w \text{ in bits}] = \frac{I(x:y|z) + I(x:z|y) + I(y:z|x)}{2} + I(x : y : z) - O(\delta) - o(n), \quad (3)$$

$$C(w | \langle t, r \rangle) \geq |w| - o(n), \quad (4)$$

where r is the bit string produced by the public source of randomness, and t is the communication transcript (concatenation of the messages sent by Alice, Bob, and Charlie).

- (ii) The size of the key in (i) is asymptotically optimal, i.e., no communication protocol can give a key w asymptotically longer than

$$\frac{1}{2}(I(x : y | z) + I(x : z | y) + I(y : z | x)) + I(x : y : z) + O(\delta) + o(n) \quad (5)$$

without losing the property of secrecy (4).

► **Remark 1.5.** The general version of [24, Theorem 5.11] applies to a triple of inputs with arbitrary (possibly non-symmetric) complexity profile. In the general case, the characterization of the optimal size of the secret key is more involved than (3), see [24]. We discuss only symmetric complexity profiles in order to avoid cumbersome formulas and focus on the most essential combinatorial ideas behind the proofs.

Ineq. (4) means that the eavesdropper (who can access the communication transcript t and the public randomness r) gets no information on the produced secret key. Similarly to Theorem 1.2, this secrecy condition remains meaningful even if the adversary is a non-uniform agent having advice s_n of size $o(n)$. ◻

The known proofs of the positive parts of Theorem 1.2 and Theorem 1.4 (the existence of protocols) are quite explicit and constructive: we know specific communication protocols that allow to produce a secret key of the optimal size. More specifically, the proofs suggested in [24] provide a protocol for Theorem 1.2(i) with communication complexity

$$\min \{C(x | y), C(y | x)\} + O(\delta) + O(\log n) \quad (6)$$

and a protocol² for Theorem 1.4(i) with communication complexity

$$C(x, y, z) - \frac{1}{2}(I(x : y | z) + I(x : z | y) + I(y : z | x)) - I(x : y : z) + O(\delta) + O(\log n). \quad (7)$$

The communication complexity (6) from Theorem 1.2(i) is known to be asymptotically optimal, see [15]. In this paper we study the communication complexity of the problem from Theorem 1.4. In fact, (7) is *not* optimal for general communication protocols; however, we show that this communication complexity is asymptotically optimal in the class of protocols with *simultaneous messages*, i.e., in the model where Alice, Bob, and Charlie send their messages in parallel, receive the messages sent by their vis-a-vis, and compute the result (secret key) without any further interaction.

² The scheme proposed in [24] is the so called *omniscience* protocol. In this protocol, all parties send simultaneously their messages (random hash-values of the inputs) so that each of them learns completely the entire triple of inputs (x, y, z) (this explains the term *omniscience*). The total length of the sent messages is less than $C(x, y, z)$, so an eavesdropper can learn only a partial information on the inputs. The gap between the total complexity of $C(x, y, z)$ and the divulged information is used to produce a secret key.

► **Theorem 1.6** (main result). *In the setting of Theorem 1.4, communication complexity of a protocol with simultaneous messages (the total number of bits sent by Alice, Bob, and Charlie) for triples of inputs (x, y, z) with a symmetric complexity profile (2) cannot be smaller than*

$$C(x, y, z) - \frac{1}{2}(I(x : y | z) + I(x : z | y) + I(y : z | x)) - I(x : y : z) - O(\delta) - O(\log n). \quad (8)$$

Communication complexity (8) is not optimal for general (multi-round) communication protocols of secret key agreement, see Proposition 8.1.

The proof of our main result combines information-theoretic techniques and spectral bounds for graphs (the expander mixing lemma). Spectral bounds *per se* are not new in communication complexity (see, e.g., the usage of Lindsey’s lemma in [3]). Information-theoretic methods are also pretty common in this area. But the combination of these two techniques seems to be less standard. The key step of the proof is the observation that in some setting, when parties hold correlated data sets, for each of them it is hard to send a message that has non-negligible mutual information with the partners’ data. In other words, a “too short” message sent by Alice would have zero mutual information with the data (y, z) given to Bob and Charlie. For secret key agreement protocols, this observation implies that the messages of every party inevitably have to be quite long. A similar argument can be used in problems that are not connected with cryptography, see Theorem 5.1.

► **Remark 1.7.** It is instructive to compare our work with [15], where similar questions were addressed in the setting of two parties. Our work was motivated by the observation that the argument from [15] fails in the setting with $k > 2$ parties. In fact, the multi-party setting is qualitatively different. This becomes clear when we consider secret key agreement with a sub-optimal size of the key. The technique of [15] (in the setting with *two* parties) implies that communication complexity of the secret key agreement cannot be reduced even if Alice and Bob agree on a key of a pretty small size, see the “threshold phenomenon” discussed in [15]. Apparently, this phenomenon does not occur in the multi-party setting.

Thus, to deal with multi-party setting, we have to revise significantly the techniques from [15]. We have to change both the *information-theoretic* and *algebraic* components of the proof. The most important new components appear in the information-theoretic part of the proof. In particular, we need to use the exact expression (5) for the size of the secret key. The key new element of our argument is the observation *Alice must send a message having large mutual information with Bob’s and Charlie’s inputs, and the cost of this task can be high*, see the discussion in Sections 3.3-3.4 (this idea was irrelevant in the setting with two parties). In the algebraic component of the proof, we adapt the definition of a spectral expander to hypergraphs and then construct a hypergraph with the required properties (see Definition 3.5 and Section 3.5). Only the bridge between the algebraic and the information-theoretic components is pretty much the same as in [15] (in the proof of Theorem 4.1 we use an argument very similar to [15, Lemma 6]). ◻

The rest of the paper is organized as follows. In Section 2 we recall several standard definitions and introduce the notation. In Section 3 we explain informally the scheme of our argument. In Section 4 we prove the main technical tool of this paper, Theorem 4.1 (which claims that in some setting, it is hard to send a message that has non-negligible mutual information with the partners’ data). In Section 5 we illustrate the application of our technique with a simple example that is not related to cryptography. In Section 6 we prove Theorem 1.6 for a restricted (“the most important”) class of complexity profiles. In Section 7 we extend this result and prove Theorem 1.6 for all (symmetric) complexity profiles. We conclude with a discussion of limitations of our technique and open problems.

2 Preliminaries and Notation

2.1 General notation

For a binary string x we denote its length $|x|$. For a finite set S we denote its cardinality $\#S$.

We manipulate with equalities and inequalities for Kolmogorov complexity. Since many of them hold up to a logarithmic term, we use the notation $A \stackrel{\text{lg}}{=} B$, $A \leq^{\text{lg}} B$, and $A \geq^{\text{lg}} B$ for $|A - B| = O(\log n)$, $A \leq B + O(\log n)$, and $B \leq A + O(\log n)$ respectively, where n is clear from the context (n is usually the length of the strings involved in the inequality).

\mathbb{F}_q denotes the field of q elements (usually $q = 2^n$). A k -dimensional vector over \mathbb{F}_q is a k -tuple $(x_1, \dots, x_k) \in \mathbb{F}_q^k$. We say that two vectors (x_1, \dots, x_k) and (y_1, \dots, y_k) in \mathbb{F}_q^k are orthogonal to each other if $x_1y_1 + \dots + x_ky_k = 0$ (the addition and multiplication are computed in the field \mathbb{F}_q). A vector is called self-orthogonal if it is orthogonal to itself. In a k -dimensional space over the field of characteristic 2 there are 2^{k-1} self-orthogonal vectors (x_1, \dots, x_k) and they form a linear subspace of co-dimension 1 (a vector is self-orthogonal iff $x_1 + \dots + x_k = 0$). A *direction* in \mathbb{F}_q^k is an equivalence class of non-zero vectors over \mathbb{F}_q that are proportional to each other (a direction can be understood as a point in the projective space of dimension $k - 1$).

$C(x)$ stands for Kolmogorov complexity of x (the length of the shortest program³ producing x) and $C(x | y)$ (the length of the shortest program producing x given input y) stands for Kolmogorov complexity of x given y . Respectively, $I(x : y)$ and $I(x : y | z)$ denote the mutual information between x and y and the conditional information between x and y given z . We use the notation $I(x : y : z) := I(x : y) - I(x : y | z)$. For a tuple of strings (x_1, \dots, x_n) its *complexity profile* is the vector consisting of the complexity values $C(x_{i_1}, \dots, x_{i_s})$ (for all $2^n - 1$ sub-tuples $1 \leq i_1 < \dots < i_s \leq n$). Kolmogorov complexity can be relativized: $C^{\mathcal{O}}(x)$ and $C^{\mathcal{O}}(x | y)$ stand for Kolmogorov complexity of x (conditional on y) assuming that the universal decompressor can access oracle \mathcal{O} . If the oracle is a finite string s , then $C^{\mathcal{O}}(x) = C(x | s) + O(1)$. For more detail on the basic facts about Kolmogorov complexity, see the full paper. A comprehensive introduction in the theory of Kolmogorov complexity can be found in [20] and [26].

2.2 Communication complexity

We use the conventional notion of a communication protocol for two or three parties, see for detailed definitions [19]. We discuss *deterministic protocols* and *randomized protocols with a public source of random bits* (see the full version of the paper for more detail).

In general, a communication protocol may consist of several rounds, when each next message of every party depends on the previously sent messages. In the *simultaneously messages* model there is no interaction: all parties send in parallel their messages that depend only on their own input data (and the random bits), and then compute the final result.

We will assume that the communication protocol has a “uniform” description. More technically, we assume that for n -bit inputs (the full description of such a protocol) has an efficient description of size $O(\log n)$. For such a protocol we do not lose much security even if the description of the protocol is available to the eavesdropper. Thus, we cannot “cheat” by embedding in the structure of the protocol any secret information hidden from the adversary.

³ In an optimal programming language, see [20, 26] for more detail.

2.3 Reminder of the spectral graph technique

Let $G = (L \cup R, E)$ be a bi-regular bipartite graph where each vertex in L has degree D_L , each vertex in R has degree D_R , and each edge $e \in E$ connects a vertex from L with a vertex from R (observe that $\#E = \#L \cdot D_L = \#R \cdot D_R$). The adjacency matrix of such a graph is a zero-one matrix $M = \begin{pmatrix} 0 & A \\ A^\top & 0 \end{pmatrix}$ where A is a matrix of dimension $(\#L) \times (\#R)$ ($A_{xy} = 1$ if and only if there is an edge between the x -th vertex in L and the y -th vertex in R). Let $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_N$ be the eigenvalues of M , where $N = \#L + \#R$ is the total number of vertices. Since M is symmetric, all λ_i are real numbers. It is well known that for a bipartite graph the spectrum is symmetric, i.e., $\lambda_i = -\lambda_{N-i+1}$ for each i , and $\lambda_1 = -\lambda_N = \sqrt{D_L D_R}$ (see, e.g., [12]). The graphs with a large gap between λ_1 and λ_2 have the property of good *mixing*, see [25].

► **Lemma 2.1** (Expander Mixing Lemma for bipartite graphs, see [12]). *Let $G = (L \cup R, E)$ be a regular bipartite graph where each vertex in L has degree D_L and each vertex in R has degree D_R . Then for each $A \subseteq L$ and $B \subseteq R$ we have $\left| E(A, B) - \frac{D_L \cdot \#A \cdot \#B}{\#R} \right| \leq \lambda_2 \sqrt{\#A \cdot \#B}$, where λ_2 is the second largest eigenvalue of the adjacency matrix of G and $E(A, B)$ is the number of edges between A and B .*

► **Corollary 2.2.** *Let $G = (L \cup R, E)$ be a graph from Lemma 2.1. Then for $A \subseteq L$ and $B \subseteq R$ such that $\#A \cdot \#B \geq \left(\frac{\lambda_2 \#R}{D_L} \right)^2$ we have $E(A, B) = O\left(\frac{D_L \cdot \#A \cdot \#B}{\#R} \right)$.*

3 Main technical tools and the scheme of the proof

In this section we sketch the main ideas used in the proof of our principal result (Theorem 1.6).

3.1 Setting the parameters

Let us assume that $\delta = O(\log n)$, i.e., all parties of the protocol “know” the complexity profile of the triple of inputs (x, y, z) up to an additive logarithmic term⁴. This assumption does not affect significantly the argument, but it helps to avoid minor technical details and makes the explanation more transparent. To simplify the notation, in this section we discuss only triples of inputs with the profile

$$\begin{aligned} C(x) &\stackrel{\text{lg}}{=} C(y) \stackrel{\text{lg}}{=} C(z) \stackrel{\text{lg}}{=} kn, & C(x, y) &\stackrel{\text{lg}}{=} C(x, z) \stackrel{\text{lg}}{=} C(y, z) \stackrel{\text{lg}}{=} (2k - 1)n, \\ C(x, y, z) &\stackrel{\text{lg}}{=} (3k - 3)n \end{aligned} \tag{9}$$

In this setting, Theorem 1.4 gives the optimal size of a secret key

$$\frac{1}{2} (I(x : y | z) + I(x : z | y) + I(y : z | x)) + I(x : y : z) \stackrel{\text{lg}}{=} 1.5n. \tag{10}$$

Our aim is to bound communication complexity for inputs with this complexity profile:

► **Theorem 3.1** (special case of Theorem 1.6). *In the setting of Theorem 1.4, communication complexity of a protocol with simultaneous messages for some triples of inputs (x, y, z) with complexity profile (9) cannot be smaller than $(3k - 4.5)n$, which matches Eq. (8).*

⁴ A logarithmic error term is, in some sense, the finest meaningful precision for Kolmogorov complexity. All our arguments can be repeated *mutatis mutandis* for any coarser precision δ such that $\log n \ll \delta(n) \ll n$.

3.2 Preliminary consideration: the need for hard inputs

The optimal size of the secret key in Theorem 1.2 and Theorem 1.4 depends only on the complexity profile of (x, y, z) and not on the combinatorial structure of the input. The situation with communication complexity (the number of bits sent by the parties) is different: it may vary significantly for different tuples of inputs with the same complexity profile. When we talk about the communication complexity of a protocol, we mean the worst-case complexity, i.e., the maximal number of sent bits among all admissible inputs. To prove a lower bound for the worst-case communication complexity, we need to provide a triple of inputs for which the parties have to send long messages. We provide a class of inputs that are guaranteed to be “hard” (for all valid protocol, for most triples of inputs from this class, communication complexity is high).

3.3 First step of the argument: conditional on Charlie’s message, the mutual information between Alice’s and Bob’s inputs must increase

We begin with an observation that might seem to have nothing to do with communication complexity. We recall the lower bound for the size of the secret key (that applies to protocols with any communication complexity). In [24] (see Theorem 1.2(ii)) it is shown that two parties, Alice and Bob, can agree on secret key of complexity k *only if* the mutual information between Alice’s input x and Bob’s input y is greater than k . The proof of this statement can be easily adapted to the following slightly more general setting:

► **Lemma 3.2.** *Assume that there is a publicly available information s (accessible to Alice, Bob, and the eavesdropper); besides this, Alice is given a private input x and Bob is given a private input y . Then, by communication via a public channel accessible to the eavesdropper, Alice and Bob cannot agree on a secret key of complexity greater than $I(x : y | s)$.*

We apply this proposition to a protocol with three parties. Let t_C denote the concatenation of the messages sent by Charlie. This is a piece of publicly available information (accessible to Alice, Bob, and the eavesdropper). Due to Lemma 3.2, Alice and Bob cannot agree on a secret key with Kolmogorov complexity greater than $I(x : y | t_C)$ (at this point we ignore whether Charlie can learn the same key or not). Hence, in the settings (9), a secret key of size (10) can be produced only if $I(x : y | t_C) \geq^{\lg} 1.5n$. Observe that in the setting (9) the mutual information between x and y is equal to n . This means that the mutual information between Alice’s and Bob’s inputs *conditional on Charlie’s message*, i.e., $I(x : y | t_C)$, is bigger than the unconditional mutual information between Alice’s and Bob’s inputs, i.e., $I(x : y)$. A pretty standard information-theoretic argument implies that the gap between $I(x : y)$ and $I(x : y | t_C)$ is not greater than the mutual information between $\langle x, y \rangle$ and t_C , and we conclude that $I(x, y : t_C) \geq^{\lg} n/2$. In other words, Charlie must send a message t_C that has $\geq n/2$ bits of mutual information with the pair of inputs of Alice and Bob. A similar argument implies that Alice must send a message t_A such that $I(y, z : t_A) \geq^{\lg} n/2$ and Bob must send a message t_B such that $I(x, z : t_B) \geq^{\lg} n/2$.

This part of the argument is based on Lemma 3.2, which re-employs an argument from [24] in a pretty direct way. So at this stage we need no substantially new ideas.

3.4 Second step of the argument: it may be difficult for Alice to send a message increasing the mutual information between Bob's and Charlie's inputs

We have shown above that in the setting (9) Alice, Bob, and Charlie can agree on a secret key of optimal size only if each of them sends a message that contains $\geq^{\text{lg}} n/2$ bits of mutual information with the inputs of two other parties

We are going to show that this may require sending *very long* messages (much longer than $n/2$ bits). This part of the argument is the main technical contribution of our paper. To explain this idea, we make a digression and discuss a similar problem in simpler settings.

Digression: how to say something that the interlocutor already knows. Let us consider randomized communication protocols with two participants playing non-symmetric roles. We call the participants Speaker and Listener and assume that Speaker holds an input string a and Listener holds another input string b . This is a one-way protocol: Speaker sends a message to Listener in one round, without any feedback. The aim of Speaker is to send to Listener a message that is *not completely unpredictable* from the point of view of Listener. More precisely, Speaker's message must have positive (and non-negligible) mutual information with Listener's input b . We start with a simple example when the task of Speaker is trivial.

► **Example 3.3.** Let Speaker is given a string $a = uv$ and Listener is given a string $b = uv$, where u , v , and w are independent incompressible strings of length n , i.e., $C(uvw) \stackrel{\text{lg}}{=} C(u) + C(v) + C(w) \stackrel{\text{lg}}{=} 3n$. Observe that

$$C(a) \stackrel{\text{lg}}{=} 2n, \quad C(b) \stackrel{\text{lg}}{=} 2n, \quad I(a : b) \stackrel{\text{lg}}{=} n \quad (11)$$

In this setting, if Speaker wants to communicate a message of length n with a *high* mutual information with Listener's y , she may send a part of u , which is known to both participants of the protocol. On the other hand, if Speaker wants to communicate a message with a *low* mutual information with Listener's b , this is also possible: Speaker may send a part of v , which is known to Speaker but not to the Listener. \lrcorner

► **Example 3.4.** Now we consider a pair (a, b) with the same complexity profile as in Example 3.3 but with a different combinatorial structure. Let a be a line in the projective plane over the finite field \mathbb{F}_{2^n} and b be a point in the same projective plane incident to a , and the pair (a, b) have the maximal possible complexity (among all incident pairs (line, point) in the plane). For these a and b we have the same complexity profile (11). Indeed, we need two elements of the field ($2n$ bits of information) to specify a line or a point, but we need only one element of the field (n bits of information) to specify a point when a line is known. However, the combinatorial properties of this pair are very different from the properties of the pair in Example 3.3.

If Speaker is given a and Listener is given b as above, then Speaker cannot send a *reasonably short* message having non-negligible mutual information with Listener's input b . In fact, if Speaker wants to send to Listener a message $m = m(a)$ having δ bits of mutual information with b , then the size of m must be at least $n + \delta$. In particular, if the message m is shorter than n , then it cannot contain any information on b , see Section 4. \lrcorner

Example 3.4 is an instance of a much more general phenomenon. Let us have a bipartite graph $G = (V_L, V_R, E)$, where the set of vertices is $V_L \cup V_R$ and the set of edges is $E \subset V_L \times V_R$. We assume that the graph is bi-regular, i.e., all vertices in V_L have the same degree D_L and all vertices in V_R have the same degree D_R (we always assume that $D_L \geq D_R$). We say that

G is a *spectral expander*⁵ if the second eigenvalue of its adjacency matrix $\lambda_2 = O(\sqrt{D_L})$. Let $(x, y) \in E$ be a “typical” edge of this graph (in the sense that its Kolmogorov complexity is close to the maximum possible value), and let x and y be the inputs given to Alice and Bob respectively. Then we have a property similar to Example 3.3: if Alice wants to send a message having δ bits of mutual information with Bob’s data y , she must send a message of size at least $\log D_R + \delta$. We prove this fact using the Expander Mixing Lemma. (Example 3.4 corresponds to the graph $G = (V_L, V_R, E)$ where V_L consists of all lines in the plane, V_R consists of all points in the plane, and E is the set of all pairs of incident lines and points; it is known that this graph is a spectral expander.) [End of **Digression**.]

Now we generalize the observations from the *Digression* above and explain the main idea of the proof of Theorem 3.1. We need the following extension of the notion of expander.

► **Definition 3.5.** Let $G = (V_1, V_2, V_3, H)$ be a hypergraph where the set of vertices consists of three disjoint parts V_1, V_2, V_3 of the same cardinality, and the set of hyperedges is a set $H \subset V_1 \times V_2 \times V_3$. We consider three bipartite graphs G_1, G_2, G_3 associated with hypergraph G : each G_i is a bipartite graph $(V_i, V_{j\ell}, E_i)$ (here $j = i + 1 \pmod 3$ and $\ell = i + 2 \pmod 3$), where $V_{j\ell}$ is the sets of $(y, z) \in V_j \times V_\ell$ that are connected in G and $(x, (y, z)) \in E_i$ if and only if the triple $\{x, y, z\}$ corresponds to a hyperedge in H . The hypergraph is called *tri-expander* if the graphs G_1, G_2, G_3 are *bi-regular spectral expanders*⁶.

We show that the communication is costly for a triple of inputs (x, y, z) that is a hyperedge in a tri-expander. To this end, we combine the idea from section 3.3 with an argument similar to the observation sketched in the *Digression*: each party must send a message having non-negligible mutual information with two other inputs (an information-theoretic argument) but this is only possible when each of the messages is very long (due to the spectral bound and the expander mixing lemma).

3.5 Construction of a tri-expander

To conclude the proof of the main result it remains to show that there exists a tri-expander with suitable parameters:

► **Proposition 3.6.** For all integer numbers $k \geq 0$ and $n \geq 1$ there exists a tri-expander $G = (V_1, V_2, V_3, H)$ such that $\#V_1 = \#V_2 = \#V_3 = \Theta(2^{kn})$, $\#H = \Theta(2^{kn} \cdot 2^{(k-1)n} \cdot 2^{(k-2)n})$, and for all $i \neq j$, for every $x \in V_i$ there exists $\Theta(2^{(k-1)n})$ vertices $y \in V_j$ such that x and y are adjacent in the hypergraph.

Proof. We construct such a tri-expander explicitly. We fix the finite field \mathbb{F}_{2^n} with $q = 2^n$ elements, the $(k + 2)$ -dimensional space \mathcal{L} over this field, and the subspace $\mathcal{L}_{so} \subset \mathcal{L}$ that consists of self-orthogonal vectors. Observe that $\#\mathcal{L}_{so} = \#\mathcal{L}/q = q^{k+1}$ (a subspace of co-dimension 1 in \mathcal{L}). Let V denote the space of all *directions* in \mathcal{L}_{so} except for the direction $(1, \dots, 1)$ (which is self-orthogonal for even k). Observe that $\#V = \Theta(q^k)$.

We let $V_1 = V_2 = V_3 = V$ and define H as the set of all triple $(x, y, z) \in V^3$ such that x, y, z are *distinct and pairwise orthogonal* directions in \mathcal{L}_{so} .

For every vector $x \in \mathcal{L}_{so}$, the condition of being orthogonal to x determines in \mathcal{L}_{so} a subspace of co-dimension 1; this subspace consists of q^k vectors (including x itself as it is self-orthogonal) and, respectively, $(q^k - 1)/(q - 1)$ directions (again, including the direction

⁵ We use the term *expander* without assuming that the degree of a graph is constant.

⁶ The usage of the expander mixing lemma for a tri-expander seems to be similar but not literally equivalent to the hypergraph generalization of the expander mixing lemma from [18].

collinear with x). If we have two non-collinear vectors $x, y \in \mathcal{L}_{so}$, then the condition of being orthogonal to x and y determines in \mathcal{L}_{so} a subspace of co-dimension 2; this subspace consists of q^{k-1} vectors (including x and y), which corresponds to $(q^{k-1} - 1)/(q - 1) = \Theta(q^{k-2})$ directions (once again, including the directions collinear with x and with y).

Thus, we have $\Theta(q^k)$ individual vertices, $\Theta(q^k \cdot q^{k-1})$ pairs of adjacent vertices, and $\Theta(q^k \cdot q^{k-1} \cdot q^{k-2})$ adjacent triples (hyperedges). It remains to compute the eigenvalues of the associated bipartite graphs.

► **Lemma 3.7.** *The hypergraph $G = (V_1, V_2, V_3, H)$ defined above is a tri-expander.*

(We give a proof in the full version of the paper.) ◀

► **Remark 3.8.** A standard counting shows that for most hyperedges (x, y, z) in the graph from Proposition 3.6 we have $C(x) \stackrel{\text{lg}}{=} \log \Theta(q^k) \stackrel{\text{lg}}{=} kn$, $C(x, y) \stackrel{\text{lg}}{=} \log \Theta(q^k \cdot q^{k-1}) \stackrel{\text{lg}}{=} (2k - 1)n$, $C(x, y, z) \stackrel{\text{lg}}{=} \log \Theta(q^k \cdot q^{k-1} \cdot q^{k-2}) \stackrel{\text{lg}}{=} (3k - 3)n$, and we get the profile (9). ◻

4 When it is hard to say anything that the interlocutor already knows

In this section we explain our main technical tool. We consider randomized communication protocols with two participants, Speaker and Listener. We assume that Speaker holds an input string a and Listener holds another input string b ; we assume also that the complexity profile of the pairs (a, b) is known to all parties. The aim of Speaker in this protocol is to send to Listener a message that has non-negligible mutual information with Listener's input b , as we discussed in Section 3.

► **Theorem 4.1.** *Let $G = (V_L, V_R, E)$ be a bipartite spectral expander such that $N = \#V_L$, $M = \#V_R$, and (D_L, D_R) are the degrees of the edges in V_L and V_R respectively. Let $(a, b) \in E$ be a "typical" edge in the graph, i.e., $C(a, b) \stackrel{\text{lg}}{=} \log \#E$, and $C(m | a) \stackrel{\text{lg}}{=} 0$. Then $I(m : b) \stackrel{\text{lg}}{\leq} \max\{0, C(m) - C(a | b)\}$. In particular, if the length of m is less than $C(a | b)$, then $I(m : b) \stackrel{\text{lg}}{=} 0$.*

► **Remark 4.2.** The statement of Theorem 4.1 remain valid if we relativize all terms of Kolmogorov complexity in this statement conditional on a string r such that $I(r : (a, b)) \stackrel{\text{lg}}{=} 0$. In what follows we present the proof without r . But every step of this argument trivially relativizes conditional on r assuming that $C(a, b | r) \stackrel{\text{lg}}{=} C(a, b) \stackrel{\text{lg}}{=} \log \#E$. ◻

Proof of Theorem 4.1. Let $n_a := \log N$, $n_b = \log M$, $n'_a = \log D_R$, $n'_b = \log D_L$, and $n_{ab} := n_a - n'_a$. Using this notation, we have

$$C(a) \stackrel{\text{lg}}{=} n_a, \quad C(b) \stackrel{\text{lg}}{=} n_b, \quad C(a | b) \stackrel{\text{lg}}{=} n'_a, \quad C(b) \stackrel{\text{lg}}{=} n_b, \quad C(b | a) \stackrel{\text{lg}}{=} n'_b, \quad I(a : b) \stackrel{\text{lg}}{=} n_{ab}.$$

Since Speaker computes the message m given the input data a , we have $C(m | a) \stackrel{\text{lg}}{=} 0$. We denote $\alpha := I(m : a | b)$ and $\beta := I(m : a : b)$. It is easy to verify that $C(m) = \alpha + \beta$.

Case 1. Assume that $C(m) \leq n'_a - 2 \cdot \text{const} \cdot \log n$ for some $\text{const} > 0$ (a constant to be specified later). In this case, to prove the theorem, we need to show that $I(m : b) \stackrel{\text{lg}}{=} 0$. In our notation this is equivalent to $\beta \stackrel{\text{lg}}{=} 0$. More technically, we are going to show that

$$\beta \leq \text{const} \cdot \log n. \tag{12}$$

For the sake of contradiction we assume that (12) is false. It is enough to consider the case when β is *somewhat large* but not *too large*, i.e., just slightly above the threshold (12). Indeed, any communication protocol violating (12) can be converted in a different protocols

with the same or a smaller value of α and with $\beta = \text{const} \cdot \log n + O(1)$. To this end, we observe that by discarding a few last bits of Speaker's message m we make the protocol only simpler. So, we may replace the initial message m with the shortest prefix of the initial message that still violates (12). Thus, in what follows, we assume w.l.o.g. that

$$\text{const} \cdot \log n < \beta \leq \text{const} \cdot \log n + O(1).$$

Let us define $A := \{a' : C(a' | m) \leq C(a | m)\}$ and $B := \{b' : C(b' | m) \leq C(b | m)\}$. We use the following standard claim:

▷ **Claim.** For A and B defined above we have $\#A = 2^{C(a|m) \pm O(\log n)} = 2^{n_a - \alpha - \beta \pm O(\log n)}$ and $\#B = 2^{C(b|m) \pm O(\log n)} = 2^{n_b - \beta \pm O(\log n)}$ (see, e.g. [24, Claim 4.7]).

From the claim we obtain $\#A \cdot \#B = 2^{n_a - \alpha - \beta + n_b - \beta \pm O(\log n)} = 2^{n_a + n_b - C(m) - \beta \pm O(\log n)}$. Since $C(m) \leq n'_a - 2 \cdot \text{const} \log n$ and $\beta < \text{const} \log n + O(1)$, we conclude

$$\begin{aligned} n_a + n_b - C(m) - \beta n \pm O(\log n) &\geq n_a + n_b - (n'_a - 2 \cdot \text{const} \cdot \log n) \\ &\quad - \text{const} \cdot \log n - O(\log n) \\ &\geq n_{ab} + n_b + \text{const} \cdot \log n - O(\log n) \geq n_{ab} + n_b. \end{aligned}$$

(To get the last inequality, we should choose the value of const in (12) so that $\text{const} \cdot \log n$ majorizes the term $O(\log n)$ in the inequality above.) Thus, $\#A \cdot \#B \geq 2^{n_{ab} + n_b} = \frac{M^2}{D_L}$.

With the Corollary 2.2 we obtain $E(A, B) = O\left(\frac{D_L \cdot \#A \cdot \#B}{M}\right) = O\left(\frac{\#A \cdot \#B}{M/D_L}\right)$. Now observe that given m and the numbers $C(a | m)$ and $C(b | m)$ we can enumerate the sets A and B and, therefore, we can describe (a, b) by the index of this edge in the list of all edges between A and B . The size of such an index is $\log E(A, B)$. Hence,

$$\begin{aligned} C(a, b | m) \stackrel{\text{lg}}{\leq} \log E(A, B) &\stackrel{\text{lg}}{\leq} (n_a + n_b - C(m) - \beta) - (n_b - n'_b) \\ &= n_a + n'_b - C(m) - \beta = C(a, b) - C(m) - \beta, \end{aligned}$$

and $C(a, b) \stackrel{\text{lg}}{\leq} C(m) + C(a, b | m) \stackrel{\text{lg}}{\leq} C(a, b) - \beta$. The terms $O(\log n)$ hidden in the notation $\stackrel{\text{lg}}{\leq}$ and $\stackrel{\text{lg}}{=}$ in this inequality do not depend on β . Thus, we get a contradiction if the constant in (12) is chosen large enough.

Case 2. Now we assume that $C(m) = n'_a + \delta$ for an arbitrary δ . Denote by m' the prefix of m of length $(n'_a - \text{const} \log n)$ and by m'' the suffix of m of length $(\delta + \text{const} \log n)$. We know from Case 1 that $I(m' : b) \stackrel{\text{lg}}{=} 0$. It remains to apply the chain rule,

$$I(m : b) \stackrel{\text{lg}}{=} I(m' : b) + I(m'' : b | m') \stackrel{\text{lg}}{=} I(m'' : b | m') \stackrel{\text{lg}}{\leq} |m''| \stackrel{\text{lg}}{=} \delta.$$

and the theorem is proven. ◀

► **Corollary 4.3.** Let $G = (V_L, V_R, E)$ be a bipartite spectral expander such that $N = \#V_L$, $M = \#V_R$, and (D_L, D_R) are the degrees of the edges in V_L and V_R respectively.

- (a) We assume that Speaker and Listener are given, respectively, a and b that are ends of a typical edge $(a, b) \in E$ in the graph. We consider a one-round communication protocol where Speaker sends to Listener a message $m = m(a)$. Then $I(m : b) \stackrel{\text{lg}}{\leq} \max\{0, C(m) - C(a | b)\}$. In particular, if the length of m is less than $C(a | b)$, then $I(m : b) \stackrel{\text{lg}}{=} 0$.
- (b) A similar statement is true if Speaker and Listener are given instead of a and b some inputs a' and b' such that $C(a' | a) \stackrel{\text{lg}}{=} 0$ and $C(b' | b) \stackrel{\text{lg}}{=} 0$ (e.g., if Speaker is given a function of a vertex $a \in V_L$ and Listener is given a function of a vertex $b \in V_R$).

5 Protocols with simultaneous messages : a warm-up example

In this section we use Theorem 4.1 from the previous section to prove a lower bound for communication complexity of the following problem. Alice and Bob hold, respectively, lines a and b in a plane (intersecting at one point c). They send to Charlie (in parallel, without interacting with each other) some messages so that Charlie can reconstruct the intersection point. We argue that the trivial protocol (where Alice and Bob send the full information on their lines) is essentially optimal.

► **Theorem 5.1.** *Let Alice and Bob be given lines in the projective plane over the finite field \mathbb{F}_{2^n} (we denote them a and b respectively), and it is known that the lines intersect at point c . Another participant of the protocol Charlie has no input information. Alice and Bob (without a communication with each other) send to Charlie messages m_A and m_B so that Charlie can find c . For every communication protocol for this problem, for some a, b we have $|m_A| + |m_B| \geq^{\lg} 4n$, which means essentially that in the worst case Alice and Bob must send to Charlie all their data (for a typical pair of lines we have $C(a) + C(b) \stackrel{\lg}{=} 4n$).*

In the setting of Theorem 5.1, the inputs of Alice and Bob contain n bits of the mutual information with c , so an easy lower bound for the communication complexity is $n + n = 2n$. However, due to the spectral properties of graphs implicitly present in this construction, the true communication complexity of this problem is twice bigger.

Sketch of the proof (see the full version of the paper for the details). In this sketch we ignore the public randomness and explain the argument for deterministic protocols. A generalization for protocols with public randomness is pretty straightforward, see full version of the paper.

Let (a, b) be a pair of lines in a projective plane over \mathbb{F}_{2^n} intersecting at a point c , such that $C(a, b) \stackrel{\lg}{=} C(a) + C(b) \stackrel{\lg}{=} 4n$ (which is the case for most pairs of lines in the plane). Observe that $I(a : c) \stackrel{\lg}{=} n$ and $I(b : c) \stackrel{\lg}{=} n$. It follows that for the messages $m_A = m_A(a)$ and $m_B = m_B(b)$ we have $I(m_A : c) \leq^{\lg} n$ and $I(m_B : c) \leq^{\lg} n$. Using standard information theoretic inequalities, one can show that Alice's message m_A and Bob's message m_B determine the point c uniquely only if $I(m_A : c) \stackrel{\lg}{=} n$ and $I(m_B : c) \stackrel{\lg}{=} n$. Thus, Alice and Bob must send messages with large enough information on c .

The graph of possible pairs (a, c) and the graph of possible pairs (b, c) (the configurations (line, point)) is the same as in Example 3.4. Hence, we can apply Theorem 4.1 (Alice and Bob play the roles of Speaker, and Charlie plays the role of Listener) and conclude that $I(m_A : c) \leq^{\lg} \max\{0, C(m_A) - n\}$ and $I(m_B : c) \leq^{\lg} \max\{0, C(m_B) - n\}$. In particular, $I(m_A : c) \stackrel{\lg}{=} n$ and $I(m_B : c) \stackrel{\lg}{=} n$ only if Kolmogorov complexities of m_A and m_B are both at least $2n$. Thus, the total communication complexity is $\geq^{\lg} 2n + 2n = 4n$. ◀

6 Secret key agreement: a lower bound for the most crucial profile

In this section we prove a lower bound for communication complexity of secret key agreement with three parties. Let us recall the setting. We assume that Alice, Bob, and Charlie are given inputs x, y, z respectively with the complexity profile (9). This is a pretty “generic” complexity profile; by choosing k , we control the gap between the complexities of x, y, z and the mutual informations shared by the inputs.

We consider communication protocols with public randomness. Denote by r the string of random bits accessible for all the parties (including the eavesdropper). We assume that Alice, Bob, and Charlie broadcast simultaneously messages $m_A = m_A(x, r)$, $m_B = m_B(y, r)$, $m_C = m_C(z, r)$ over a public communication channel. Then each of them computes the final result

$$\text{key}_{\text{Alice}}(x, r, m_B, m_C), \text{key}_{\text{Bob}}(y, r, m_A, m_C), \text{key}_{\text{Charlie}}(z, r, m_A, m_B).$$

We say that a protocol is successful if $\text{key}_{\text{Alice}} = \text{key}_{\text{Bob}} = \text{key}_{\text{Charlie}} = w$ (i.e., the parties agree on a common key w) and $C(w \mid \langle m_A, m_B, m_C, r \rangle) \stackrel{\text{lg}}{=} |w|$ (i.e., the eavesdropper gets no information on this key).

Theorem 1.4 claims that for any $\epsilon > 0$ there exists a protocol that is successful with probability $(1 - \epsilon)$, and the size of the key is equal to (5), which gives for the profile (9) the value $1.5n$. Moreover, this value of the key is optimal (up to an additive term $O(\log n)$).

It was shown in [24] that a secret key of this size can be obtained in an *omniscience* protocol. In this protocol, the parties broadcast messages so that each of them learns completely the entire triple of inputs (x, y, z) . The total length of the broadcasted messages bits is less than $C(x, y, z)$, so an eavesdropper can learn only a partial information on the inputs. More specifically, communication complexity of the omniscience protocol is (7), which is $(3k - 4.5)n$ for a triple satisfying (9). The gap between $C(x, y, z) \stackrel{\text{lg}}{=} (3k - 3)n$ and the amount of the divulged information is used to produce the secret key of size $1.5n$.

The omniscience protocol used in [24] provides an *upper bound* on the communication complexity of secret key agreement. In what follows we prove the matching *lower bound* (for protocols with simultaneous messages) and show that $(3k - 4.5)n$ is the optimal communication complexity for a protocol of secret key agreement protocols with simultaneous messages for inputs satisfying (9). The proof follows the scheme sketched in Section 3. The first ingredient of this proof is Lemma 3.2 (see p. 8).

Sketch of proof of Lemma 3.2. This lemma is a relativized version of [24, Theorem 4.2]. One can follow the argument from [24] step by step, substituting s as a supplementary condition in each term of Kolmogorov complexity appearing in the proof. ◀

▶ **Corollary 6.1.** *Consider a communication protocol with three parties where Alice is given x , Bob is given y , and Charlie is given z . Denote by m_C the concatenation of all messages broadcasted by Charlie during the communication. If the parties agree on a secret key w on which the eavesdropper gets no information (even given access to the messages sent by all parties), then $C(w) \leq^{\text{lg}} I(x : y \mid r, m_C)$.*

Proof. We apply Lemma 3.2 substituting m_C instead of the public information s . ◀

▶ **Theorem 3.1 rephrased.** *Let Alice, Bob, and Charlie be given x , y , and z respectively such that (x, y, z) is a hyperedge of the hypergraph $G = (V_1, V_2, V_3, H)$ from Proposition 3.6 (the pairwise disjoint self-orthogonal directions in a $(k + 2)$ -dimensional vector space over \mathbb{F}_2^n). We consider non-interactive communication protocols where Alice, Bob, and Charlie send messages m_A , m_B , and m_C respectively and produce a secret key w with the optimal complexity $C(w) \stackrel{\text{lg}}{=} 1.5n$. Then $C(m_A) \geq^{\text{lg}} (k - 1.5)n$, $C(m_B) \geq^{\text{lg}} (k - 1.5)n$, $C(m_C) \geq^{\text{lg}} (k - 1.5)n$, and the communication complexity of the protocol is at least $(3k - 4.5)n - O(\log n)$.*

Proof. To simplify the notation, we ignore the bits r provided by the public source of randomness and explain the proof for deterministic protocols. Our argument trivially relativizes given any instance of random bits r independent of (x, y) (which is true with a probability close to 1), cf. the proof of Theorem 5.1 in the full version of the paper.

From Corollary 6.1 we know that the size of the key (in our case $1.5n$) cannot be greater than $I(x : y | m_C)$. By the construction of the tri-expander, $I(x : y) \stackrel{\text{lg}}{=} n$. Therefore, the difference between $I(x : y)$ and $I(x : y | m_C)$ is at least $0.5n$.

► **Lemma 6.2.** *For all binary strings x, y, s it holds $I(x : y | s) - I(x : y) \leq \text{lg} I(s : xy)$.*

(See the proof of the lemma in the full version of the paper.) We combine Corollary 6.1 with Lemma 6.2 and obtain $I(m_C : xy) \geq \text{lg} 0.5n$.

Then, we apply Theorem 4.1 to the bipartite graph G_3 associated with the tri-expander G (see p. 11); here Charlie plays the role of Speaker, and Alice and Bob together play the role of Listener. Since $I(m_C : xy) \geq \text{lg} 0.5n$, we obtain $C(m_C) \geq \text{lg} C(z | x, y) + 0.5n \stackrel{\text{lg}}{=} (k - 1.5)kn$. A similar argument applies to $C(m_A)$ and $C(m_B)$, and we are done. ◀

7 Secret key agreement: a lower bound for all symmetric profiles

Proof of Theorem 1.6. If the complexity profile of (x, y, z) is symmetric then it can be specified by three real numbers:

$$\begin{cases} C(x | y, z) \stackrel{\text{lg}}{=} C(y | x, z) \stackrel{\text{lg}}{=} C(z | x, y) \stackrel{\text{lg}}{=} \alpha, \\ I(x : y | z) \stackrel{\text{lg}}{=} I(x : z | y) \stackrel{\text{lg}}{=} I(y : z | x) \stackrel{\text{lg}}{=} \beta, \quad I(x : y : z) \stackrel{\text{lg}}{=} \gamma. \end{cases} \quad (13)$$

In what follows we say that the triple of numbers (α, β, γ) represent *symmetric complexity profile* of the triple (x, y, z) .

In Theorem 3.1 we proved that communication complexity (7) of the omniscience protocol is optimal in case $\alpha = (k - 2)n$, $\beta = n$, and $\gamma = 0$. We reduce the problem with arbitrary α, β, γ to the special case settled in Theorem 3.1.

► **Lemma 7.1.** *If communication complexity (7) is optimal (in the worst case) for some triples of inputs (x, y, z) with complexity profile (13), then*

- (i) *for every positive $\delta \leq n$, the omniscience protocol is also optimal for some triples of inputs (x', y', z') with symmetric complexity profile $(\alpha', \beta', \gamma') = (\alpha - \delta, \beta, \gamma)$;*
- (ii) *for every positive δ , the omniscience protocol is also optimal for some triples of inputs (x', y', z') with symmetric complexity profile $(\alpha', \beta', \gamma') = (\alpha, \beta, \gamma + \delta)$;*
- (iii) *if $\alpha \stackrel{\text{lg}}{=} (k - 2)n$, $\beta \stackrel{\text{lg}}{=} n$, $\gamma \stackrel{\text{lg}}{=} 0$ (as in Theorem 3.1), then for every positive $\delta \leq \beta/2$ the omniscience protocol is also optimal for some triples of inputs (x', y', z') with symmetric complexity profile $(\alpha', \beta', \gamma') = (\alpha, \beta + \delta, \gamma - 3\delta)$.*

In Lemma 7.1 we show that the existence of an “excessively efficient” protocol for triples of inputs with modified symmetric profiles $(\alpha', \beta', \gamma')$ would imply an “excessively efficient” protocol for the original symmetric profile (9), which is impossible due to Theorem 3.1. The proof of this lemma uses mostly techniques of Kolmogorov complexity that are not specific for communication problems. The argument is based on repeated application of Muchnik’s theorem on conditional descriptions ([23]). Due to the lack of space, the proof of the lemma is deferred to full version of the paper.

It is not hard to verify that starting with a triple (x, y, z) from Theorem 3.1 and then applying the reductions from Lemma 7.1, we can obtain any realizable profiles (2). Indeed, we begin with a triple of pairwise orthogonal directions (x, y, z) with $\alpha = (k - 2)n$, $\beta = n$, $\gamma = 0$ for a suitable n and k , then apply Lemma 7.1 (ii) or Lemma 7.1 (iii) to get a triple (x', y', z') with a suitable $I(x' : y' : z')$ (case (ii) serves to make the triple mutual information positive, and case (iii) is needed if we want to make it negative), and further apply Lemma 7.1 (i) to trim the value of α . Thus, Theorem 3.1 implies optimality of (7) for triples of inputs (x, y, z) with arbitrary symmetric complexity profile (2). ◀

8 Conclusion and open problems

We proved that the standard omniscience protocol provides the optimal worst-case communication complexity of the problem of secret key agreement (with three parties) in the class of protocols with simultaneous messages. A natural direction for further research is the study of the limits of our approach. A more specific open problem is to settle the communication complexity of multi-party secret key agreement for multi-round protocols. Indeed, in the multi-party settings, the existing communication protocols can be actually improved at the cost of increasing the number of rounds. In particular, the communication complexity (7) is no longer the optimal for multi-round protocols:

► **Proposition 8.1.** *In the setting of Theorem 3.1 there is a multi-round communication protocol (not a simultaneous messages protocol) with communication complexity $(2k - 2.5)n + O(\log n)$, where the parties agree on a secret key of the optimal size $1.5n - O(\log n)$.*

(See the proof in the full version of the paper.) Our technique implies *some* lower bounds for communication complexity of interactive protocols, but this bound does not match the known upper bounds.

Another open problem is to establish the trade-off between the size of the secret key and the optimal communication complexity. For two-parties protocols, communication complexity of secret key agreement cannot be reduced even if Alice and Bob agree on a very small secret key, see the “threshold phenomenon” in [15]; for protocols with $k \geq 3$ parties the situation is different, and communication complexity may be improved if the size of the secret key is suboptimal. It would be also interesting to extend our results to the communication model with private sources of randomness.



References

- 1 Rudolf Ahlswede and Imre Csiszár. Common randomness in information theory and cryptography - I: secret sharing. *IEEE Trans. Inf. Theory*, 39(4):1121–1132, 1993. doi:10.1109/18.243431.
- 2 Luis Antunes, Sophie Laplante, Alexandre Pinto, and Liliana C. M. Salvador. Cryptographic security of individual instances. In Yvo Desmedt, editor, *Information Theoretic Security - Second International Conference, ICITS 2007, Madrid, Spain, May 25-29, 2007, Revised Selected Papers*, volume 4883 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2007. doi:10.1007/978-3-642-10230-1_17.
- 3 László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory (preliminary version). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 337–347. IEEE Computer Society, 1986. doi:10.1109/SFCS.1986.15.
- 4 Charles H. Bennett, François Bessette, Gilles Brassard, Louis Salvail, and John A. Smolin. Experimental quantum cryptography. *J. Cryptol.*, 5(1):3–28, 1992. doi:10.1007/BF00191318.
- 5 Matthieu Bloch, Onur Günlü, Aylin Yener, Frédérique Oggier, H. Vincent Poor, Lalitha Sankar, and Rafael F. Schaefer. An overview of information-theoretic security and privacy: Metrics, limits and applications. *IEEE Journal on Selected Areas in Information Theory*, 2(1):5–22, 2021. doi:10.1109/JSAIT.2021.3062755.
- 6 Imre Csiszár and Prakash Narayan. Secrecy capacities for multiple terminals. *IEEE Trans. Inf. Theory*, 50(12):3047–3061, 2004. doi:10.1109/TIT.2004.838380.
- 7 Igor Devetak and Andreas Winter. Distillation of secret key and entanglement from quantum states. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 461(2053):207–235, January 2005. doi:10.1098/rspa.2004.1372.

- 8 Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976. doi:10.1109/TIT.1976.1055638.
- 9 Yan Zong Ding. Error correction in the bounded storage model. In Joe Kilian, editor, *Theory of Cryptography*, pages 578–599, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 10 Yevgeniy Dodis, Bhavana Kanukurthi, Jonathan Katz, Leonid Reyzin, and Adam Smith. Robust fuzzy extractors and authenticated key agreement from close secrets. *IEEE Transactions on Information Theory*, 58(9):6207–6222, 2012. doi:10.1109/TIT.2012.2200290.
- 11 Yevgeniy Dodis and Adam D. Smith. Correcting errors without leaking partial information. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 654–663. ACM, 2005. doi:10.1145/1060590.1060688.
- 12 Shai Evra, Konstantin Golubev, and Alexander Lubotzky. Mixing properties and the chromatic number of ramanujan complexes, 2014. arXiv:1407.7700.
- 13 Noah Golowich and Madhu Sudan. Round complexity of common randomness generation: The amortized setting. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1076–1095. SIAM, 2020. doi:10.1137/1.9781611975994.66.
- 14 Peter Grünwald and Paul M. B. Vitányi. Shannon information and kolmogorov complexity. *CoRR*, cs.IT/0410002, 2004. URL: <http://arxiv.org/abs/cs.IT/0410002>.
- 15 Emirhan Gürpınar and Andrei E. Romashchenko. Communication complexity of the secret key agreement in algorithmic information theory. In Javier Esparza and Daniel Král’, editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPICs*, pages 44:1–44:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.MFCS.2020.44.
- 16 Ryszard Horodecki, Paweł Horodecki, Michał Horodecki, and Karol Horodecki. Quantum entanglement. *Reviews of Modern Physics*, 81(2):865–942, June 2009. doi:10.1103/revmodphys.81.865.
- 17 Tanya Ignatenko and Frans M. J. Willems. Biometric security from an information-theoretical perspective. *Found. Trends Commun. Inf. Theory*, 7:135–316, 2012. URL: <https://api.semanticscholar.org/CorpusID:51848802>.
- 18 A. Wigderson J. Friedman. On the second eigenvalue of hypergraphs. *Combinatorica* 15, pages 43–65, 1995. doi:10.1007/BF01294459.
- 19 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.
- 20 Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition*. Texts in Computer Science. Springer, 2019. doi:10.1007/978-3-030-11298-1.
- 21 Ueli M. Maurer. Secret key agreement by public discussion from common information. *IEEE Trans. Inf. Theory*, 39(3):733–742, 1993. doi:10.1109/18.256484.
- 22 Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, 1978. doi:10.1145/359460.359473.
- 23 Andrei A. Muchnik. Conditional complexity and codes. *Theor. Comput. Sci.*, 271(1-2):97–109, 2002. doi:10.1016/S0304-3975(01)00033-0.
- 24 Andrei E. Romashchenko and Marius Zimand. An operational characterization of mutual information in algorithmic information theory. *J. ACM*, 66(5):38:1–38:42, 2019. doi:10.1145/3356867.
- 25 A. Wigderson S. Hoory, N. Linial. Expander graphs and their applications. *Bulletin of the American Mathematical Society* 43, pages 439–561, August 2006.
- 26 Alexander Shen, Vladimir Andreevich Uspensky, and Nikolay Vereshchagin. *Kolmogorov Complexity and Algorithmic Randomness*. American Mathematical Society, 2017. URL: <https://hal-lirmm.ccsd.cnrs.fr/lirmm-01803620>.

- 27 Benjamin Smith. Pre- and post-quantum diffie-hellman from groups, actions, and isogenies. In Lilya Budaghyan and Francisco Rodríguez-Henríquez, editors, *Arithmetic of Finite Fields - 7th International Workshop, WAIFI 2018, Bergen, Norway, June 14-16, 2018, Revised Selected Papers*, volume 11321 of *Lecture Notes in Computer Science*, pages 3–40. Springer, 2018. doi:10.1007/978-3-030-05153-2_1.
- 28 Madhu Sudan, Badih Ghazi, Noah Golowich, and Mitali Bafna. Communication-rounds tradeoffs for common randomness and secret key generation. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1861–1871. SIAM, 2019. doi:10.1137/1.9781611975482.112.
- 29 Madhu Sudan, Himanshu Tyagi, and Shun Watanabe. Communication for generating correlation: A unifying survey. *IEEE Trans. Inf. Theory*, 66(1):5–37, 2020. doi:10.1109/TIT.2019.2946364.

Fault-tolerant k -Supplier with Outliers

Deeparnab Chakrabarty  

Department of Computer Science, Dartmouth College, Hanover, NH, USA

Luc Cote  

Thayer School of Engineering, Dartmouth College, Hanover, NH, USA

Ankita Sarkar   

Department of Computer Science, Dartmouth College, Hanover, NH, USA

Abstract

We present approximation algorithms for the Fault-tolerant k -Supplier with Outliers ($FkSO$) problem. This is a common generalization of two known problems – k -Supplier with Outliers, and Fault-tolerant k -Supplier – each of which generalize the well-known k -Supplier problem. In the k -Supplier problem the goal is to serve n clients C , by opening k facilities from a set of possible facilities F ; the objective function is the farthest that any client must travel to access an open facility. In $FkSO$, each client v has a *fault-tolerance* ℓ_v , and now desires ℓ_v facilities to serve it; so each client v 's contribution to the objective function is now its distance to the ℓ_v^{th} closest open facility. Furthermore, we are allowed to choose m clients that we will serve, and only those clients contribute to the objective function, while the remaining $n - m$ are considered outliers.

Our main result is a $(4t - 1)$ -approximation for the $FkSO$ problem, where t is the number of distinct values of ℓ_v that appear in the instance. At $t = 1$, i.e. in the case where the ℓ_v 's are uniformly some ℓ , this yields a 3-approximation, improving upon the 11-approximation given for the uniform case by Inamdar and Varadarajan [2020], who also introduced the problem. Our result for the uniform case matches tight 3-approximations that exist for k -Supplier, k -Supplier with Outliers, and Fault-tolerant k -Supplier.

Our key technical contribution is an application of the *round-or-cut* schema to $FkSO$. Guided by an LP relaxation, we reduce to a simpler optimization problem, which we can solve to obtain distance bounds for the “round” step, and valid inequalities for the “cut” step. By varying how we reduce to the simpler problem, we get varying distance bounds – we include a variant that gives a $(2^t + 1)$ -approximation, which is better for $t \in \{2, 3\}$. In addition, for $t = 1$, we give a more straightforward application of round-or-cut, yielding a 3-approximation that is much simpler than our general algorithm.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Clustering, approximation algorithms, round-or-cut

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.23

Funding *Deeparnab Chakrabarty*: Partially supported by NSF grant CCF-2041920.

Ankita Sarkar: Partially supported by NSF grant CCF-2041920.

1 Introduction

Clustering problems form a class of discrete optimization problems that appear in many application areas ranging from operations research [28, 32, 30] to machine learning [21, 1, 31, 24]. They also have formed a sandbox where numerous algorithmic ideas, especially ideas in approximation algorithms, have arisen and developed over the years. One of the first clustering problems to have been studied is the k -Supplier problem [19]: in this problem, one is given a set of points in a metric space $(C \cup F, d)$, where C is the set of “clients” and F is the set of “facilities”, and a number k . The objective is to “open” a collection $S \subseteq F$ of k centers so as to minimize the maximum distance between a client $v \in V$ to its nearest



© Deeparnab Chakrabarty, Luc Cote, and Ankita Sarkar;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 23; pp. 23:1–23:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



open center in S , that is, minimize $\max_{v \in C} \min_{f \in S} d(f, v)$. It has been known since the mid-80's, due to an influential paper of Hochbaum and Shmoys [19], that this problem has a 3-approximation and no better approximation is possible¹.

One motivation behind the objective function above is that $d(v, S) := \min_{f \in S} d(f, v)$ indicates how (un)desirable the client v perceives the the set of open facilities, and the k -Supplier objective tries to take the egalitarian view of trying to minimize the unhappiest client. However, in certain applications, a client v would perhaps be interested not only in having one open facility in a small neighborhood but a larger number. For instance, the client may be worried about some open facilities closing down. This leads to the *fault-tolerant* versions of clustering problems. In this setting, each client v has an integer ℓ_v associated with it, and the desirability of a subset S for v is not determined by the nearest facility in S , but rather the ℓ_v^{th} nearest facility. That is, we sort the facilities in S in increasing order of $d(f, v)$ and let $d_{\ell_v}(v, S)$ denote the ℓ_v^{th} distance in this order (so $d(v, S) = d_1(v, S)$). The Fault-tolerant k -Supplier (FkS) problem is to find $S \subseteq F$ with $|S| = k$ so as to minimize $\max_{v \in C} d_{\ell_v}(v, S)$. As far as we know, the Fault-tolerant k -Supplier problem has not been *explicitly* studied in the literature², however, as we show in Section 2, there is a simple 3-approximation based on the same scheme developed by Hochbaum and Shmoys [19].

One drawback of the k -Supplier objective is that it is extremely sensitive to outliers; since one is trying to minimize the maximum, a single far-away client makes the optimal value large. To allay this, people have considered the “outlier version” of the problem, k -Supplier with Outliers (k SO). In k SO, one is given an additional integer parameter m , and the goal of the algorithm is to open a subset S of k facilities *and* recognize a subset $T \subseteq C$ of m -clients, so as to minimize $\max_{v \in T} d(v, S)$. That is, all clients outside T are deemed outliers and one doesn't consider their distance to the solution. The outlier version is algorithmically interesting and is not immediately captured by the Hochbaum-Shmoys technique. Nevertheless, in 2001, Charikar, Khuller, Mount, and Narsimhan [10] described a combinatorial, greedy-like 3-approximation for k SO³. Since then, outlier versions of many clustering problems have been considered, and it has been a curious feature that the approximability of the outlier version has been of the same order as the approximability of the original version without outliers.

In this paper, as suggested by the title, we study the *Fault-tolerant k -Supplier with Outliers* (FkSO) problem which generalizes FkS and k SO. This problem was explicitly studied only recently by Inamdar and Varadarajan [20]; but that work only studies the *uniformly fault-tolerant* case where all ℓ_v 's are the same (say, ℓ). The main result of [20] was a “reduction” to the “non-fault-tolerant” version of the clustering problem with outliers, and their result is that an α -approximation for the k SO problem translates⁴ to a $(3\alpha + 2)$ -approximation for the FkSO problem with uniform fault-tolerance. Setting $\alpha = 3$ from the aforementioned work [10] on k SO, one gets an 11-approximation for the uniform case of FkSO.

¹ Howard Karloff is attributed the hardness result in [19].

² although the fault-tolerant facility location and k -median have been extensively studied [22, 17, 36, 18]; more on this in Section 1.1.

³ A different LP-based approach was taken by Chakrabarty, Goyal, and Krishnaswamy [6] and vastly generalized by Chakrabarty and Negahbani [7]; more on this in Sections 1.1 and 2.

⁴ Actually, they [20] only study the “ k -Center” case when $F = C$, and in that case the result is $(2\alpha + 2)$; their proofs do reveal that for the Supplier version, one obtains $(3\alpha + 2)$.

Our Contributions

We begin by providing a simple LP-based 3-approximation for the $FkSO$ problem when the fault-tolerances are uniform, that is, $\ell_v = \ell$ for all $v \in C$. This improves the known 11-approximation [20]. Even this special case is interesting in that when the uniform fault-tolerance ℓ divides k , then the “natural LP” suffices to obtain a 3-approximation using a rounding scheme similar to a prior rounding algorithm for kSO [6]. However, when ℓ doesn’t divide k , then we need to add in valid inequalities akin to Chvátal-Gomory cuts [13, 16] in integer programming. Nevertheless, the rounding algorithm is simple and is described in Section 3.

Our main contribution is to the general $FkSO$ problem, when ℓ_v ’s can be different for different clients. This problem becomes much more complex for the simple reason that if two clients v and v' are located very close together, but $\ell_v < \ell_{v'}$, then opening ℓ_v facilities around v would still render v' unhappy – this does not happen in the uniform case. Therefore, the Hochbaum-Shmoys procedure [19], or more precisely the LP-guided Hochbaum-Shmoys rounding that is known for kSO [6], simply doesn’t apply under non-uniform fault-tolerance. Indeed, the natural LP relaxation and its natural strengthening, which give us the 3-approximation for the uniform case, has large integrality gaps even when the ℓ_v ’s take only two values; we show this in Section 3.1.

Our main result is a $(4t - 1)$ -approximation for the $FkSO$ problem when there are t distinct⁵ ℓ_v ’s (that is, $|\{\ell_v : v \in C\}| = t$). When $t = 1$, we recover the 3-approximation mentioned above. This is not the most desirable result (one would hope a $O(1)$ -approximation for any t), but as the above integrality gap example illustrates, even when $t = 2$, strong LPs have bad integrality gaps. We also use the same schema to give a $(2^t + 1)$ -approximation, which gives better approximation factors for $t \in \{2, 3\}$.

Our main technical contribution is to apply the *round-or-cut* schema introduced for clustering problems by Chakrabarty and Negahbani [7] to $FkSO$. In particular, this schema uses a fractional solution $\{\text{cov}_v\}_{v \in C}$ which indicates the extent to which each client v is an “inlier” (that is, in the final set T of at least m clients). Earlier works [6, 7] use this fractional solution to guide the Hochbaum-Shmoys-style [19] rounding algorithm, creating a partition on the set of clients and solving a simpler optimization problem on this partition. We also use the same schemata, except that our partitioning scheme is a more general one warranted by the non-uniform fault-tolerances; nevertheless, we show that we either obtain the desired approximation factor (the “round” step), or we can prove that the cov_v ’s cannot arise as a combination of integral solutions (the “cut” step). Once we do this, the round-or-cut schema implies a polynomial time approximation factor. We also show that t is the limiting factor in our approach; more precisely, the diameter of the parts of the desired partition dictates the upper bound on the approximation factor, and in Appendix B we construct an instance such that the diameter needs to be $\Omega(t)$. We leave the possibility of obtaining $O(1)$ -approximations for $FkSO$, or alternately proving a super-constant hardness, as an intriguing open problem.

⁵ We should point to the reader that one can’t simply solve t different uniform $FkSO$ versions and “stick them together” to get such a result; although it is a natural idea, note that a priori we do not know how many outliers one will obtain from each fault-tolerant class, and enumerating is infeasible.

1.1 Related Work

The Hochbaum-Shmoys algorithm [19] gives a 3-approximation for the k -Supplier problem, and has been extended to give approximation algorithms for multiple related problems. Plesník [33] gave one such extension, obtaining a 3-approximation⁶ when each client v has weight $w(v)$, and this scales the client’s “unhappiness”, so that the objective function becomes $\max_{v \in C} (w(v) \cdot \min_{f \in S} d(v, f))$. In another direction, Chakrabarty, Goyal, and Krishnaswamy [6] gave an extension to k -Supplier with Outliers, using an LP relaxation to indicate which clients are outliers, and obtaining a Hochbaum-Shmoys-like 3-approximation. This was vastly extended by Chakrabarty and Negahbani [7], implying a 3-approximation for multiple problems, including k SO with knapsack constraints on the facilities. Bajpai, Chekuri, Chakrabarty, and Negahbani [4] generalized the aforementioned weighted version [33] to handle outliers, matroid constraints, and knapsack constraints, obtaining constant approximation ratios for each.

In the early 2000s, Jain and Vazirani [22] introduced the notion of fault-tolerance for the Uncapacitated Facility Location (UFL) problem. The notion has thereafter been studied for various related problems: UFL [17, 36, 5]; UFL with multiset solutions, often called facility *placement* or *allocation* [37, 38, 34]; k -Median [36, 27, 18]; matroid and knapsack Median [15]; and k -Center [26, 11, 25, 27]. In particular relevance to this paper, the Fk SO problem was studied by Inamdar and Varadarajan [20]. In addition, prior work also addresses alternate notions of fault-tolerance and outlier-type constraints. In a 2020 preprint, Deng [14] combines fault-tolerance with an outlier-type constraint requiring that the number of *client-facility connections*, rather than the weight of satisfied clients, be at least some m . An altogether different notion of fault-tolerance has also been studied [12, 29, 35], where clients each want just one facility, but an adversary secretly causes some $k' \leq k$ of the chosen facilities to fail.

The round-or-cut schema that this paper applies, has found widespread usage in clustering problems, including in problems related to k -Supplier. For example, the weighted version of k -Supplier [33, 4] can be extended to impose different budgets to different weight classes – i.e. there is no longer one k , but one k_i per distinct weight w_i . This version admits a constant-factor approximation for certain special cases [8, 23] via the round-or-cut schema. Round-or-cut has also been used for k -Supplier with covering constraints [3], and for the Capacitated Facility Location problem [2]. In the continuous clustering realm, where facilities can be picked from a potentially infinite-sized ambient metric space, round-or-cut has been used to circumvent the infinitude of the instance [9].

2 Preliminaries

Before we formally define our main problem, let us set up some important notation.

► **Definition 1.** *Given a subset $S \subseteq F$, a client $v \in C$, and $a \in [k]$, let $d_a(v, S)$ be the distance of v to its a^{th} closest neighbor in S (breaking ties arbitrarily and consistently). So $d_1(v, S) = d(v, S)$. Also let $N_a(v, S) \subseteq S$ denote the a facilities in S that are closest to v .*

► **Definition 2 (Fault-tolerant k -Supplier and Fault-tolerant k -Supplier with Outliers).** *In the Fault-tolerant k -Supplier (FkS) problem, we are given a finite metric space $(C \cup F, d)$, where C is a set of n clients and F is a set of $\text{poly}(n)$ facilities. We are also given a parameter $k \in \mathbb{N}$, and fault-tolerances $\{\ell_v \in [k]\}_{v \in C}$. The goal is to open k facilities, i.e. pick $S \subseteq F : |S| \leq k$, minimizing $\max_{v \in C} d_{\ell_v}(v, S)$.*

⁶ The work [33] studies the k -Center case, where $F = C$, and gives a 2-approximation; but the proofs imply a 3-approximation for the Supplier version.

In the Fault-tolerant k -Supplier with Outliers (FkSO) problem, we are given an FkS instance along with an additional parameter $m \in [n]$. The goal is to pick S of size k as before, along with inliers $T \subseteq C : |T| \geq m$, minimizing $\max_{v \in T} d_{\ell_v}(v, S)$.

In the absence of fault-tolerances and outliers, i.e. in the k -Supplier problem, the Hochbaum-Shmoys algorithm [19] achieves a 3-approximation as follows. It starts with a guess r of the optimum value, large enough that every client j has a facility within distance r of itself, but otherwise arbitrary. Then it picks an arbitrary client j , opens a facility within distance r of j , and deletes the set of “children” of j , which is $\text{child}(j) := B(j, 2r) \cap C = \{v \in C : d(v, j) \leq 2r\}$. Then it repeats this with the remaining clients, until there are no clients left. Observe that the j 's picked over the iterations – call them the set R – has the following *well-separated* property.

► **Definition 3** (r -well-separated set). A set $X \subseteq C$ is r -well-separated if for distinct $x, y \in X$, we have $d(x, y) > 2r$. Where r is clear from context, we simply say that X is well-separated.

Since R is well-separated, it takes $|R|$ clients to provide every $j \in R$ with a facility in $B(j, r)$. So if $|R| > k$, then the guess of r is too small – we can double r and retry the algorithm. On the other hand, if $|R| \leq k$, then the guess is either correct or too large, so we halve r and retry. This binary search yields the correct r , and the following guarantee: $\{\text{child}(j)\}_{j \in R}$ partitions C , and for a $v \in \text{child}(j)$, since $d(v, j) \leq 2r$ and we opened a facility in $B(j, r)$, there is a facility within distance $3r$ of v . This means that we have a 3-approximation.

The Hochbaum-Shmoys algorithm described above, generalizes to give a 3-approximation for FkS via the following modifications: instead of picking j 's into R in arbitrary order, we pick them in decreasing order of ℓ_v 's; we also open ℓ_j facilities in each $B(j, r) : j \in R$, instead of just one. This guarantees that, if $v \in \text{child}(j)$, $\ell_v \leq \ell_j$, allowing us to extend the Hochbaum-Shmoys [19] guarantee to FkS. We now formally state this algorithm.

■ **Algorithm 1** Hochbaum-Shmoys [19] modified for FkS.

Input: C

```

1:  $U \leftarrow C$ 
2:  $R \leftarrow \emptyset$ 
3:  $S \leftarrow \emptyset$ 
4: while  $U \neq \emptyset$  do
5:    $j \leftarrow \text{argmax}_{v \in U} \ell_v$ 
6:    $R \leftarrow R \cup \{j\}$ 
7:    $i_1, i_2, \dots, i_{\ell_j} \leftarrow \ell_j$  arbitrary facilities in  $B(j, r) \cap F$    ▷ they exist by choice of  $r$ 
8:    $S \leftarrow S \cup \{i_1, i_2, \dots, i_{\ell_j}\}$ 
9:    $\text{child}(j) \leftarrow \{v \in U : d(v, j) \leq 2r\}$ 
10:   $U \leftarrow U \setminus \text{child}(j)$ 

```

Output: $S \subseteq F$

To show that this algorithm yields a 3-approximation, we need to argue that $\forall v \in C$, $d_{\ell_v}(v, S) \leq 3r$. To see this, consider $j \in R : v \in \text{child}(j)$. Algorithm 1 guarantees that $\ell_v \leq \ell_j$, so $d_{\ell_v}(v, S) \leq d_{\ell_j}(v, S)$. By triangle inequalities, this is at most $d(v, j) + d_{\ell_j}(j, S)$. By construction of $\text{child}(j)$, $d(v, j) \leq 2r$; and since we open ℓ_j facilities in Line 7, $d_{\ell_j}(j, S) \leq r$. We have just shown that

► **Theorem 4.** *The FkS problem admits a 3-approximation.*

One way to achieve a 3-approximation for the k -Supplier with Outliers problem, described in [6], is as follows: under a guess of r as before, a linear program relaxation is used to assign variables $\text{cov}_v \in [0, 1]$ to each client v , representing whether v is “covered”, i.e. whether there is an open facility in $B(v, r)$. The LP-guided Hochbaum-Shmoys algorithm considers clients in decreasing order of these cov_v 's, and we wait to pick facilities until the loop terminates. Then, facilities are opened near those $j \in R$ that have the k largest $|\text{child}(j)|$. The LP relaxation is used to ensure that $\geq m$ clients are served in this way. This *does not* generalize directly to $FkSO$ because the decreasing order of ℓ_v 's that we employed above for FkS can conflict with the decreasing order of cov_v 's (indeed, one may just expect clients v with large ℓ_v 's would be more likely to be outliers, that is, have low cov_v 's). So in our algorithm for $FkSO$, we elect to follow the cov_v order, and explicitly force $\ell_v \leq \ell_j$ for v 's that we pick into $\text{child}(j)$. This choice breaks the well-separated property of R , so our techniques are devoted to obtaining other well-separated sets that can guide our rounding; details of this can be found in Section 4. When all the ℓ_v 's are the same, though, we can indeed use the natural LP relaxation (with a slight strengthening to take care of divisibility issues), and we show this in the next section.

3 3-approximation for $UFkSO$

In this section, we address the *uniform* case, where all fault-tolerances in the instance are the same, i.e.

► **Definition 5** (Uniformly Fault-tolerant k -Supplier with Outliers ($UFkSO$)). *The Uniformly Fault-tolerant k -Supplier with Outliers problem is a special case of the $FkSO$ problem where, for an $\ell \in \mathbb{N}$, $\forall v \in C$, $\ell_v = \ell$.*

We prove that

► **Theorem 6.** *The $UFkSO$ problem admits a 3-approximation.*

Our algorithm begins by rounding a solution to the following LP relaxation, closely mimicking the 3-approximation for kSO [6] described in the last paragraph of the previous section. This rounding suffices when $\ell \mid k$. When $\ell \nmid k$, we identify a valid inequality for the round-or-cut framework. In the LP, the variables $\{\text{cov}_v\}_{v \in C}$ denote whether or not a client $v \in C$ is *covered* i.e. served within distance r ; and variables $\{x_i\}_{i \in F}$ denote whether or not a facility $i \in F$ is open. $B(v, r)$ is the *ball* of radius r around v , containing all points within distance r of v , i.e. $B(v, r) := \{x \in C \cup F : d(v, x) \leq r\}$.

$$\sum_{v \in C} \text{cov}_v \geq m \quad (\text{WL1})$$

$$\sum_{i \in F} x_i \leq k \quad (\text{WL2})$$

$$\forall v \in C, \quad \sum_{i \in F \cap B(v, r)} x_i \geq \ell \text{cov}_v \quad (\text{WL3})$$

$$\forall v \in C : d_\ell(v, F) > r, \quad \text{cov}_v = 0 \quad (\text{WL4})$$

$$\forall v \in C, i \in F, \quad 0 \leq \text{cov}_v, x_i \leq 1 \quad (\text{WL5})$$

Here, (WL1) enforces that at least m clients must be covered, and (WL2) enforces that at most k facilities can be opened. (WL3) and (WL4) connect the cov_v variables with the x_i variables, ensuring that a client cannot be covered unless there are sufficient facilities opened

within distance r of it. Finally, (WL5) enforces that a client can be covered only once, and a facility can be opened only once. Claim 7 shows that this LP is a valid relaxation of our problem. We defer its proof to Appendix A.

▷ **Claim 7.** An instance of $UFkSO$ is feasible iff it admits an integral solution satisfying (WL1)-(WL5).

Given a solution $(\{\text{cov}_v\}_{v \in C}, \{x_i\}_{i \in F})$ satisfying (WL1)-(WL5), we round as per Algorithm 2. This algorithm constructs a well-separated set of *representatives* $R_{\text{cov}} \subseteq C$. Each client v that has $\text{cov}_v > 0$ becomes the *child* of some representative, yielding a partition $\{\text{child}(j)\}_{j \in R_{\text{cov}}}$ of these clients. Then, facilities $S_{\text{cov}} \subseteq F$ are opened in a manner that serves the $\lfloor \frac{k}{\ell} \rfloor$ largest $\text{child}(j)$ sets within distance $3r$.

■ **Algorithm 2** 3-approximation for $UFkSO$.

Input: $(\{\text{cov}_v\}_{v \in C}, \{x_i\}_{i \in F})$ satisfying (WL1)-(WL5)

- 1: $R_{\text{cov}} \leftarrow \emptyset$
- 2: $U \leftarrow \{v \in C : \text{cov}_v > 0\}$
- 3: **while** $U \neq \emptyset$ **do** ▷ filtering
- 4: $j \leftarrow \text{argmax}_{v \in U} \text{cov}_v$
- 5: $R_{\text{cov}} \leftarrow R_{\text{cov}} \cup \{v\}$
- 6: $\text{child}(j) \leftarrow B(j, 2r) \cap U$
- 7: $U \leftarrow U \setminus \text{child}(j)$
- 8: $S_{\text{cov}} \leftarrow \emptyset$
- 9: $R' \leftarrow R_{\text{cov}}$
- 10: **while** $|S_{\text{cov}}| < \lfloor \frac{k}{\ell} \rfloor \cdot \ell$ **do** ▷ picking facilities to open
- 11: $j \leftarrow \text{argmax}_{j' \in R'} |\text{child}(j')|$
- 12: $R' \leftarrow R' \setminus \{j\}$
- 13: $S_{\text{cov}} \leftarrow S_{\text{cov}} \cup N_\ell(j, F)$
- 14: **return** S_{cov}

Output: $S_{\text{cov}} \subseteq F$ ▷ open facilities

We argue that Algorithm 2 opens at most k facilities, and that if $N_\ell(j, F)$ is opened, then $\text{child}(j)$ is served within distance $3r$. Formally,

- ▶ **Lemma 8.** Given $(\{\text{cov}_v\}_{v \in C}, \{x_i\}_{i \in F})$ satisfying (WL1)-(WL5),
- $|S_{\text{cov}}| \leq k$, and
 - Let R'_{cov} be the clients j for which $N_\ell(j, F)$ was added to S_{cov} in Line 13. Then $\forall j \in R'_{\text{cov}}, \forall v \in \text{child}(j), d_\ell(v, S_{\text{cov}}) \leq 3r$.

Proof.

- Line 13 adds ℓ facilities to S_{cov} in each iteration. So Line 10 ensures that $|S_{\text{cov}}| \leq k$.
- Consider $v \in \text{child}(j), j \in R'_{\text{cov}}$. By triangle inequalities, $d_\ell(v, S_{\text{cov}}) \leq d(v, j) + d_\ell(j, S_{\text{cov}})$. By Line 6, $d(v, j) \leq 2r$. Since $N_\ell(j, F) \subseteq S_{\text{cov}}, d_\ell(j, S_{\text{cov}}) \leq d_\ell(j, F)$; and by Line 2, $\text{cov}_j > 0$, i.e. by (WL4), $d_\ell(j, F) \leq r$. ◀

It remains to show that $\sum_{j \in R'_{\text{cov}}} |\text{child}(j)| \geq m$. We have

$$\sum_{j \in R_{\text{cov}}} |\text{child}(j)| \text{cov}_j \geq \sum_{v \in C: \text{cov}_v > 0} \text{cov}_v = \sum_{v \in C} \text{cov}_v \geq m, \quad (1)$$

where the first inequality is by Line 4 and the last inequality is by (WL1). We also have

$$\sum_{j \in R_{\text{cov}}} \text{cov}_j \leq \sum_{j \in R_{\text{cov}}} \sum_{i \in F \cap B(j,r)} \frac{x_i}{\ell} \leq \sum_{i \in F} \frac{x_i}{\ell} \leq \frac{k}{\ell}, \quad (2)$$

where the first inequality is by (WL3); the second inequality is because R_{cov} is well-separated; and the last inequality is by (WL2). So we can view the LHS in (1) as a weighted sum of $|\text{child}(j)|$ values, the weights being cov_j 's. Since this weighted sum is $\geq m$ and the weights sum to $\leq k/\ell$, the $\lfloor \frac{k}{\ell} \rfloor$ largest child-sets must contain at least $\frac{m}{k/\ell} \cdot \lfloor \frac{k}{\ell} \rfloor$ elements. Hence, if $\ell \mid k$, we are done.

In fact, we observe the following even when $\ell \nmid k$: if we can replace the RHS in (2) with $\lfloor \frac{k}{\ell} \rfloor$, then the weighted-sum argument would yield $\frac{m}{\lfloor \frac{k}{\ell} \rfloor} \cdot \lfloor \frac{k}{\ell} \rfloor = m$. To achieve this, observe that the argument in (2) applies to any well-separated set $R \subseteq C$, yielding $\sum_{j \in R} \text{cov}_j \leq k/\ell$. Also, for any integral solution, the RHS can be replaced by its floor. Thus the following are valid inequalities:

$$\forall R \subseteq C : R \text{ is well-separated, } \sum_{j \in R} \text{cov}_j \leq \left\lfloor \frac{k}{\ell} \right\rfloor. \quad (\text{WLCut})$$

We have showed that if (WLCut) holds for $R = R_{\text{cov}}$ then we are done, i.e.

► **Lemma 9.** *Given $(\{\text{cov}_v\}_{v \in C}, \{x_i\}_{i \in F})$ satisfying (WL1)-(WL5), if (WLCut) holds for $R = R_{\text{cov}}$ where R_{cov} is constructed as per Lines 1-7, then S_{cov} is a 3-approximation.*

Using this, we now present our overall algorithm via a round-or-cut schema.

Proof of Theorem 6. Given $(\{\text{cov}_v\}_{v \in C}, \{x_i\}_{i \in F})$ satisfying (WL1)-(WL5), we round as per Lines 1-7 to obtain $R_{\text{cov}} \subseteq C$. If (WLCut) holds for $R = R_{\text{cov}}$, then we continue Algorithm 2 to obtain S_{cov} that satisfies the desired guarantees via Lemmas 8 and 9. Otherwise, we know that the valid inequality (WLCut) for $R = R_{\text{cov}}$ is violated. So we pass it to the ellipsoid algorithm as a separating hyperplane, obtaining fresh cov_v 's with which we restart Algorithm 2. By the guarantees of the ellipsoid algorithm, in polynomial time, we either round to get S_{cov} , or detect that the guess of r is too small. ◀

We conclude this section by exhibiting that the above algorithm fails for the general problem. In particular, we exhibit an infinite integrality gap when there are just two different fault-tolerances in the instance.

3.1 Gap example for $Fk\text{SO}$

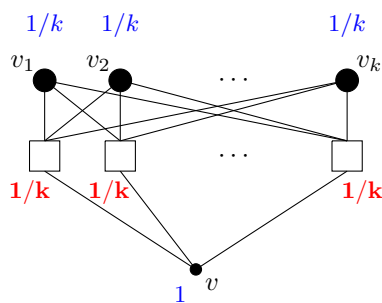
Consider (WL3) generalized to $Fk\text{SO}$:

$$\forall v \in C, \quad \sum_{i \in F \cap B(v,r)} x_{i_v} \geq \ell_v \text{cov}_v; \quad (\text{WL3}')$$

and a similar generalization of (WLCut):

$$\forall R \subseteq C : R \text{ is well-separated, } \left\lceil \sum_{v \in R} \ell_v \text{cov}_v \right\rceil \leq k. \quad (\text{WLCut}')$$

These, along with (WL1)-(WL2) and (WL4)-(WL5), generalize the earlier LP to $Fk\text{SO}$. We now show an infinite integrality gap w.r.t. this LP.



■ **Figure 1** One of the k identical gadgets in the gap example, showing LP values in **red** (x values) and **blue** (cov values). The “edges” represent distance 1, and all other distances are determined by making triangle inequalities tight. The fault-tolerances are $\ell_{v_1} = \ell_{v_2} = \dots = \ell_{v_k} = k$, and $\ell_v = 1$.

Consider k identical gadgets, each like in Figure 1, infinitely apart from each other. Let $m = 2k$. The small client in each gadget (v in Figure 1) has fault-tolerance 1. The big clients in each gadget (v_1, v_2, \dots, v_k in Figure 1) have fault-tolerance k . Within a gadget, an integral solution only benefits from either picking one facility to serve just the small client, or picking all facilities to serve all $(k + 1)$ clients. So over all gadgets, an integral solution can either pick one facility per gadget, or pick all facilities in exactly one gadget, either way serving $k < m$ clients. Since all facilities are within distance 1 of the clients in their gadget, the above is true for an integral solution with any radius dilation $\alpha \geq 1$.

But the LP can assign $x_i = 1/k$ to each of the k^2 facilities in the instance. This allows it to assign $\text{cov}_v = 1$ to all the small clients, and $\text{cov}_{v_1} = \text{cov}_{v_2} = \dots = \text{cov}_{v_k} = 1/k$ to all the big clients, thus serving $k \cdot 1 + k^2 \cdot \frac{1}{k} = 2k = m$ clients.

4 Fault-tolerant k -Supplier with Outliers

In this section, we address FkSO in its full generality. We use t to denote the number of distinct fault-tolerances in the instance, i.e. $|\{\ell_v : v \in C\}| = t$. We prove that

► **Theorem 10.** *The FkSO problem admits a $(\min\{4t - 1, 2^t + 1\})$ -approximation.*

4.1 Strong LP Relaxation and the Round-or-Cut Schema

To circumvent the gap example in Section 3, we adapt the following stronger linear program idea from Chakrabarty and Negahbani [7]. As before, r is the guess of the optimal solution, and we have the same fractional variables cov_v indicating coverage. However, we assert that these cov_v 's arise as a convex combination of integral solutions. More precisely, we have *exponentially many* auxiliary variables $\{z_S\}_{S \subseteq F: |S| \leq k}$ indicating possible locations of open facilities and the fractional amount to which they are open. When such a solution is opened, a client v is “covered” if there are ℓ_v facilities in an r -neighborhood. To this end, for a client v , we define the collection $\mathcal{F}_v := \{S \subseteq F : |S| \leq k \wedge |S \cap B(v, r)| \geq \ell_v\}$ of solutions which can serve v . Therefore, the coverage cov_v is simply the total fractional weight of sets in \mathcal{F}_v . Formally, if r is a correct guess, then the following (huge) LP has a feasible solution.

23:10 Fault-tolerant k -Supplier with Outliers

$$\sum_{v \in C} \text{cov}_v \geq m \quad (\text{L1})$$

$$\forall v \in C, \text{cov}_v = \sum_{S \in \mathcal{F}_v} z_S \quad (\text{L2})$$

$$\sum_{S \subseteq F: |S| \leq k} z_S \leq 1 \quad (\text{L3})$$

$$\forall S \subseteq F, \forall v \in C, 0 \leq z_S, \text{cov}_v \leq 1 \quad (\text{L4})$$

(L1) enforces that at least m clients must be covered. (L2) connects the cov_v and z_S variables, ensuring that a client v can only be covered via solutions in \mathcal{F}_v . (L3)-(L4) enforce convexity. (L4) also enforces that each client can be covered at most once.

► **Observation 11.** *All cov_v 's that satisfy (L1)-(L4) also satisfy (WL1)-(WL5).*

Also observe that we cannot efficiently figure out whether the above system is feasible or not; indeed, if so we would solve the Fault-tolerant k -Supplier with Outliers problem optimally. Nevertheless, one can use the round-or-cut schema to obtain an *approximation* algorithm. In order to do so, the first step is to use the *dual* of the above system to obtain the collection of all valid inequalities on the cov_v 's. Recall, a valid inequality is one that every feasible cov_v must satisfy; the lemma below from the literature [6], in some sense, eliminates all the z_S variables from the above program.

► **Lemma 12** ([7, Lemma 10]). *Given real numbers $\{\lambda_v\}_{v \in C}$ such that*

$$\forall S \subseteq F, \sum_{v \in C: S \in \mathcal{F}_v} \lambda_v < m, \quad (\lambda 1)$$

the following is a valid inequality for (L1)-(L4):

$$\sum_{v \in C} \lambda_v \text{cov}_v < m. \quad (\lambda 2)$$

Given $\{\lambda_v\}_{v \in C}$, one cannot easily check ($\lambda 1$), and thus, a priori, one cannot see the usefulness of the above lemma. We now briefly describe its usefulness to the round-or-cut schema. The algorithm begins with values of $\{0 \leq \text{cov}_v \leq 1\}_{v \in C}$ that satisfy (L1) – such cov is straightforward to find. We then try to use these cov_v 's to “round” and obtain a solution where clients are covered within distance $\alpha \cdot r$ for desired factor α , and if we fail, then we find a valid inequality that “cuts” cov_v away from the above system. If we can do so, then we can feed this separating hyperplane to the ellipsoid algorithm which would give us new cov_v 's. Repeating the above procedure a polynomial number of times, we would either obtain an α -approximation, or prove that the above system is empty implying our guess r was too small. For FkSO, the “round” step is via the abstract concept of a “good partition” where the “radius” of the partition dictates the approximation factor; this definition and resulting rounding algorithm is described in Section 4.2. For the “cut” step, we show that if our rounding algorithm fails, then we can use this failure to generate $\{\lambda_v\}_{v \in C}$'s that satisfy ($\lambda 1$) but not ($\lambda 2$), leveraging our definition of “good partitions”. This gives our separating hyperplane using Lemma 12, and we succeed in cutting, and thus we can run the round-or-cut schema. Subsequently, we construct good partitions. In Section 4.3, we describe two methods to do this: one with “radius” $(4t - 1)$ and the other with radius $(2^t + 1)$. In Appendix B, we show a limitation of our approach, via an example where this “radius” can be $\Omega(t)$.

Before proceeding, we make one simplification: at the beginning of every rounding step, we discard any clients that have $\text{cov}_v = 0$, and hereafter assume, without loss of generality, that $\forall v \in C, \text{cov}_v > 0$.

4.2 Good Partitions and Implementing Round-or-cut

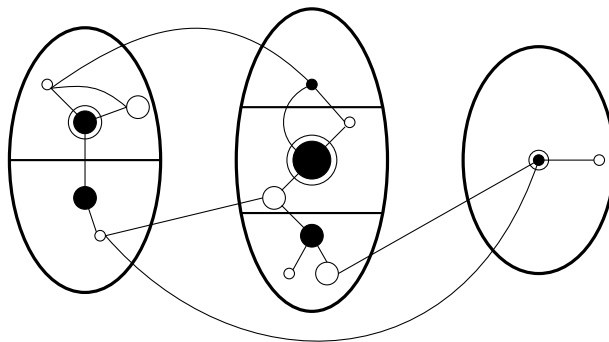
Given cov_v 's for every $v \in C$, we define a notion of a “good partition”. Before formally defining it, we explain this operationally, hopefully giving intuition for the definition. We start with a finer partition, and the good partition \mathcal{P} coarsens it. As in previous algorithms discussed so far, we have $R \subseteq C$, a set of *representatives*. The finer partition is $\{\text{child}(j)\}_{j \in R}$, as motivated by our algorithms for *FkS* in Section 2 and *UFkSO* in Section 3. This time, however, we want favorable properties from both of those algorithms to coincide – we want, for $j \in R$ and $v \in \text{child}(j)$, $\text{cov}_v \leq \text{cov}_j$ as well as $\ell_v \leq \ell_j$. These desired properties of the finer partition are formalized as Property 1.

The property above breaks the “well-separated” property of R , which was crucial in our other algorithms in Sections 2 and 3. Therefore, instead of requiring R to be well-separated, we coalesce the child-sets of certain representatives, to get a *coarsening* \mathcal{P} of $\{\text{child}(j)\}_{j \in R}$ such that representatives *across* different parts of \mathcal{P} are indeed well-separated. This is Property 2.

Our approximation ratio is then determined by the diameter of the parts P 's in the good partition; so we impose a radius bound on each $P \in \mathcal{P}$, requiring that the highest-fault-tolerance client in each P be not too far from the rest of P . This is Property 3. We are now ready to present the formal definition.

► **Definition 13** ((ρ, cov) -good partition). *Given a parameter $\rho \in \mathbb{R}$, and $\{0 \leq \text{cov}_v \leq 1\}_{v \in C}$ satisfying (L1), a partition \mathcal{P} of C is (ρ, cov) -good if there exists $R \subseteq C$ such that the following hold.*

1. *Every $v \in C$ is assigned to be the child of a $j \in R$, forming a partition $\{\text{child}(j)\}_{j \in R}$ of C that refines \mathcal{P} . Also, $\forall j \in R, \forall v \in \text{child}(j)$, $\text{cov}_j \geq \text{cov}_v$ and $\ell_j \geq \ell_v$.*
2. *For any two $j, j' \in R$ that lie in different parts of \mathcal{P} , $d(j, j') > 2r$.*
3. *For each $P \in \mathcal{P}$, let $j_P := \text{argmax}_{v \in P} \ell_v$ (breaking ties arbitrarily). Then $\forall v \in P$, $d(j_P, v) \leq \rho r$.*



■ **Figure 2** An example of a $(6, \text{cov})$ -good partition \mathcal{P} (Definition 13). The ellipses represent \mathcal{P} , and their subdivisions represent the child sets. All the circles are clients, with the filled-in circles being R , and among those, the double borders indicate the j_P 's. cov values are 1 on R and $1/2$ elsewhere. ℓ_v values are indicated by the sizes of the circles. The “edges” represent distance $2r$, and all other distances are obtained by making triangle inequalities tight.

We observe here that the child-sets constructed in Section 3 are themselves a good partition, so for *UFkSO*, we did not need to coarsen it. This will not necessarily be the case for child-sets that we construct in Section 4.3. We also observe that

23:12 Fault-tolerant k -Supplier with Outliers

► **Observation 14.** *In a (ρ, cov) -good partition \mathcal{P} , by Property 1, the j_P 's in Property 3 can be chosen such that $\forall P \in \mathcal{P}, j_P \in R$. So we can assume, without loss of generality, that all j_P 's are in R .*

We prove that a good partition suffices to achieve our desired approximation. That is,

► **Theorem 15.** *If we have a feasible instance with a (ρ, cov) -good partition, then in polynomial time, we can either obtain a $(\rho+1)$ -approximation, or identify a valid inequality for (L1)-(L4) that is violated by cov .*

To prove Theorem 15, we solve a budgeting problem on the (ρ, cov) -good partition. We want to distribute our budget of k facilities among the $P \in \mathcal{P}$, assigning each $P \in \mathcal{P}$ with k_P facilities that are within distance $(\rho+1)r$ of the clients in P . Here k_P must be at most $\ell_P := \ell_{j_P}$, because at most ℓ_P facilities are guaranteed to exist within a bounded distance of clients in P . The payoff from assigning k_P facilities to P in this way is that the clients $\{v \in P : \ell_v \leq k_P\}$ are served within distance $(\rho+1)r$. So if $\sum_{P \in \mathcal{P}} |\{v \in P : \ell_v \leq k_P\}| \geq m$, we have our $(\rho+1)$ -approximation. Therefore, we want our choice of k_P 's to maximize $\sum_{P \in \mathcal{P}} |\{v \in P : \ell_v \leq k_P\}|$, and this maximum to be $\geq m$. However, our analysis can only handle clients from well-separated sets; so instead, we maximize the following lower-bound on our desired quantity: $\sum_{P \in \mathcal{P}} \sum_{j \in R \cap P : \ell_j \leq k_P} |\text{child}(j)|$, where we under-count by only considering $v \in \text{child}(j)$ served if j is served. Formally, our budgeting problem is the following.

► **Definition 16** (Budgeting over a (ρ, cov) -good partition). *Given a (ρ, cov) -good partition \mathcal{P} , let $\ell_P := \max_{v \in P} \ell_v$. Find $\{k_P \leq \ell_P\}_{P \in \mathcal{P}}$ such that $\sum_{P \in \mathcal{P}} k_P \leq k$, maximizing $\sum_{P \in \mathcal{P}} \sum_{j \in R \cap P : \ell_j \leq k_P} |\text{child}(j)|$. Let $\text{opt}_B(\mathcal{P})$ denote this maximum.*

In Lemma 17, we show that if $\text{opt}_B(\mathcal{P}) \geq m$, then we can round. Then in Lemma 18, we see that if $\text{opt}_B(\mathcal{P}) < m$, then we can cut. Lemma 19 shows that $\text{opt}_B(\mathcal{P})$ can be found efficiently. Together, these three lemmas yield the proof of Theorem 15.

► **Lemma 17.** *Given a (ρ, cov) -good partition \mathcal{P} , if $\text{opt}_B(\mathcal{P}) \geq m$, then we have a $(\rho+1)$ -approximation.*

Proof. Let $\{k_P\}_{P \in \mathcal{P}}$ be an optimal solution to the budgeting problem (Definition 16). Define $S := \cup_{P \in \mathcal{P}} N_{k_P}(j_P, F)$. So $|S| \leq k$. We show that S serves $\geq m$ clients within distance $(\rho+1)r$.

Define $T := \uplus_{P \in \mathcal{P}} \uplus_{j \in R \cap P : \ell_j \leq k_P} \text{child}(j)$. Then $|T| = \sum_{P \in \mathcal{P}} \sum_{j \in R \cap P : \ell_j \leq k_P} |\text{child}(j)| = \text{opt}_B(\mathcal{P}) \geq m$. We complete this proof by showing that $\forall v \in T, d_{\ell_v}(v, S) \leq (\rho+1)r$. For this, fix $v \in T$. By triangle inequalities, we have that $d_{\ell_v}(v, S) \leq d(v, j_P) + d_{\ell_v}(j_P, S)$. By Property 3, $d(v, j_P) \leq \rho r$, so it remains to show that $d_{\ell_v}(j_P, S) \leq r$.

By definition of T , $d_{\ell_v}(j_P, S) \leq d_{k_P}(j_P, S)$. Since $N_{k_P}(j_P, F) \subseteq S$, $d_{k_P}(j_P, S) \leq d_{k_P}(j_P, F)$. By definitions of k_P and ℓ_P , $d_{k_P}(j_P, F) \leq d_{\ell_P}(j_P, F) = d_{\ell_{j_P}}(j_P, F)$. But $\text{cov}_{j_P} > 0$; so by Observation 11 and (WL4), $d_{\ell_{j_P}}(j_P, F) \leq r$. ◀

► **Lemma 18.** *Given a (ρ, cov) -good partition \mathcal{P} , if $\text{opt}_B(\mathcal{P}) < m$, then we find a valid inequality for (L1)-(L4) that is violated by cov .*

Proof. We appeal to Lemma 12 mentioned in Section 4.1. $\forall v \in C$, define

$$\lambda_v := \begin{cases} |\text{child}(v)| & \text{if } v \in R, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Note that

$$\begin{aligned}
\sum_{v \in C} \lambda_v \text{cov}_v &= \sum_{j \in R} \lambda_j \text{cov}_j = \sum_{j \in R} |\text{child}(j)| \text{cov}_j = \sum_{j \in R} \sum_{v \in \text{child}(j)} \text{cov}_j \\
&\geq \sum_{j \in R} \sum_{v \in \text{child}(j)} \text{cov}_v && \dots \text{ by Property 1} \\
&= \sum_{v \in C} \text{cov}_v && \dots \text{ by Definition 13} \\
&\geq m, && \dots \text{ by (L1)}
\end{aligned}$$

i.e. these λ_v 's violate $(\lambda 2)$. So by Lemma 12, it suffices to show that $(\lambda 1)$ holds for these λ_v 's.

Suppose not, i.e. $\exists S_0 \subseteq F : |S_0| \leq k$ and $\sum_{v \in C: S_0 \in \mathcal{F}_v} \lambda_v \geq m$. Then, devise a candidate solution $\{k'_P\}_{P \in \mathcal{P}}$ for the budgeting problem in Definition 16, as follows. For each $P \in \mathcal{P}$, if $\exists j \in R \cap P$ such that $S_0 \in \mathcal{F}_j$, then set k'_P to be the largest fault-tolerance among such j 's; that is, where $j'_P := \arg\max_{j \in R \cap P: S_0 \in \mathcal{F}_j} \ell_j$, set $k'_P := \ell_{j'_P}$. Otherwise, i.e. when there is no such j and j'_P is not well-defined, set $k'_P := 0$. By definitions, $\forall P \in \mathcal{P}$, $k'_P \leq \ell_P$.

Also, by Property 2, $\{B(j'_P, r)\}_{P \in \mathcal{P}}$ is pairwise disjoint. Since $S_0 \in \mathcal{F}_{j'_P}$ for each $P \in \mathcal{P}$, we then have $\sum_{P \in \mathcal{P}} k'_P \leq \sum_{P \in \mathcal{P}} |S_0 \cap B(j'_P, r)| \leq |S_0| \leq k$. So $\{k'_P\}_{P \in \mathcal{P}}$ is indeed a candidate solution for the budgeting problem. We evaluate the objective function of the budgeting problem (see Definition 16) on $\{k'_P\}_{P \in \mathcal{P}}$:

$$\begin{aligned}
\sum_{P \in \mathcal{P}} \sum_{j \in R \cap P: \ell_j \leq k'_P} |\text{child}(j)| &= \sum_{P \in \mathcal{P}} \sum_{j \in R \cap P: \ell_j \leq k'_P} \lambda_j \\
&\geq \sum_{P \in \mathcal{P}} \sum_{j \in R \cap P: S_0 \in \mathcal{F}_j} \lambda_j && \dots \text{ by choice of } k'_P \text{'s} \\
&= \sum_{j \in R: S_0 \in \mathcal{F}_j} \lambda_j && \dots \text{ by Definition 13} \\
&= \sum_{v \in C: S_0 \in \mathcal{F}_v} \lambda_v && \dots \text{ by choice of } \lambda_v \text{'s} \\
&\geq m && \dots \text{ by supposition.}
\end{aligned}$$

So $\{k'_P\}_{P \in \mathcal{P}}$ is a candidate solution to the budgeting problem, for which the objective function evaluates to $\geq m$, contradicting $\text{opt}_B(\mathcal{P}) < m$. Hence $(\lambda 1)$ holds for our chosen λ_v 's, and $(\lambda 2)$ is the desired valid inequality that is violated by cov . \blacktriangleleft

► **Lemma 19.** *The budgeting problem in Definition 16 can be solved in polynomial time.*

Proof. We proceed via dynamic programming. Let $N := |\mathcal{P}|$. Without loss of generality, say $\mathcal{P} = \{P_1, P_2, \dots, P_N\}$. For brevity, $\forall a \in [N]$, we say $L_a := \ell_{P_a}$. To handle base cases in our DP, we set the convention that $P_0 := \emptyset$. Now define the entries in our DP table: $\forall \nu \in [N] \cup \{0\}$ and $\forall b \in [k] \cup \{0\}$,

$$M[\nu, b] := \max_{\{k_a \leq L_a\}_{a=1}^\nu: \sum_{a=1}^\nu k_a \leq b} \sum_{a=1}^\nu \sum_{j \in R \cap P_a: \ell_j \leq k_a} |\text{child}(j)|. \quad (\text{DP-defn})$$

The desired entry is $M[N, k]$, as the corresponding $\{k_a\}_{a=1}^N$ becomes, upon renaming as $\{k_{P_a} = k_a\}_{a=1}^N$, the k_P 's that we want.

23:14 Fault-tolerant k -Supplier with Outliers

The base cases are: $M[0, 0] = 0$; $\forall \nu \in [N]$, $M[\nu, 0] = 0$; and $\forall b \in [k]$, $M[0, b] = 0$. The DP table has $O(Nk) = O(nk)$ entries; so in polynomial time, we can fill it via the following recurrence.

$$M[\nu, b] := \max_{\ell=0}^{\min(b, L_\nu)} \left(M[\nu - 1, b - \ell] + \sum_{j \in R \cap P_\nu: \ell_j \leq \ell} |\text{child}(j)| \right). \quad (\text{DP-rec})$$

We also remember, for each entry $M[\nu, b]$, the ℓ that maximizes the RHS of (DP-rec). Note, in (DP-defn), that the RHS for $M[N, k]$ corresponds, up to renaming, with the RHS in the objective function (see Definition 16). Thus it remains to show that (DP-rec) is correct wrt (DP-defn).

- To show that LHS \leq RHS, consider the solution $\{k_a^*\}_{a=1}^\nu$ corresponding to $M[\nu, b]$. By (DP-defn), $k_\nu^* \leq \min(b, L_\nu)$. So $\{k_a^*\}_{a=1}^{\nu-1}$ is a candidate solution for $M[\nu - 1, b - k_\nu^*]$, i.e. $\sum_{a=1}^{\nu-1} \sum_{j \in R \cap P_a: \ell_j \leq k_a^*} |\text{child}(j)| \leq M[\nu - 1, b - k_\nu^*]$, so

$$\begin{aligned} \text{LHS} = M[\nu, b] &= \sum_{a=1}^\nu \sum_{j \in R \cap P_a: \ell_j \leq k_a^*} |\text{child}(j)| \\ &\leq M[\nu - 1, b - k_\nu^*] + \sum_{j \in R \cap P_\nu: \ell_j \leq k_\nu^*} |\text{child}(j)| \leq \text{RHS} \end{aligned}$$

since the RHS is a maximum.

- To show that RHS \leq LHS, fix an $\ell \in \{0, \dots, \min(b, L_\nu)\}$, and let $\{k'_a\}_{a=1}^{\nu-1}$ be the solution corresponding to $M[\nu - 1, b - \ell]$. Setting $k'_\nu = \ell$ yields $\{k'_a\}_{a=1}^\nu$, a candidate solution for $M[\nu, b]$. So

$$M[\nu - 1, b - \ell] + \sum_{j \in R \cap P_\nu: \ell_j \leq \ell} |\text{child}(j)| = \sum_{a=1}^\nu \sum_{j \in R \cap P_a: \ell_j \leq k'_a} |\text{child}(j)| \leq M[\nu, b] = \text{LHS}$$

since $M[\nu, b]$ is a maximum by (DP-defn).

As the RHS maximizes over $\ell \in \{0, \dots, \min(b, L_\nu)\}$, we are done. \blacktriangleleft

Proof of Theorem 15. Given a (ρ, cov) -good partition \mathcal{P} , we solve the budgeting problem (Definition 16), which we can do efficiently due to Lemma 19, and obtain $\text{opt}_B(\mathcal{P})$. If $\text{opt}_B(\mathcal{P}) \geq m$, Lemma 17 guarantees a $(\rho + 1)$ -approximation; otherwise, Lemma 18 gives a valid inequality that is violated by cov . We pass the valid inequality as a separating hyperplane to the ellipsoid algorithm, and restart our rounding process with fresh cov_v 's. By the guarantees of ellipsoid, in polynomial time, we either round to obtain a $(\rho + 1)$ -approximation, or detect that the guess of r is too small. \blacktriangleleft

4.3 Obtaining a good partition

► **Theorem 20.** Given $\{0 \leq \text{cov}_v \leq 1\}_{v \in C}$, in polynomial time, we can obtain the following:

1. a $(4t - 2, \text{cov})$ -good partition, and
2. a $(2^t, \text{cov})$ -good partition.

Theorem 15 follows from Lemmas 21 and 22.

► **Lemma 21.** Algorithm 3 yields a $(4t - 2, \text{cov})$ -good partition.

■ **Algorithm 3** Finding a $(4t - 2, \text{cov})$ -good partition.

Input: $\{0 \leq \text{cov}_v \leq 1\}_{v \in C}$

- 1: $U \leftarrow C$
- 2: $R \leftarrow \emptyset$
- 3: **while** $U \neq \emptyset$ **do**
- 4: $j \leftarrow \text{argmax}_{v \in U} \text{cov}_v$
- 5: $R \leftarrow R \cup \{j\}$
- 6: $\text{child}(j) \leftarrow \{v \in U : d(v, j) \leq 2tr \wedge \ell_v \leq \ell_j\}$
- 7: $U \leftarrow U \setminus \text{child}(j)$
- 8: $\mathcal{P} \leftarrow \emptyset$
- 9: $G \leftarrow (R, E := \{\{j, j'\} : d(j, j') \leq 2r\})$ ▷ undirected graph
- 10: $\mathcal{C} \leftarrow$ connected components of G
- 11: $\mathcal{P} \leftarrow \{\cup_{j \in V} \text{child}(j)\}_{V \in \mathcal{C}}$

Output: A partition \mathcal{P} of C .

Proof. Consider \mathcal{P} , the output of Algorithm 3, and the child and R constructed alongside. Line 7 ensures that $\{\text{child}(j)\}_{j \in R}$ is a partition of C . Line 11 ensures that this partition is a refinement of \mathcal{P} . Lines 4 and 6 construct child as desired, ensuring that $\forall j \in R, \forall v \in \text{child}(j), \text{cov}_j \geq \text{cov}_v$ and $\ell_j \geq \ell_v$. So Property 1 holds.

Now consider $P_1, P_2 \in \mathcal{P}, x_1 \in R \cap P_1, x_2 \in R \cap P_2 : P_1 \neq P_2$. By Lines 9-10, $R \cap P_1$ and $R \cap P_2$ are distinct connected components in \mathcal{C} , so $\{x_1, x_2\} \notin E$, i.e. $d(x_1, x_2) > 2r$. This shows that Property 2 holds.

Finally, consider $P \in \mathcal{P}$, and $v \in P$ s.t. $v \in \text{child}(j_1)$ for $j_1 \in R$. By Line 11, $j_1 \in R \cap P$. Also consider a different $j_2 \in R \cap P$. By Lines 9-10, $R \cap P \in \mathcal{C}$. In G , consider π , the shortest j_1 - j_2 path passing entirely through $R \cap P$. We claim that

▷ **Claim.** π contains at most t vertices.

Proof. Suppose not. Then, by the pigeonhole principle, π contains vertices $u, v \in R \cap P$ s.t. $u \neq v$ and $\ell_u = \ell_v$. Choose such u, v minimizing $d(u, v)$, and consider the u - v subpath π' of π . If π' contains $> t$ vertices, then we can replace j_1, j_2 with u, v and repeat our argument to obtain a smaller $d(u, v)$ – contradicting our choice of u, v . So π' contains $\leq t$ vertices, i.e. $d(u, v) \leq 2(t-1)r$; but since $u, v \in R$, this contradicts Line 6. ◀

So $d(j_1, j_2) \leq 2(t-1)r$, i.e. by Line 6, $d(v, j_2) \leq d(v, j_1) + d(j_1, j_2) \leq 2tr + 2(t-1)r = (4t-2)r$. We have just showed that, $\forall v \in P, j \in R \cap P, d(v, j) \leq (4t-2)r$. By Observation 14, this implies Property 3 for $\rho = (4t-2)$. ◀

► **Lemma 22.** *Algorithm 4 yields a $(2^t, \text{cov})$ -good partition.*

Proof. Consider \mathcal{P} , the output of Algorithm 4, and the child and R constructed alongside. Note that, since Line 8 only creates edges to **Roots**, and Line 9 updates **Roots** accordingly, (R, E) is indeed a forest.

Line 12 ensures that $\{\text{child}(j)\}_{j \in R}$ is a partition of C . Line 14 ensures that this partition is a refinement of \mathcal{P} . Lines 6 and 11 construct child as desired, ensuring that $\forall j \in R, \forall v \in \text{child}(j), \text{cov}_j \geq \text{cov}_v$ and $\ell_j \geq \ell_v$. So Property 1 holds.

Now consider $P_1, P_2 \in \mathcal{P}, x_1 \in R \cap P_1, x_2 \in R \cap P_2$. Without loss of generality, suppose x_2 was added to R after x_1 ; if $d(x_1, x_2) \leq 2r$, then by Lines 8 and 10, we would have $d(x_2, x_1) \in E$, i.e. x_1, x_2 would lie in the same connected component in \mathcal{T} . So by Lines 13-14, $P_1 = P_2$. This shows that Property 2 holds.

■ **Algorithm 4** Finding a $(2^t, \text{cov})$ -good partition.

```

1:  $U \leftarrow C$ 
2:  $(R, E) \leftarrow (\emptyset, \emptyset)$  ▷ initializing an empty directed forest
3:  $\forall v \in U, \text{height}(v) \leftarrow 0$  ▷ height in the forest;  $\text{height}(v) = 0 \implies v \notin R$ 
4:  $\text{Roots} \leftarrow \emptyset$  ▷ tracking roots in the forest
5: while  $U \neq \emptyset$  do
6:    $j \leftarrow \text{argmax}_{v \in U} \text{cov}_v$ 
7:    $R \leftarrow R \cup \{j\}$ 
8:    $E \leftarrow E \cup \{(j, j') : j' \in \text{Roots} \wedge d(j, j') \leq 2^{\text{height}(j')} r\}$ 
9:    $\text{Roots} \leftarrow (\text{Roots} \setminus \{j' : (j, j') \in E\}) \cup \{j\}$ 
10:   $\text{height}(j) \leftarrow 1 + \max_{(j, j') \in E} \text{height}(j')$  ▷ convention: max over  $\emptyset$  is 0
11:   $\text{child}(j) \leftarrow \{v \in U : d(v, j) \leq 2^{\text{height}(j)} r \wedge \ell_v \leq \ell_j\}$ 
12:   $U \leftarrow U \setminus \text{child}(j)$ 
13:  $\mathcal{T} \leftarrow$  connected components in the forest  $(R, E)$  ▷ each component induces a tree
14:  $\mathcal{P} \leftarrow \{\cup_{j \in V} \text{child}(j)\}_{V \in \mathcal{T}}$ 

```

Finally, note that

▷ **Claim 23.** $(j, j') \in E \implies \ell_j > \ell_{j'}$.

Proof. Since $(j, j') \in E$, we know that j' was added to R before j , and $d(j, j') \leq 2^{\text{height}(j')} r$. So if $\ell_j \leq \ell_{j'}$, then by Line 11, we would have $j \in \text{child}(j')$, contradicting the fact that $j \in R$. ◁

Now fix $P \in \mathcal{P}$, and consider j_P which, by Observation 14, lies in $R \cap P$, and hence by Lines 13-14, $R \cap P$ induces a tree in (R, E) . Claim 23 tells us that j_P is the root in this tree, and that $\text{height}(j_P) \leq t$. So by Line 8, for any $j \in R \cap P$, $d(j_P, j) \leq (2^{\text{height}(j_P)} - 2^{\text{height}(j)}) r \leq (2^t - 2^{\text{height}(j)}) r$. Now consider $v \in P : v \in \text{child}(j)$ for a $j \in R \cap P$. Then $d(v, j) \leq 2^{\text{height}(j)} r$, so $d(v, j_P) \leq d(v, j) + d(j, j_P) \leq (2^t - 2^{\text{height}(j)} + 2^{\text{height}(j)}) r = 2^t r$. Thus Property 3 holds for $\rho = 2^t$. ◀

5 Conclusion

In this paper, we have studied the Fault-tolerant k -Supplier with Outliers problem and presented a $(4t - 1)$ -approximation when there are t distinct fault tolerances. While this gives the optimal 3-approximation for the uniform version of the problem (improving upon the recent result [20]), the parameter t could be as large as k . To obtain our result, we needed to resort to the powerful hammer of the round-or-cut schema, and indeed used a very strong LP relaxation. This was necessary since, as we saw in Section 3.1, natural LP relaxations and their strengthenings have unbounded integrality gaps. We also show a $\Omega(t)$ -bottleneck to our approach (Appendix B), and this raises the intriguing question: are there $O(1)$ -approximations for the $FkSO$ problem? As noted in Section 1, the authors are not aware of clustering problems where the version without outliers has a constant approximation (as we saw in Section 2, FkS does), but the outlier version doesn't. Perhaps $FkSO$ is such a candidate example. This also raises the question of designing inapproximability results for metric clustering problems, which has not been explored much. We leave all these as interesting avenues of further study.

References

- 1 Ravinder Ahuja, Aakarsha Chug, Shaurya Gupta, Pratyush Ahuja, and Shruti Kohli. Classification and clustering algorithms of machine learning with their applications. *Nature-inspired computation in data mining and machine learning*, pages 225–248, 2020. doi:10.1007/978-3-030-28553-1_11.
- 2 Hyung-Chan An, Mohit Singh, and Ola Svensson. LP-Based Algorithms for Capacitated Facility Location. In *Proc., IEEE Symposium on Foundations of Computer Science (FOCS)*, 2014. doi:10.1137/151002320.
- 3 Georg Anegg, Haris Angelidakis, Adam Kurpisz, and Rico Zenklusen. A technique for obtaining true approximations for k-center with covering constraints. *Math. Programming*, pages 1–25, 2022. doi:10.1007/s10107-021-01645-y.
- 4 Tanvi Bajpai, Deeparnab Chakrabarty, Chandra Chekuri, and Maryam Negahbani. Revisiting Priority k-Center: Fairness and Outliers. In *Proc., International Colloquium on Automata, Languages and Programming (ICALP)*, pages 21:1–21:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.21.
- 5 Jaroslav Byrka, Aravind Srinivasan, and Chaitanya Swamy. Fault-tolerant facility location: a randomized dependent LP-rounding algorithm. In *Proc., MPS Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 244–257, 2010. doi:10.1007/978-3-642-13036-6.
- 6 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The Non-Uniform k-Center Problem. *ACM Trans. on Algorithms (TALG)*, 2020. Preliminary version in ICALP 2016. doi:10.1145/3392720.
- 7 Deeparnab Chakrabarty and Maryam Negahbani. Generalized Center Problems with Outliers. *ACM Trans. on Algorithms (TALG)*, 2019. Prelim. version in ICALP 2018. doi:10.1145/3338513.
- 8 Deeparnab Chakrabarty and Maryam Negahbani. Robust k-center with two types of radii. *Math. Programming*, 197(2):991–1007, 2023. Special Issue for Proc. IPCO 2021. doi:10.1007/s10107-022-01799-3.
- 9 Deeparnab Chakrabarty, Maryam Negahbani, and Ankita Sarkar. Approximation Algorithms for Continuous Clustering and Facility Location Problems. In *Proc., European Symposium on Algorithms*, pages 33:1–33:15, 2022. doi:10.4230/LIPIcs.ESA.2022.33.
- 10 Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for Facility Location Problems with Outliers. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 642–651, 2001. URL: <https://dl.acm.org/doi/abs/10.5555/365411.365555>.
- 11 Shiva Chaudhuri, Naveen Garg, and Ramamoorthi Ravi. The p -neighbor k -center problem. *Inform. Process. Lett.*, 65(3):131–134, 1998. doi:10.1016/S0020-0190(97)00224-X.
- 12 Shiri Chechik and David Peleg. Robust fault tolerant uncapacitated facility location. *Theoretical Computer Science*, 543:9–23, 2014. Preliminary version appeared in STACS 2010. doi:10.1016/j.tcs.2014.05.013.
- 13 Vasek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(4):305–337, 1973. doi:10.1016/0012-365X(73)90167-2.
- 14 Shichuan Deng. Fault-Tolerant Center Problems with Robustness and Fairness. *CoRR*, abs/2011.00817v2, 2020. Citation specific to version 2; latest version is a significantly different work. arXiv:2011.00817v2.
- 15 Shichuan Deng. Constant approximation for fault-tolerant median problems via iterative rounding. *Operations Research Letters*, 50(4):384–390, 2022. doi:10.1016/j.orl.2022.05.002.
- 16 Ralph E Gomory. Some polyhedra related to combinatorial problems. *Linear Algebra Appl.*, 2(4):451–558, 1969. doi:10.1016/0024-3795(69)90017-2.
- 17 Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation algorithm for the fault-tolerant facility location problem. *Journal of Algorithms*, 48(2):429–440, 2003. Preliminary version appeared in SODA 2001. doi:10.1016/S0196-6774(03)00056-7.

- 18 MohammadTaghi Hajiaghayi, Wei Hu, Jian Li, Shi Li, and Barna Saha. A constant factor approximation algorithm for fault-tolerant k -median. *ACM Trans. on Algorithms (TALG)*, 12(3):1–19, 2016. doi:10.1145/2854153.
- 19 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986. doi:10.1145/5925.5933.
- 20 Tanmay Inamdar and Kasturi Varadarajan. Fault tolerant clustering with outliers. In *Proc., Workshop on Approximation and Online Algorithms (WAOA)*, pages 188–201. Springer, 2020. doi:10.1007/978-3-030-39479-0_13.
- 21 Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999. doi:10.1145/331499.331504.
- 22 Kamal Jain and Vijay V Vazirani. An approximation algorithm for the fault tolerant metric facility location problem. *Algorithmica*, 38:433–439, 2004. Preliminary version appeared in APPROX 2000. doi:10.1007/s00453-003-1070-1.
- 23 Xinrui Jia, Lars Rohwedder, Kshiteej Sheth, and Ola Svensson. Towards Non-Uniform k -Center with Constant Types of Radii. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 228–237. SIAM, 2022. doi:10.1137/1.9781611977066.16.
- 24 Christopher Jung, Sampath Kannan, and Neil Lutz. Service in Your Neighborhood: Fairness in Center Location. In *Proceedings, Foundations of Responsible Computing, FORC 2020*, volume 156, pages 5:1–5:15, 2020. doi:10.4230/LIPIcs.FORC.2020.5.
- 25 Samir Khuller, Robert Pless, and Yoram J Sussmann. Fault tolerant k -center problems. *Theoretical Computer Science*, 242(1-2):237–245, 2000. doi:10.1016/S0304-3975(98)00222-9.
- 26 Sven Oliver Krumke. On a generalization of the p -center problem. *Inform. Process. Lett.*, 56(2):67–71, 1995. doi:10.1016/0020-0190(95)00141-X.
- 27 Nirman Kumar and Benjamin Raichel. Fault Tolerant Clustering Revisited. In *Canadian Conference on Computational Geometry (CCCG)*, 2013. doi:10.48550/arXiv.1307.2520.
- 28 Richard CT Lee. Clustering analysis and its applications. In *Advances in Information Systems Science: Volume 8*, pages 169–292. Springer, 1981. doi:10.1007/978-1-4613-9883-7_4.
- 29 Yu Li, Dachuan Xu, Donglei Du, and Naihua Xiu. Improved approximation algorithms for the robust fault-tolerant facility location problem. *Inform. Process. Lett.*, 112(10):361–364, 2012. doi:10.1016/j.ipl.2012.02.004.
- 30 Chiun-Ming Liu. Clustering techniques for stock location and order-picking in a distribution center. *Comput. Oper. Res.*, 26(10-11):989–1002, 1999. doi:10.1016/S0305-0548(99)00026-X.
- 31 Sepideh Mahabadi and Ali Vakilian. (Individual) Fairness for k -Clustering. In *Proc., International Conference on Machine Learning (ICML)*, pages 7925–7935, 2020. URL: <https://dl.acm.org/doi/abs/10.5555/3524938.3525549>.
- 32 Boris Mirkin. *Mathematical classification and clustering*, volume 11. Springer Science & Business Media, 1996. doi:10.1007/978-1-4613-0457-9.
- 33 Ján Plesník. A heuristic for the p -center problems in graphs. *Discrete Applied Mathematics*, 17(3):263–268, 1987. doi:10.1016/0166-218X(87)90029-1.
- 34 Bartosz Rybicki and Jaroslaw Byrka. Improved approximation algorithm for fault-tolerant facility placement. In *Proc., Workshop on Approximation and Online Algorithms (WAOA)*, pages 59–70. Springer, 2015. doi:10.1007/978-3-319-18263-6_6.
- 35 Chinmay Sonar, Subhash Suri, and Jie Xue. Fault Tolerance in Euclidean Committee Selection. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 95:1–95:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.95.
- 36 Chaitanya Swamy and David B. Shmoys. Fault-Tolerant Facility Location. *ACM Trans. on Algorithms (TALG)*, 4(4), August 2008. doi:10.1145/1383369.1383382.
- 37 Shihong Xu and Hong Shen. The fault-tolerant facility allocation problem. In *Proc., Int. Symposium on Algorithms and Computation (ISAAC)*, pages 689–698. Springer, 2009. doi:10.1007/978-3-642-10631-6_70.
- 38 Li Yan and Marek Chrobak. Approximation algorithms for the fault-tolerant facility placement problem. *Inform. Process. Lett.*, 111(11):545–549, 2011. doi:10.1016/j.ipl.2011.03.005.

A Proof of Claim 7

Consider a feasible solution S^* that serves inliers T^* . Set

- $\forall v \in C, \text{cov}_v = \mathbf{1}_{v \in T^*}$, and
- $\forall i \in F, y_i = \mathbf{1}_{i \in S^*}$.

These satisfy (WL1), (WL2), and (WL5) by construction. Now note that, for a $v \in T^*$, $N_{\ell_v}(v, F) \subseteq S$; so (WL3) is satisfied. Furthermore, for a $v \in C$, if $d_{\ell_v}(v, F) > r$ then $v \notin T^*$, satisfying (WL4).

Conversely, given an integral solution satisfying (WL1)-(WL5), we can construct $S^* = \{i \in F : y_i = 1\}$, and $T^* = \{v \in C : \text{cov}_v = 1\}$. (WL2) implies $|S^*| \leq k$, and (WL1) implies $w(T^*) \geq W$. For any $v \in T^*$,

$$\begin{aligned} |S^* \cap B(v, r)| &= \sum_{i \in F \cap B(v, r)} y_i && \dots \text{by construction of } S^* \\ &\geq \ell \text{cov}_v = \ell, && \dots \text{by (WL3) and construction of } T^* \end{aligned}$$

so $d_{\ell_v}(v, S^*) \leq r$.

B Limiting Example for Good-Partition Rounding

In order to achieve a better approximation factor than $\Omega(t)$, we will need to move beyond the overall schema of using a good partition (Definition 13) to round solutions to (L1)-(L4). This can be seen via the following example, illustrated in Figure 3. Here $r = 1, n = t, m = 1$. C is the set $\{v_1, \dots, v_t\}$, with each client v_a having fault-tolerance $\ell_{v_a} = a$. F is the union of t sets $\{F_a\}_{a=1}^t$, where $F_a = \{i_{a1}, i_{a2}, \dots, i_{ak}\}$, for a total of tk facilities in F . Each client v_a has distance 2 to v_{a+1} and v_{a-1} , and distance 1 to each facility in F_a . Remaining distances are determined by making triangle inequalities tight.

Consider the following (cov, z) satisfying (L1)-(L4). We set $z_{F_a} = \frac{1}{aH_t}$ for each $a \in [t]$, where H_t is the t^{th} Harmonic number; and set all other z_S 's to zero. This allows us to set $\text{cov}_{v_a} = \frac{1}{aH_t}$ for each $a \in [t]$. Under this (cov, z) , observe that $\forall v_a, v_b \in C, v_a \neq v_b \wedge \text{cov}_a \geq \text{cov}_b \implies \ell_a < \ell_b$; so Property 1 can only hold if all clients are in the same piece of the partition, i.e. $\mathcal{P} = \{C\}$. This means that a (ρ, cov) -good partition can only be attained for $\rho \geq 2(t - 1)$, so upon applying Theorem 15, this approach attains a $(2t - 1)$ -approximation at best.

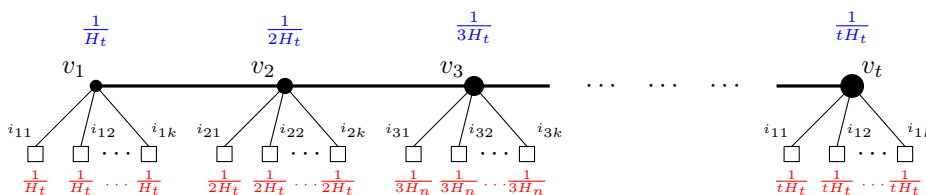


Figure 3 An example showing the limitations of good partitions, with a solution to (L1)-(L4) shown in red (z values) and blue (cov values). The thin “edges” represent distance 1, the thick “edges” represent distance 2, and all other distances are determined by making triangle inequalities tight. The fault-tolerances are $\ell_{v_1} = 1, \ell_{v_2} = 2, \dots, \ell_{v_t} = t$.

Approximate Circular Pattern Matching Under Edit Distance


Panagiotis Charalampopoulos ✉ 

Birkbeck, University of London, UK

Solon P. Pissis ✉ 

CWI, Amsterdam, The Netherlands

Vrije Universiteit, Amsterdam, The Netherlands

Jakub Radoszewski ✉ 

University of Warsaw, Poland

Wojciech Rytter ✉ 

University of Warsaw, Poland

Tomasz Waleń ✉ 

University of Warsaw, Poland

Wiktor Zuba ✉ 

CWI, Amsterdam, The Netherlands

Abstract

In the k -Edit Circular Pattern Matching (k -Edit CPM) problem, we are given a length- n text T , a length- m pattern P , and a positive integer threshold k , and we are to report all starting positions of the substrings of T that are at edit distance at most k from some cyclic rotation of P . In the decision version of the problem, we are to check if any such substring exists. Very recently, Charalampopoulos et al. [ESA 2022] presented $\mathcal{O}(nk^2)$ -time and $\mathcal{O}(nk \log^3 k)$ -time solutions for the reporting and decision versions of k -Edit CPM, respectively. Here, we show that the reporting and decision versions of k -Edit CPM can be solved in $\mathcal{O}(n + (n/m)k^6)$ time and $\mathcal{O}(n + (n/m)k^5 \log^3 k)$ time, respectively, thus obtaining the first algorithms with a complexity of the type $\mathcal{O}(n + (n/m) \text{poly}(k))$ for this problem. Notably, our algorithms run in $\mathcal{O}(n)$ time when $m = \Omega(k^6)$ and are superior to the previous respective solutions when $m = \omega(k^4)$. We provide a meta-algorithm that yields efficient algorithms in several other interesting settings, such as when the strings are given in a compressed form (as straight-line programs), when the strings are dynamic, or when we have a quantum computer.

We obtain our solutions by exploiting the structure of approximate circular occurrences of P in T , when T is relatively short w.r.t. P . Roughly speaking, either the starting positions of approximate occurrences of rotations of P form $\mathcal{O}(k^4)$ intervals that can be computed efficiently, or some rotation of P is almost periodic (is at a small edit distance from a string with small period). Dealing with the almost periodic case is the most technically demanding part of this work; we tackle it using properties of locked fragments (originating from [Cole and Hariharan, SICOMP 2002]).

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases circular pattern matching, approximate pattern matching, edit distance

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.24

Related Version Full Version: <https://arxiv.org/abs/2402.14550>

Funding *Solon P. Pissis*: Supported by the PANGAIA and ALPACA projects that have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively.

Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2022/46/E/ST6/00463.

Wiktor Zuba: Received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement Grant Agreement No 101034253.

Acknowledgements We thank Tomasz Kociumaka for helpful discussions.



© Panagiotis Charalampopoulos, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba; licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 24; pp. 24:1–24:22



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

In the classic pattern matching (PM) problem, we are given a length- n text T and a length- m pattern P , and we are to report all starting positions (called occurrences) of the fragments of T that are identical to P . This problem can be solved in the optimal $\mathcal{O}(n)$ time by, e.g., the famous Knuth-Morris-Pratt algorithm [29]. In many real-world applications, we are interested in locating not only the fragments of T which are identical to P , but also the fragments of T which are identical to any cyclic rotation of P . In this setting, the rotations of P form an equivalence class, represented by a single circular string. In the circular PM (CPM) problem, we are to report all occurrences of the fragments of T that are identical to some cyclic rotation of P . The CPM problem can also be solved in $\mathcal{O}(n)$ time [14].

Applications where circular strings are considered include the comparison of DNA sequences in bioinformatics [23, 4] as well as the comparison of shapes represented through directional chain codes in image processing [36, 35]. In both applications, it is not sufficient to look for exact (circular) matches. In bioinformatics, we need to account for DNA sequence divergence (e.g., in the comparison of different species or individuals); and in image processing, we need to account for small differences in the comparison of images (e.g., in classifying handwritten digits). This gives rise to the notion of edit distance on circular strings [34, 3].

We say that string U is a (cyclic) rotation of string V if $U = XY$ and $V = YX$ for some strings X, Y , and write $V = \text{rot}^i(U)$, where $i = |X|$; e.g., $U = \text{abcde}$, $X = \text{ab}$, $Y = \text{cde}$, $V = \text{cdeab} = \text{rot}^2(U)$. The edit (Levenshtein) distance $\delta_E(U, V)$ of two strings U and V is the minimal number of letter insertions, deletions and substitutions required to transform U to V . For two strings U and V and an integer $k > 0$, we write $U =_k V$ if $\delta_E(U, V) \leq k$ and we write $U \approx_k V$ if there exists a rotation U' of U such that $U' =_k V$.

For a string U composed of letters $U[0], \dots, U[|U| - 1]$, by $U[i..j] = U[i..j + 1]$ we denote the fragment of U corresponding to the substring $U[i] \dots U[j]$. We say that $T[p..p']$ is a circular k -edit occurrence of pattern P if $P \approx_k T[p..p']$. By $\text{CircOcc}_k(P, T)$ we denote the set of starting positions of circular k -edit occurrences of P in T . Let us define k -Edit CPM (cf. Figure 1).

k -Edit CPM

Input: A text T of length n , a pattern P of length m , and a positive integer k .

Output: A representation of the set $\text{CircOcc}_k(P, T)$. (**Reporting version**)
 Any position $i \in \text{CircOcc}_k(P, T)$, if there is any. (**Decision version**)

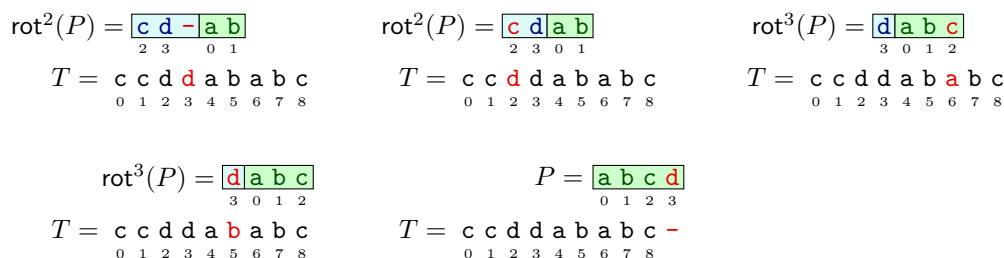


Figure 1 Illustration of the 1-edit circular occurrences of pattern $P = \text{abcd}$ in text $T = \text{ccddababc}$. We have $\text{CircOcc}_1(P, T) = \{1, 2, 3, 5, 6\}$. The letters involved in an edit operation are coloured red.

Related work. The *Hamming distance* of two equal-length strings U and V is the number of mismatches between U and V ; that is, the minimal number of letter substitutions required to transform U to V . Accounting for surplus or missing letters on top of substitutions poses significant challenges. For example, the Hamming distance of two length- n strings can be computed in $\mathcal{O}(n)$ time with a trivial algorithm, while it is known that their edit distance cannot be computed in $\mathcal{O}(n^{2-\epsilon})$ time, for any $\epsilon > 0$, under the Strong Exponential Time Hypothesis [5]. The situation is similar for (non-circular) approximate pattern matching. The k -Mismatch PM problem is quite well-understood as the upper bound of $\tilde{\mathcal{O}}(n + kn/\sqrt{m})$ due to Gawrychowski and Uznański [22], who provided a smooth tradeoff between the algorithms of Amir et al. [2] with running time $\tilde{\mathcal{O}}(n\sqrt{k})$ and Clifford et al. [18] with running time $\tilde{\mathcal{O}}(n + (n/m)k^2)$, is matched by a lower bound for so-called “combinatorial” algorithms.¹ Algorithms that are faster by polylogarithmic factors have been presented in [11, 12, 16]. In contrast, the complexity of the k -Edit PM problem is not yet settled: the current records are the classic $\mathcal{O}(nk)$ -time algorithm of Landau and Vishkin [33] and the very recent $\tilde{\mathcal{O}}(n + (n/m)k^{3.5})$ -time algorithm of Charalampopoulos et al. [17] improving the classic $\mathcal{O}(n + (n/m)k^4)$ -time algorithm of Cole and Hariharan [20]. However, there is no known lower bound for k -Edit PM ruling out an $\mathcal{O}(n + (n/m)k^2)$ -time algorithm.

Recent results in pattern matching under both the Hamming distance and the edit distance for various settings [8, 9, 13, 15, 16, 17, 19, 27, 30, 39] were fuelled by a novel characterization of the structure of approximate occurrences. It is folklore knowledge that if $n \leq 3m/2$, either pattern P has a single exact occurrence in T or both P and the portion of T spanned by occurrences of P are periodic (with the same period). In 2019, Bringmann et al. [9] showed that either P has *few* approximate occurrences (under the Hamming distance) or it is *approximately periodic*. Later, Charalampopoulos et al. [16] tightened this result and proved an analogous statement for approximate occurrences under the edit distance.

Let us now focus on approximate circular pattern matching. The CPM problem under the Hamming distance is called the k -Mismatch CPM problem. An $\mathcal{O}(nk)$ -time algorithm and an $\tilde{\mathcal{O}}(n + (n/m)k^3)$ -time algorithm were proposed for the reporting version of k -Mismatch CPM by Charalampopoulos et al. in [13] and [15], respectively, whereas an $\tilde{\mathcal{O}}(n + (n/m)k^2)$ -time algorithm for its decision version was given in [15]. Further, the authors of [7, 26] presented efficient average-case algorithms for k -Mismatch CPM. The k -Edit CPM problem was considered in [15], where an $\mathcal{O}(nk^2)$ -time algorithm and an $\mathcal{O}(nk \log^3 k)$ -time algorithm were presented for the reporting and decision version, respectively. Until now, no algorithm with worst-case runtime $\mathcal{O}(n + (n/m)k^{\mathcal{O}(1)})$ was known for k -Edit CPM. Such an algorithm is superior over $\mathcal{O}(nk^{\mathcal{O}(1)})$ -time algorithms when the number of allowed errors is small in comparison to the length of the pattern. Here, we propose the first such algorithms.

Our result. In order to represent the output of our algorithm compactly, we need the notion of an *interval chain*. For two integer sets A and B , let $A \oplus B = \{a + b : a \in A, b \in B\}$. We extend this notation for an integer b to $A \oplus b = b \oplus A = A \oplus \{b\}$. An interval chain for an interval I and non-negative integers a and q is a set of the form

$$\text{Chain}(I, a, q) = I \cup (I \oplus q) \cup (I \oplus 2q) \cup \dots \cup (I \oplus aq).$$

Here q is called the *difference* of the interval chain. For example the set of underlined intervals in Figure 4 corresponds to $\text{Chain}(\underline{[3..8]}, 2, 8) = \underline{[3..8]} \cup \underline{[11..16]} \cup \underline{[19..24]}$.

Our main algorithmic result can be stated as follows (cf. Table 1).

¹ Throughout this work, the $\tilde{\mathcal{O}}(\cdot)$ notation hides factors polylogarithmic in the length of the input strings.

24:4 Approximate Circular Pattern Matching Under Edit Distance

■ **Table 1** The upper-bound landscape of pattern matching (PM) and circular PM (CPM) with k edits. In the decision version of k -Edit CPM, the algorithms only find if there exists at least one occurrence and return a witness; otherwise the algorithms report all the occurrences.

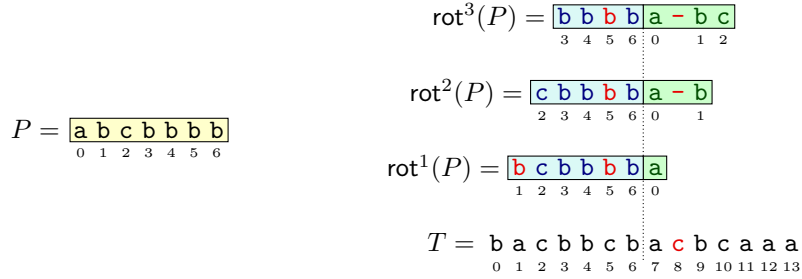
k -Edit PM	Reference	Note	k -Edit CPM	Reference	Note
$\mathcal{O}(n^2)$	[38]	for any k	$\mathcal{O}(nk^2)$	[15]	
$\mathcal{O}(nk^2)$	[32]		$\tilde{\mathcal{O}}(nk)$	[15]	decision
$\mathcal{O}(nk)$	[33]		$\mathcal{O}(n + k^6 \cdot n/m)$	This work	
$\tilde{\mathcal{O}}(n + k^{\frac{25}{3}} \cdot n/m^{\frac{1}{3}})$	[37]		$\tilde{\mathcal{O}}(n + k^5 \cdot n/m)$	This work	decision
$\mathcal{O}(n + k^4 \cdot n/m)$	[20]				
$\tilde{\mathcal{O}}(n + k^{3.5} \cdot n/m)$	[17]				

► **Theorem 1.** *The reporting version of the k -Edit CPM problem can be solved in $\mathcal{O}(n + (n/m)k^6)$ time, with the output represented as a union of $\mathcal{O}((n/m)k^6)$ interval chains. The decision version of the k -Edit CPM problem can be solved in $\mathcal{O}(n + (n/m)k^5 \log^3 k)$ time.*

The following notion of an *anchor* (see also Figure 2) is crucial for understanding the structure of (approximate) circular pattern matching.

► **Definition 2.** *A circular k -edit occurrence $T[p..p']$ of P is anchored at position i (called *anchor*) if $\delta_E(T[p..i], Y) + \delta_E(T[i..p'], X) \leq k$, where $P = XY$ for some X, Y . We denote*

$$\text{Anchored}_k(P, T, i) = \{p : T[p..p'] \text{ is anchored at } i \text{ for some } p'\}.$$



■ **Figure 2** The starting positions of circular 2-edit occurrences of pattern P anchored at position 7 in text T are $\text{Anchored}_2(P, T, 7) = \{0, 1, 2, 3, 4\}$; the occurrences at positions 1, 2, 3 are shown.

► **Example 3.** Let $P = \text{a}^{99}\text{b}$ and $T = P^2$. Then $|\text{CircOcc}_0(P, T)| = 101$, while we have only two anchors (0 and 100).

Our algorithm exploits the approximate periodic structure of the two strings in scope. On the way to our main algorithmic result we prove (in the end of Section 2) the following structural result for k -Edit CPM:

► **Theorem 4.** *Consider a pattern P of length m , a positive integer threshold k , and a text T of length $n \leq cm + k$, for a constant $c \geq 1$. Then, either there are only $\mathcal{O}(k^2)$ anchors of circular k -edit occurrences of P in T or some rotation of P is at edit distance $\mathcal{O}(k)$ from a string with period $\mathcal{O}(m/k)$.*

The PILLAR model. We work in the PILLAR model that was introduced in [16] with the aim of unifying approximate pattern matching algorithms across different settings. In this model, we assume that the following primitive PILLAR operations can be performed efficiently, where the argument strings are fragments of strings in a given collection \mathcal{X} :

- $\text{Extract}(S, \ell, r)$: Retrieve string $S[\ell..r]$.
 - $\text{LCP}(S, T)$, $\text{LCP}_R(S, T)$: Compute the length of the longest common prefix/suffix of S, T .
 - $\text{IPM}(S, T)$: Assuming that $|T| \leq 2|S|$, compute the starting positions of all exact occurrences of S in T , expressed as an arithmetic progression.
 - $\text{Access}(S, i)$: Retrieve the letter $S[i]$; $\text{Length}(S)$: Compute the length $|S|$ of the string S .
- The runtime of algorithms in this model can be expressed in terms of the number of primitive PILLAR operations. The result underlying Theorem 1 can be stated as follows.

► **Theorem 5.** *If $n \leq m \leq 2n$, the reporting and decision versions of the k -Edit CPM problem can be solved in $\mathcal{O}(k^6)$ time and $\mathcal{O}(k^5 \log^3 k)$ time in the PILLAR model, respectively.*

Theorem 5 implies Theorem 1 as well as efficient algorithms for k -Edit CPM in internal, dynamic, fully compressed, and quantum settings based on known implementations of the PILLAR model in these settings, as discussed in Appendix C.

Our approach. Every circular k -edit occurrence of P in T is anchored at some position i of T . In the reporting and decision version of the problem, we use the following respective results.

► **Lemma 6** ([14, Lemma 30]). *Given a text T of length n , a pattern P of length m , an integer $k > 0$, and a position i of T , we can compute in $\mathcal{O}(k^2)$ time in the PILLAR model the set $\text{Anchored}_k(P, T, i)$, represented as a union of $\mathcal{O}(k^2)$ intervals, possibly with duplicates.*

For an interval I denote by $\text{AnyAnchored}_k(P, T, I)$ an arbitrarily chosen position in the set $\bigcup_{i \in I} \text{Anchored}_k(P, T, i)$; if this set is empty then the result is *none*.

► **Lemma 7** ([15, Section 4]). *Given a text T of length n , a pattern P of length m , an integer $k > 0$, and an interval I containing up to k positions of T , we can compute $\text{AnyAnchored}_k(P, T, I)$ in $\mathcal{O}(k^2 \log^3 k)$ time in the PILLAR model.*

It will be convenient and sufficient to deal separately with fragments of T of length $\mathcal{O}(m)$, so we can assume w.l.o.g. that $n = \mathcal{O}(m)$. Let $P = P_1P_2$ be a decomposition of the pattern with $|P_1| = \lfloor m/2 \rfloor$. By using Lemma 6 to compute k -edit circular occurrences that are anchored at one of $\mathcal{O}(k^2)$ carefully chosen anchors, we reduce our problem to searching for k -edit (non-circular) occurrences of any length- m substring of a certain fragment V of $P_2P_1P_2$ in a suitable fragment U of T , where both V and U are approximately periodic (there is also a symmetric case where V is a substring of $P_1P_2P_1$).

We achieve this as follows. Let us denote the set of standard (non-circular) k -edit occurrences of a string X in a string Y by

$$\text{Occ}_k(X, Y) = \{i \in [0..|Y|] : Y[i..i'] =_k X \text{ for some } i' \geq i\}.$$

We compute the set $\text{Occ}_k(P_1, T)$ using an algorithm for pattern matching with k edits [16]. If this set is small, it yields a small set of *anchors* for k -edit occurrences of rotations of P that contain P_1 . We also do the same for P_2 . Then, we can apply Lemma 6 to each anchor.

The challenging case is when $\text{Occ}_k(P_1, T)$ is large. The structural result for k -Edit PM then implies that P_1 and the portions of T spanned by approximate occurrences of P_1 are *almost periodic*, i.e., they are at small edit distance from a substring of string Q^∞ , where

Q is a short string. We extend the periodicity in each of $P_2P_1P_2$ and T , allowing for more edits. The reduction is then completed by accounting for some technical considerations and, possibly, calling Lemma 6 $\mathcal{O}(k^2)$ more times.

In order to develop some intuition for how to deal with the almost periodic case, let us briefly discuss how it is dealt with in the case where we are looking for approximate (circular) occurrences under the Hamming distance. The mismatches of each of the two strings (P and T or U and V) with a substring of Q^∞ are called *misperiods*. Now, consider some candidate starting position i of P in T , assuming that both $P[0..|Q|)$ and $T[i..i+|Q|)$ are approximate copies of Q : the number of mismatches of P and $T[i..i+m)$ can be inferred by just looking at the misperiods: it is just the total number of misperiods in P and $T[i..i+m)$ minus the misperiods that are aligned and thus “cancel out”.

For approximate PM under the edit distance, the situation is much more complicated as deletions and insertions can be applied, and hence we cannot have an analogous statement about misperiods “cancelling out”. Following works on (non-circular) k -edit PM, we employ so-called *locked* fragments (see [16, 20]).

Roughly speaking, we partition each of U and V into locked fragments and powers of Q , such that the total length of locked fragments is small and, if a locked fragment is to be aligned with a substring of Q^∞ , we would rather align it with a power of Q . Then, intuitively, one has to overcome technical challenges arising from the nature of the overlap of the locked fragments with a specific circular k -edit occurrence.

We consider different cases depending on whether the fragments of U and V that yield a match imply that any pair of locked fragments (one in U and one in V) overlap. A crucial observation is that, roughly speaking, as we slide a length- m fragment of V over U , $|Q|$ positions at a time, such that the locked fragments in the window in U remain unchanged and do not overlap with locked fragments in V , the edit distance remains unchanged.

2 Reduction of k -Edit CPM to the PeriodicSubMatch Problem

A string $S = S[0..|S|-1]$ is a sequence of *letters* over some alphabet. The string $S[i]S[i+1]\cdots S[j]$, for any indices i, j such that $i \leq j$, is called a *substring* of S . By $S[i..j] = S[i..j+1) = S(i-1..j]$ we denote a *fragment* of S that can be viewed as a positioned substring $S[i]S[i+1]\cdots S[j]$ (it is represented in $\mathcal{O}(1)$ space). We also denote $S^{(j)} = S[j..j+m)$. An integer p such that $0 < p \leq |S|$ is called a *period* of S if $S[i] = S[i+p]$, for all $i \in [0..|S|-p)$. We define *the period* of S as the smallest such p . A string Q is called *primitive* if $Q = W^k$ for a string W and a positive integer k implies that $k = 1$. By $\text{rot}^j(X)$ we denote the string $X[j..|X|)X[0..j)$. We generalize the rotation operation rot to arbitrary integer exponents r as $\text{rot}^r(X) = \text{rot}^{r \bmod |X|}(X)$.

By $\delta_E(X, Y^*)$, $\delta_E(X, *Y)$ and $\delta_E(X, *Y^*)$ we denote the minimum edit distance between string X and any prefix, suffix and substring of string $Y^{|X|+|Y|}$, respectively.

We say that a string U is *almost Q -periodic* if $\delta_E(U, Q^*) \leq 112k$. We write $a \equiv_d b \pmod{q}$ if $a - b \equiv i \pmod{q}$, where $\min(i, q - i) \leq d$ (in other words, a and b are d -approximately congruent modulo q). For example, $11 \equiv_3 21 \pmod{8}$, but $11 \equiv_1 21 \pmod{8}$ does not hold.

A pair of indices (p, x) satisfying $p \in \text{Occ}_k(V^{(x)}, U)$ and $p \equiv_{77k} x + r \pmod{q}$ will be called an approximate match (*app-match*, in short).

The following auxiliary problem, PERIODICSUBMATCH, is illustrated in Figure 3.

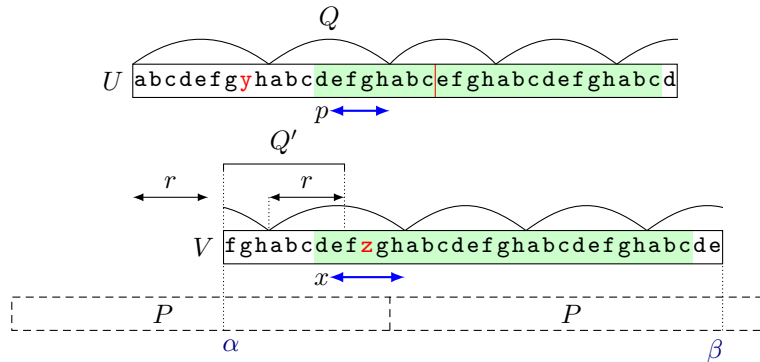
PERIODICSUBMATCH

Input: A primitive string Q , integers m, r, k, α, β , and strings U, V such that

- $m \leq |U| \leq \frac{7}{4}m + 3(k+1)$, $m \leq |V| \leq \frac{3}{2}m$, $q = |Q| \leq \frac{m}{256k}$, $r \in [0..q)$,
- U is almost Q -periodic,
- $V = P^2[\alpha.. \beta]$ (hence, length- m substrings of V are rotations of P),
- V is almost Q' -periodic, where $Q' := \text{rot}^r(Q)$.

Output: $\{p \in \text{Occ}_k(V^{(x)}, U) : p \equiv_{77k} x + r \pmod{q}, x \leq |V| - m\}$.

► Remark 8. Due to the condition that V is a fragment of P^2 , we can apply the operation Anchored_k to compute efficiently the output of PERIODICSUBMATCH in the case when a position j_1 in V is aligned with a position i_1 in U . The efficiency of the whole approach is based on the efficiency of the operation Anchored_k .



■ Figure 3 We have $m = 25$, $k = 2$ and $r = 5$. Edits with respect to the approximate periodicity are marked in red. Green rectangles show that $V^{(x)} =_2 U[p..p+23]$. We have $p = x + r + 1$, so $p \equiv_1 x + r \pmod{q}$. The distances (in blue) from p and x to the starts of next approximate periods Q are the same up to $\Theta(k)$. For the example purposes, we waive the constraint $q = |Q| \leq \frac{m}{256k}$.

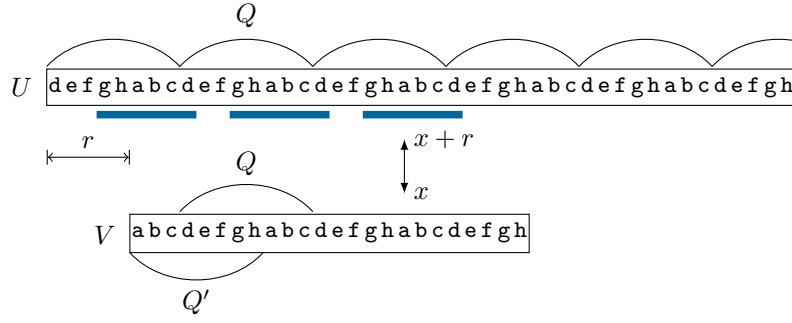
The strings U and V are both close to substrings of Q^∞ . The condition $p \equiv_{\Theta(k)} x + r \pmod{q}$ means that we are only interested in k -edit occurrences $U[p..p']$ of $V^{(x)}$ such that the two substrings are approximately synchronized with respect to the approximate period Q ; see Figure 3. (In particular, no other k -edit occurrences exist.) The constants originate from Theorem 10 and some additional requirements imposed in the proof of Lemma 12.

► Example 9. A very simple double fully periodic case, where both U and V are substrings of Q^∞ , is depicted in Figure 4. Again, we waive the constraint $q = |Q| \leq \frac{m}{256k}$.

The following theorem follows as a combination of several results of [16], see Appendix A.

► Theorem 10 ([16]). If $|T| = n < \frac{3}{2}m + k$, then in $\mathcal{O}(k^4)$ time in the PILLAR model we can compute a representation of the set $\text{Occ}_k(P, T)$. If $|\text{Occ}_k(P, T)|/k > 642045 \cdot (n/m) \cdot k$, the algorithm also returns:

- a primitive string Q satisfying $|Q| \leq m/(256k)$, $\delta_E(P, *Q^*) = \delta_E(P, Q^*) < 2k$, and
- a fragment \bar{T} of T such that $\delta_E(\bar{T}, *Q^*) \leq \delta_E(\bar{T}, Q^*) \leq 24k$, $|\text{Occ}_k(P, T)| = |\text{Occ}_k(P, \bar{T})|$. Moreover, $i \equiv_{24k} 0 \pmod{|Q|}$ for each $i \in \text{Occ}_k(P, \bar{T})$.



■ **Figure 4** A double fully periodic case. Let $k = 2$, $q = |Q| = 8$, and $r = 4$. For $m = 23$, the set of k -edit occurrences of any length- m fragment of V (2 possibilities) in U is the (underlined) interval chain. For $m = 16$ it is a single interval. Position x in V is synchronized with respect to the periodicity with any position p in U such that $p \equiv x + r \pmod{q}$.

▶ **Remark 11.** An $\mathcal{O}((n/m)k^{3.5}\sqrt{\log k \log m})$ -time algorithm for computing a representation of the set $\text{Occ}_k(P, T)$ using $\mathcal{O}(k^3)$ arithmetic progressions was presented in [17]. The simpler result from [16] is sufficient for our needs.

▶ **Lemma 12.** *If $n = \mathcal{O}(m)$, then k -Edit CPM can be reduced in $\mathcal{O}(k^4)$ time in the PILLAR model to at most two instances of the PERIODICSUBMATCH problem. The output to k -Edit CPM is a union of the outputs of the two PERIODICSUBMATCH instances and $\mathcal{O}(k^4)$ intervals.*

Sketch of the proof. The proof resembles the proof of [15, Lemma 12] for Hamming distance. Let us partition P to two (roughly) equal chunks, P_1 of length $\lfloor m/2 \rfloor$ and P_2 of length $\lceil m/2 \rceil$. Each circular k -edit occurrence of P in T implies a standard k -edit occurrence of at least one of P_1 and P_2 . We focus on the case when it implies such an occurrence of P_1 , noting that the computations for P_2 are symmetric.

For a fragment T' of T , we denote by $\text{Implied}_k(P_1, T')$ the set of circular k -edit occurrences of P in T in which a k -edit occurrence of P_1 is contained in T' .

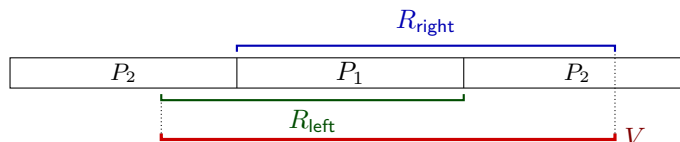
We cover T with fragments of length $\lfloor \frac{3}{2}|P_1| \rfloor + k$ starting at multiples of $\lfloor \frac{1}{2}|P_1| \rfloor$. (The last fragments can be shorter.) For each of the fragments T' of T , we will compute a representation of a set A such that $\text{Implied}_k(P_1, T') \subseteq A \subseteq \text{CircOcc}_k(P, T)$. If $|\text{Occ}_k(P_1, T')| = \mathcal{O}(k^2)$, we use the following fact whose proof is based on anchors.

▷ **Claim 13.** *If the set $\text{Occ}_k(P_1, T')$ for a fragment T' of T has size $\mathcal{O}(k^2)$ and is given, then a set of positions A such that $\text{Implied}_k(P_1, T') \subseteq A \subseteq \text{CircOcc}_k(P, T)$, represented as a union of $\mathcal{O}(k^4)$ intervals, can be computed in $\mathcal{O}(k^4)$ time in the PILLAR model.*

Proof. We compute the set $\text{Anchored}_k(P, T, i + s)$ for each position $i \in \text{Occ}_k(P_1, T')$, where s is the starting position of T' in T (i.e., $T' = T[s..s + |T'|]$). By Lemma 6, this set is represented as a union of $\mathcal{O}(k^2)$ intervals and can be computed in $\mathcal{O}(k^2)$ time in the PILLAR model. Since $|\text{Occ}_k(P_1, T')| = \mathcal{O}(k^2)$, the union A of all these sets contains $\mathcal{O}(k^4)$ intervals and is computed in $\mathcal{O}(k^4)$ total time. Clearly, A satisfies the required inclusions. \triangleleft

If $|\text{Occ}_k(P_1, T')| = \mathcal{O}(k^2)$, Claim 13 can be applied. Otherwise, we can assume that $|\text{Occ}_k(P_1, T')| > ck^2$ for a sufficiently large constant c . Then, by Theorem 10, P_1 and the relevant part \bar{T}' of T' are both almost Q -periodic.

Computing V . String V is obtained by extending the approximate periodicity of the middle fragment P_1 in $P_2P_1P_2$ towards both directions. (Note that all rotations of P that contain its first half P_1 are substrings of $P_2P_1P_2$.) In each direction, we stop extending when either $c'k$ errors to a prefix (suffix) of $Q^{|V|}$ are accumulated, for a specified constant c' , or we reach the end of the string. In the former case, we obtain a so-called repetitive region R_{right} with a prefix P_1 (R_{left} with a suffix P_1 , respectively); see Figure 5.



■ **Figure 5** String V (shown in brown) and repetitive regions R_{left} and R_{right} in $P_2P_1P_2$.

Intuitively, a repetitive region is a fragment that is sufficiently long and almost periodic, but also sufficiently far from being periodic. Thus a rotation of P that contains P_1 either contains one of the repetitive regions or it is contained in V . A repetitive region is known [16] to have $\mathcal{O}(k^2)$ occurrences in a string of length $\mathcal{O}(m)$, so the former case can be solved in $\mathcal{O}(k^4)$ time with the aid of anchors as in Claim 13. The latter case will lead to PERIODICSUBMATCH.

Computing U . We obtain U by extending the approximate periodicity of \bar{T}' to T towards both directions. We extend it to the left until one of the following conditions is satisfied: the appended substring is at edit distance at least $c'k$ from all suffixes of $(\text{rot}^x(Q))^{2n}$, for all $x \in [-34k \dots 34k]$, the beginning of T is reached, or roughly $m/2 + k$ letters have been inspected.

The extension to the left is symmetric. Finally, we prove that all occurrences in $\text{Implied}_k(P_1, T')$ that correspond to length- m substrings of V are contained in U . The approximate congruence $\text{mod } |Q|$ in PERIODICSUBMATCH follows from the analogous condition in Theorem 10. ◀

Let us now restate and prove our structural result.

► **Theorem 4.** *Consider a pattern P of length m , a positive integer threshold k , and a text T of length $n \leq cm + k$, for a constant $c \geq 1$. Then, either there are only $\mathcal{O}(k^2)$ anchors of circular k -edit occurrences of P in T or some rotation of P is at edit distance $\mathcal{O}(k)$ from a string with period $\mathcal{O}(m/k)$.*

Proof. Theorem 4 readily follows from the proof of Lemma 12. If $|V| \geq m$, then some rotation of P is almost periodic. Otherwise, we only have $\mathcal{O}(k^2)$ anchors for approximate circular occurrences (stemming from occurrences of some of P_1, P_2 , or a repetitive region obtained by extending either of P_1 or P_2 in some direction). ◀

3 Locked Fragments

The notion of locked fragments originates from [20]. We use them as defined in [16]. Let us state [16, Lemma 6.9]² with $d_S = 112k$, for $k > 0$; this characterization of locked fragments will be sufficient for our purposes. See Figure 6 for an illustration.

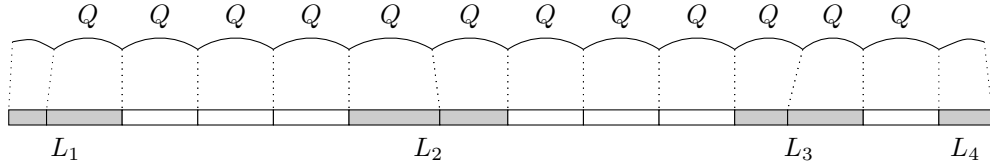
² The original lemma also concluded that L_1 is a so-called k -locked prefix; however, this property is not needed here (and, in particular, a k -locked string is also locked).

24:10 Approximate Circular Pattern Matching Under Edit Distance

► **Lemma 14** (see [16, Lemmas 5.6 and 6.9]). *Let S denote a string, Q denote a primitive string, $q = |Q|$, and suppose that $\delta_E(S, *Q^*) \leq 112k$ and $|S| \geq 225kq$ for some positive integer k .*

Then there is an algorithm which in $\mathcal{O}(k^2)$ time in the PILLAR model computes disjoint locked fragments L_1, \dots, L_ℓ of S satisfying:

- (a) $S = L_1 Q^{\alpha_1} L_2 Q^{\alpha_2} \dots L_{\ell-1} Q^{\alpha_{\ell-1}} L_\ell$, where $\alpha_i \in \mathbb{Z}_{>0}$ for all i ,
- (b) $\delta_E(S, *Q^*) = \sum_{i=1}^{\ell} \delta_E(L_i, *Q^*)$ and $\delta_E(L_i, *Q^*) > 0$ for all $i \in (1.. \ell)$,
- (c) $\ell = \mathcal{O}(k)$ and $\sum_{i=1}^{\ell} |L_i| \leq 676kq$.



■ **Figure 6** Illustration of Lemma 14. We have a decomposition $S = L_1 \cdot Q^3 \cdot L_2 \cdot Q^3 \cdot L_3 \cdot Q^1 \cdot L_4$. L_1 is an approximate suffix of $Q^{|S|}$, L_4 is an approximate prefix of Q^∞ , and internal gray parts are *approximate* powers of Q . The remaining (white) fragments are *exact* powers of Q .

Let us consider the decompositions obtained by applying Lemma 14 to strings U and V from PERIODICSUBMATCH w.r.t. the string Q . Strings U and V are almost Q -periodic and almost Q' -periodic, respectively, so $\delta_E(U, *Q^*), \delta_E(V, *Q^*) \leq 112k$. Moreover, $|U|, |V| \geq m \geq 256kq > 225kq$. Thus, U and V satisfy the assumptions of the lemma. If any of the decompositions starts with a locked prefix of length smaller than q (possibly empty) or ends with a locked suffix of length smaller than q , we extend the locked fragment by a copy of Q and possibly by a neighbouring locked fragment if this copy was the only copy separating them. The total length of the locked fragments increases by at most $2q \leq 2kq$, so it is bounded by $678kq$.

4 Overlap Case of PERIODICSUBMATCH

We consider all possible *offsets* Δ (integers $\Delta \in (-|V|..|U|)$) by which we can shift V , looking for a length- m substring of V that approximately matches a substring of U .

We denote $\text{Ext}_t(X) = \bigcup_{x \in X} \{y : |x - y| \leq t\}$. Denote also by $\text{locked}(U)$, $\text{locked}(V)$ the set of positions in all locked fragments in U , V , respectively.

► **Definition 15.** Δ is a t -overlap offset if there are positions p, x such that $p - x = \Delta$, and

$$p \in X \oplus \{-m, 0, m\}, \quad x \in Y \oplus \{-m, 0, m\} \quad \text{where } X = \text{Ext}_t(\text{locked}(U)), Y = \text{locked}(V).$$

Otherwise Δ is a t -non-overlap offset.

An integer Δ is called a *valid offset* if $\Delta \equiv_{77k} r \pmod{q}$. (Recall the definition of r in PERIODICSUBMATCH.) For two integer sets A and B , let $A \ominus B = \{a - b : a \in A, b \in B\}$.

► **Observation 16.** For any intervals I, J , the set $\text{Ext}_t(I \ominus J)$ is an interval of size $|I| + |J| - 1 + 2t$ that can be computed in $\mathcal{O}(1)$ time.

► **Lemma 17.** The set of valid t -overlap offsets can be represented as a union of $\mathcal{O}(k^2 + k^2t/q)$ intervals of length $\mathcal{O}(k)$ each. This representation can be computed in $\mathcal{O}(k^2 + k^2t/q)$ time in the PILLAR model.

Proof. Let $\ell_1, \dots, \ell_{n_1}$ and $\ell'_1, \dots, \ell'_{n_2}$ be the lengths of locked fragments in U and V , respectively, and $s_1 = \sum_{i=1}^{n_1} \ell_i$, $s_2 = \sum_{i=1}^{n_2} \ell'_i$. By point (c) in Lemma 14, we have $n_1 + n_2 = \mathcal{O}(k)$ and $s_1 + s_2 = \mathcal{O}(kq)$. By Observation 16, the set of t -overlap offsets is a union of $\mathcal{O}(k^2)$ intervals of total length proportional to:

$$\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} (\ell_i + \ell'_j + t) = n_1 n_2 t + n_2 \sum_{i=1}^{n_1} \ell_i + n_1 \sum_{j=1}^{n_2} \ell'_j \leq n_1 n_2 t + (n_1 + n_2)(s_1 + s_2) = \mathcal{O}(k^2(t+q)).$$

The intervals can be computed in $\mathcal{O}(k^2)$ time. An interval of length ℓ contains $\mathcal{O}(k + \ell k/q)$ valid offsets grouped into $\mathcal{O}(1 + \ell/q)$ intervals of length $\mathcal{O}(k)$ each. These maximal intervals of offsets can be computed in $\mathcal{O}(1 + \ell/q)$ time via elementary modular arithmetics. Therefore, the number of intervals of t -overlap offsets that are valid is proportional to

$$\left(\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} 1 \right) + \mathcal{O}(k^2 + k^2 t/q) = \mathcal{O}(k^2 + k^2 t/q)$$

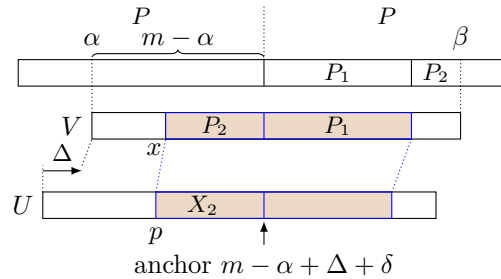
and all of them can be computed in $\mathcal{O}(k^2 + k^2 t/q)$ time. ◀

An app-match (p, x) is called a **t -overlap app-match** if and only if $p - x$ is a t -overlap offset. In this section, we consider t -overlap app-matches. In Section 5, we consider t -non-overlap app-matches: app-matches (p, x) such that $p - x$ is a t -non-overlap offset, for $t = \Theta(qk)$.

It follows from the statement of PERIODICSUBMATCH that if (p, x) is an app-match, then $p - x$ is a valid offset. The following fact, together with Lemma 6, implies a fast algorithm for computing the following set for a given offset Δ :

$$\{p \in \text{Occ}_k(V^{(x)}, U) : \Delta = p - x, \Delta \equiv_{77k} r \pmod{q}\}.$$

► **Fact 18.** *If (p, x) is an app-match, $\Delta = p - x$ and $\Delta' = m - \alpha + \Delta$, then the corresponding circular k -edit occurrence $U[p..p']$ is anchored at a position in $[\Delta' - k.. \Delta' + k]$; see Figure 7.*



■ **Figure 7** The anchor in U is at position $m - \alpha + \Delta + \delta$, where $\delta = |X_2| - |P_2| \in [-k..k]$ (since $\delta_E(X_2, P_2) \leq k$).

Using Lemmas 6 and 7 we obtain the following corollary.

► **Corollary 19.** *Let I be an interval of size $\mathcal{O}(k)$. All positions p for which there exists an app-match (p, x) such that $p - x \in I$, represented as a union of $\mathcal{O}(k^3)$ intervals, can be computed in $\mathcal{O}(k^3)$ time in the PILLAR model. Moreover, one can check if there is any app-match (p, x) with $p - x \in I$ in $\mathcal{O}(k^2 \log^3 k)$ time in the PILLAR model.*

The solution of the overlap case is presented in Algorithm 1. Lemma 17 together with Fact 18 and Corollary 19 imply the following lemma.

■ **Algorithm 1** Overlap case: reporting version.

```

Compute the decompositions of  $U$  and  $V$  into locked fragments;
// Compute the set  $\Lambda$  of  $(t+k)$ -overlap offsets, being a union of  $\mathcal{O}(k^2)$  intervals:
foreach locked fragment  $U[i_{\min} \dots i_{\max}]$  do
  foreach locked fragment  $V[j_{\min} \dots j_{\max}]$  do
     $\Lambda := \Lambda \cup ([i_{\min} - j_{\max} - (t+k) \dots i_{\max} - j_{\min} + (t+k)] \oplus \{-m, 0, m\})$ ;
// Compute the set  $\Gamma$  of valid  $(t+k)$ -overlap offsets,
// represented as a union of  $\mathcal{O}(k^2 + k^2(t+k)/q)$  intervals of size  $\mathcal{O}(k)$  each:
foreach interval  $I$  of offsets in  $\Lambda$  do
   $\Gamma := \Gamma \cup \{\text{maximal intervals representing } \{i \in I : i \equiv_{77k} r \pmod{q}\}\}$ ;

foreach interval  $[i_{\min} \dots i_{\max}]$  of offsets in  $\Gamma$ , with  $i_{\max} - i_{\min} = \mathcal{O}(k)$  do
   $J := [i_{\min} \dots i_{\max}] \oplus (m - \alpha)$ ;
  report  $\bigcup_{a \in J} \text{Anchored}_k(P, U, a)$ ;
```

► **Lemma 20.** *Let B be the output of Algorithm 1. Then $B \subseteq \text{CircOcc}_k(P, U)$ and every t -overlap app-match occurrence p is in B .*

Moreover, if $t = \mathcal{O}(kq)$, Algorithm 1 works in $\mathcal{O}(k^6)$ time in the PILLAR model with the output represented as a union of $\mathcal{O}(k^6)$ intervals.

Proof. Consider a t -overlap app-match (p, x) . Then, there exists an anchor a such that $p \in \text{Anchored}_k(P, U, a)$, and $y = a - (m - \alpha)$ is a $(t+k)$ -overlap offset, since we have

$$\delta_E(V[x \dots m - \alpha], U[p \dots a]) + \delta_E(V[m - \alpha \dots x + m], U[a \dots p']) \leq k.$$

Now, y is in some interval $[i_{\min} \dots i_{\max}] \in \Gamma$, as the union of the elements of Γ comprises the set of valid $(t+k)$ -overlap offsets. Then, since $y \in [i_{\min} \dots i_{\max}]$, we have $a = y + (m - \alpha) \in [i_{\min} \dots i_{\max}] \oplus (m - \alpha)$, and hence a is in one of the sets J constructed in the penultimate line of Algorithm 1. In the case when $t = \mathcal{O}(kq)$, using Lemma 17, we compute, in $\mathcal{O}(k^3)$ time, $\mathcal{O}(k^3)$ intervals of anchors, of size $\mathcal{O}(k)$ each. The time complexity and the fact that the algorithm returns the output as a union of $\mathcal{O}(k^6)$ intervals follows by a direct application of Corollary 19 to each interval of anchors. ◀

To obtain the next corollary, we replace the last line of Algorithm 1 by:

```

if  $\text{AnyAnchored}_k(P, U, J) \neq \text{none}$  then return  $\text{AnyAnchored}_k(P, U, J)$ ;
```

► **Corollary 21.** *If $t = \mathcal{O}(kq)$, one can check if $B \neq \emptyset$ and, if so, return an arbitrary element of B , in $\mathcal{O}(k^5 \log^3 k)$ time in the PILLAR model.*

5 Non-Overlap Case of PERIODICSUBMATCH

Recall that an app-match (p, x) is called a t -non-overlap app-match if and only if $p - x$ is a t -non-overlap offset. In this section we assume $t = \Theta(kq)$. The set of t -non-overlap offsets is too large, but it has a short representation.

► **Lemma 22.** *The set of t -non-overlap offsets can be partitioned into $\mathcal{O}(k^2)$ maximal intervals in $\mathcal{O}(k^2 \log \log k)$ time in the PILLAR model.*

Proof. There are $\mathcal{O}(k)$ locked fragments in U and V . By Observation 16, every pair of locked fragments, one from U and one from V , induces an interval of t -overlap offsets that can be computed in $\mathcal{O}(1)$ time. The complement of the union of these offsets can be computed in $\mathcal{O}(k^2 \log \log k)$ time by sorting the endpoints of the intervals using integer sorting [24]. ◀

We denote by $\mathbf{NonOv}(t)$ the set of maximal intervals yielded by the above lemma. For simplicity, we mostly discuss the decision version of the problem in this section; the correctness proof for the reporting version requires a few further technical arguments.

Let $\lambda_k = (112k + 3) \cdot (3k + 10) \cdot q + 678kq$.

► **Lemma 23.** *If $\lambda_k > \frac{m}{2}$, PERIODICSUBMATCH can be solved in $\mathcal{O}(k^5)$ time in the PILLAR model, with the output represented as a union of $\mathcal{O}(k^5)$ intervals.*

Proof. We have $m = \mathcal{O}(k^2q)$. As $\mathcal{O}(k)$ out of every q consecutive offsets are valid and they can be grouped in at most two intervals, there are $\mathcal{O}(mk/q) = \mathcal{O}(k^3)$ valid offsets, which are grouped into $\mathcal{O}(k^2)$ intervals of size $\mathcal{O}(k)$ each. Let the set of such intervals be \mathcal{J} . The time complexity and output size follow from an application of Corollary 19 to the $\mathcal{O}(k)$ -size interval of anchors corresponding to each $J \in \mathcal{J}$, as in the last three lines of Algorithm 1. ◀

Henceforth we assume that $\lambda_k \leq \frac{m}{2}$. Let W be the longest fragment of V such that each length- m fragment of V contains W , i.e., $W = V[|V| - m .. m]$.

► **Observation 24.** *If $\lambda_k \leq \frac{m}{2}$, then W contains a fragment equal Q^{3k+9} that is disjoint from locked fragments in V .*

Proof. We have $|W| \geq \frac{m}{2}$ since $|V| \leq \frac{3}{2}m$. By Lemma 14, V contains at most $112k + 2$ locked fragments. Their total length does not exceed $678kq$. By the pigeonhole principle, as $\lambda_k = (112k + 3) \cdot ((3k + 10) \cdot q) + 678kq \leq |W|$, string W contains a substring of length at least $(3k + 10)q$ that is disjoint from locked fragments. By Lemma 14, this substring is a substring of Q^∞ and thus contains a copy of Q^{3k+9} . ◀

► **Definition 25 (sample).** *We select an arbitrary fragment $V[j .. j']$ equal Q^{3k+9} of W that is disjoint from locked fragments in V ; then the middle fragment $V[j_1 .. j_2]$ of $V[j .. j']$ equal Q^{k+1} becomes an additional locked fragment. The fragment $V[j_1 .. j_2]$ is called the sample.*

When computing t -overlap offsets with the algorithm of Section 4, we treat the sample as a locked fragment; the total length of the locked fragments is then still $\mathcal{O}(kq)$.

Henceforth we replace P by its rotation $\text{rot}^y(P)$, where $y = (j_1 + \alpha) \bmod m$. Let us note that after this change, the sets $\mathbf{Anchored}_k$ can be computed equally efficiently as the sets $\mathbf{Anchored}_k$ for the original P . This follows from the fact that the algorithm underlying Lemma 6 does not use IPM queries, and the remaining queries from the PILLAR model can easily be implemented in $\mathcal{O}(1)$ time if an input string is given by its cyclic rotation.

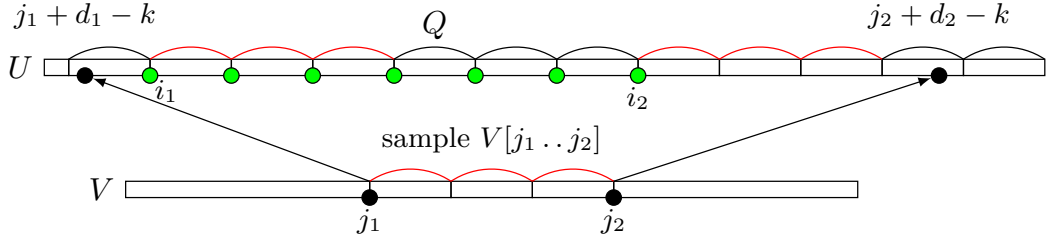
For an interval $I = [i_1 .. i_2]$ and a string S , by $S[I]$ we denote $S[i_1 .. i_2]$. We denote $\hat{q} = 2(k + 6)(q + 3)$; the constants originate from the proof of Lemma 29.

► **Observation 26.** *Let $[d_1 .. d_2] \in \mathbf{NonOv}(t)$. If $V[j_1 .. j_2]$ is the sample in V , then $U[j_1 + d_1 - t .. j_2 + d_2 + t]$ does not contain a position in a locked fragment, since we defined the sample as an (exceptional) locked fragment.*

► **Definition 27.** *For an interval $D = [d_1 .. d_2]$, denote*

$$\text{scope}(D) = \text{Ext}_k([j_1 .. j_2] \oplus D), \quad \text{CritPos}(D) = \text{Occ}_0(Q^{k+1}, U[\text{scope}(D)]).$$

The positions in $\bigcup_{D \in \mathbf{NonOv}(\hat{q})} \text{CritPos}(D)$ are called critical positions; see Figure 8.



■ **Figure 8** Illustration of basic parameters in the algorithm: $D = [d_1 \dots d_2]$, $I = \text{Ext}_k([j_1 \dots j_2] \oplus D)$. We have $I \cap \text{locked}(U) = \emptyset$. $\text{CritPos}(D)$ consists of critical positions shown as green circles.

The main idea of the proof of the next lemma is as follows: in an app-match for an offset from D , at least one copy of Q from the sample must match a copy of Q in $\text{scope}(D)$ exactly. For $D \in \text{NonOv}(\hat{q})$, $\text{scope}(D)$ is a substring of Q^∞ . This implies that the whole sample matches a fragment of $\text{scope}(D)$ exactly, which is how critical positions were defined.

► **Lemma 28.** *For each position p for which there is a \hat{q} -non-overlap app-match (p, x) , we have $p \in \bigcup \{ \text{Anchored}_k(P, U, i) : i \text{ is a critical position} \}$.*

The lemma says that it would be enough to consider $\text{Anchored}_k(P, T, i)$ for all critical positions i . Unfortunately, the total number of critical positions can be too large; however, they are grouped into $\mathcal{O}(k^2)$ arithmetic progressions and it is enough to consider the first and the last position in each such progression. In the decision version we use Algorithm 2.

■ **Algorithm 2** Non-overlap case: decision version.

```

Compute decompositions of  $U$  and  $V$  into locked fragments and the sample;
Compute  $\text{NonOv}(\hat{q})$ ;
foreach interval of non-overlap offsets  $D \in \text{NonOv}(\hat{q})$  do
     $i_1 := \min \text{CritPos}(D)$ ;  $i_2 := \max \text{CritPos}(D)$ ;
    if  $\text{AnyAnchored}_k(P, U, i_1) \neq \text{none}$  then return  $\text{AnyAnchored}_k(P, U, i_1)$ ;
    if  $\text{AnyAnchored}_k(P, U, i_2) \neq \text{none}$  then return  $\text{AnyAnchored}_k(P, U, i_2)$ ;
return none;
    
```

► **Lemma 29.** *Assume that $\lambda_k \leq \frac{m}{2}$. Algorithm 2 works in $\mathcal{O}(k^4)$ time in the PILLAR model and returns a circular k -edit occurrence of P in U if any \hat{q} -non-overlap app-match exists.*

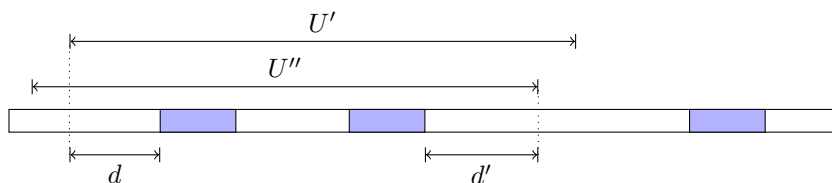
Proof of Theorem 5, decision version. If $\lambda_k \leq \frac{m}{2}$, Lemma 29 and Corollary 21 cover the decision version of PERIODICSUBMATCH for \hat{q} -non-overlap offsets and \hat{q} -overlap offsets, respectively. Together with Lemma 23 used for the corner case that $\lambda_k > \frac{m}{2}$, they yield a solution to a decision version of PERIODICSUBMATCH. The decision version from Theorem 5 is obtained through the reduction to PERIODICSUBMATCH of Lemma 12, as the time complexities of all the algorithms in the PILLAR model are $\mathcal{O}(k^5 \log^3 k)$. ◀

5.1 Overview of the proof of Lemma 29

The complexity of Algorithm 2 directly follows from Lemma 6 (computing Anchored_k), Lemma 14 (computing decompositions into locked fragments) and Lemma 22 (computing $\text{NonOv}(\hat{q})$).

For a fragment $F = U[I]$ ($F = V[I]$, respectively), we denote by $locked(F)$ the set $I \cap locked(U)$ ($I \cap locked(V)$, respectively).

► **Definition 30.** Two fragments F_1, F_2 (both of U or both of V) are called locked-equivalent if $locked(F_1) = locked(F_2)$ and there are no locked positions in a prefix and a suffix of length $(k + 4)q$ in F_1 and in F_2 ; see Figure 9.



■ **Figure 9** The gray boxes correspond to locked fragments, while $d, d' \geq (k + 4)q$. The fragments U' and U'' are locked-equivalent.

We extend Definition 2 and say that a circular k -edit occurrence $T[p..p']$ of P is x -anchored at position i if $\delta_E(T[p..i], P[x..m]) + \delta_E(T[i..p'], P[0..x]) \leq k$. For a fragment $Y = X[i..j]$ and integer y , we denote $shift(Y, y) = X[i + y..j + y]$.

Let $i_1 = \min \text{CritPos}(D)$, $i_2 = \max \text{CritPos}(D)$ as in Algorithm 2. The next lemma shows that in many cases, if $U[p..p']$ forms a \hat{q} -non-overlap app-match that is anchored at a critical position i such that $i_1 < i < i_2$, then the same fragment or a fragment shifted by q positions forms a \hat{q} -non-overlap app-match anchored at a critical position $i \pm q$.

► **Lemma 31.** Let $V[j_1..j_2] = Q^{k+1}$ be the sample, $\mathbf{C} = \text{CritPos}(D)$ where $D \in \text{NonOv}(\hat{q}/2)$, and $i \in \mathbf{C}$. If $I = [p..p']$ and $U[I]$ is x -anchored at i , then for any $y \in \{q, -q\}$:

- (a) If $U[I]$ and $U[I \oplus y]$ are locked-equivalent and $i + y \in \mathbf{C}$, then $U[I \oplus y]$ is x -anchored at $i + y$.
- (b) If $V' = V^{(x)}$ and $shift(V', y)$ are locked-equivalent and $i - y \in \mathbf{C}$, then $U[I]$ is $(x + y)$ -anchored at $i - y$.

Sketch of the proof. For part (a), $y = -q$, it suffices to show that:

$$\delta_E(U[p..i], V[x..j_1]) = \delta_E(U[p - q..i - q], V[x..j_1]), \tag{1}$$

$$\delta_E(U[i..p'], V[j_1..x + m]) = \delta_E(U[i - q..p' - q], V[j_1..x + m]). \tag{2}$$

For example, in (1), fragments $X := U[p..i]$ and $X' := U[p - q..i - q]$ contain the same locked fragments of U , just shifted by q positions. Each of X, X' has a length- $\Theta(kq)$ prefix and suffix without locked positions; for the prefix, this follows from the locked-equivalence assumption, whereas for the suffix, we use Observation 26. To prove (1), we notice that an optimal alignment of X and $Y := V[x..j_1]$ can be divided into parts, such that every second part is a power of Q , and in the remaining parts, there are locked fragments in only one of the strings. This follows from the fact that $p - x$ is a $\Theta(kq)$ -non-overlap offset, so the locked fragments of X and the locked fragments of Y are “well-separated”. Finally, we show that with such an alignment, shifting X by q positions changes an optimal alignment in a structured manner and so the edit distance to Y stays the same. ◀

The sets Anchored_k contain too little information for proving the correctness of the algorithm. It is important that for any of the $\mathcal{O}(k^2)$ intervals of positions of app-matches $[p_l..p_r]$ returned by a call to $\text{Anchored}_k(P, U, i)$, there exist positions $[p'_l..p'_r]$ and values

$[x_l \dots x_r]$ of cyclic rotations such that $U[p_l \dots p'_l]$ is x_l -anchored at i , $U[p_l + 1 \dots p'_l + 1]$ is $(x_l + 1)$ -anchored at i , etc. Therefore we define

$$\text{Anchored}'_k(P, T, i) = \{ (p, p', x) : T[p \dots p'] \text{ is } x\text{-anchored at } i \}.$$

For a triad (I, J, L) of intervals of the same size, we denote the combined set of triples

$$\text{zip}(I, J, L) = \{ (a+t, b+t, c+t) : 0 \leq t < |I| \}, \text{ where } (a, b, c) = (\min(I), \min(J), \min(L)).$$

For example $\text{zip}([1 \dots 3], [5 \dots 7], [2 \dots 4]) = \{(1, 5, 2), (2, 6, 3), (3, 7, 4)\}$. (Treating I, J, L as lists, this can be written in Python as `set(zip(I, J, L))`.) Just like Lemma 31 states a relation of single elements of the sets Anchored_k for anchors at two consecutive critical positions, the next lemma shows what happens to intervals of positions in Anchored_k (together with end-positions of app-matches and the rotations of P).

Denote by $\text{L-cut}_q(I)$, $\text{R-cut}_q(I)$ the operations of removing from the interval I its prefix/suffix of length q , possibly obtaining an empty interval. For example, $\text{L-cut}_2([2 \dots 5]) = [4 \dots 5]$.

► **Lemma 32.** *Let $D \in \text{NonOv}(\hat{q})$, $i_1 = \min \text{CritPos}(D)$, $i_2 = \max \text{CritPos}(D)$. Assume that for some $i \in \text{CritPos}(D)$ such that $i \neq i_1, i_2$, we have $\text{zip}(I_1, I_2, I_3) \subseteq \text{Anchored}'_k(P, U, i)$, where $|I_1| = |I_2| = |I_3| \geq q$. Then:*

$$\begin{aligned} \text{zip}(\text{L-cut}_q(I_1), \text{L-cut}_q(I_2), \text{R-cut}_q(I_3)) &\subseteq \text{Anchored}'_k(P, U, i + q), \\ \text{zip}(\text{R-cut}_q(I_1), \text{R-cut}_q(I_2), \text{L-cut}_q(I_3)) &\subseteq \text{Anchored}'_k(P, U, i - q). \end{aligned}$$

For every $p \in \text{Anchored}_k(P, U, i)$ that satisfies the assumption of Lemma 31(b) and $i_1 < i < i_2$, that lemma immediately shows that $p \in \text{Anchored}_k(P, U, i - q) \cap \text{Anchored}_k(P, U, i + q)$. Unfortunately, this assumption does not always hold. However, Lemma 32 shows that this is true for all but at most q elements $p \in \text{Anchored}_k(P, U, i)$.

To prove Lemma 32, roughly speaking, we compute a *superposable partition* of intervals $I_1, I_2, I_3, I_3 \oplus m$, such that in each part, locked fragments can occur only in the parts originating from one of the strings U, V . As before, this is possible thanks to the fact that the offset is non-overlapping; here we use the fact that the definition of t -non-overlap offsets (Definition 15) covers the cases $(\Delta' \pm m) \oplus [-t \dots t]$. Finally, we apply the appropriate point of Lemma 31 to positions in each part in bulk.

Correctness of Algorithm 2. By Lemma 32, if $I \subseteq \text{Anchored}_k(P, U, i)$ for an interval I , then $\text{L-cut}_q(I) \subseteq \text{Anchored}_k(P, U, i + q)$ and $\text{R-cut}_q(I) \subseteq \text{Anchored}_k(P, U, i - q)$. In the proof of Lemma 29, we use Lemma 31 on positions in the first and last q positions of I to show that one of the following conditions hold:

$$(\star) I \subseteq \text{Anchored}_k(P, U, i \pm q) \text{ or } (\star\star) I \ominus q \subseteq \text{Anchored}_k(P, U, i - q).$$

In case (\star) , by induction we show that $I \subseteq \text{Anchored}_k(P, U, i_1) \cup \text{Anchored}_k(P, U, i_2)$. In case $(\star\star)$, we show by induction that $J := I \oplus (i_1 - i) \subseteq \text{Anchored}_k(P, U, i_1)$. This way we prove the correctness of Algorithm 2.

Reporting version. In the reporting version of Algorithm 2, we prove that the condition $(\star\star)$ implies an interval chain of positions $\text{Chain}(J, (i_2 - i_1)/q, q)$ and show a way to efficiently verify this condition given interval I (see Algorithm 3 in Appendix B). Finally, the reporting version of Theorem 5 follows from the reporting version of the overlap case (Lemma 20), the correctness and the complexity of Algorithm 3, the usage of Lemma 23 for the corner case when $\lambda_k > \frac{m}{2}$, and the reduction to `PERIODICSUBMATCH` (Lemma 12).

► **Remark 33.** In both versions (decision, reporting), the bottleneck of the algorithm's running time is the overlap case, while the most technically demanding part is the non-overlap case.

References

- 1 Andris Ambainis. Quantum query algorithms and lower bounds. In *Classical and New Paradigms of Computation and their Complexity Hierarchies*, pages 15–32, 2004. doi:10.1007/978-1-4020-2776-5_2.
- 2 Amihod Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.
- 3 Lorraine A. K. Ayad, Carl Barton, and Solon P. Pissis. A faster and more accurate heuristic for cyclic edit distance computation. *Pattern Recognition Letters*, 88:81–87, 2017. doi:10.1016/j.patrec.2017.01.018.
- 4 Lorraine A. K. Ayad and Solon P. Pissis. MARS: Improving multiple circular sequence alignment using refined sequences. *BMC Genomics*, 18(1):86, 2017. doi:10.1186/s12864-016-3477-5.
- 5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM Journal on Computing*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.
- 6 Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52:3457–3467, 1995. doi:10.1103/PhysRevA.52.3457.
- 7 Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Fast algorithms for approximate circular string matching. *Algorithms for Molecular Biology*, 9:9, 2014. doi:10.1186/1748-7188-9-9.
- 8 Gabriel Bathie, Tomasz Kociumaka, and Tatiana Starikovskaya. Small-space algorithms for the online language distance problem for palindromes and squares. In *34th International Symposium on Algorithms and Computation, ISAAC 2023*, volume 283 of *LIPICs*, pages 10:1–10:17, 2023. doi:10.4230/LIPICs.ISAAC.2023.10.
- 9 Karl Bringmann, Philip Wellnitz, and Marvin Künnemann. Few matches or almost periodicity: Faster pattern matching with mismatches in compressed texts. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1126–1145. SIAM, 2019. doi:10.1137/1.9781611975482.69.
- 10 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.
- 11 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 643–656. ACM, 2020. doi:10.1145/3357713.3384266.
- 12 Timothy M. Chan, Ce Jin, Virginia Vassilevska Williams, and Yinzhan Xu. Faster algorithms for text-to-pattern Hamming distances. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 2188–2203. IEEE, 2023. doi:10.1109/FOCS57990.2023.00136.
- 13 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Waleń, and Wiktor Zuba. Circular pattern matching with k mismatches. *Journal of Computer and System Sciences*, 115:73–85, 2021. doi:10.1016/j.jcss.2020.07.003.
- 14 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Approximate circular pattern matching. *CoRR*, abs/2208.08915, 2022. arXiv:2208.08915, doi:10.48550/ARXIV.2208.08915.
- 15 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Approximate circular pattern matching. In *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPICs*, pages 35:1–35:19, 2022. doi:10.4230/LIPICs.ESA.2022.35.

- 16 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 978–989. IEEE, 2020. Full version: arXiv:2004.08350v2. doi:10.1109/FOCS46700.2020.00095.
- 17 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster pattern matching under edit distance: A reduction to dynamic puzzle matching and the seaweed monoid of permutation matrices. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 698–707. IEEE, 2022. Full version: arXiv:2204.03087v1. doi:10.1109/FOCS54457.2022.00072.
- 18 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k -mismatch problem revisited. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 2039–2052. SIAM, 2016. doi:10.1137/1.9781611974331.ch142.
- 19 Raphaël Clifford, Paweł Gawrychowski, Tomasz Kociumaka, Daniel P. Martin, and Przemysław Uznański. The dynamic k -mismatch problem. In *33rd Annual Symposium on Combinatorial Pattern Matching, CPM 2022*, volume 223 of *LIPIcs*, pages 18:1–18:15, 2022. doi:10.4230/LIPIcs.CPM.2022.18.
- 20 Richard Cole and Ramesh Hariharan. Approximate string matching: A simpler faster algorithm. *SIAM Journal on Computing*, 31(6):1761–1782, 2002. doi:10.1137/S0097539700370527.
- 21 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łacki, and Piotr Sankowski. Optimal dynamic strings. In *29th ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1509–1528. SIAM, 2018. doi:10.1137/1.9781611975031.99.
- 22 Paweł Gawrychowski and Przemysław Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 62:1–62:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.62.
- 23 Roberto Grossi, Costas S. Iliopoulos, Robert Mercas, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, and Fatima Vayani. Circular sequence comparison: algorithms and applications. *Algorithms for Molecular Biology*, 11:12, 2016. doi:10.1186/s13015-016-0076-6.
- 24 Yijie Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *Journal of Algorithms*, 50(1):96–105, 2004. doi:10.1016/j.jalgor.2003.09.001.
- 25 Ramesh Hariharan and V. Vinay. String matching in $\tilde{O}(\sqrt{n} + \sqrt{m})$ quantum time. *Journal of Discrete Algorithms*, 1(1):103–110, 2003. doi:10.1016/S1570-8667(03)00010-8.
- 26 Tommi Hirvola and Jorma Tarhio. Approximate online matching of circular strings. In *Experimental Algorithms - 13th International Symposium, SEA 2014*, pages 315–325. Springer, 2014. doi:10.1007/978-3-319-07959-2_27.
- 27 Ce Jin and Jakob Nogler. Quantum speed-ups for string synchronizing sets, longest common substring, and k -mismatch matching. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 5090–5121. SIAM, 2023. doi:10.1137/1.9781611977554.ch186.
- 28 Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. In *STOC 2022: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1657–1670. ACM, 2022. doi:10.1145/3519935.3520061.
- 29 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. doi:10.1137/0206024.
- 30 Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. Small-space and streaming pattern matching with k edits. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 885–896. IEEE, 2021. doi:10.1109/FOCS52979.2021.00090.
- 31 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551. SIAM, 2015. Full version: arXiv:1311.6235. doi:10.1137/1.9781611973730.36.

- 32 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.
- 33 Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989. doi:10.1016/0196-6774(89)90010-2.
- 34 Maurice Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35(2):73–78, 1990. doi:10.1016/0020-0190(90)90109-B.
- 35 Vicente Palazón-González and Andrés Marzal. Speeding up the cyclic edit distance using LAESA with early abandon. *Pattern Recognition Letters*, 62:1–7, 2015. doi:10.1016/j.patrec.2015.04.013.
- 36 Vicente Palazón-González, Andrés Marzal, and Juan Miguel Vilar. On hidden Markov models and cyclic strings for shape recognition. *Pattern Recognition*, 47(7):2490–2504, 2014. doi:10.1016/j.patcog.2014.01.018.
- 37 Süleyman Cenk Sahinalp and Uzi Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm (extended abstract). In *37th Annual Symposium on Foundations of Computer Science, FOCS 1996*, pages 320–328. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548491.
- 38 Peter H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373, 1980. doi:10.1016/0196-6774(80)90016-4.
- 39 Teresa Anna Steiner. Differentially private approximate pattern matching. In *15th Innovations in Theoretical Computer Science Conference, ITCS 2024*, volume 287 of *LIPICs*, pages 94:1–94:18, 2024. doi:10.4230/LIPICs.ITCS.2024.94.

A Origin of Theorem 10

An algorithm that efficiently computes a representation of $\text{Occ}_k(P, T)$ is encapsulated in [16, Main Theorem 9]³. The first step of this algorithm is the analysis of the pattern specified in [16, Lemma 6.4], which results in computing either a set of *breaks*, a set of *repetitive regions*, or a primitive string Q that is of length at most $m/(128k)$ and satisfies $\delta_E(P, *Q^*) < 2k$. In the presence of breaks or repetitive regions, we have $|\text{Occ}_k(P, T)|/k \leq 642045 \cdot (n/m) \cdot k$, see [16, Lemmas 5.21 and 5.24]. In the case where the analysis of the pattern returns an approximate period Q , we can use [16, Lemma 6.5] to find a rotation Q_1 of Q such that $\delta_E(P, *Q_1^*) = \delta_E(P, Q_1^*)$. Set $Q := Q_1$. Now, let us also compute all k -edit occurrences of the reversal of P in the reversal of T . Then, we can trim T , obtaining a string \bar{T} so that all k -edit occurrences of P in T are preserved in \bar{T} , and P has a k -edit occurrence both as a prefix and as a suffix of \bar{T} . We can then directly apply [16, Theorem 5.2] with $d = 8k$ to obtain the stated properties of \bar{T} ; for the fact that $\delta_E(\bar{T}, Q^*) \leq 24k$ holds see the fourth paragraph of the proof of that theorem. The length of Q can be instead bounded by $m/(256k)$ with all other constants remaining unchanged; this is because the bottleneck for the number of occurrences in the case where P is not almost periodic stems from repetitive regions and is not sensitive to the exact length of Q . That is, it is only the number of occurrences in the case where the analysis of the pattern yields $2k$ breaks that can be larger (by a multiplicative factor of 2), but the bound stated above is dominant.

³ When referring to statements of [16], we use their numbering in the full (arxiv) version of the paper.

B Reporting Version

Algorithm 3 is a reporting version of Algorithm 2. Algorithm 3 outputs all \hat{q} -non-overlap app-matches as a collection of $\mathcal{O}(k^4)$ interval chains (some of which can be single intervals).

■ **Algorithm 3** Non-overlap case: reporting version.

```

foreach interval of offsets  $D \in \text{NonOv}(\hat{q})$  do
   $i_1 := \min \text{CritPos}(D); i_2 := \max \text{CritPos}(D);$ 
   $Z_1 := \text{Anchored}_k(P, U, i_1);$ 
   $Z_2 := \text{Anchored}_k(P, U, i_2);$ 
  report  $Z_1 \cup Z_2;$ 
  foreach interval  $I = [p_l \dots p_r]$  in  $Z_1$ , with  $p_l > 0$  and  $p_r + m + k \leq |U|$  do
    if  $(\{p_l - 1\} \cup I) \cap \text{locked}(U) = \emptyset$  then
      report  $\text{Chain}(I, (i_2 - i_1)/q, q);$ 

```

C k -Edit CPM in Other Settings

Theorem 5 is stated in the PILLAR model. In the standard setting, all PILLAR operations can be implemented in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ preprocessing [14, Section 3]; this yields Theorem 1.

We now present our results for the internal, dynamic, fully compressed, and quantum settings. In each case, in the reporting version of the problem, the output is represented as a union of $\mathcal{O}((|T|/|P|) \cdot k^6)$ interval chains.

With the same implementations of operations in the internal setting as in the standard setting, we obtain an efficient implementation.

► **Theorem 34 (Internal Setting).** *Given two substrings P and T of a length- n string S , reporting and decision versions of k -Edit CPM for P and T can be solved in $\mathcal{O}((|T|/|P|)k^6)$ time and $\mathcal{O}((|T|/|P|)k^5 \log^3 k)$ time, respectively, after $\mathcal{O}(n)$ preprocessing on S .*

Let \mathcal{X} be a growing collection of non-empty persistent strings; it is initially empty, and then undergoes updates by means of the following operations:

- **Makestring**(U): Insert a non-empty string U to \mathcal{X}
- **Concat**(U, V): Insert string UV to \mathcal{X} , for $U, V \in \mathcal{X}$
- **Split**(U, i): Insert $U[0..i]$ and $U[i..|U|]$ to \mathcal{X} , for $U \in \mathcal{X}$ and $i \in [0..|U|]$.

By N we denote an upper bound on the total length of all strings in \mathcal{X} throughout all updates executed by an algorithm. A collection \mathcal{X} of non-empty persistent strings of total length N can be dynamically maintained with operations **Makestring**(U), **Concat**(U, V), **Split**(U, i) requiring time $\mathcal{O}(\log N + |U|)$, $\mathcal{O}(\log N)$ and $\mathcal{O}(\log N)$, respectively, so that PILLAR operations can be performed in time $\mathcal{O}(\log^2 N)$. All stated time complexities hold with probability $1 - 1/N^{\Omega(1)}$; see [21, 16]. Moreover, Kempa and Kociumaka [28, Section 8 in the arXiv version] presented an alternative deterministic implementation, which supports operations **Makestring**(U), **Concat**(U, V), **Split**(U, i) in $\mathcal{O}(|U| \log^{\mathcal{O}(1)} \log N)$, $\mathcal{O}(\log |UV| \log^{\mathcal{O}(1)} \log N)$, and $\mathcal{O}(\log |U| \log^{\mathcal{O}(1)} \log N)$ time, respectively, so that PILLAR operations can be performed in time $\mathcal{O}(\log N \log^{\mathcal{O}(1)} \log N)$. With these implementations, we obtain the following result.

► **Theorem 35** (Dynamic Setting). *A collection \mathcal{X} of non-empty persistent strings of total length N can be dynamically maintained with operations $\text{Makestring}(U)$, $\text{Concat}(U, V)$, $\text{Split}(U, i)$ requiring time $\mathcal{O}(\log N + |U|)$, $\mathcal{O}(\log N)$ and $\mathcal{O}(\log N)$, respectively, so that, given two strings $P, T \in \mathcal{X}$ and an integer threshold $k > 0$, we can solve k -Edit CPM in $\mathcal{O}((|T|/|P|) \cdot k^6 \log^2 N)$ time for the reporting variant and $\mathcal{O}((|T|/|P|) \cdot k^5 \log^3 k \log^2 N)$ time for the decision variant. All stated time complexities hold with probability $1 - 1/N^{\Omega(1)}$. Randomization can be avoided at the cost of a $\log^{\mathcal{O}(1)} \log N$ multiplicative factor in all the update times, with k -Edit CPM queries answered in $\mathcal{O}((|T|/|P|) \cdot k^6 \log N \log^{\mathcal{O}(1)} \log N)$ time (reporting version) or $\mathcal{O}((|T|/|P|) \cdot k^5 \log^3 k \log N \log^{\mathcal{O}(1)} \log N)$ time (decision version).*

A straight line program (SLP) is a context-free grammar G that consists of a set Σ of terminals and a set $N_G = \{A_1, \dots, A_n\}$ of non-terminals such that each $A_i \in N_G$ is associated with a unique production rule $A_i \rightarrow f_G(A_i) \in (\Sigma \cup \{A_j : j < i\})^*$. We can assume without loss of generality that each production rule is of the form $A \rightarrow BC$ for some symbols B and C (that is, the given SLP is in Chomsky normal form). Every symbol $A \in S_G := N_G \cup \Sigma$ generates a unique string, which we denote by $\text{gen}(A) \in \Sigma^*$. The string $\text{gen}(A)$ can be obtained from A by repeatedly replacing each non-terminal with its production. We say that G generates $\text{gen}(G) := \text{gen}(A_n)$.

In the fully compressed setting, given a collection of straight-line programs (SLPs) of total size n generating strings of total length N , each PILLAR operation can be performed in $\mathcal{O}(\log^2 N \log \log N)$ time after an $\mathcal{O}(n \log N)$ -time preprocessing [14, Section 3]. If we applied Theorem 1 directly in the fully compressed setting, we would obtain $\Omega(N/M)$ time, where N and M are the uncompressed lengths of the text and the pattern, respectively. Instead, we can adapt an analogous procedure provided in [16, Section 7.2] for (non-circular) pattern matching with edits to obtain the following result.

► **Theorem 36** (Fully Compressed Setting). *Let G_T denote a straight-line program of size n generating a string T , let G_P denote a straight-line program of size m generating a string P , let $k > 0$ denote an integer threshold, and set $N := |T|$ and $M := |P|$. We can solve k -Edit CPM in $\mathcal{O}(m \log N + nk^6 \log^2 N \log \log N)$ time (counting version) or $\mathcal{O}(m \log N + nk^5 \log^3 k \log^2 N \log \log N)$ time (decision version). A representation of the occurrences in the form of interval chains can be returned in $\mathcal{O}((N/M) \cdot k^6)$ extra time.*

We say an algorithm on an input of size n succeeds *with high probability* if the success probability can be made at least $1 - 1/n^c$ for any desired constant $c > 1$.

In what follows, we assume the input strings can be accessed in a quantum query model [1, 10]. We are interested in the time complexity of our quantum algorithms [6].

► **Observation 37** ([27, Observation 2.3]). *For any two strings S, T of length at most n , $\text{LCP}(S, T)$ or $\text{LCP}_R(S, T)$ can be computed in $\tilde{\mathcal{O}}(\sqrt{n})$ time in the quantum model with high probability.*

Hariharan and Vinay [25] gave a near-optimal quantum algorithm for the decision version of exact PM. We formalize this next.

► **Theorem 38** ([25]). *The decision version of PM can be solved in $\tilde{\mathcal{O}}(\sqrt{n})$ time in the quantum model with high probability. If the answer is YES, then the algorithm returns a witness occurrence.*

By employing Theorem 38 and binary search to find the period of S [31] and thus its full list of occurrences expressed as an arithmetic progression in T , we obtain the following.


24:22 Approximate Circular Pattern Matching Under Edit Distance

► **Observation 39.** *For any two strings S, T of length at most n , with $|T| \leq 2|S|$, $\text{IPM}(S, T)$ can be computed in $\tilde{\mathcal{O}}(\sqrt{n})$ time in the quantum model with high probability.*

All other PILLAR operations are performed trivially in $\mathcal{O}(1)$ quantum time. Thus while all PILLAR operations can be implemented in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ -time preprocessing in the standard setting by a classic algorithm, in the quantum setting, all PILLAR operations can be implemented in $\tilde{\mathcal{O}}(\sqrt{m})$ quantum time *with no preprocessing*, as we always deal with strings of length $\mathcal{O}(m)$. We obtain the following results.

► **Theorem 40 (Quantum Setting).** *The reporting version of the k -Edit CPM problem can be solved in $\tilde{\mathcal{O}}((n/\sqrt{m})k^6)$ time in the quantum model with high probability. The decision version of the k -Edit CPM problem can be solved in $\tilde{\mathcal{O}}((n/\sqrt{m})k^5)$ time in the quantum model with high probability.*

Depth-3 Circuit Lower Bounds for k -OV

Tameem Choudhury 

Department of Computer Science and Engineering, IIT Hyderabad, India

Karteek Sreenivasaiah 

Department of Computer Science and Engineering, IIT Hyderabad, India

Abstract

The 2-Orthogonal Vectors (2-OV) problem is the following: given two tuples A and B of n Boolean vectors, each of dimension d , decide if there exist vectors $u \in A$, and $v \in B$, such that u and v are orthogonal. This problem, and its generalization k -OV defined analogously for k tuples, are central problems in the area of fine-grained complexity. One of the major conjectures in fine-grained complexity is that k -OV cannot be solved by a randomised algorithm in $n^{k-\epsilon} \text{poly}(d)$ time for any constant $\epsilon > 0$.

In this paper, we are interested in unconditional lower bounds against k -OV, but for weaker models of computation than the general Turing Machine. In particular, we are interested in circuit lower bounds to computing k -OV by Boolean circuit families of depth 3 of the form OR-AND-OR, or equivalently, a *disjunction of CNFs*.

We show that for all $k \leq d$, any disjunction of t -CNFs computing k -OV requires size $\Omega((n/t)^k)$. In particular, when k is a constant, any disjunction of k -CNFs computing k -OV needs to use $\Omega(n^k)$ CNFs. This matches the brute-force construction, and for each fixed $k > 2$, this is the first unconditional $\Omega(n^k)$ lower bound against k -OV for a computation model that can compute it in size $O(n^k)$. Our results partially resolve a conjecture by Kane and Williams [17] (page 12, conjecture 10) about depth-3 AC^0 circuits computing 2-OV.

As a secondary result, we show an exponential lower bound on the size of $\text{AND} \circ \text{OR} \circ \text{AND}$ circuits computing 2-OV when d is very large. Since 2-OV reduces to k -OV by projections trivially, this lower bound works against k -OV as well.

2012 ACM Subject Classification Theory of computation \rightarrow Circuit complexity; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases fine grained complexity, k -OV, circuit lower bounds, depth-3 circuits

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.25

Related Version *Previous Version:* <https://ecc.weizmann.ac.il/report/2023/014/#revision3>

Acknowledgements The authors would like to thank the anonymous referees for their valuable comments that helped improve the presentation of this paper in several respects.

1 Introduction

The area of fine-grained complexity is a branch of computational complexity that studies the complexity of functions with a finer lens than the usual approach that makes a coarse distinction between polynomial time and super-polynomial time. The area has been focused on functions in P that find uses in a variety of contexts. In the seminal paper by Vassilevska Williams and Williams [26], they show eight problems that are subcubic time equivalent to one another. Hence a truly subcubic time algorithm for any one of these problems will also imply a subcubic algorithm for the others.

The holy grail of computation complexity is to show *unconditional* lower bounds to resources used in computing an *explicit* function. Unfortunately, the state of affairs in terms of unconditional lower bounds for computation, in its full generality, is rather bleak. The best known unconditional lower bounds for the running time of computing an explicit function are



© Tameem Choudhury and Karteek Sreenivasaiah;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;
Article No. 25; pp. 25:1–25:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



merely linear. Even for functions such as SAT that do not have any polynomial time running algorithms till date, we do not know how to show super-linear lower bounds. We do know from the time hierarchy theorem¹ that there are languages in $\text{DTIME}(n^2)$ that are not in $\text{DTIME}(n^c)$ for any $c < 2$. However the languages constructed in a proof of the time hierarchy are not natural, and not as explicit as we would like. Results such as [26] and [7] that show equivalences among several important functions help in identifying candidate functions that might witness the time hierarchy theorem for their time class. One such candidate function for quadratic time² is the *2-Orthogonal Vectors problem*.

The 2-Orthogonal Vectors problem $2\text{-OV}_{n,d}$ is defined as follows: Given as input two tuples $A \subseteq \{0,1\}^d$ and $B \subseteq \{0,1\}^d$ of n vectors each, decide if there is a vector $a \in A$ and a vector $b \in B$ such that a and b are orthogonal. To define a generalization of this problem, we think of each vector from $\{0,1\}^d$ as a characteristic vector of a subset from $[d]$. Then a natural generalization of $2\text{-OV}_{n,d}$ is the problem $k\text{-OV}_{n,d}$ that takes as input k tuples $A_1, A_2, \dots, A_k \subseteq \{0,1\}^d$ of n vectors each, and the task is to decide if there exists vectors $a_1 \in A_1, a_2 \in A_2, \dots, a_k \in A_k$ such that $a_1 \cap a_2 \cap \dots \cap a_k = \emptyset$. The problems 2-OV and $k\text{-OV}$ have emerged as central problems in fine-grained complexity. An important hypothesis is that no deterministic, or randomized, algorithm computing $2\text{-OV}_{n,d}$ can run in time $O(n^{2-\epsilon} \text{poly}(d))$ for any $\epsilon > 0$. This is essentially saying that the brute force algorithm is also the best. Interestingly, Ryan Williams in [24], shows that under the *strong exponential time hypothesis* (SETH)³, 2-OV (3-OV) requires $n^{2-o(1)}$ time ($n^{3-o(1)}$ time respectively).

In the absence of techniques that can show unconditional lower bounds, two natural directions of research emerge: (i) Conditional lower bounds to help us understand connections between various such problems, and “bottlenecks” to better algorithms. (ii) Unconditional lower bounds for weaker models of computation.

The first line of research has seen a tremendous body of results. There are numerous fine-grained reductions, and lower bounds, conditioned on SETH, and the hardness of functions such as $2\text{-OV}_{n,d}$, and $k\text{-OV}_{n,d}$. In the 2018 survey [25], Vassilevska Williams aptly describes it as “*an explosion of hardness results based on OV*”, and lists nineteen problems whose complexity is connected to that of $k\text{-OV}$. The fact that better algorithms for so many problems would imply better algorithms for $k\text{-OV}$, is perhaps not surprising. Intuitively, the $k\text{-OV}$ function looks “canonical” in a certain sense, and has managed to hide itself inside several other problems that look quite different at the surface. These include seemingly unrelated problems such as Longest Common Subsequence [1], Edit Distance [2], Fréchet distance [4, 5], Regular Expressions Matching [3], to name a few. Their survey [25] is an excellent source for those looking for a thorough treatment of fine-grained complexity, and in particular, this line of research.

The second direction, of showing lower bounds against weaker models of computation, seems to be lacking the same attention. To the best of our knowledge, the only paper that addresses this line is that of Kane and Williams [17]. In their paper they show tight lower bounds for formulas and branching programs computing 2-OV . We do not know any non-trivial lower bounds for computing 2-OV by models stronger than branching programs.

Note that if a uniform circuit family of bounded fan-in, and size $O(s(n, d))$ computes $k\text{-OV}_{n,d}$, then an algorithm that simply evaluates the circuit, computes $k\text{-OV}_{n,d}$ in time $\tilde{O}(s(n, d))$. So if the $k\text{-OV}$ hypothesis is true, then we can expect any uniform circuit family computing $k\text{-OV}_{n,d}$ to have size $\Omega(n^k)$. This begs the question:

¹ Such hierarchy theorems go through for the unit cost RAM model as well.

² We are being imprecise here so as to remain informal. The input length of $2\text{-OV}_{n,d}$ is actually nd . So “quadratic in n ” is not the same as $\text{DTIME}(n^2)$

³ [15],[6] For every $\epsilon > 0$, $\exists k$ such that $k\text{-SAT}$ problem on n variables cannot be solved in $O(2^{(1-\epsilon)n})$ time

What is the largest class of circuits for which we can show $\Omega(n^k \text{poly}(d))$ size lower bounds against computing $k\text{-OV}_{n,d}$?

One class of Boolean circuits that has been extensively studied in terms of lower bounds is AC^0 (gates from $\{\wedge, \vee, \neg\}$, unbounded fan-in, $O(1)$ -depth). In fact we know exponential lower bounds against this class of circuits. So a good target would be to show that $k\text{-OV}_{n,d}$ requires AC^0 circuits of size $\Omega(n^k \text{poly}(d))$. We note that $k\text{-OV}_{n,d}$ can indeed be computed by depth-3 AC^0 circuits of size $n^k d$, as shown later in equation 2. Can we show matching lower bounds?

The best known lower bound against depth-3 AC^0 circuits is $2^{\Omega(\sqrt{n})}$ for computing majority. This bound can be obtained by several classic techniques from the 80s including the switching lemma by Håstad [13], the polynomial method by Razborov [21] and Smolensky [22], and finite-limit vectors by [14]. One of the most important problems in circuit complexity is to prove $2^{\omega(n/\log \log n)}$ lower bounds to the size of depth-3 AC^0 circuits computing an explicit function. This would imply superlinear lower bounds against $O(\log n)$ depth circuits (of bounded fan-in) due to the depth reduction procedure described by Valiant [23] (alternatively, see Chapter 11 of Jukna [16]). With the aim of making progress on this front, Goldreich and Wigderson proposed a new framework in [11] where they define a new model of arithmetic circuits that use *multilinear gates*, as opposed to allowing gates computing sum or product alone, and a new complexity measure on this model. The main motivation being that lower bounds to their complexity measure implies lower bounds to a specific class of Boolean depth-3 circuits that they call *D-canonical*. The best lower bounds obtained for this class of depth-3 Boolean circuits, using their framework, is $\Omega(2^{n^{3/5}})$ by Goldreich and Tal [10]. In fact, the brute force depth-3 AC^0 circuits computing the negation of $k\text{-OV}$, described later in equation 3, bears close resemblance to D-canonical circuits since it is a product of set-multilinear functions, but over the Boolean algebra, as opposed to $\text{GF}(2)$.

More recently, the status of depth-3 $\text{AC}^0[\oplus]$ circuits (gates computing xor are allowed in addition to the usual \wedge, \vee, \neg) got an update. The lower bound for computing majority using depth-3 $\text{AC}^0[\oplus]$ circuits was improved from $2^{\Omega(n^{1/4})}$ to $2^{\Omega(\sqrt{n})}$ by Oliveira, Santhanam and Srinivasan [20]. This closed the gap between upper and lower bounds up to a logarithmic factor in the exponent.

While techniques such as the switching lemma and the polynomial method work in a “bottom-up” fashion, the techniques in [14] is a “top-down” approach specifically for depth-3 AC^0 circuits. To the best of our knowledge, the only top-down strategies for circuit lower bounds are the *Karchmer-Wigderson game* by Karchmer and Wigderson [18], the *discriminator lemma* for depth-2 threshold circuits by Hajnal, Masse, Pudlák, Szegedy, Turán [12], and *finite-limits* by Håstad, Jukna, Pudlak [14]. Our results in this paper can be seen as a non-trivial application of the techniques of Håstad, Jukna, Pudlak [14].

Kane and Williams [17] conjecture that any depth-3 AC^0 circuit computing $2\text{-OV}_{n,d}$ requires $\Omega(n^2)$ wires (see page 12, conjecture 10 in [17]). Observe that $2\text{-OV}_{n,d}$ (and $k\text{-OV}_{n,d}$) can be computed by $\text{OR} \circ \text{AND} \circ \text{OR}$ circuits with $2n^2 d$ wires (and $kn^k d$ wires respectively):

$$2\text{-OV}_{n,d}(A, B) = \bigvee_{i_1, i_2 \in [n]} \bigwedge_{j \in [d]} (\neg a_{i_1}[j] \vee \neg b_{i_2}[j]) \quad (1)$$

$$k\text{-OV}_{n,d}(A_1, \dots, A_k) = \bigvee_{i_1, \dots, i_k \in [n]} \bigwedge_{j \in [d]} (\neg a_{i_1}[j] \vee \dots \vee \neg a_{i_k}[j]) \quad (2)$$

Hence, informally, their conjecture for $2\text{-OV}_{n,d}$, and by extension $k\text{-OV}_{n,d}$, is that the brute-force circuit is also the best.

A second important question in [17] is about generalizing lower bounds from 2-OV to k -OV. As they have noted, generalizing their lower bounds to $k > 2$ would beat the state of the art in branching program lower bounds. Our results for depth-3 AC^0 circuits generalize to $k > 2$, and scale well when the bottom fan-in is bounded.

Our Results

In this paper, we show lower bounds against the size of certain classes of depth-3 AC^0 circuit families computing k -OV $_{n,d}$. Our main result shows lower bounds against a restricted class of $OR \circ AND \circ OR$ circuits computing k -OV $_{n,d}$, while our secondary result deals with $AND \circ OR \circ AND$ circuits computing a special case of k -OV $_{n,d}$. Our main result is the following:

► **Theorem 1.** *For all $k \leq d$, any $OR \circ AND \circ OR$ circuit with bottom fan-in t computing k -OV $_{n,d}$ requires top fan-in $\Omega\left(\left(\frac{n}{t}\right)^k\right)$.*

Circuit families of the type $OR \circ AND \circ OR$ can also be understood as a *disjunction of CNFs*. Therefore Theorem 1 is equivalent to the following statement:

“Any disjunction of t -CNFs computing k -OV $_{n,d}$ requires size $\Omega\left((n/t)^k\right)$.”

(Here, by ‘ t -CNF’, we mean a CNF whose clauses have at most t literals, and by ‘size’ we mean the number of CNFs being used.)

The brute-force circuit described earlier in equation 2 for k -OV $_{n,d}$, is a disjunction of n^k many k -CNFs, and the lower bound from Theorem 1 for this model is $\Omega((n/k)^k)$. Hence for all constant $k > 1$, the complexity of computing k -OV $_{n,d}$ as a disjunction of k -CNFs is $\Theta(n^k)$.

The proof technique used for Theorem 1 actually goes through for a more general class of depth-3 circuits where the bottom gates can have arbitrary fan-in as long as the number of negated literals among their inputs is at most t . We describe this in the next subsection. The more general theorem is the following. Let \mathcal{C}_t^- be the set of all unate functions (see Definition 7) that are negative unate on at most t variables.

► **Theorem 2.** *For all $k \leq d$, any $OR \circ AND \circ \mathcal{C}_t^-$ circuit computing k -OV $_{n,d}$ requires top fan-in $\Omega\left(\left(\frac{n}{t}\right)^k\right)$.*

It is important to note that the usual trick of using random restrictions to get rid of the bottom fan-in restriction in Theorem 1 is very unlikely to work as it is known that 2-OV becomes easy to compute by AC^0 circuits with high probability under random restrictions [17] (section 3).

As a secondary result, we show an exponential lower bound on the size of $AND \circ OR \circ AND$ circuits computing 2-OV $_{n,d}$ when d is very large:

► **Theorem 3.** *For all $\ell \leq d$, any $AND \circ OR \circ AND$ circuit computing 2-OV $_{n,d}$ requires size $s \in \Omega(\min\{2^\ell, \left(\frac{d}{n\ell}\right)^n\})$. In particular, for $\ell = d/2n$ and $d \in \Omega(n^2)$, $s \in \Omega(2^n)$.*

Since 2-OV $_{n,d}$ reduces to k -OV $_{n,d}$ by projections trivially, the above theorem holds for k -OV $_{n,d}$ as well. It must also be noted that the input size for 2-OV $_{n,d}$ is $2nd$ which, for the choice of d in Theorem 3, is $2n^3$. Hence with respect to an input size of n , the lower bound for 2-OV $_{n,d}$ from Theorem 3 is actually $2^{\Omega(n^{1/3})}$.

An important fact to be noted about depth-3 AC^0 circuits is that, in general, the computational power of $OR \circ AND \circ OR$ circuits and $AND \circ OR \circ AND$ circuits are incomparable. As demonstrated by [14], the *iterated intersection* function on $2n$ variables (see Definition 26)

is computable by $\text{AND} \circ \text{OR} \circ \text{AND}$ circuits of linear size, but any $\text{OR} \circ \text{AND} \circ \text{OR}$ circuit family computing it requires size $2^{\Omega(\sqrt{n})}$. A more thorough discussion on this topic can be found in Chapter 11.5 of Jukna [16]. This is true in the context of $\text{k-OV}_{n,d}$ and our bounds also: an idea used by Kane and Williams (Proposition 4 in [17]) can be used to show $\text{AND} \circ \text{OR} \circ \text{AND}$ circuits computing $\text{k-OV}_{n,d}$ that are smaller than our lower bounds in Theorem 1 for $\text{OR} \circ \text{AND} \circ \text{OR}$ circuits when $d \in O(1)$.

We show in Section 5 a general construction for $\text{k-OV}_{n,d}$ that achieves a trade-off between top fan-in and bottom fan-in. This shows that in general, for circuits with bottom fan-in t our lower bound against the top fan-in of $\text{OR} \circ \text{AND} \circ \text{OR}$ circuits computing $\text{k-OV}_{n,d}$ is at least a factor of t^{k-1}/k away from the corresponding upper bound.

Techniques

We note that throughout this paper, we work with the function $\text{k-Int}_{n,d}$ defined as the negation of $\text{k-OV}_{n,d}$. We do this because $\text{k-Int}_{n,d}$ is a monotone function, and hence allows us several conveniences with regard to notation. Thus our lower bounds to $\text{AND} \circ \text{OR} \circ \text{AND}$ circuits computing $\text{k-Int}_{n,d}$ transfer directly to $\text{OR} \circ \text{AND} \circ \text{OR}$ circuits computing $\text{k-OV}_{n,d}$. More formally, $\text{k-Int}_{n,d}$ is defined as

$$\text{k-Int}_{n,d}(A_1, \dots, A_k) = \bigwedge_{i_1, \dots, i_k \in [n]} \bigvee_{j \in [d]} (a_{i_1}[j] \wedge \dots \wedge a_{i_k}[j]) \quad (3)$$

Main result. For our main result, the strategy we use is that of *finite limit vectors*. This is a top-down strategy that was used by Håstad, Jukna, and Pudlák in [14] for proving depth-3 AC^0 circuit lower bounds. We briefly describe the approach.

Assume an $\text{AND} \circ \text{OR} \circ \text{AND}$ circuit $C = C_1 \wedge \dots \wedge C_{s(n)}$ computes a function f . Then for any $\mathcal{N} \subseteq f^{-1}(0)$, by an averaging argument, there is a C_i that correctly outputs 0 on at least $1/s$ fraction of inputs in \mathcal{N} . Hence showing an upper bound to $|C_i^{-1}(0) \cap \mathcal{N}|$ implies a lower bound to $s(n)$ as $s \geq |\mathcal{N}|/|C_i^{-1}(0) \cap \mathcal{N}|$.

The technique of *finite limits* by [14] is used to show that C_i cannot be correct on many inputs in \mathcal{N} . The idea is to show that if $C_i^{-1}(0) \cap \mathcal{N}$ is large, then we can construct a 1-input y such that for any set of t input positions, it looks identical to *some string* in $C_i^{-1}(0) \cap \mathcal{N}$. Such a string y is called a *t-limit* for the set $C_i^{-1}(0) \cap \mathcal{N}$. Then if the bottom gates in C_i can each see only t bits of the input, the string y fools all of them into evaluating to 0 *simultaneously*, and hence C_i will output 0 on y . This is a contradiction since $y \in C^{-1}(1)$ by construction, but $C_i(y) = 0$ implies $C(y) = 0$. It is not hard to see that if the t -limit string y has the additional property that $y \geq x$ for all $x \in C_i^{-1}(0) \cap \mathcal{N}$, and each bottom gate in C_i has at most t positive literals among its inputs, the same argument goes through. We call such a y an *upper t-limit* for the set $C_i^{-1}(0) \cap \mathcal{N}$ (as opposed to the term ‘*lower t-limit*’ used in [14] for the case when $y \leq x$). We shall also use the term “bottom positive fan-in” to indicate how many of the input literals are allowed to be positive for each bottom gate.

The key idea behind our construction of a t -limit is to first model any subset of *maxterms* of $\text{k-Int}_{n,d}$ as a k -partite hypergraph such that the maxterms in the subset and the hyperedges are in bijection. We call this hypergraph as a “Support Graph”, and construct it in Section 3.2. Then we construct a t -limit for the case of $2\text{-Int}_{n,d}$ by using König’s theorem on this graph. To deal with the general case of $\text{k-Int}_{n,d}$, we first show a sunflower lemma on this support graph, and then use the sunflower structure to construct a t -limit. We show a version of the sunflower lemma on our hypergraph that is *very slightly* less demanding than the standard sunflower lemma [9]. We note that this does not improve the asymptotic complexity of our final bound.

We remark here that all t -limit strings that we construct in this paper are also upper t -limit strings. Hence all our lower bounds for k -Int $_{n,d}$ go through for the circuit class $\text{AND} \circ \text{OR} \circ \mathcal{C}_t^+$ where \mathcal{C}_t^+ is the set of all unate functions that are positive unate on at most t variables. Informally, this means that the bottom gates can compute any unate functions, have unbounded fan-in, but at most t of the inputs can be positive literals. (The dual statement for k -OV $_{n,d}$ is Theorem 2 stated in the previous section.) As an example, lower bounds using this technique will also work against depth-3 circuits where the top and middle layers are AND and OR respectively, and the bottom layer consists of homogeneous linear threshold functions, each of which is defined by a vector of weights that has at most t positive weights.

An important observation about the technique described above is that it is impervious to the fan-in of the middle OR gates. So we could use a suitable DNF for each bottom gate and convert an $\text{AND} \circ \text{OR} \circ \mathcal{C}_t^+$ circuit to an $\text{AND} \circ \text{OR} \circ \text{AND}$ circuit with bottom positive fan-in at most t and a possibly larger middle fan-in. Since the technique gives lower bounds to top fan-in regardless of middle fan-in, all lower bounds that we can derive against $\text{AND} \circ \text{OR} \circ \text{AND}$ circuits with bottom positive fan-in t using this technique, transfer to $\text{AND} \circ \text{OR} \circ \mathcal{C}_t^+$ without any change. Hence throughout this paper, we focus our attention to $\text{AND} \circ \text{OR} \circ \text{AND}$ circuits.

Secondary result. The exponential lower bound of [14] for $\text{OR} \circ \text{AND} \circ \text{OR}$ circuits computing the iterated intersection function $S_{n,d}$ for $d \in \sqrt{n}$ is of particular interest to us. The function $S_{n,d}$ bears a close resemblance to 2 -Int $_{n,d}$. While $S_{n,d}$ is the *iterated* intersection, 2 -Int $_{n,d}$ can be seen as “all-pairs” intersection.

We show a reduction (via projections) from $S_{n,d/n}$ to 2 -Int $_{n,d}$. The blow-up in the dimension of vectors is rather large, and we can conclude non-trivial lower bounds only for $d \in \omega(n)$.

2 Preliminaries

We often interpret a d -dimensional vector $u \in \{0,1\}^d$ as the characteristic vector of a subset of $[d]$.

► **Definition 4** (k -OV $_{n,d}$). For tuples $A_1, A_2, \dots, A_k \subseteq \{0,1\}^d$ where $\forall i \in [k], |A_i| = n$.

$$k\text{-OV}_{n,d}(A_1, A_2, \dots, A_k) = 1 \iff \exists a_1 \in A_1, \exists a_2 \in A_2, \dots, \exists a_k \in A_k, \text{ such that} \\ a_1 \cap a_2 \cap \dots \cap a_k = \emptyset$$

For notational convenience, we work with the negation of k -OV $_{n,d}$ throughout the paper. We use k -Int $_{n,d}$ to denote the negation of k -OV $_{n,d}$, and is defined as follows:

► **Definition 5** (k -Int $_{n,d}$). For tuples $A_1, A_2, \dots, A_k \subseteq \{0,1\}^d$ where $\forall i \in [k], |A_i| = n$.

$$k\text{-Int}_{n,d}(A_1, A_2, \dots, A_k) = 1 \iff \forall a_1 \in A_1, \forall a_2 \in A_2, \dots, \forall a_k \in A_k, \text{ we have} \\ a_1 \cap a_2 \cap \dots \cap a_k \neq \emptyset$$

An input to the function k -Int $_{n,d}$ has nk vectors, each of dimension d . Hence $nk d$ many input bits in total.

For any $x, y \in \{0,1\}^d$, we write $x \leq y$ if $\forall i, x_i \leq y_i$. Similarly, we write $x \oplus y$ to denote the string obtained by a point-wise xor between x and y .

► **Definition 6** (Monotone function). We say that a Boolean function f is monotone if $\forall x, y \in \{0, 1\}^d$ such that $x \leq y$, we have $f(x) \leq f(y)$.

The notion of monotone can be generalized to the notion of being *unate*:

► **Definition 7** (Unate function). A Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is unate if there exists a monotone Boolean function $g : \{0, 1\}^n \rightarrow \{0, 1\}$ and a string $s \in \{0, 1\}^n$ such that for all inputs x , we have $f(x) = g(x \oplus s)$.

Further, a unate function is *positive unate* (negative unate) on a variable x_i if $s_i = 0$ ($s_i = 1$ respectively).

For monotone functions such as $\text{k-Int}_{n,d}$, we can define *maximal 0-inputs*:

► **Definition 8** (Maximal 0-input). Let f be a monotone Boolean function. An input x is a maximal 0-input for f if $f(x) = 0$ and for all strings y such that $x < y$, $f(y) = 1$.

Throughout this article, we will use the term “*maxterm*” and “maximal 0-inputs” interchangeably. This deviates from the standard definition of *maxterm*, but is very convenient in our context.

For a vector $u \in \{0, 1\}^d$, and a set of indices $S \subseteq [d]$, we denote the restriction of u to the indices in S as $u|_S$.

► **Definition 9** (t -limit). A vector $y \in \{0, 1\}^m$ is said to be a t -limit for a set $B \subseteq \{0, 1\}^m$ if and only if $\forall S \subseteq [m]$ with $|S| = t$, $\exists x \in B$ such that $y \neq x$ but $y|_S = x|_S$. Further, $y \in \{0, 1\}^m$ is said to be an upper t -limit if $y \geq x$.

Note that if a string y is a t -limit for a set B and $B \subseteq B'$, then y is also a t -limit for B' .

We will be using König’s theorem in our proofs, which is stated as follows:

► **Proposition 10** ([19], [8]). The maximum cardinality of a matching in a bipartite graph \mathcal{G} is equal to the minimum cardinality of a vertex cover of its edges.

We will always assume that the depth-3 circuits we consider are layered. i.e., inputs are read directly by only the gates at the bottom layer, and every layer reads outputs from the layer below it. This assumption does not affect asymptotic complexity. We say a depth-3 circuit C has *bottom positive fan-in* (bottom negation fan-in) t if for every gate in the bottom layer, at most t of its inputs are positive literals (negated literals respectively).

We denote the permutation group on k distinct elements with S_k . Let $\mathcal{P} = (P_1, \dots, P_k)$ be an ordered partition of $[d]$ into k parts. For any permutation $\sigma \in S_k$, we use \mathcal{P}_σ to denote the ordered partition obtained by permuting the parts of \mathcal{P} using σ . i.e., $\mathcal{P}_\sigma \triangleq (P_{\sigma(1)}, \dots, P_{\sigma(k)})$

3 AND ◦ OR ◦ AND circuits

To describe the lower bound for $\text{k-Int}_{n,d}$ against AND ◦ OR ◦ AND circuits, we first identify a special set of maxterms (maximal 0-inputs) of $\text{k-Int}_{n,d}$. We do this by explicitly constructing such inputs.

3.1 Maxterms of $\text{k-Int}_{n,d}$

Fix any choice of integers $k, d \in \mathbb{N}$ such that $1 < k \leq d$. For any choice of $n_1, \dots, n_k \in [n]$, and any ordered partition $\mathcal{P} = (P_1, \dots, P_k)$ of $[d]$ into k parts, we will construct an input $N = (A_1, \dots, A_k)$ where $A_i \subseteq \{0, 1\}^d$ with $|A_i| = n$ such that N is a maxterm for $\text{k-Int}_{n,d}$. Throughout, we will denote the j ’th vector in A_i by a_i^j .

The input $N = (A_1, \dots, A_k) \in \{0, 1\}^{nkd}$ is constructed as follows:

- Set every vector other than $a_1^{n_1}, \dots, a_k^{n_k}$ to all 1s.
- In each $a_i^{n_i}$, set the indices contained in P_i to 0s. Set every other position to 1. Formally, for all $i \in [k]$, set $a_i^{n_i}|_{P_i} \leftarrow \vec{0}$ and $a_i^{n_i}|_{[d] \setminus P_i} \leftarrow \vec{1}$.

We shall call $((n_1, \dots, n_k), \mathcal{P})$ the *support* of N , and denote it by $\text{sup}(N)$.

To see that N is indeed a maxterm of $\mathbf{k}\text{-Int}_{n,d}$, observe that since \mathcal{P} is a partition of $[d]$, for every position $\ell \in [d]$, there is a *unique* $i \in [k]$ such that $\ell \in P_i$. Therefore, by construction of N , $a_i^{n_i}[\ell] = 0$. So for every position ℓ , there is some vector among $a_1^{n_1}, \dots, a_k^{n_k}$ that is 0 in position ℓ , and hence $a_1^{n_1} \cap \dots \cap a_k^{n_k} = \emptyset$. Moreover, due to i being unique for each such ℓ , we also have $a_j^{n_j}[\ell] = 1$ for all $j \neq i$. So changing $a_i^{n_i}[\ell]$ from 0 to 1 results in the vectors intersecting at ℓ . Combining this with the fact that every vector in N other than $a_1^{n_1}, \dots, a_k^{n_k}$ is the all-1s vector, we conclude that N is indeed a maximal 0-input.

We will be particularly interested in a subset of such maxterms of $\mathbf{k}\text{-Int}_{n,d}$ that are formed by the permutations of the parts of some fixed partition into non-empty parts. We define this formally as follows.

► **Definition 11** (Permutation-maxterms). *Fix an ordered partition $\mathcal{P} = (P_1, \dots, P_k)$ of $[d]$ into k non-empty parts. A permutation-maxterm with respect to \mathcal{P} is any maxterm N constructed as above that has $\text{sup}(N) = ((n_1, \dots, n_k), \mathcal{P}_\sigma)$ for some $n_1, \dots, n_k \in [n]$ and $\sigma \in \mathbf{S}_k$.*

We shall use $\mathcal{N}_{\mathcal{P}}^{n,k,d}$ to denote the set of all permutation-maxterms of $\mathbf{k}\text{-Int}_{n,d}$ with respect to some ordered partition \mathcal{P} of $[d]$ into k non-empty parts. We drop the subscript, and superscripts if it is clear from context.

Note that for any partition \mathcal{P} as in the definition above, $|\mathcal{N}_{\mathcal{P}}^{n,k,d}| = n^k k!$ as there are n^k many k -tuples (n_1, \dots, n_k) and $k!$ many permutations in \mathbf{S}_k .

► **Remark 12.** The proofs in this paper do not depend on the exact partition chosen. Any arbitrary ordered partition of $[d]$ into k non-empty parts will work. For a further simplification, one could assume $k = d$, and fix the permutation $\mathcal{P} = (P_1, \dots, P_k)$ to be $P_i = \{i\}$ for all $i \in [d]$.

3.2 Support Graph

We define a k -partite hypergraph to encode, and reason about, the relationship between permutation-maxterms of $\mathbf{k}\text{-Int}_{n,d}$. Here, by k -partite hypergraph we mean that every hyperedge must contain exactly one vertex from each part.

Fix $k \geq 2$ and $d \geq k$, and any ordered partition \mathcal{P} of $[d]$ into k non-empty parts. For any subset $S \subseteq \mathcal{N}_{\mathcal{P}}^{n,k,d}$ of permutation-maxterms of $\mathbf{k}\text{-Int}_{n,d}$, we define the *support graph* of S as a k -partite hypergraph $\mathcal{G}_S = (V_1 \cup \dots \cup V_k, E)$ such that each maxterm in S corresponds to a hyperedge in the graph. Recall that an input to $\mathbf{k}\text{-Int}_{n,d}$ consists of k tuples, each having n vectors of dimension d . In each V_i , we include a total of nk vertices as follows: for each $j \in [n]$, we include k vertices in V_i indexed as $v_i^{j,1}, \dots, v_i^{j,k}$. So for all $i \in [k]$, we have $|V_i| = nk$ and hence the graph \mathcal{G}_S is defined on nk^2 many vertices.

We define the set E of hyperedges as follows:

$$\left(v_1^{n_1, b_1}, \dots, v_k^{n_k, b_k} \right) \in E \iff \exists \text{ maxterm } N \in S \text{ such that} \\ \text{sup}(N) = ((n_1, \dots, n_k), \mathcal{P}_\sigma) \text{ and } b_i = \sigma(i) \forall i \in [k]$$

► **Remark 13.** Note that the set of maxterms $S \subseteq \mathcal{N}_{\mathcal{P}}$ and the set of hyperedges in \mathcal{G}_S are in bijection. More precisely, a maxterm N with $\text{sup}(N) = ((n_1, \dots, n_k), \mathcal{P}_\sigma)$ corresponds to the hyperedge $\left(v_1^{n_1, \sigma(1)}, \dots, v_k^{n_k, \sigma(k)} \right)$ and vice-versa.

► **Definition 14** (Co-disjoint). We call two vectors $u \in \{0, 1\}^d$ and $v \in \{0, 1\}^d$ as co-disjoint if and only if $\bar{u} \cap \bar{v} = \emptyset$. i.e., the set of positions where u is 0, and the set where v is 0 are disjoint.

For two tuples of vectors $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$ where $a_i, b_i \in \{0, 1\}^d$, we say A and B are co-disjoint if for all $i \in [n]$, a_i and b_i are co-disjoint.

Maxterms $M = (M_1, \dots, M_k)$ and $N = (N_1, \dots, N_k)$, both from $\mathcal{N}_{\mathcal{P}}^{n,k,d}$, are said to be co-disjoint if and only if for all $i \in [k]$, M_i and N_i are co-disjoint.

Intuitively, the graph \mathcal{G}_S records where the 0s in each of the maxterms in S appear. This gives us the following close connection between co-disjointness of vectors across maxterms, and disjointness of their hyperedges.

► **Lemma 15.** Let $S \subseteq \mathcal{N}_{\mathcal{P}}^{n,k,d}$, and let $\mathcal{G}_S = (V_1 \cup \dots \cup V_k, E)$ be its support graph. Let $M = (M_1, \dots, M_k)$ and $N = (N_1, \dots, N_k)$ be two maxterms from S and let E_M , and E_N respectively, denote their corresponding hyperedges in \mathcal{G}_S . Then for each $i \in [k]$, we have the following two properties:

1. If E_M and E_N share a vertex in V_i , then $M_i = N_i$.
2. If E_M and E_N contain different vertices from V_i , then M_i and N_i are co-disjoint.

Proof. Let $\text{sup}(M) = (a_1, \dots, a_k, \mathcal{P}_\sigma)$ and $\text{sup}(N) = (b_1, \dots, b_k, \mathcal{P}_\pi)$.

Proof of (1). If E_M and E_N share a vertex in V_i for some $i \in [k]$, then $v_i^{a_i, \sigma(i)} = v_i^{b_i, \pi(i)}$ and so we have $a_i = b_i$ and $\sigma(i) = \pi(i)$. Let $\ell = a_i = b_i$, and let $q = \sigma(i) = \pi(i)$. Then by construction of the maxterms M and N , all vectors in M_i other than m_i^ℓ are all 1s, and similarly all vectors in N_i other than n_i^ℓ are all 1s. The vector m_i^ℓ and n_i^ℓ both have 0s in indices from the part P_q , and 1s elsewhere. So $m_i^\ell = n_i^\ell$. Hence the tuple M_i and N_i are identical.

Proof of (2). If E_M and E_N have different vertices from V_i , then $v_i^{a_i, \sigma(i)} \neq v_i^{b_i, \pi(i)}$. So either $a_i \neq b_i$ or $\sigma(i) \neq \pi(i)$ (or both). The claim holds in both cases:

- If $a_i \neq b_i$, then recall that by construction, the only vector that has 0s in M_i is the vector $m_i^{a_i}$. Every other vector in M_i , and in particular $m_i^{b_i}$ is the all 1s vector by construction. So the tuples of vectors M_i and N_i cannot both be 0 in any vector in any position.
- Else $a_i = b_i$ and $\sigma(i) \neq \pi(i)$. By our construction of maxterms, the 0s in the vectors $m_i^{a_i}$ and $n_i^{b_i}$ are in the indices given by $P_{\sigma(i)}$ and $P_{\pi(i)}$ respectively. Since \mathcal{P} is a partition, and $\sigma(i) \neq \pi(i)$, $P_{\sigma(i)} \cap P_{\pi(i)} = \emptyset$. Therefore there cannot be an index where both $m_i^{a_i}$ and $n_i^{b_i}$ are both 0. ◀

The following lemma follows directly from Lemma 15:

► **Lemma 16.** Let $S \subseteq \mathcal{N}_{\mathcal{P}}^{n,k,d}$ be a set of maxterms such that all hyperedges in \mathcal{G}_S are pairwise vertex-disjoint. Then the maxterms in S are pairwise co-disjoint. (i.e., for all positions $\ell \in [nk]$, there is at most one maxterm in S that has 0 in the ℓ 'th position.)

Proof. Let $M, N \in S$ be any two maxterms, and let the vertex set of \mathcal{G}_S be $V = V_1 \cup \dots \cup V_k$. The hyperedges E_M and E_N , corresponding to M , and N respectively, are vertex-disjoint from the premise. So for each $i \in [k]$, E_M and E_N contain different vertices from V_i . Applying Lemma 15 to \mathcal{G}_S , we obtain that M_i and N_i are co-disjoint for all $i \in [k]$. Hence there is no position where both M and N are 0 by definition of co-disjoint. ◀

3.3 Warm-up: 2-Int $_{n,d}$

We give a self-contained proof of our lower bound for the case of 2-Int $_{n,d}$ that demonstrates the strategy behind the proof for the general case.

► **Theorem 17.** *For all $d > 1$, any AND \circ OR \circ AND circuit with bottom fan-in t computing 2-Int $_{n,d}$ requires top fan-in at least $2n^2/t^2$.*

Proof. Let $C = C_1 \wedge C_2 \wedge \cdots \wedge C_s$ be an AND \circ OR \circ AND circuit with bottom fan-in t computing 2-Int $_{n,d}$. Let $\mathcal{P} = (P_1, P_2)$ be any ordered partition of $[d]$ into two non-empty parts. Consider the permutation-maxterms $\mathcal{N} = \mathcal{N}_{\mathcal{P}}^{n,2,d}$ of 2-Int $_{n,d}$ as described in definition 11. Since \mathcal{N} is a subset of the 0-inputs of 2-Int $_{n,d}$, the circuit C outputs 0 on every input in \mathcal{N} . By an averaging argument, there exists $i \in [s]$ such that C_i correctly outputs 0 on at least $1/s$ fraction of inputs in \mathcal{N} . We will show that $|C_i^{-1}(0) \cap \mathcal{N}| \leq t^2$. Then the theorem follows as:

$$\frac{2n^2}{s} = \frac{1}{s} |\mathcal{N}| \leq |C_i^{-1}(0) \cap \mathcal{N}| \leq t^2.$$

Let $S = C_i^{-1}(0) \cap \mathcal{N}$. Suppose, for the sake of contradiction, $|S| > t^2$. We will show later in the proof that in such a case, there is a t -limit $y \in C^{-1}(1)$ for S . This leads to a contradiction as follows: let $C_i = g_1 \vee g_2 \vee \cdots \vee g_\ell$ with each g_j having fan-in at most t . Then, by definition of t -limit, for all $T \subseteq [nkd]$ with $|T| = t$, there exists $x \in S$ such that $x|_T = y|_T$. Now each of the gates g_j is a function of at most t variables, and we know that for all inputs $x \in S$, we have $g_j(x) = 0$ for all $j \in [\ell]$. Since y looks identical to some string in S when restricted to these t positions, all the g_j will output 0 on y too. This forces $C_i(y) = 0$, but this cannot happen since $y \in C^{-1}(1)$. Hence it could not have been the case that $|S| > t^2$.

We now construct a t -limit for any $S \subseteq \mathcal{N}$ when $|S| > t^2$. Let $S \subseteq \mathcal{N}$ be any set with size $|S| > t^2$ and let \mathcal{G}_S be its support graph. Note that since $k = 2$, \mathcal{G}_S is a bipartite graph with simple edges rather than hyperedges, and every maxterm in S corresponds to an edge in \mathcal{G}_S and vice versa. We claim at least one of the following is true for \mathcal{G}_S :

- (i) There exists a matching of size $t + 1$ in \mathcal{G}_S .
- (ii) There exists a vertex of degree at least $t + 1$ in \mathcal{G}_S .

Indeed this is a consequence of Kőnig's Theorem (stated in Proposition 10): suppose the size of a maximum matching is at most t , then the minimum vertex-cover has size at most t . Since there are $|S|$ many edges in \mathcal{G}_S , there must be a vertex v in the vertex cover with degree at least $\frac{|S|}{t}$. Since $|S| > t^2$, it must be that $\deg(v) > t$ which satisfies (ii). In both the above cases, we construct a string $y \in C^{-1}(1)$ that is a t -limit for S .

- Case (i): Consider the set S' of maxterms corresponding to the edges in a maximum matching of \mathcal{G}_S . Then S' is a set of at least $t + 1$ pairwise co-disjoint maxterms. Then $y \triangleq \vec{1}$ is a t -limit for S' . To see why, consider any set of t positions. By Lemma 16, at each of these positions, at most one of the maxterms in S' can be 0. Since there are $t + 1$ such maxterms and only t positions, there must be a maxterm where the value at all the given positions is 1, thus looking identical to y .
- Case (ii): Let the vertex set of \mathcal{G}_S be $V = V_1 \cup V_2$. Without loss of generality, let the vertex v with $\deg(v) > t$ be in V_1 . Let E be the edges that have v as one endpoint, and let $M_E \subseteq S$ be the maxterms corresponding to the edges in E . Then by property (1) of Lemma 15, the first tuple of vectors in all these maxterms is the same. Let A_1 be the first tuple of vectors. We construct the input $y = (Y_1, Y_2)$ as follows: set $Y_1 \leftarrow A_1$, and set $Y_2 \leftarrow \vec{1}$.

Since the string y was obtained by taking first tuple of a *maxterm*, and setting every vector in the 2nd tuple to 1, it must be a 1-input.

To see that y is a t -limit, take any subset of indices $T \subseteq [2nd]$ with $|T| = t$. We will show that one of the maxterms in M_E looks identical to y in these t positions. For every position from $[nd]$ (the 1st tuple of vectors), every maxterm in M_E is identical to y since $Y_1 = A_1$. So assume that all indices in T are from the range $\{nd + 1, \dots, 2nd\}$. By construction, y is all-1s in this range of indices. Since edges in E have distinct endpoints in V_2 , property (2) of Lemma 15 tells us that the second tuple of vectors in the maxterms in T are pairwise co-disjoint. This is similar to case (i): we have $|M_E| \geq t + 1$ many maxterms such that for any position in T , at most one of them is 0, and there are only t positions in T . So by the pigeon-hole principle, there must be a maxterm in M_E that has 1 in all positions from T , thus looking identical to y in these positions. ◀

Since $2\text{-OV}_{n,d}$ is the negation of $2\text{-Int}_{n,d}$, the following is an immediate corollary of Theorem 17.

► **Corollary 18.** *For all $d > 1$, any $\text{OR} \circ \text{AND} \circ \text{OR}$ circuit with bottom fan-in t computing $2\text{-OV}_{n,d}$ requires top fan-in at least $2n^2/t^2$.*

► **Remark 19.** It is easy to see that the t -limit string y constructed in the proof of Theorem 17 is in fact an *upper* t -limit. Therefore the lower bound shown for $2\text{-Int}_{n,d}$ works against a slightly more general class of circuits – $\text{AND} \circ \text{OR} \circ \text{AND}$ circuits that have each bottom AND-gate seeing at most t positive literals. Analogously the lower bound for $2\text{-OV}_{n,d}$ works against $\text{OR} \circ \text{AND} \circ \text{OR}$ circuits where each bottom gate has at most t negated inputs.

3.4 General case: $k\text{-Int}_{n,d}$

We will need the following lemma on k -partite hypergraphs:

► **Lemma 20.** *Let G be a k -partite hypergraph with m many hyperedges. Then for all $t > 0$ at least one of the following holds:*

- (i) *There are more than t vertex-disjoint hyperedges in G .*
- (ii) *There is a vertex u such that $\deg(u) > \lfloor \frac{m}{kt} \rfloor$.*

Proof. Let G be a k -partite hypergraph with m hyperedges. Let S be a largest set of vertex-disjoint hyperedges in G . If $|S| > t$, then the lemma is true. Suppose $|S| \leq t$. Let V_S be the set of vertices participating in the hyperedges in S . Since each hyperedge contains exactly k many vertices, $|V_S| \leq kt$. Also, since S is a largest such set, each of the remaining hyperedges must contain at least one vertex from V_S . Therefore, by an averaging argument, there is a vertex $u \in V_S$ that is part of at least $\frac{m - |S|}{|V_S|}$ many hyperedges outside S , and 1 hyperedge in S . Therefore, we have:

$$\deg(u) \geq \frac{m - |S|}{|V_S|} + 1 \geq \frac{m - t}{kt} + 1 = \frac{m}{kt} - \frac{1}{k} + 1 > \left\lfloor \frac{m}{kt} \right\rfloor \quad \blacktriangleleft$$

We use Lemma 20 to show that if we start with enough hyperedges, then there is a subset of them such that in each part, either all of them coincide, or they are all distinct.

► **Lemma 21.** *Let $k \geq 2$, and let $G = (V_1 \cup \dots \cup V_k, E)$ be a k -partite hypergraph with $|E| > \frac{k!t^k}{2}$. Then there exists $S \subseteq E$ with $|S| > t$ such that for each $i \in [k]$, exactly one of the following holds:*

1. *There exists a vertex $u \in V_i$ such that all hyperedges in S share the vertex u .*
2. *No two hyperedges in S share the same vertex in V_i .*

Proof. Induction on k . Base case $k = 2$ is a consequence of König's theorem (Proposition 10): Since $k = 2$, G is just a bipartite graph. If there is a matching in G of size more than t , then let S be the edges in such a matching. Clearly the edges in S are vertex-disjoint and statement (2) holds. Else the maximum matching has size $\leq t$. Then Proposition 10 implies that the minimum vertex cover has size at most t . By an averaging argument, there must exist a vertex u such that $\deg(u) > |E|/t > \frac{k!t^k}{2t} = \frac{2t^2}{2t} = t$. Define S to be the set of edges that share u . Without loss of generality, let $u \in V_1$. Then all edges in S must have distinct vertices in V_2 . Therefore in V_1 , they all coincide, and in V_2 they are all distinct.

Case $k > 2$. Apply Lemma 20 to G . If (i) holds, then we have a set S of more than t vertex-disjoint hyperedges. This means for all $i \in [k]$, statement (2) holds and we are done.

Suppose (ii) holds, then there is a vertex u such that $\deg(u) > \lfloor m/kt \rfloor = \frac{(k-1)!t^{k-1}}{2}$. Let S be the set of all hyperedges that contain vertex u . Then $|S| = \deg(u)$. Let $z \in [k]$ be such that $u \in V_z$.

We construct a $(k-1)$ -partite hypergraph $G' = (V', E')$ by removing V_z , and the z 'th coordinate from each edge. More formally:

$$V' \triangleq V_1 \cup \dots \cup V_{z-1} \cup V_{z+1}, \dots \cup V_k$$

$$E' \triangleq \{(v_1, \dots, v_{z-1}, v_{z+1}, \dots, v_k) \mid (v_1, \dots, v_{z-1}, u, v_{z+1}, v_k) \in S\}$$

(Informally, an edge $e' \in E'$ is just an edge $e \in S$ with its z 'th coordinate removed.)

We define $V'_i = V_i$ for all $i \neq z$ as the $k-1$ parts of V' . Note that $|E'| = |S|$. This is because $\forall e_1, e_2 \in S$ such that $e_1 \neq e_2$, the edges e_1 and e_2 share the vertex u in V_z . So there must exist $j \neq z$ such that e_1 and e_2 use different vertices in V_j . Hence $e'_1 \neq e'_2$. Further, observe that for any $i \neq z$, $e'_1, e'_2 \in E'$ share a vertex in V'_i if and only if e_1 and e_2 share the same vertex in V_i .

Now G' is a $(k-1)$ -partite hypergraph with $|E'| = |S| > \frac{(k-1)!t^{k-1}}{2}$ many hyperedges. By induction on G' , there must exist a set $S' \subseteq E'$ with $|S'| > t$ such that for each $i \neq z$, either all hyperedges in S' share a vertex in V'_i , or they use distinct vertices in V'_i . We already know that since $S' \subseteq S$, all edges in S' share the same vertex in V_z , namely u . Hence for all $i \in [k]$, the edges in S' satisfy (1) or (2). ◀

► **Remark 22.** The statement of Lemma 21 can be seen as a sunflower lemma. Take any vertex u in the graph G that participates in at least one hyperedge from S . Then exactly one of the following holds: (i) The vertex u participates in exactly one hyperedge in S , or (ii) The vertex u participates in all hyperedges in S . The standard sunflower lemma would require more than $k!t^k$ hyperedges, while our statement needs half of that.

We now describe how to construct an upper t -limit in the general case.

► **Lemma 23.** *Let $\mathcal{M} \subseteq \mathcal{N}_{\mathcal{P}}^{n,k,d}$ be any set of permutation-maxterms of k -Int $_{n,d}$ for any $k \geq 2$ and $d \geq k$. If $|\mathcal{M}| > \frac{k!t^k}{2}$, then there is a string $y \in k$ -Int $_{n,d}^{-1}(1)$ that is an upper t -limit for \mathcal{M} .*

Proof. Let $G_{\mathcal{M}} = (V, E)$ be the k -partite support graph of \mathcal{M} (defined in section 3.2), and let $V = V_1 \cup \dots \cup V_k$. By Lemma 21, there exists a set of hyperedges $S \subseteq E$ with $|S| \geq t+1$ such that for each $i \in [k]$, either all edges in S share the same vertex in V_i , or no two edges share a vertex of V_i . Let M_S be the set of maxterms corresponding to S .

Let $B \subseteq [k]$ be the set of all indices $i \in [k]$ such that all edges in S share the same vertex in V_i . Then \overline{B} contains indices of parts where the edges in S use distinct vertices. (Observe that \overline{B} is non-empty because otherwise all maxterms would share all vertices, and hence would be one and the same. But we know that $|S| \geq t+1 > 1$, so this cannot happen.) By

property (1) of Lemma 15, this implies that for each $i \in B$, the i 'th tuple of vectors in the maxterms in M_S are identical. For each $i \in B$, denote the i 'th tuple of vectors in all these maxterms as A_i .

We construct the string $y = (Y_1, \dots, Y_k)$ as follows:

$$\forall i \in B, \text{ set } Y_i \leftarrow A_i$$

$$\forall j \in \overline{B}, \text{ set } Y_j \leftarrow \vec{1}$$

y is a 1-input of $\mathbf{k}\text{-Int}_{n,d}$

Observe that y can also be obtained by starting with any maxterm $N = (N_1, \dots, N_k)$ from S , and setting to 1s all vectors in N_j for all $j \in \overline{B}$. Since N is a maxterm (maximal 0-input), the string y must be a 1-input. This also means that the string y is point-wise greater than or equal to any maxterm in S .

y is a t -limit

Let $T \subseteq [nkd]$ with $|T| = t$ be a set of any t positions. For all $i \in B$, the string y is identical to *every* maxterm in M_S . So assume that T only has positions that fall into tuples indexed by \overline{B} . By property (2) of Lemma 15, the maxterms in M_S are pairwise co-disjoint on all such positions. i.e., for any position $\ell \in T$, at most one maxterm in M_S can be 0. So we have t positions, and $|M_S| = |S| \geq t + 1$ maxterms. By pigeon-hole principle, there exists a maxterm in M_S that is 1 on all these t positions, thus looking identical to y .

Since y is point-wise greater or equal to every maxterm in S , we conclude that indeed y is an upper t -limit for \mathcal{M} . \blacktriangleleft

► **Lemma 24.** *Let C be any OR \circ AND circuit with bottom positive fan-in t computing a function f on n variables. Let y be any string that is an upper t -limit for $f^{-1}(0)$. Then $C(y) = 0$.*

Proof. Let g be any bottom AND-gate of C . Let $P \subseteq [n]$ ($Q \subseteq [n]$) be the variables whose positive literals (negated literals resp.) are input to g . Then $|P| \leq t$ by assumption.

Since y is an upper t -limit for $g^{-1}(0)$, it must be that for every set T of t positions there exists a string $x^{(T)} \in g^{-1}(0)$ such that $y|_T = x^{(T)}|_T$. In particular, this holds for the set P . So in all positions from P , the gate g sees no difference between y and $x^{(T)}$.

The gate g sees negative literals of all variables from Q . Since y is an *upper* t -limit, we have $x^{(T)}|_Q \leq y|_Q$. Hence for all $i \in Q$ such that $\neg x_i = 0$, we also have $\neg y_i = 0$. Hence $g(y) \leq g(x^{(T)}) = 0$ as $x^{(T)} \in g^{-1}(0)$. \blacktriangleleft

► **Theorem 25.** *For all k, d such that $k \leq d$, any AND \circ OR \circ AND circuit with bottom positive fan-in t computing $\mathbf{k}\text{-Int}_{n,d}$ requires top fan-in $\Omega\left(\left(\frac{n}{t}\right)^k\right)$.*

Proof. Let $C = C_1 \wedge \dots \wedge C_s$ be an AND \circ OR \circ AND circuit with bottom positive fan-in t , computing $\mathbf{k}\text{-Int}_{n,d}$. Consider the set $\mathcal{N} = \mathcal{N}_{\mathcal{P}}^{n,k,d}$ of all permutation-maxterms of $\mathbf{k}\text{-Int}_{n,d}$ with respect to any ordered permutation \mathcal{P} of $[d]$ into k non-empty parts (see Definition 11, and Remark 12). Since C outputs 0 on all inputs from \mathcal{N} , there must be some OR \circ AND $_t$ subcircuit C_i that correctly outputs 0 on at least $1/s$ fraction of inputs in \mathcal{N} . We will show that $|C_i^{-1}(0) \cap \mathcal{N}| \leq k! t^k / 2$, and the theorem follows since:

$$\frac{k! n^k}{s} = \frac{1}{s} |\mathcal{N}| \leq |C_i^{-1}(0) \cap \mathcal{N}| \leq \frac{k! t^k}{2}$$

25:14 Depth-3 Circuit Lower Bounds for k -OV

Let $\mathcal{M} = C_i^{-1}(0) \cap \mathcal{N}$. Suppose, for the sake of contradiction, $|\mathcal{M}| > k!t^k/2$. Since $\mathcal{M} \subseteq \mathcal{N}$, we apply Lemma 23 to conclude that there exists a string $y \in \text{k-Int}_{n,d}^{-1}(1)$ that is an upper t -limit y for \mathcal{M} . Then by Lemma 24, it must be that $C(y) = 0$. But this is a contradiction since $y \in \text{k-Int}_{n,d}^{-1}(1)$. ◀

Since $\text{k-OV}_{n,d}$ is the negation of $\text{k-Int}_{n,d}$, the following is an immediate corollary of Theorem 25.

► **Theorem 1.** *For all $k \leq d$, any $\text{OR} \circ \text{AND} \circ \text{OR}$ circuit with bottom fan-in t computing $\text{k-OV}_{n,d}$ requires top fan-in $\Omega\left(\left(\frac{n}{t}\right)^k\right)$.*

4 OR \circ AND \circ OR circuits

In this section, we show that any $\text{OR} \circ \text{AND} \circ \text{OR}$ circuit requires exponential size to compute $2\text{-Int}_{n,d}$ for any $d \in \Omega(n^2)$. This result is a consequence of a known lower bound for the iterated intersection function defined as follows:

► **Definition 26** (Iterated Intersection). *Let $A = (a_1, a_2, \dots, a_n)$ and $B = (b_1, b_2, \dots, b_n)$ be tuples of vectors from $\{0, 1\}^d$,*

$$S_{n,d}(A, B) = 1 \iff \forall i \in [n] \text{ we have } a_i \cap b_i \neq \emptyset$$

Observe that $S_{n,d}(A, B)$ differs from $2\text{-Int}_{n,d}(A, B)$ in that the intersection between two vectors a_i and b_j when $i \neq j$ does not affect the value of $S_{n,d}$ at all. Recall the definition of $2\text{-Int}_{n,d}(A, B)$:

$$2\text{-Int}_{n,d}(A, B) = 1 \iff \forall i, j \in [n] \text{ we have } a_i \cap b_j \neq \emptyset$$

The function $S_{n,d}$ can also be defined using an $\text{AND} \circ \text{OR} \circ \text{AND}_2$ circuit of size nd :

$$S_{n,d}(A, B) = \bigwedge_{i=1}^n \bigvee_{j=1}^d a_i[j] \wedge b_i[j].$$

The result by Håstad, Jukna, Pudlák in [14] shows the following lower bound for computing $S_{n,d}$ by $\text{OR} \circ \text{AND} \circ \text{OR}$ circuits:

► **Proposition 27** ([14]). *For all $\ell \leq nd$, any $\text{OR} \circ \text{AND} \circ \text{OR}$ circuit computing $S_{n,d}$ requires size $\min\{2^\ell, (d/\ell)^n\}$.*

In particular, Proposition 27 shows that $S_{\sqrt{n}, \sqrt{n}}$ requires $2^{\Omega(\sqrt{n})}$ size $\text{OR} \circ \text{AND} \circ \text{OR}$ circuits. This can be used to show lower bounds for $2\text{-Int}_{n,d}$:

► **Theorem 28.** *Let C be an $\text{OR} \circ \text{AND} \circ \text{OR}$ circuit computing $2\text{-Int}_{n,d}$. Then for all $\ell \leq d$, size of C is at least $\min\{2^\ell, \left(\frac{d}{n\ell}\right)^n\}$.*

Proof. We show this by reducing $S_{n, \lfloor d/n \rfloor}$ to $2\text{-Int}_{n,d}$ via projections. Let $d' = \lfloor d/n \rfloor$. Take any instance $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$ with $a_i, b_i \in \{0, 1\}^{d'}$ of $S_{n,d'}$. We create two tuples of d -dimensional vectors $A' = (a'_1, \dots, a'_n)$ and $B' = (b'_1, \dots, b'_n)$ that serve as an instance of $2\text{-Int}_{n,d}$ as follows – for all $i \in [n]$, define $a'_i = 1^{(i-1)d'} a_i 1^{(n-i)d'}$ and $b'_i = 0^{(i-1)d'} b_i 0^{(n-i)d'}$. Note that the dimension of each a_i and b_i is $nd' \leq d$.

Observe that a_i and b_i are disjoint if and only if a'_i and b'_i are disjoint. So if (A, B) was a 0-instance of $S_{n,d'}$, then (A', B') is a 0-instance of $2\text{-Int}_{n,d}$.

Further, if $b_j \neq \vec{0}$ for some $j \in [n]$, then for all $i \neq j$, we have $a'_i \cap b'_j \neq \emptyset$. To see this, observe that if $b_j \neq \vec{0}$, then there is some position $p \in [(j-1)d' + 1, jd']$ such that $b'_j[p] = 1$. But by construction, the vector a'_i is 1 everywhere outside the interval $[(i-1)d' + 1, id']$. Since $i \neq j$, the vector a'_i must be 1 at position p .

If (A, B) was a 1-instance of $S_{n,d'}$, then all a_i intersect b_i . This means all b_i are non-zero vectors. Thus for all $i, j \in [n]$, $a'_i \cap b'_j \neq \emptyset$.

The above reduction shows that C can be used to compute $S_{n, \lfloor d/n \rfloor}$. Applying Proposition 27 to C tells us that C must have size at least $\min\{2^\ell, \left(\frac{d}{n\ell}\right)^n\}$ for all $\ell \leq d$. ◀

Our reduction in proof of Theorem 28 inflates the dimension of vectors by a factor of n making the obtained bound trivial when $d \in O(n)$. However, we can still conclude an exponential lower bound by substituting $\ell = d/2n$ that gives us a lower bound of $\min\{2^{d/2n}, 2^n\} \in 2^{\Omega(n)}$ when $d \in \Omega(n^2)$.

Since $2\text{-OV}_{n,d}$ is the negation of $2\text{-Int}_{n,d}$, the following is an immediate corollary.

▶ **Theorem 3.** *For all $\ell \leq d$, any $\text{AND} \circ \text{OR} \circ \text{AND}$ circuit computing $2\text{-OV}_{n,d}$ requires size $s \in \Omega(\min\{2^\ell, \left(\frac{d}{n\ell}\right)^n\})$. In particular, for $\ell = d/2n$ and $d \in \Omega(n^2)$, $s \in \Omega(2^n)$.*

5 A General Upper Bound

In this section, we describe a more general construction of a depth-3 circuit to compute $k\text{-Int}_{n,d}$ that allows a trade-off between the top fan-in and bottom fan-in. We recall the construction given by equation 3 here:

$$k\text{-Int}_{n,d}(A_1, \dots, A_k) = \bigwedge_{i_1, \dots, i_k \in [n]} \bigvee_{j \in [d]} (a_{i_1}[j] \wedge \dots \wedge a_{i_k}[j]) \quad (3)$$

We generalize this construction to obtain the following trade-off between top and bottom fan-in:

▶ **Proposition 29.** *For any integer $1 \leq t \leq n^k$, the function $k\text{-Int}_{n,d}$ can be computed by a monotone depth-3 $\text{AND} \circ \text{OR} \circ \text{AND}$ circuit with top fan-in $\lceil \frac{n^k}{t} \rceil$, middle fan-in d^t , and bottom fan-in at most kt .*

Proof. Let C be the circuit described in equation 3. Observe that each $\text{OR} \circ \text{AND}$ subcircuit of C is checking whether a particular choice $a_{i_1} \in A_1, a_{i_2} \in A_2, \dots, a_{i_k} \in A_k$ of vectors are intersecting or not. Since there are n^k many such choices, the top fan-in in C is n^k . Checking if a particular choice of k vectors intersects at some fixed coordinate uses an AND of fan-in k , and hence the bottom fan-in in C is k .

To obtain the trade-off in the theorem statement, the idea is to construct an $\text{AND} \circ \text{OR} \circ \text{AND}$ circuit where each $\text{OR} \circ \text{AND}$ subcircuit checks whether t many such choices of vectors intersect. Each choice can be written as a k -tuple of vectors $(a_{i_1}, \dots, a_{i_k})$. For convenience, let's assume that t divides n^k . Let $T = \{T_1, T_2, \dots, T_{n^k/t}\}$ be a partition of the set of n^k possible k -tuples of vectors into n^k/t parts with each T_l containing exactly t many k -tuples. For the vectors in any particular k -tuple in T_l to have non-empty intersection, there must exist a position $i \in [d]$ where all the k vectors in the k -tuple are 1. Hence to check if each of the k -tuples of vectors in T_l have non-zero intersection, it suffices to check if there exist t positions $i_1, i_2, \dots, i_t \in [d]$ such that the j 'th k -tuple of vectors intersect in i_j .

Let $A_l^j[i]$ be the AND of the bits in the i^{th} position of the vectors in the j^{th} tuple in T_l . This is an AND gate with fan-in k because there are k many vectors in each tuple. We construct the following circuit where the ℓ^{th} OR \circ AND subcircuit checks if each k -tuple of vectors in T_ℓ have non-zero intersection:

$$G_t = \bigwedge_{l \in \{1, \dots, \frac{n^k}{t}\}} \bigvee_{i_1, i_2, \dots, i_t \in [d]} (A_l^1[i_1] \wedge A_l^2[i_2] \wedge \dots \wedge A_l^t[i_t])$$



Observe that G_t has top fan-in as n^k/t , middle fan-in as d^t , and bottom fan-in kt as desired. \blacktriangleleft

References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.
- 2 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58. ACM, 2015. doi:10.1145/2746539.2746612.
- 3 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.56.
- 4 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.76.
- 5 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete fréchet distance. *J. Comput. Geom.*, 7(2):46–76, 2016. doi:10.20382/jocg.v7i2a4.
- 6 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the exact complexity of evaluating quantified k -cnf. In *Parameterized and Exact Computation - 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*, volume 6478 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2010. doi:10.1007/978-3-642-17493-3_7.
- 7 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 21–40. SIAM, 2019. doi:10.1137/1.9781611975482.2.
- 8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 9 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.
- 10 Oded Goldreich and Avishay Tal. On constant-depth canonical boolean circuits for computing multilinear functions. In *Computational Complexity and Property Testing - On the Interplay Between Randomness and Computation*, volume 12050 of *Lecture Notes in Computer Science*, pages 306–325. Springer, 2020. doi:10.1007/978-3-030-43662-9_17.
- 11 Oded Goldreich and Avi Wigderson. On the size of depth-three boolean circuits for computing multilinear functions. In *Computational Complexity and Property Testing - On the Interplay Between Randomness and Computation*, volume 12050 of *Lecture Notes in Computer Science*, pages 41–86. Springer, 2020. doi:10.1007/978-3-030-43662-9_6.

- 12 András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993. doi:10.1016/0022-0000(93)90001-D.
- 13 Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986. doi:10.1145/12130.12132.
- 14 Johan Håstad, Stasys Jukna, and Pavel Pudlák. Top-down lower bounds for depth-three circuits. *Computational Complexity*, 5(2):99–112, 1995. doi:10.1007/BF01268140.
- 15 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 16 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.
- 17 Daniel M. Kane and Richard Ryan Williams. The orthogonal vectors conjecture for branching programs and formulas. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 48:1–48:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.48.
- 18 Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM J. Discret. Math.*, 3(2):255–265, 1990. doi:10.1137/0403021.
- 19 Dénes König. Graphen und matrizen. *Mat. Fiz. Lapok*, 38(10), 1931.
- 20 Igor Carboni Oliveira, Rahul Santhanam, and Srikanth Srinivasan. Parity helps to compute majority. In *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA*, volume 137 of *LIPIcs*, pages 23:1–23:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CCC.2019.23.
- 21 Alexander A. Razborov. On the method of approximations. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 167–176. ACM, 1989. doi:10.1145/73007.73023.
- 22 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 77–82. ACM, 1987. doi:10.1145/28395.28404.
- 23 Leslie G. Valiant. Exponential lower bounds for restricted monotone circuits. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 110–117. ACM, 1983. doi:10.1145/800061.808739.
- 24 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.
- 25 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3447–3487. World Scientific, 2018.
- 26 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.67.

A Myhill-Nerode Theorem for Generalized Automata, with Applications to Pattern Matching and Compression

Nicola Cotumaccio  

Gran Sasso Science Institute, L'Aquila, Italy
Dalhousie University, Halifax, Canada

Abstract

The model of generalized automata, introduced by Eilenberg in 1974, allows representing a regular language more concisely than conventional automata by allowing edges to be labeled not only with characters, but also strings. Giammaresi and Montalbano introduced a notion of determinism for generalized automata [STACS 1995]. While generalized deterministic automata retain many properties of conventional deterministic automata, the uniqueness of a minimal generalized deterministic automaton is lost.

In the first part of the paper, we show that the lack of uniqueness can be explained by introducing a set $\mathcal{W}(\mathcal{A})$ associated with a generalized automaton \mathcal{A} . The set $\mathcal{W}(\mathcal{A})$ is always trivially equal to the set of all prefixes of the language recognized by the automaton, if \mathcal{A} is a conventional automaton, but this need not be true for generalized automata. By fixing $\mathcal{W}(\mathcal{A})$, we are able to derive for the first time a full Myhill-Nerode theorem for generalized automata, which contains the textbook Myhill-Nerode theorem for conventional automata as a degenerate case.

In the second part of the paper, we show that the set $\mathcal{W}(\mathcal{A})$ leads to applications for pattern matching and data compression. Wheeler automata [TCS 2017, SODA 2020] are a popular class of automata that can be compactly stored using $e \log \sigma(1 + o(1)) + O(e)$ bits (e being the number of edges, σ being the size of the alphabet) in such a way that pattern matching queries can be solved in $\tilde{O}(m)$ time (m being the length of the pattern). In the paper, we show how to extend these results to generalized automata. More precisely, a Wheeler generalized automata can be stored using $\epsilon \log \sigma(1 + o(1)) + O(e + rn)$ bits so that pattern matching queries can be solved in $\tilde{O}(rm)$ time, where ϵ is the total length of all edge labels, r is the maximum length of an edge label and n is the number of states.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Pattern matching; Theory of computation → Data compression

Keywords and phrases Generalized Automata, Myhill-Nerode Theorem, Regular Languages, Wheeler Graphs, FM-index, Burrows-Wheeler Transform

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.26

Related Version *Full Version:* <https://arxiv.org/abs/2302.06506>

Funding This work was partially funded by Dante Labs.

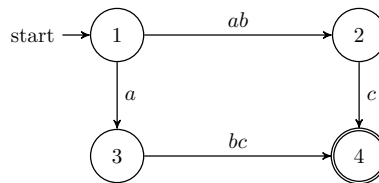
1 Introduction

The class of regular languages can be defined starting from *non-deterministic finite automata* (NFAs). In his monumental work [18] on automata theory (which dates back to 1974), Eilenberg proposed a natural generalization of NFAs where edges can be labeled not only with characters but with (possibly empty) finite strings, the so-called *generalized non-deterministic finite automata* (GNFAs). While classical automata are only a special case of generalized automata, it is immediate to realize that generalized automata can only recognize regular languages, because it is well-known that epsilon transitions do not add expressive



power [31], and a string-labeled edge can be decomposed into a path of edges labeled only with characters. However, generalized automata can represent regular languages more concisely than classical automata. A standard measure of the complexity of a regular language is the minimum number of states of some automaton recognizing the language, and generalized automata may have fewer states than conventional automata. In generalized automata, we assume that both the number of states and the number of edges are finite, but the number of edges cannot be bounded by some function of the number of states and the size of the finite alphabet (and so edge labels may be arbitrarily long). As a consequence, in principle it is not clear whether the problem of determining the minimum number states of some generalized automaton recognizing a given language is decidable. In [29], Hashiguchi showed that the problem is decidable by proving that there must exist a state-minimal generalized automaton for which the lengths of edge labels can be bounded by a function that only depends on the size of the syntactic monoid recognizing the language.

An NFA is a *deterministic finite automaton (DFA)* if no state has two distinct outgoing edges with the same label. This local notion of determinism extends to global determinism, that is, given a string α , there exists at most one path labeled α that can be followed starting from the initial state. However, this is not true for generalized automata (see Figure 1). When considering generalized automata, we must add the additional requirement that no state has two distinct outgoing edges such that one edge label is a prefix of the other edge label. By adding this requirement, we retrieve global determinism, thus obtaining *generalized deterministic finite automata (GDFAs)*.



■ **Figure 1** No state has two distinct outgoing edges with the same label, but there are two distinct paths labeled abc from the initial state.

For every regular language, there exists a *unique* deterministic automaton recognizing the language and having the minimum number of states among all deterministic automata recognizing the language, the *minimal DFA* of the language. More generally, a classical textbook result in automata theory is the *Myhill-Nerode theorem*. Let $\text{Pref}(\mathcal{L})$ be the set of all strings prefixing at least one string in the language \mathcal{L} . We have the following result.

► **Theorem 1** (Myhill-Nerode theorem). *Let $\mathcal{L} \subseteq \Sigma^*$. The following are equivalent:*

1. \mathcal{L} is recognized by an NFA.
2. The Myhill-Nerode equivalence $\equiv_{\mathcal{L}}$ has finite index.
3. There exists a right-invariant equivalence relation \sim on $\text{Pref}(\mathcal{L})$ of finite index such that \mathcal{L} is the union of some \sim -classes.
4. \mathcal{L} is recognized by a DFA.

Moreover, if one of the above statements is true (and so all the above statements are true), then there exists a unique minimal DFA recognizing \mathcal{L} (that is, two DFAs having the minimum number of states among all DFAs that recognize \mathcal{L} must be isomorphic).

The problem of studying the notion of determinism in the setting of generalized automata was approached by Giammaresi and Montalbano [28, 27]. The notion of isomorphism can be naturally extended to GDFAs (intuitively, two GDFAs are isomorphic if they are the same

G DFA up to renaming the states), and the natural question is whether one can analogously define the minimal G DFA of a regular language. This is not possible: in general, there can exist two or more non-isomorphic state-minimal G DFAs recognizing a given regular language. Consider the two distinct G DFAs in Figure 2. It is immediate to check that the two (non-isomorphic) G DFAs recognize the same language, and it can be shown that the no G DFA with less than three states can recognize the same language [28].



■ **Figure 2** Two state-minimal GDFAs recognizing the same regular language.

The non-uniqueness of a state-minimal GDFAs seems to imply a major difference in the behavior of generalized automata compared to conventional automata, so it looks like there is no hope to derive a structural result like the Myhill-Nerode theorem in the model of the generalized automata. It is natural to wonder whether the lack of uniqueness should be interpreted as a weakness of the model of generalized automata, or rather as a consequence of some deeper property. Consider a conventional automaton \mathcal{A} that recognizes a language \mathcal{L} . As typical in automata theory, we can assume that all states are reachable from the initial state, and all states are either final or they allow reaching a final state. Then, the set $\mathcal{W}(\mathcal{A})$ of all strings that can be read starting from the initial state and reaching some state is exactly equal to $\text{Pref}(\mathcal{L})$. However this is no longer true in the model of generalized automata: typically, we do not have $\mathcal{W}(\mathcal{A}) = \text{Pref}(\mathcal{L})$, but only $\mathcal{W}(\mathcal{A}) \subseteq \text{Pref}(\mathcal{L})$. For example, consider Figure 2. In both automata we have $a^3 \in \text{Pref}(\mathcal{L})$, but we have $a^3 \in \mathcal{W}(\mathcal{A})$ only for the G DFA on the left.

Given $\mathcal{W} \subseteq \text{Pref}(\mathcal{L})$, we say that a GNFA \mathcal{A} recognizing \mathcal{L} is a \mathcal{W} -GNFA if $\mathcal{W}(\mathcal{A}) = \mathcal{W}$. We will show that, if \mathcal{L} is recognized by a \mathcal{W} -G DFA, then there exists a *unique* state-minimal \mathcal{W} -G DFA recognizing \mathcal{L} . In particular, our result will imply the uniqueness of the minimal automaton for standard DFAs, because for DFAs it must necessarily be $\mathcal{W} = \text{Pref}(\mathcal{L})$.

We will actually prove much more. We will show that, once we fix \mathcal{W} , then nondeterminism and determinism still have the same expressive power, and it is possible to derive a characterization in terms of equivalence relations. In other words, we will prove a *Myhill-Nerode theorem for generalized automata*. To this end, we will introduce the notion of *locally bounded set* (Definition 8), which we can use to prove the following result.

► **Theorem 2** (Myhill-Nerode theorem for generalized automata). *Let $\mathcal{L} \subseteq \Sigma^*$ and let $\mathcal{W} \subseteq \Sigma^*$ be a locally bounded set such that $\mathcal{L} \cup \{\epsilon\} \subseteq \mathcal{W} \subseteq \text{Pref}(\mathcal{L})$. The following are equivalent:*

1. \mathcal{L} is recognized by a \mathcal{W} -GNFA.
2. The Myhill-Nerode equivalence $\equiv_{\mathcal{L}, \mathcal{W}}$ has finite index.
3. There exists a right-invariant equivalence relation \sim on \mathcal{W} of finite index such that \mathcal{L} is the union of some \sim -classes.
4. \mathcal{L} is recognized by a \mathcal{W} -G DFA.

Moreover, if one of the above statements is true (and so all the above statements are true), then there exists a unique minimal \mathcal{W} -G DFA recognizing \mathcal{L} (that is, two \mathcal{W} -G DFAs having the minimum number of states among all \mathcal{W} -G DFAs that recognize \mathcal{L} must be isomorphic).

In particular, we will show that there is no loss of generality in assuming that $\mathcal{W} \subseteq \Sigma^*$ is a locally bounded set such that $\mathcal{L} \cup \{\epsilon\} \subseteq \mathcal{W} \subseteq \text{Pref}(\mathcal{L})$, because these are *necessary* conditions for the existence of a \mathcal{W} -GNFA. We conclude that our Myhill-Nerode theorem for GNFA provides the first structural result for the model of generalized automata.

In the second part of the paper, we show that the set $\mathcal{W}(\mathcal{A})$ sheds new light on the *String Matching in Labeled Graphs (SMLG)* problem. The SMLG problem has a fascinating history that dates back to more than 30 years ago. Loosely speaking, the SMLG problem can be defined as follows: given a directed graph whose nodes are labeled with nonempty strings and given a pattern string, decide whether the pattern can be read by following a path on the graph and concatenating the node labels. The SMLG problem is a natural generalization of the classical pattern matching problem on texts (which requires determining whether a pattern occurs in a text) because texts can be seen as graphs consisting of a single path. The pattern matching problem on text can be efficiently solved in $O(n + m)$ time (n being the length of the text, m being the length of the pattern) by using the Knuth-Morris-Pratt algorithm [34]. The SMLG problem is more challenging, and the complexity can be affected by the specific variant of the pattern matching problem under consideration or the class of graphs to which the problem is restricted. For example, in the (approximate) variant where one allows errors in the graph the problem becomes NP-hard [5], so generally errors are only allowed in the pattern. The SMLG problem was studied extensively during the nineties [35, 1, 39, 5, 41, 37]; Amir et al. showed how to solve the (exact) SMLG problem on arbitrary graphs in $O(\epsilon + me)$ time [5], where e is the number of edges in the graph, m is the length of the pattern, and ϵ is the total length of all labels in the graph. Recently, the SMLG problem has been back in the spotlight. Equi et al. [21, 19, 20] showed that, *on arbitrary graphs*, for every $\epsilon > 0$ the SMLG cannot be solved in $O(me^{1-\epsilon})$ or $O(m^{1-\epsilon}e)$ time, unless the Orthogonal Vectors hypothesis fails. In applications (especially in bioinformatics) we often need faster algorithms, so the SMLG problem has been restricted to class of graphs on which it can be solved more efficiently. For example, *Elastic Founder graphs* can be used to represent multiple sequence alignments (MSA), a central model of biological evolution, and on Elastic Founder graphs the SMLG problem can be solved in linear time under a number of assumptions which only have a limited impact on the generality of the model [23, 22, 42].

The pattern matching problem on texts has been revolutionized by the invention of the Burrows-Wheeler Transform [10] and the FM-index [24, 25], which allow solving pattern matching queries efficiently on *compressed* text, thus establishing a new paradigm in bioinformatics (where the huge increase of genomic data requires the development of space-efficient algorithms) [43]. Recently, these ideas were extended to NFAs. In particular, *Wheeler NFAs* are a popular class of automata on which the SMLG problem can be solved in linear time, while only storing a *compact* representation of the Wheeler NFA [26, 3]. A special case of Wheeler NFAs are de Bruijn graphs [9], which are used to perform Eulerian sequence assembly [32, 40, 7]. Wheeler NFAs are also of relevant theoretical interest: for example, the powerset construction applied to a Wheeler NFA leads to a linear blow-up in the number of states of the equivalent DFA, and the equivalent DFA is Wheeler [4]; on arbitrary NFAs, the blow-up can be exponential.

The missing step is to determine whether it is possible to generalize the Burrows-Wheeler Transform and the FM-index to GNFA, so that the resulting data structures can also be applied to Elastic Founder graphs and other classes of graphs where labels can be arbitrary strings. Indeed, in data compression it is common to consider edge-labeled graphs where one compresses unary paths in the graph to save space and the path is replaced by a single edge labeled with the concatenation of all labels. For example, some common data structures that

are stored using this mechanism are Patricia trees, suffix trees and pangenomes [38, 6, 36]. In the following, we only consider GNFA without ϵ -transitions (that is, GNFA where no edge is labeled with the empty string ϵ) because in general a GNFA may contain arbitrarily long paths consisting of adjacent ϵ -transitions, so the SMLG problem becomes more difficult and our mechanism, based on the FM-index, fails. We say that a GNFA is an r -GNFA if all edge labels have length at most r (so a GNFA is a conventional NFA if and only if it is a 1-GNFA). Let m , e and ϵ as above, let n be the number of states, and let $\sigma = |\Sigma|$. In Section 4, we will extend the notion of Wheelerness to GNFA. The key ingredient will be the same set $\mathcal{W}(\mathcal{A})$ that we use in our Myhill-Nerode theorem: we will consider a partial order $\preceq_{\mathcal{A}}$ which sorts the set of all states with respect to the *co-lexicographical order* of the strings in $\mathcal{W}(\mathcal{A})$ (See Definition 21). We will then prove the following result.

► **Theorem 3** (FM-index of generalized automata). *Let \mathcal{A} be a Wheeler r -GNFA. Then, we can encode \mathcal{A} by using $\epsilon \log \sigma(1 + o(1)) + O(e + rn)$ bits so that later on, given a pattern $\alpha \in \Sigma^*$ of length m , we can solve the SMLG problem on \mathcal{A} in $O(m \log \log \sigma)$ time. Within the same time bound we can also decide whether $\alpha \in \mathcal{L}(\mathcal{A})$.*

If $r = 1$ (that is, if \mathcal{A} is a conventional Wheeler NFA), we conclude that we can encode \mathcal{A} by using $e \log \sigma(1 + o(1)) + O(e)$ bits (we assume that every state is reachable from the initial state, thus $e \geq n - 1$) so that we can solve the SMLG problem in $O(m \log \log \sigma)$ time, that is, we retrieve the time and space bound which were already known for Wheeler automata [26, 15]. If $r = O(1)$, we can still solve pattern matching queries in linear time (for constant alphabets), thus breaking the lower bound by Equi et al., while only storing a compact representation of \mathcal{A} .

Due to space constraints, some proofs and some auxiliary results can be found in the full version [14].

2 Preliminary Definitions

Let Σ be a finite alphabet. We denote by Σ^* the set of all finite strings on Σ . We denote by ϵ the empty string, and $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$ is the set of all nonempty finite strings on Σ . For $i \geq 0$, let $\Sigma^i \subseteq \Sigma^*$ be the set of all strings of length i (we will often interpret Σ^i as a new alphabet). If $\mathcal{L} \subseteq \Sigma^*$, let $\text{Pref}(\mathcal{L})$ be the set of all prefixes of some string in \mathcal{L} . Note that if $\mathcal{L} \neq \emptyset$, then $\epsilon \in \text{Pref}(\mathcal{L})$. We say that $\mathcal{L} \subseteq \Sigma^*$ is *prefix-free* if no string in \mathcal{L} is a strict prefix of another string in \mathcal{L} . Note that if \mathcal{L} is prefix-free and $\epsilon \in \mathcal{L}$, then $\mathcal{L} = \{\epsilon\}$. If $\mathcal{L} \subseteq \Sigma^*$, the *prefix-free kernel* of \mathcal{L} is the set $\mathcal{K}(\mathcal{L})$ of all strings in \mathcal{L} whose strict prefixes are all not in \mathcal{L} . Note that $\mathcal{K}(\mathcal{L})$ is always prefix-free, and \mathcal{L} is prefix-free if and only if $\mathcal{L} = \mathcal{K}(\mathcal{L})$.

Let us recall the definition of generalized deterministic finite automaton (GDFA) [27, 28].

► **Definition 4.** *A generalized non-deterministic finite automaton (GNFA) is a 4-tuple $\mathcal{A} = (Q, E, s, F)$, where Q is a finite set of states, $E \subseteq Q \times Q \times \Sigma^*$ is a finite set of string-labeled edges, $s \in Q$ is the initial state and $F \subseteq Q$ is a set of final states. Moreover, we assume that each $u \in Q$ is reachable from the initial state and is co-reachable, that is, it is either final or allows reaching a final state.*

A generalized deterministic finite automaton (GDFA) is a GNFA such that for every $u \in Q$ no edge leaving u is labeled with ϵ , distinct edges leaving u have distinct labels, and the set of all strings labeling some edge leaving u is prefix-free.

The assumption that every state is reachable and co-reachable is standard in automata theory because all states that do not satisfy this requirement can be removed without changing the recognized language. A conventional NFA (DFA, respectively) is a GNFA

(G DFA, respectively) where all edges are labeled with characters from Σ . Note that we explicitly require a GNFA to have finitely many edges (in conventional NFAs, the finiteness of the number of states automatically implies the finiteness of the number of edges, the alphabet being finite). If we allowed a GNFA to have infinitely many edges, then any nonempty (possibly non-regular) language would be recognized by a GNFA with two states, where the first state is initial, the second state is final, all edges go from the first state to the second state, and a string labels an edge if and only if it is in the language. By requiring a GNFA to have finitely many edges, the class of recognized languages is exactly the class of regular languages, because it is easy to transform a GNFA into a NFA with ϵ -transitions that recognizes the same language by proceeding as follows: for every edge $(u', u, \rho) \in E$, with $\rho = r_1, \dots, r_{|\rho|} \in \Sigma^+$, where $r_1, \dots, r_{|\rho|} \in \Sigma$ and $|\rho| \geq 2$, we delete the edge (u', u, ρ) , we add $|\rho| - 1$ new states $z_1, \dots, z_{|\rho|-1}$ and then we add the edges $(u', z_1, r_1), (z_1, z_2, r_2), \dots, (z_{|\rho|-1}, u, r_{|\rho|})$ (none of the new states is made initial or final).

Let us introduce some notation that will be helpful in the paper.

► **Definition 5.** Let $\mathcal{A} = (Q, E, s, F)$ be a GNFA.

1. For every $\alpha \in \Sigma^*$, let I_α be the set of all states that can be reached from the initial state by following edges whose labels, when concatenated, yield α . In other words, for some $t \geq 0$ there exist edges $(s, u_1, \alpha_1), (u_1, u_2, \alpha_2), (u_2, u_3, \alpha_3), \dots, (u_{t-1}, u_t, \alpha_t)$ such that $\alpha = \alpha_1 \alpha_2 \alpha_3 \dots \alpha_t$.
 2. Let $\mathcal{L}(\mathcal{A})$ be the language recognized by \mathcal{A} , that is, $\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^* \mid I_\alpha \cap F \neq \emptyset\}$.
 3. For every $u \in Q$, let I_u be the set of all strings that can be read from the initial state to u by concatenating edge labels, that is, $I_u = \{\alpha \in \Sigma^* \mid u \in I_\alpha\}$. Note that for every $u \in Q$ we have $\emptyset \subsetneq I_u \subseteq \text{Pref}(\mathcal{L}(\mathcal{A}))$ because every state is reachable and co-reachable.
- When \mathcal{A} is not clear from the context, we write $I_\alpha^{\mathcal{A}}$ and $I_u^{\mathcal{A}}$.

Lastly, following the introduction of the paper, we can naturally define the SMLG problem for GNFA's.

► **Definition 6.** Let \mathcal{A} be a GNFA. The String Matching in Labeled Graphs (SMLG) problem for GNFA's is defined as follows: build a data structure that encodes \mathcal{A} such that, given a string α , we can efficiently compute the set of all states reached by a path suffixed by α .

3 The Myhill-Nerode Theorem for Generalized Automata

The Myhill-Nerode theorem for conventional automata (Theorem 1) provides some algebraic properties that $\text{Pref}(\mathcal{L})$ must satisfy for $\mathcal{L} \subseteq \Sigma^*$ to be a regular language. Intuitively, the reason why $\text{Pref}(\mathcal{L})$ captures the regularity of a regular language (that is, the link between the algebraic characterization and the automata characterization of regular languages) is that, given an NFA $\mathcal{A} = (Q, E, s, F)$ that recognizes \mathcal{L} , we have $\bigcup_{u \in Q} I_u = \text{Pref}(\mathcal{L})$ because if $\alpha \in \text{Pref}(\mathcal{L})$, one can read α on \mathcal{A} starting from the initial state. However, if more generally $\mathcal{A} = (Q, E, s, F)$ is a GNFA that recognizes \mathcal{L} , then we only have $\bigcup_{u \in Q} I_u \subseteq \text{Pref}(\mathcal{L})$, because if $\alpha \in \text{Pref}(\mathcal{L})$, then one can read α on \mathcal{A} starting from the initial state, but it may happen that we have read only a strict prefix of the label of the last edge, if the label is a string of length at least two.

Let us give the following definition.

► **Definition 7.** Let $\mathcal{A} = (Q, E, s, F)$ be a GNFA. Define:

$$\mathcal{W}(\mathcal{A}) = \bigcup_{u \in Q} I_u.$$

We say that \mathcal{A} is a $\mathcal{W}(\mathcal{A})$ -GNFA.

Note that for every $\alpha \in \Sigma^*$ we have $I_\alpha \neq \emptyset$ if and only if $\alpha \in \mathcal{W}(\mathcal{A})$. Moreover, $\mathcal{L}(\mathcal{A}) \cup \{\epsilon\} \subseteq \mathcal{W}(\mathcal{A}) \subseteq \text{Pref}(\mathcal{L}(\mathcal{A}))$, because (i) if $\alpha \in \mathcal{L}(\mathcal{A})$, then $I_\alpha \cap F \neq \emptyset$ and in particular $\alpha \in \mathcal{W}(\mathcal{A})$, (ii) $\epsilon \in I_s$, (iii) $I_u \subseteq \text{Pref}(\mathcal{L}(\mathcal{A}))$ for every $u \in Q$. Let us prove an additional property of $\mathcal{W}(\mathcal{A})$. Pick $\alpha \in \mathcal{W}(\mathcal{A})$, and consider the set $T_\alpha = \{\rho \in \Sigma^+ \mid \alpha\rho \in \mathcal{W}(\mathcal{A})\}$. Consider the prefix-free kernel $\mathcal{K}(T_\alpha)$. If $\rho \in \mathcal{K}(T_\alpha)$, then $I_{\alpha\rho} \neq \emptyset$, but for every $\rho' \in \Sigma^+$ being a strict nonempty prefix of ρ we have $I_{\alpha\rho'} = \emptyset$. This implies that $|\rho| \leq r$, where r is the maximum of the lengths of edge labels. We conclude that $\mathcal{K}(T_\alpha)$ must be finite, because Σ is finite. This leads to the following definition.

► **Definition 8.**

1. Let $\mathcal{W} \subseteq \Sigma^*$. We say that \mathcal{W} is locally bounded if for every $\alpha \in \mathcal{W}$ we have that $\mathcal{K}(T_\alpha)$ is finite, where $T_\alpha = \{\rho \in \Sigma^+ \mid \alpha\rho \in \mathcal{W}\}$.
2. Let $\mathcal{A} = (Q, E, s, F)$ be a GNFA, and let $\mathcal{W} \subseteq \Sigma^*$ be a locally bounded set such that $\mathcal{L}(\mathcal{A}) \cup \{\epsilon\} \subseteq \mathcal{W} \subseteq \text{Pref}(\mathcal{L}(\mathcal{A}))$. We say that \mathcal{A} is a \mathcal{W} -GNFA if $\mathcal{W}(\mathcal{A}) = \mathcal{W}$.

► **Remark 9.** Let $\mathcal{A} = (Q, E, s, F)$ be a GDFA. Let $\alpha \in \mathcal{W}(\mathcal{A})$. If $\alpha = \epsilon$, then $I_\epsilon = \{s\}$ because no edge is labeled with ϵ . If $|\alpha| > 1$, then there exist a prefix $\alpha_1 \in \Sigma^*$ of α and $u_1 \in Q$ such that $(s, u_1, \alpha_1) \in E$, and since \mathcal{A} is a GDFA, we have $\alpha_1 \in \Sigma^+$, and both α_1 and u_1 are unique. In particular, $\alpha_1 \in \mathcal{W}(\mathcal{A})$. If α_1 is a strict prefix of α , we can repeat the argument starting from u_1 . We conclude that for every $\alpha \in \mathcal{W}(\mathcal{A})$ we have $|I_\alpha| = 1$. As a consequence, if $u, v \in Q$ are distinct, then $I_u \cap I_v = \emptyset$. In the following, if \mathcal{A} is a GDFA and $\alpha \in \mathcal{W}(\mathcal{A})$, we will identify I_α and the state being its unique element.

Moreover, our argument shows that, if \mathcal{A} is a GDFA, then for every $\alpha \in \mathcal{W}(\mathcal{A})$ such that $|\alpha| > 0$, the longest strict prefix of α in $\mathcal{W}(\mathcal{A})$ is the unique strict prefix α' of α in $\mathcal{W}(\mathcal{A})$ such that, letting $\rho \in \Sigma^+$ be the string for which $\alpha = \alpha'\rho$, we have $(I_{\alpha'}, I_\alpha, \rho) \in E$. This implies that if \mathcal{A} is a GDFA and $\alpha \in \mathcal{W}(\mathcal{A})$, then $\mathcal{K}(T_\alpha) = \{\rho \in \Sigma^+ \mid \alpha\rho \in \mathcal{W}(\mathcal{A}), (I_\alpha, I_{\alpha\rho}, \rho) \in E\}$.

In the classical Myhill-Nerode we consider equivalence relations defined on $\text{Pref}(\mathcal{L})$. In our setting, we will need to define equivalence relations on subsets of $\text{Pref}(\mathcal{L})$. This leads to the following general definition.

► **Definition 10.** Let $\mathcal{W} \subseteq \Sigma^*$ and let \sim be an equivalence relation on \mathcal{W} . We say that \sim is right-invariant if:

$$(\forall \alpha, \beta \in \mathcal{W})(\forall \phi \in \Sigma^*)(\alpha \sim \beta \rightarrow ((\alpha\phi \in \mathcal{W} \iff \beta\phi \in \mathcal{W}) \wedge (\alpha\phi \in \mathcal{W} \implies \alpha\phi \sim \beta\phi))).$$

► **Remark 11.** Notice that the property defining a right-invariant equivalence relation is trivially true if ϕ is the empty string, so it can be rephrased as follows:

$$(\forall \alpha, \beta \in \mathcal{W})(\forall \phi \in \Sigma^+)(\alpha \sim \beta \rightarrow ((\phi \in T_\alpha \iff \phi \in T_\beta) \wedge (\phi \in T_\alpha \implies \alpha\phi \sim \beta\phi))).$$

Let us prove that \sim is right-invariant if and only if:

$$(\forall \alpha, \beta \in \mathcal{W})(\forall \phi \in \Sigma^+)(\alpha \sim \beta \rightarrow ((\phi \in \mathcal{K}(T_\alpha) \iff \phi \in \mathcal{K}(T_\beta)) \wedge (\phi \in \mathcal{K}(T_\alpha) \implies \alpha\phi \sim \beta\phi))).$$

(\implies) Let $\alpha, \beta \in \mathcal{W}$ such that $\alpha \sim \beta$, and let $\phi \in \mathcal{K}(T_\alpha)$. We must prove that $\phi \in \mathcal{K}(T_\beta)$ and $\alpha\phi \sim \beta\phi$. Since $\phi \in \mathcal{K}(T_\alpha) \subseteq T_\alpha$, we immediately obtain $\phi \in T_\beta$ and $\alpha\phi \sim \beta\phi$, so we only have to prove that $\phi \in \mathcal{K}(T_\beta)$. Since for every $\phi' \in \Sigma^+$ we have $\phi' \in T_\alpha$ if and only if $\phi' \in T_\beta$, then $T_\alpha = T_\beta$, and so $\mathcal{K}(T_\alpha) = \mathcal{K}(T_\beta)$. As a consequence, from $\phi \in \mathcal{K}(T_\alpha)$ we conclude $\phi \in \mathcal{K}(T_\beta)$.

(\impliedby) Let $\alpha, \beta \in \mathcal{W}$ such that $\alpha \sim \beta$, and let $\phi \in T_\alpha$. We must prove that $\phi \in T_\beta$ and $\alpha\phi \sim \beta\phi$. Let ϕ_1, \dots, ϕ_s be all prefixes of ϕ such that $\alpha\phi_i \in \mathcal{W}$ for every $1 \leq i \leq s$, where ϕ_i is a strict prefix of ϕ_{i+1} for every $1 \leq i \leq s-1$. Note that $s \geq 2$, $\phi_1 = \epsilon$ and $\phi_s = \phi$. For

every $1 \leq i \leq s-1$, let $\psi_i \in \Sigma^+$ be such that $\phi_{i+1} = \phi_i \psi_i$. Notice that by definition we have $\psi_i \in \mathcal{K}(T_{\alpha\phi_i})$ for every $1 \leq i \leq s-1$. Since $\alpha, \beta \in \mathcal{W}$, $\alpha \sim \beta$ and $\psi_1 \in \mathcal{K}(T_{\alpha\phi_1}) = \mathcal{K}(T_\alpha)$, we obtain $\psi_1 \in \mathcal{K}(T_\beta)$ and $\alpha\phi_2 = \alpha\psi_1 \sim \beta\psi_1 = \beta\phi_2$. Since $\alpha\phi_2, \beta\phi_2 \in \mathcal{W}$, $\alpha\phi_2 \sim \beta\phi_2$ and $\psi_2 \in \mathcal{K}(T_{\alpha\phi_2})$, we obtain $\psi_2 \in \mathcal{K}(T_{\beta\phi_2})$ and $\alpha\phi_3 = \alpha\phi_2\psi_2 \sim \beta\phi_2\psi_2 = \beta\phi_3$. By continuing like that, we conclude that $\phi \in T_\beta$ and $\alpha\phi = \alpha\phi_s \sim \beta\phi_s = \beta\phi$.

In general, an equivalence relation is not right-invariant. Let us show how to define a canonical right-invariant equivalence relation starting from *any* equivalence relation.

► **Lemma 12.** *Let $\mathcal{W} \subseteq \Sigma^*$ and let \sim be an equivalence relation on \mathcal{W} . For every $\alpha, \beta \in \mathcal{W}$, let:*

$$\alpha \sim_r \beta \iff (\forall \phi \in \Sigma^*)((\alpha\phi \in \mathcal{W} \iff \beta\phi \in \mathcal{W}) \wedge (\alpha\phi \in \mathcal{W} \implies \alpha\phi \sim \beta\phi)).$$

Then \sim_r is an equivalence relation on \mathcal{W} , it is right-invariant and it is the coarsest right-invariant equivalence relation on \mathcal{W} refining \sim . We say that \sim_r is the right-invariant refinement of \sim .

The *Myhill-Nerode equivalence* plays a major role in the classical Myhill-Nerode theorem. Let us show how we can extend it when \mathcal{W} is not necessarily equal to $\text{Pref}(\mathcal{L})$.

► **Definition 13.** *Let $\mathcal{L}, \mathcal{W} \subseteq \Sigma^*$. The Myhill-Nerode equivalence on \mathcal{L} and \mathcal{W} is the equivalence relation $\equiv_{\mathcal{L}, \mathcal{W}}$ on \mathcal{W} defined as the right-invariant refinement of $\sim_{\mathcal{L}, \mathcal{W}}$, where $\sim_{\mathcal{L}, \mathcal{W}}$ is the equivalence relation on \mathcal{W} such that for every $\alpha, \beta \in \mathcal{W}$:*

$$\alpha \sim_{\mathcal{L}, \mathcal{W}} \beta \iff (\alpha \in \mathcal{L} \iff \beta \in \mathcal{L}).$$

If $\mathcal{W} = \text{Pref}(\mathcal{L})$, then we retrieve the classical Myhill-Nerode equivalence relation for \mathcal{L} . Let us describe some elementary properties of $\equiv_{\mathcal{L}, \mathcal{W}}$.

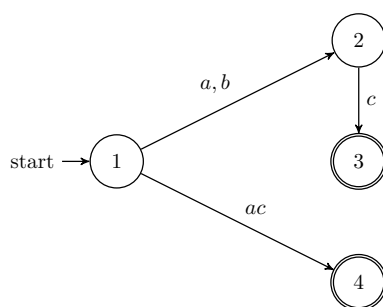
► **Lemma 14.** *Let $\mathcal{L}, \mathcal{W} \subseteq \Sigma^*$. Then $\equiv_{\mathcal{L}, \mathcal{W}}$ is right-invariant and \mathcal{L} is the union of some $\equiv_{\mathcal{L}, \mathcal{W}}$ -classes.*

Proof. First, $\equiv_{\mathcal{L}, \mathcal{W}}$ is right-invariant because it is a right-invariant refinement by definition. Moreover, \mathcal{L} is the union of some $\sim_{\mathcal{L}, \mathcal{W}}$ -classes, and so also of some $\equiv_{\mathcal{L}, \mathcal{W}}$ -classes because $\equiv_{\mathcal{L}, \mathcal{W}}$ refines $\sim_{\mathcal{L}, \mathcal{W}}$. ◀

Let \mathcal{A} be a conventional NFA, and define the equivalence relation $\sim_{\mathcal{A}}$ on $\text{Pref}(\mathcal{L}(\mathcal{A}))$ as follows: for every $\alpha, \beta \in \text{Pref}(\mathcal{L}(\mathcal{A}))$, let $\alpha \sim_{\mathcal{A}} \beta$ if and only if $I_\alpha = I_\beta$. This equivalence relation is an intermediate tool in the Myhill-Nerode theorem for conventional automata, and it can be also defined for a generalized automata \mathcal{A} by considering the equivalence relation $\sim_{\mathcal{A}}$ on $\mathcal{W}(\mathcal{A})$ such that for every $\alpha, \beta \in \mathcal{W}(\mathcal{A})$ we have $\alpha \sim_{\mathcal{A}} \beta$ if and only if $I_\alpha = I_\beta$. If \mathcal{A} is an NFA (or an NFA with ϵ -transitions), then $\sim_{\mathcal{A}}$ is right-invariant, because for every $\alpha \in \text{Pref}(\mathcal{L}(\mathcal{A}))$ and for every prefix α' of α , any path from the initial state to a node in I_α must go through a node in $I_{\alpha'}$. However, in general this property is not true if \mathcal{A} is a GNFA, so $\sim_{\mathcal{A}}$ need not be right-invariant if \mathcal{A} is a GNFA (see Figure 3). Since right-invariance is crucial in the Myhill-Nerode theorem, we will consider the right-invariant refinement of $\sim_{\mathcal{A}}$.

► **Definition 15.** *Let $\mathcal{A} = (Q, E, s, F)$ be a GNFA. Let $\equiv_{\mathcal{A}}$ be the right-invariant refinement of $\sim_{\mathcal{A}}$, where $\sim_{\mathcal{A}}$ is the equivalence relation on $\mathcal{W}(\mathcal{A})$ such that for every $\alpha, \beta \in \mathcal{W}(\mathcal{A})$:*

$$\alpha \sim_{\mathcal{A}} \beta \iff I_\alpha = I_\beta.$$



■ **Figure 3** A GNFA \mathcal{A} such that $\sim_{\mathcal{A}}$ is not right-invariant. Indeed, $a, b, ac, bc \in \mathcal{W}(\mathcal{A})$ and $a \sim_{\mathcal{A}} b$, but $ac \not\sim_{\mathcal{A}} bc$.

► **Remark 16.** Let us prove that if \mathcal{A} is GDFA, then $\sim_{\mathcal{A}}$ is right-invariant. Let $\alpha, \beta \in \mathcal{W}$ be such that $I_{\alpha} = I_{\beta}$, and let $\phi \in \mathcal{K}(T_{\alpha})$. By Remark 11, we only have to prove that $\phi \in \mathcal{K}(T_{\beta})$ and $I_{\alpha\phi} = I_{\beta\phi}$. By Remark 9, we have $\alpha\phi \in \mathcal{W}(\mathcal{A})$ and $(I_{\alpha}, I_{\alpha\phi}, \phi) \in E$. Hence $(I_{\beta}, I_{\alpha\phi}, \phi) \in E$, we obtain $I_{\alpha\phi} = I_{\beta\phi}$ and again by Remark 9, we conclude $\phi \in \mathcal{K}(T_{\beta})$. Notice that, in fact, the generalized automaton \mathcal{A} in Figure 3 is not a GDFA.

Since $\equiv_{\mathcal{A}}$ is the right-invariant refinement of $\sim_{\mathcal{A}}$, we conclude that, if \mathcal{A} is a GDFA, then $\equiv_{\mathcal{A}}$ and $\sim_{\mathcal{A}}$ are the same equivalence relation.

Let us study the properties of $\equiv_{\mathcal{A}}$.

► **Lemma 17.** *Let $\mathcal{A} = (Q, E, s, F)$ be a GNFA. Then, $\equiv_{\mathcal{A}}$ is right-invariant, it refines $\equiv_{\mathcal{L}(\mathcal{A}), \mathcal{W}(\mathcal{A})}$, it has finite index and $\mathcal{L}(\mathcal{A})$ is the union of some $\equiv_{\mathcal{A}}$ -classes.*

The following lemma is crucial to derive our Myhill-Nerode theorem for generalized automata.

► **Lemma 18.** *Let $\mathcal{L} \subseteq \Sigma^*$ and let $\mathcal{W} \subseteq \Sigma^*$ be a locally bounded set such that $\mathcal{L} \cup \{\epsilon\} \subseteq \mathcal{W} \subseteq \text{Pref}(\mathcal{L})$. Assume that \mathcal{L} is the union of some classes of a right-invariant equivalence relation \sim on \mathcal{W} of finite index. Then, \mathcal{L} is recognized by a \mathcal{W} -GDFA $\mathcal{A}_{\sim} = (Q_{\sim}, E_{\sim}, s_{\sim}, F_{\sim})$ such that:*

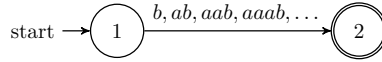
1. $|Q_{\sim}|$ is equal to the index of \sim .
2. $\equiv_{\mathcal{A}_{\sim}}$ and \sim are the same equivalence relation.

Moreover, if \mathcal{B} is a \mathcal{W} -GDFA that recognizes \mathcal{L} , then $\mathcal{A}_{\equiv_{\mathcal{B}}}$ is isomorphic to \mathcal{B} .

Proof (Sketch). The desired \mathcal{W} -GDFA $\mathcal{A}_{\sim} = (Q_{\sim}, E_{\sim}, s_{\sim}, F_{\sim})$ is defined as follows.

- $Q_{\sim} = \{[\alpha]_{\sim} \mid \alpha \in \mathcal{W}\}$.
- $s_{\sim} = [\epsilon]_{\sim}$.
- $E_{\sim} = \{([\alpha]_{\sim}, [\alpha\rho]_{\sim}, \rho) \mid \alpha \in \mathcal{W}, \rho \in \mathcal{K}(T_{\alpha})\}$.
- $F_{\sim} = \{[\alpha]_{\sim} \mid \alpha \in \mathcal{L}\}$. ◀

► **Remark 19.** In the statement of Lemma 18 we cannot remove the assumption that \mathcal{W} is locally bounded, because we have shown that if \mathcal{A} is a GNFA, then $\mathcal{W}(\mathcal{A})$ is locally bounded. However, if \mathcal{W} is not locally bounded, then \mathcal{A}_{\sim} is still a well-defined automaton with finitely many states, but it has infinitely many edges. For example, $\mathcal{W} = \{\epsilon\} \cup a^*b$ is not locally bounded because $T_{\epsilon} = a^*b$ and $\mathcal{K}(T_{\epsilon}) = a^*b$ is an infinite set. If $\mathcal{L} = a^*b$, then $\mathcal{L} \cup \{\epsilon\} \subseteq \mathcal{W} \subseteq \text{Pref}(\mathcal{L})$. Moreover, $\equiv_{\mathcal{L}, \mathcal{W}}$ has finite index (the equivalence classes are $\{\epsilon\}$ and a^*b), and by Lemma 14 we know that $\equiv_{\mathcal{L}, \mathcal{W}}$ is right-invariant and \mathcal{L} is the union of some $\equiv_{\mathcal{L}, \mathcal{W}}$ -classes. We conclude that $\mathcal{A}_{\equiv_{\mathcal{L}, \mathcal{W}}}$ is well-defined, but it has infinitely many edges (see Figure 4).



■ **Figure 4** An example where \mathcal{W} is not locally bounded.

We can now state our Myhill-Nerode theorem for generalized automata.

► **Theorem 20** (Myhill-Nerode theorem for generalized automata). *Let $\mathcal{L} \subseteq \Sigma^*$ and let $\mathcal{W} \subseteq \Sigma^*$ be a locally bounded set such that $\mathcal{L} \cup \{\epsilon\} \subseteq \mathcal{W} \subseteq \text{Pref}(\mathcal{L})$. The following are equivalent:*

1. \mathcal{L} is recognized by a \mathcal{W} -GNFA.
2. The Myhill-Nerode equivalence $\equiv_{\mathcal{L}, \mathcal{W}}$ has finite index.
3. There exists a right-invariant equivalence relation \sim on \mathcal{W} of finite index such that \mathcal{L} is the union of some \sim -classes.
4. \mathcal{L} is recognized by a \mathcal{W} -GDFA.

Moreover, if one of the above statements is true (and so all the above statements are true), then there exists a unique minimal \mathcal{W} -GDFA recognizing \mathcal{L} (that is, two \mathcal{W} -GDFA having the minimum number of states among all \mathcal{W} -GDFA that recognize \mathcal{L} must be isomorphic).

Proof.

- (1) \rightarrow (2) Let \mathcal{A} be a \mathcal{W} -GDFA recognizing \mathcal{L} . By Lemma 17 we have that $\equiv_{\mathcal{A}}$ has finite index and it refines $\equiv_{\mathcal{L}, \mathcal{W}}$, so also $\equiv_{\mathcal{L}, \mathcal{W}}$ has finite index.
- (2) \rightarrow (3) By Lemma 14 the desired equivalence relation is $\equiv_{\mathcal{L}, \mathcal{W}}$.
- (3) \rightarrow (4) It follows from Lemma 18.
- (4) \rightarrow (1) Every \mathcal{W} -GDFA is a \mathcal{W} -GNFA.

Now, let us prove that the minimum automaton is $\mathcal{A}_{\equiv_{\mathcal{L}, \mathcal{W}}}$ as defined in Lemma 18. First, $\mathcal{A}_{\equiv_{\mathcal{L}, \mathcal{W}}}$ is well-defined because $\equiv_{\mathcal{L}, \mathcal{W}}$ is right-invariant and \mathcal{L} is the union of some $\equiv_{\mathcal{L}, \mathcal{W}}$ -classes by Lemma 14, and $\equiv_{\mathcal{L}, \mathcal{W}}$ has finite index by one of the statements that we assume to be true. Now, by Lemma 18 the number of states of $\mathcal{A}_{\equiv_{\mathcal{L}, \mathcal{W}}}$ is equal to the index of $\equiv_{\mathcal{L}, \mathcal{W}}$, or equivalently, of $\equiv_{\mathcal{A}_{\equiv_{\mathcal{L}, \mathcal{W}}}}$. On the other hand, let \mathcal{B} be any \mathcal{W} -GDFA recognizing \mathcal{L} non-isomorphic to $\mathcal{A}_{\equiv_{\mathcal{L}, \mathcal{W}}}$. Then $\equiv_{\mathcal{B}}$ is a refinement of $\equiv_{\mathcal{L}, \mathcal{W}}$ by Lemma 17, and it must be a strict refinement of $\equiv_{\mathcal{L}, \mathcal{W}}$, otherwise $\mathcal{A}_{\equiv_{\mathcal{L}, \mathcal{W}}}$ would be equal to $\mathcal{A}_{\equiv_{\mathcal{B}}}$, which by Lemma 18 is isomorphic to \mathcal{B} , a contradiction. We conclude that the index of $\equiv_{\mathcal{L}, \mathcal{W}}$ is smaller than the index of $\equiv_{\mathcal{B}}$, so again by Lemma 18 the number of states of $\mathcal{A}_{\equiv_{\mathcal{L}, \mathcal{W}}}$ is smaller than the number of states of $\mathcal{A}_{\equiv_{\mathcal{B}}}$ and so of \mathcal{B} . ◀

Myhill-Nerode theorem for conventional automata (Theorem 1) is a special case of Theorem 20, because if \mathcal{A} is an NFA, then $\mathcal{W}(\mathcal{A}) = \text{Pref}(\mathcal{L}(\mathcal{A}))$. Moreover, Theorem 20 is consistent with the example in Figure 2, because, calling \mathcal{A}_1 and \mathcal{A}_2 the two GDFA in Figure 2, we have shown that $\mathcal{W}(\mathcal{A}_1) \neq \mathcal{W}(\mathcal{A}_2)$.

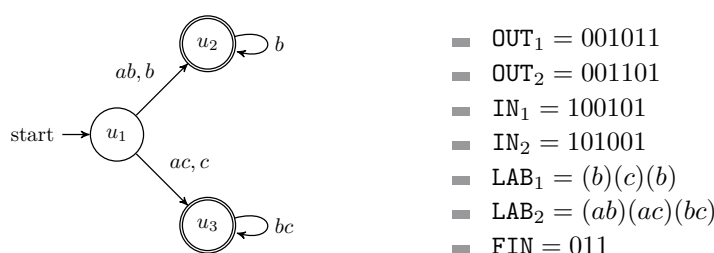
4 The FM-index of Generalized Automata

In this section, we prove Theorem 3. In order to present the main ideas, it will suffice to consider GDFA. The more general case of GNFA without ϵ -transitions requires minor technical modifications and is considered in the extended version [14].

Let V be a set. We say that a (binary) relation \leq on V is a *partial* order if \leq is reflexive, antisymmetric and transitive. A partial order \leq is a *total* order if for every $u, v \in V$ we have $(u \leq v) \vee (v \leq u)$. We say that $U \subseteq V$ is \leq -convex if for every $u, v, z \in V$, if $u \leq v \leq z$ and $u, z \in U$, then $v \in U$. For every $u, v \in V$, we write $u < v$ if $(u \leq v) \wedge (u \neq v)$.

Let us define Wheeler GDFA. As customary in the literature on Wheeler automata [3, 17], we assume that there exists a fixed total order \preceq on Σ (in our examples, we always assume $a \prec b \prec c \prec \dots$), and \preceq is extended *co-lexicographically* to Σ^* (that is, for every $\alpha, \beta \in \Sigma^*$ we have $\alpha \prec \beta$ if the reverse string α^R is lexicographically smaller than the reverse string β^R). Let $\mathcal{A} = (Q, E, s, F)$ be a GDFA. Let $\preceq_{\mathcal{A}}$ be the reflexive relation on Q such that, for every $u, v \in Q$ with $u \neq v$, we have $u \prec_{\mathcal{A}} v$ if and only if $(\forall \alpha \in I_u)(\forall \beta \in I_v)(\alpha \prec \beta)$. Since each I_u is nonempty, it is immediate to realize that $\preceq_{\mathcal{A}}$ is a partial order, but in general it is not a total order. We can then give the following definition (see Figure 5 for an example).

► **Definition 21.** Let $\mathcal{A} = (Q, E, s, F)$ be a GDFA. We say that \mathcal{A} is Wheeler if $\preceq_{\mathcal{A}}$ is a total order.



■ **Figure 5** Left: A Wheeler GDFA \mathcal{A} . States are numbered following the total order $\preceq_{\mathcal{A}}$. Right: The Burrows-Wheeler Transform (BWT) of \mathcal{A} (see Definition 29).

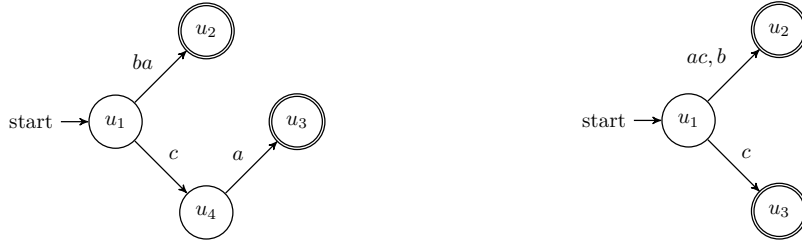
If \mathcal{A} is a conventional DFA, it is not immediately clear that Definition 21 is equivalent to the *local* definition of Wheeler DFA commonly used in the literature [3, 11, 16]. According to the local definition, a DFA $\mathcal{A} = (Q, E, s, F)$ is Wheeler if there exists a total order \leq on Q such that (i) s comes first in the total order, (ii) for every $(u', u, a), (v', v, b) \in E$, if $u < v$, then $a \preceq b$ and (iii) for every $(u', u, a), (v', v, a) \in E$, if $u < v$, then $u' < v'$. Alanko et al. [3, Corollary 3.1] proved that, if such a total order \leq exists, then it is unique and it is equal to $\preceq_{\mathcal{A}}$, so we only have to prove that if $\preceq_{\mathcal{A}}$ is a total order, then it satisfies properties (i), (ii), (iii). This follows from the following lemma.

► **Lemma 22.** Let $\mathcal{A} = (Q, E, s, F)$ be a Wheeler GDFA. Then:

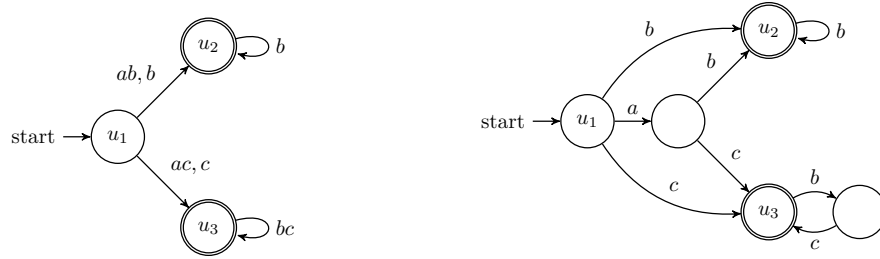
1. s comes first in the total order $\preceq_{\mathcal{A}}$.
2. For every $(u', u, \rho), (v', v, \rho') \in E$, if $u \prec_{\mathcal{A}} v$ and ρ' is not a strict suffix of ρ , then $\rho \preceq \rho'$.
3. For every $(u', u, \rho), (v', v, \rho) \in E$, if $u \prec_{\mathcal{A}} v$, then $u' \prec_{\mathcal{A}} v'$.

In case 2 of Lemma 22 we cannot remove the assumption that ρ' is not a strict prefix of ρ ; as a consequence, we cannot use Lemma 22 to provide an equivalent, local definition of Wheeler GDFA (see Figure 6). The local definition of Wheeler DFA easily implies that the problem of deciding whether a given DFA is Wheeler can be solved in polynomial time [3], but since we do not have a local definition of Wheeler GDFA, it is not clear whether the corresponding problem on GDFA is also solvable in polynomial time (and we saw in the introduction that computational problems on generalized automata are usually hard). However, we can prove that the problem is still tractable by reducing it to computing the *partial* order $\preceq_{\mathcal{A}^*}$ on a conventional DFA \mathcal{A}^* equivalent to a given GDFA \mathcal{A} .

► **Lemma 23.** Let $\mathcal{A} = (Q, E, s, F)$ be a GDFA, and let ϵ be the total length of all edge labels. In $O(\epsilon \log \epsilon)$ time we can decide whether \mathcal{A} is Wheeler and, if so, we can compute $\preceq_{\mathcal{A}}$.



■ **Figure 6** *Left:* A Wheeler G DFA $\mathcal{A} = (Q, E, s, F)$. States are numbered following the total order $\preceq_{\mathcal{A}}$. Note that $(u_1, u_2, ba), (u_4, u_3, a) \in E$, $u_2 \prec_{\mathcal{A}} u_3$, a is a strict suffix of ba and $a \prec ba$. *Right:* A G DFA $\mathcal{A} = (Q, E, s, F)$ such that states are numbered following a total order \leq such that (i) s comes first in the total order \leq , (ii) for every $(u', u, \rho), (v', v, \rho') \in E$, if $u < v$ and ρ' is not a strict suffix of ρ , then $\rho \preceq \rho'$ and (iii) for every $(u', u, \rho), (v', v, \rho) \in E$, if $u < v$, then $u' < v'$. Note that \mathcal{A} is not Wheeler because u_2 and u_3 are not $\preceq_{\mathcal{A}}$ -comparable, since $b \prec c < ac$, $c \in I_{u_3}$ and $b, ac \in I_{u_2}$.



■ **Figure 7** *Left:* The G DFA \mathcal{A} in Figure 5. *Right:* The DFA \mathcal{A}^* built starting from \mathcal{A} in the proof of Lemma 23.

Proof. Let us build a DFA \mathcal{A}^* starting from \mathcal{A} (see Figure 7 for an example). In general, if $C \subseteq \Sigma^+$, we can build a trie such that the set of all strings that can be read following a path starting from the root is equal to the set of all strings prefixing at least one element in C . If C is prefix-free, then a string is in C if and only if it can be read from the root to a leaf; we say that such a leaf *spells* the considered string. For every $u \in Q$, let C_u be the prefix-free set of all strings labeling some edge leaving u . We build an automaton $\mathcal{A}^* = (Q^*, E^*, s^*, F^*)$ by picking \mathcal{A} , removing all edges, and building a trie for every nonempty C_u in such a way that (a) the root of the trie is u , (b) the leaf that spells ρ is the unique $v \in Q$ such that $(u, v, \rho) \in E$ and (c) every internal state of the trie is a new state. Notice that $Q \subseteq Q^*$; we define $s^* = s$ and $F^* = F$. By construction, \mathcal{A}^* is a DFA, and for every $u \in Q$ we have $I_u^{\mathcal{A}} = I_u^{\mathcal{A}^*}$ (so $\mathcal{L}(\mathcal{A}^*) = \mathcal{L}(\mathcal{A})$). As a consequence, \mathcal{A} is Wheeler if and only if the restriction of the partial order $\preceq_{\mathcal{A}^*}$ to Q is a total order. Since \mathcal{A}^* is a conventional DFA, we can compute the partial order $\preceq_{\mathcal{A}^*}$ in polynomial time [17, 33, 13, 8]. For example, the algorithm in [8] runs in $O(|E^*| \log |Q^*|)$ time, and the claimed time bound follows because $|Q^*| - 1 \leq |E^*| = \epsilon$. ◀

The remaining of the section is devoted to proving that the SMLG problem can be solved efficiently on Wheeler G DFAs. If $\mathcal{A} = (Q, E, s, F)$ is a Wheeler G DFA, we write $Q = \{Q[1], Q[2], \dots, Q[n]\}$, where $Q[1] \prec_{\mathcal{A}} Q[2] \prec_{\mathcal{A}} \dots \prec_{\mathcal{A}} Q[n]$; if $1 \leq i \leq j \leq n$, let $Q[i, j] = \{Q[i], Q[i+1], \dots, Q[j-1], Q[j]\}$. If $\alpha, \beta \in \Sigma^*$, we write $\alpha \dashv \beta$ if and only if α is a suffix of β .

► **Definition 24.** Let $\mathcal{A} = (Q, E, s, F)$ be a Wheeler G DFA, and let $\alpha \in \Sigma^*$. Define:

- $G^{\prec}(\alpha) = \{u \in Q \mid (\forall \beta \in I_u)(\beta \prec \alpha)\}$;
- $G_{\dashv}(\alpha) = \{u \in Q \mid (\exists \beta \in I_u)(\alpha \dashv \beta)\}$;
- $G_{\dashv}^{\prec}(\alpha) = G^{\prec}(\alpha) \cup G_{\dashv}(\alpha) = \{u \in Q \mid (\forall \beta \in I_u)(\beta \prec \alpha) \vee (\exists \beta \in I_u)(\alpha \dashv \beta)\}$.

Intuitively, the set $G_{\downarrow}(\alpha)$ is the set of states that the SMLG problem must return on input α , and $G^{\prec}(\alpha)$ is the set of all states reached only by strings smaller than α . The following lemma shows that, as in the case of conventional Wheeler automata, both $G^{\prec}(\alpha)$ and $G_{\downarrow}(\alpha)$ are intervals with respect to the total order $\preceq_{\mathcal{A}}$, and there is no state between $G^{\prec}(\alpha)$ and $G_{\downarrow}(\alpha)$.

► **Lemma 25.** *Let $\mathcal{A} = (Q, E, s, F)$ be a Wheeler GDFA, and let $\alpha \in \Sigma^*$. Then:*

1. $G^{\prec}(\alpha) \cap G_{\downarrow}(\alpha) = \emptyset$.
2. $G_{\downarrow}(\alpha)$ is $\preceq_{\mathcal{A}}$ -convex.
3. If $u, v \in Q$ are such that $u \prec_{\mathcal{A}} v$ and $v \in G^{\prec}(\alpha)$, then $u \in G^{\prec}(\alpha)$. In other words, $G^{\prec}(\alpha) = Q[1, |G^{\prec}(\alpha)|]$.
4. If $u, v \in Q$ are such that $u \prec_{\mathcal{A}} v$ and $v \in G_{\downarrow}^{\prec}(\alpha)$, then $u \in G_{\downarrow}^{\prec}(\alpha)$. In other words, $G_{\downarrow}^{\prec}(\alpha) = Q[1, |G_{\downarrow}^{\prec}(\alpha)|]$.

In order to compute $G_{\downarrow}(\alpha)$, it will suffice to compute $G^{\prec}(\alpha)$ and $G_{\downarrow}^{\prec}(\alpha)$. Let us show how to compute $G^{\prec}(\alpha)$ and $G_{\downarrow}^{\prec}(\alpha)$; to this end, we will constantly use property 3 in Lemma 22, which is also crucial for conventional Wheeler automata (it is a generalization of the LF mapping in the FM-index [25]). First, let $G^*(\alpha)$ be the set of all states reached by an edge labeled with a string suffixed by α . Formally:

$$G^*(\alpha) = \{v \in Q \mid (\exists v' \in Q)(\exists \rho \in \Sigma^*)((v', v, \rho) \in E \wedge (\alpha \dashv \rho))\}.$$

Clearly, $G^*(\alpha) \subseteq G_{\downarrow}(\alpha)$. Now, let us give the following definition.

► **Definition 26.** *Let $\mathcal{A} = (Q, E, s, F)$ be a Wheeler GDFA. Let $U \subseteq Q$ and $\rho \in \Sigma^+$. We denote by $\text{out}(U, \rho)$ the number of edges labeled with ρ that leave states in U , and we denote by $\text{in}(U, \rho)$ the number of edges labeled with ρ that enter states in U .*

If $\alpha \in \Sigma^*$ and $0 \leq k \leq |\alpha|$, let $p(\alpha, k)$ and $s(\alpha, k)$ be the prefix and the suffix of α of length k , respectively. If $\mathcal{A} = (Q, E, s, F)$ is a GDFA and $u \in Q$, let $\lambda(u)$ be the set of all strings in Σ^+ labeling an edge reaching u ; we denote by $\min_{\lambda(u)}$ and $\max_{\lambda(u)}$ the (co-lexicographically) smallest and largest strings in $\lambda(u)$, respectively.

The next lemma formalizes the following intuition: to compute $G^{\prec}(\alpha)$, we have to consider all states whose incoming edges have a label smaller than α ; moreover, if the label is $s(\alpha, k)$ (for some k), then the start state must be in $G^{\prec}(p(\alpha, |\alpha| - k))$.

► **Lemma 27.** *Let $\mathcal{A} = (Q, E, s, F)$ be a Wheeler GDFA, and let $\alpha \in \Sigma^*$. For $0 < k < |\alpha|$, let $f_k = \text{out}(Q[1, |G^{\prec}(p(\alpha, |\alpha| - k))|], s(\alpha, k))$. Then, $|G^{\prec}(\alpha)|$ is the largest integer $0 \leq j \leq |Q|$ such that (i) $\text{in}(Q[1, j], s(\alpha, k)) \leq f_k$, for every $0 < k < |\alpha|$, and (ii) if $j \geq 1$, then $\max_{\lambda(Q[j])} \prec \alpha$.*

The following crucial lemma shows that, in order to compute $|G_{\downarrow}^{\prec}(\alpha)|$, we only have to consider $|G^{\prec}(\alpha)|$, the biggest (w.r.t $\preceq_{\mathcal{A}}$) state in $G^*(\alpha)$ and the states in $G_{\downarrow}^{\prec}(\alpha) \setminus G^*(\alpha)$.

► **Lemma 28.** *Let $\mathcal{A} = (Q, E, s, F)$ be a Wheeler GDFA, and let $\alpha \in \Sigma^*$. For $0 < k < |\alpha|$, let $f_k = \text{out}(Q[1, |G^{\prec}(p(\alpha, |\alpha| - k))|], s(\alpha, k))$ and $g_k = \text{out}(Q[1, |G_{\downarrow}^{\prec}(p(\alpha, |\alpha| - k))|], s(\alpha, k))$. Then, $g_k \geq f_k$ for every $0 < k < |\alpha|$. Moreover, $|G_{\downarrow}^{\prec}(\alpha)|$ is equal to the maximum among:*

1. $|G^{\prec}(\alpha)|$;
2. the largest integer $0 \leq i \leq |Q|$ such that, if $i \geq 1$, then $Q[i] \in G^*(\alpha)$;
3. the smallest integer $0 \leq j \leq |Q|$ such that, for every $0 < k < |\alpha|$ such that $g_k > f_k$, we have $\text{in}(Q[1, j], s(\alpha, k)) \geq g_k$.

We are now ready to generalize the Burrows-Wheeler Transform and the FM-index to Wheeler GDFAs. Let $\mathcal{A} = (Q, E, s, F)$ be a GDA. We say that \mathcal{A} is a r -GDA if all edge labels have length at most r . Fix $1 \leq i \leq r$. Let $E(i) = \{(u', u, \rho) \in E \mid \rho \in \Sigma^i\}$ be the number of edges labeled with a string of length i , and let $\Sigma(i) = \{\rho \in \Sigma^i \mid (\exists u', u \in Q)((u', u, \rho) \in E(i))\}$ be the set of all strings of length i labeling some edge. Let $e_i = |E(i)|$ and $\sigma_i = |\Sigma(i)|$; we have $\sigma_i \leq \min\{\sigma^i, e_i\}$. The i -outdegree (i -indegree, respectively) of a state is equal to the number of edges in $E(i)$ leaving (reaching, respectively) the state. The sum of the i -outdegrees of all the states and the sum of the i -indegrees of all the states are both equal to e_i . Lastly, $\sum_{i=1}^r e_i = e$ and the total length of all edge labels is $\epsilon = \sum_{i=1}^r e_i i$.

Let us define the Burrows-Wheeler Transform of a Wheeler GDA (see Figure 5 for an example).

► **Definition 29** (Burrows-Wheeler Transform of a Wheeler GDA). *Let $\mathcal{A} = (Q, E, s, F)$ be a Wheeler GDA. The Burrows-Wheeler Transform $BWT(\mathcal{A})$ of \mathcal{A} consists of the following strings.*

1. For every $1 \leq i \leq r$, the bit string $\text{OUT}_i \in \{0, 1\}^{e_i+n}$ that stores the i -outdegrees in unary. More precisely, (i) OUT_i contains exactly n characters equal to 1, (ii) OUT_i contains exactly e_i characters equal to 0, and (iii) the number of zeros between the $(\ell - 1)$ -th character equal to one (or the beginning of the sequence if $\ell = 1$) and the ℓ -th character equal to 1 yields the i -outdegree of $Q[\ell]$.
2. For every $1 \leq i \leq r$, the bit string $\text{IN}_i \in \{0, 1\}^{e_i+n}$ that stores the i -indegrees in unary. More precisely, (i) IN_i contains exactly n characters equal to 1, (ii) IN_i contains exactly e_i characters equal to 0, and (iii) the number of zeros between the $(\ell - 1)$ -th character equal to one (or the beginning of the sequence if $\ell = 1$) and the ℓ -th character equal to 1 yields the i -indegree of $Q[\ell]$.
3. For every $1 \leq i \leq r$, the string $\text{LAB}_i \in (\Sigma^i)^{e_i}$ that stores the edge labels of length i (with their multiplicities). More precisely, we sort of all edges in $E(i)$ by the index of the start states (w.r.t to $\preceq_{\mathcal{A}}$). Edges with the same start state are sorted by label. Lastly, we obtain LAB_i by concatenating the labels of all the edges following this edge order.
4. The bit string $\text{FIN} \in \{0, 1\}^n$ that marks the final states, that is, the i -th bit of FIN is equal to 1 if and only if $Q[i] \in F$.

We can now prove that the BWT of a Wheeler GDA \mathcal{A} is a valid encoding of \mathcal{A} , just like the BWT of a string is a valid encoding of the string.

► **Theorem 30.** *Let $\mathcal{A} = (Q, E, s, F)$ be a Wheeler GDA. If we only know $BWT(\mathcal{A})$, then we can retrieve \mathcal{A} (up to isomorphism). In other words, $BWT(\mathcal{A})$ is an encoding of \mathcal{A} .*

Proof. Consider states $Q[1], \dots, Q[n]$. By Lemma 22 we have $s = Q[1]$, and by using FIN we can retrieve the set of all final states F . We only have to show that we can retrieve E . In other words, for every $1 \leq i', i \leq n$ and for every $\rho \in \Sigma^+$ we must determine whether $(Q[i'], Q[i], \rho) \in E$. It will suffice to retrieve the set $E(i)$ for every $1 \leq i \leq r$, because E is the (disjoint) union of the $E(i)$'s. From LAB_i we can retrieve the labels all edges in $E(i)$, with their multiplicities. From IN_i we can retrieve the i -indegree of each $Q[\ell]$. By Lemma 22, for every $\rho \in \Sigma^i$ labeling some edge reaching some node $Q[\ell]$ and for every $\rho' \in \Sigma^i$ labeling some edge reaching $Q[\ell + 1]$ it must be $\rho \preceq \rho'$. Since we know the labels of all edges in $E(i)$ with multiplicities and we know the i -indegrees, then we can retrieve the labels of all edges entering each $Q[\ell]$, with multiplicities. From OUT_i we can retrieve the i -outdegrees of each $Q[\ell]$, and the order used in the definition of LAB_i implies that we can retrieve the labels of all edges leaving each $Q[\ell]$. Since we know the labels of all edges entering each $Q[\ell]$ and we

know the labels of all edges leaving each $Q[\ell]$, then for every $\rho \in \Sigma^i$ we know the set of all states reached by an edge labeled ρ , with multiplicities, and the set of all states having an outgoing edge labeled ρ . By Lemma 22, for every $1 \leq j < k \leq n$, if $Q[j]$ is reached by an edge labeled ρ leaving the state $Q[j']$ and $Q[k]$ is reached by an edge labeled ρ leaving the state $Q[k']$, then it must be $j' < k'$. As a consequence, we can retrieve the set $E(i)_\rho$ of all edges labeled ρ for every $\rho \in \Sigma^i$, and so we can retrieve $E(i)$, because $E(i)$ is the (disjoint) union of the $E(i)_\rho$'s. ◀

We are ready to prove the main theorem of this section (the full proof is in the extended version [14]). Recall that n is the number of vertices, e is the number of edges, and ϵ is the total length of all edge labels.

► **Theorem 31** (FM-index of Wheeler GDFAs). *Let \mathcal{A} be a Wheeler r -GDFA. Then, we can encode \mathcal{A} by using $\epsilon \log \sigma(1 + o(1)) + O(e + rn)$ bits so that later on, given a pattern $\alpha \in \Sigma^*$ of length m , we can solve the SMLG problem on \mathcal{A} in $O(rm(\log r + \log \log \sigma))$ time. Within the same time bound we can also decide whether $\alpha \in \mathcal{L}(\mathcal{A})$.*

Proof (Sketch). By Lemma 25, we only need to compute $|G^{\prec}(\alpha)|$ and $|G_{\rightarrow}^{\prec}(\alpha)|$. In $O(m)$ steps, we recursively compute $|G^{\prec}(p(\alpha, k))|$ and $|G_{\rightarrow}(p(\alpha, k))|$ for every $0 \leq k \leq |\alpha|$. By Lemma 27 and Lemma 28, we can reduce this problem to the problem of solving a number of elementary queries, such as computing $f_k = \text{out}(Q[1, |G^{\prec}(p(\alpha, |\alpha| - k))|], s(\alpha, k))$ for every $0 < k < |\alpha|$. But we only need to compute f_k for $0 < k < r + 1$, because otherwise $|s(\alpha, k)| > r$ and all edge labels have length at most r . In general, we need to solve every elementary query at most $O(r)$ time. By augmenting the Burrows-Wheeler-Transform of \mathcal{A} with compact data structures for rank/select queries and succinct dictionaries, we can solve every elementary query in $O(\log r + \log \log \sigma)$ time. ◀

5 Conclusions and Future Work

In this paper, we considered the model of generalized automata, and we introduced the set $\mathcal{W}(\mathcal{A})$. We showed that $\mathcal{W}(\mathcal{A})$ plays the same role of $\text{Pref}(\mathcal{L}(\mathcal{A}))$ in conventional NFAs: the set $\mathcal{W}(\mathcal{A})$ can be used to derive a Myhill-Nerode theorem, and it represents the starting point for extending the FM-index to generalized automata.

Further lines of research include extending the Burrows-Wheeler Transform and the FM-index to arbitrary GNFA. Indeed, the Burrows-Wheeler Transform and the FM-index were recently generalized from Wheeler NFAs to arbitrary NFAs by means of the so-called *co-lex orders* [15, 17] and *co-lex relations* [12]. However, we remark that the efficient time bounds for the SMLG problem that we derived in this paper cannot hold for arbitrary GNFA due to the (conditional) lower bounds by Equi et al. that we recalled in the introduction.

Giammaresi and Montalbano described an effective procedure for computing a minimal GDFA equivalent to a given GDFA [28, 27], but we do not know if there exists an efficient algorithm for minimizing a GDFA. The Myhill-Nerode theorem for generalized automata implies that for every minimal GDFA \mathcal{A} there exists a set $\mathcal{W} \subseteq \Sigma^*$ such that \mathcal{A} is isomorphic to the minimal \mathcal{W} -GDFA recognizing $\mathcal{L}(\mathcal{A})$. At the same time, given a \mathcal{W} -GDFA recognizing \mathcal{L} , we may build the minimal \mathcal{W} -GDFA recognizing \mathcal{L} by extending Hopcroft's algorithm [30] to GDFAs. If we could prove that, for every admissible $\mathcal{W} \subseteq \Sigma^*$, the number of states of the minimal \mathcal{W} -GDFA recognizing \mathcal{L} is comparable to the number of states of a minimal GDFA recognizing \mathcal{L} , then we would obtain a fast algorithm that significantly reduces the number of states of a GDFA without changing the recognized language.

This paper leaves many questions of theoretical interest open. The class of *Wheeler languages* is the class of all regular languages that are recognized by some Wheeler NFA [4]. Wheeler languages enjoy several properties: for example, they admit a characterization in terms of *convex* equivalence relations [4]. In addition, every Wheeler language is also recognized by some DFA, and, in particular, there exists a unique state-minimal DFA recognizing a given Wheeler language [2]. The main limitation of Wheeler languages is that they represent only a small subclass of regular languages: for example, a unary language (that is, a language over an alphabet of size one) is Wheeler if and only if it is either finite or co-finite [4]. The intuitive reason why most regular languages are not Wheeler is that the definition of a Wheeler NFA \mathcal{A} implies strong requirements on the set $\text{Pref}(\mathcal{L}(\mathcal{A}))$ (which lead to the characterization in terms of convex equivalence relations). However, when we switch to GNFA, it is $\mathcal{W}(\mathcal{A})$ that plays the role of $\text{Pref}(\mathcal{L}(\mathcal{A}))$, and it may hold $\mathcal{W}(\mathcal{A}) \subsetneq \text{Pref}(\mathcal{L}(\mathcal{A}))$, thus now the same requirement only apply to a smaller subset. The natural question is whether Wheeler GNFA extend the class of Wheeler languages. The answer is affirmative: there exists a regular \mathcal{L} language such that \mathcal{L} is not Wheeler (that is, no Wheeler NFA recognizes \mathcal{L}), but \mathcal{L} is recognized by a Wheeler GDFA. Define $\mathcal{L} = \{a^{2^n} \mid n \geq 0\}$. Then, \mathcal{L} is not Wheeler [4], but \mathcal{L} is recognized by the GDFA consisting of a single state, both initial and final, with a self-loop labeled aa . As a consequence, the class of all languages recognized by a Wheeler GNFA is strictly larger than the class of Wheeler languages. We can call the languages in this new class *generalized Wheeler languages*: the next step is to understand which properties of Wheeler languages are still true and how it is possible to characterize this new class.

References



- 1 Tatsuya Akutsu. A linear time pattern matching algorithm between a string and a tree. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Combinatorial Pattern Matching*, pages 1–10, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.
- 2 Jarno Alanko, Nicola Cotumaccio, and Nicola Prezza. Linear-time minimization of wheeler dfas. In *2022 Data Compression Conference (DCC)*, pages 53–62, 2022. doi:10.1109/DCC52660.2022.00013.
- 3 Jarno Alanko, Giovanna D’Agostino, Alberto Policriti, and Nicola Prezza. *Regular Languages meet Prefix Sorting*, pages 911–930. SIAM, 2020. doi:10.1137/1.9781611975994.55.
- 4 Jarno Alanko, Giovanna D’Agostino, Alberto Policriti, and Nicola Prezza. Wheeler languages. *Information and Computation*, 281:104820, 2021. doi:10.1016/j.ic.2021.104820.
- 5 Amihood Amir, Moshe Lewenstein, and Noa Lewenstein. Pattern matching in hypertext. *Journal of Algorithms*, 35(1):82–99, 2000. doi:10.1006/jagm.1999.1063.
- 6 Jasmijn A. Baaijens, Paola Bonizzoni, Christina Boucher, Gianluca Della Vedova, Yuri Pirola, Raffaella Rizzi, and Jouni Sirén. Computational graph pangenomics: a tutorial on data structures and their applications. *Nat. Comput.*, 21(1):81–108, 2022. doi:10.1007/s11047-022-09882-6.
- 7 Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, Alexey V. Pyshkin, Alexander V. Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev, and Pavel A. Pevzner. SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012. PMID: 22506599. doi:10.1089/cmb.2012.0021.
- 8 Ruben Becker, Manuel Cáceres, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Francisco Olivares, and Nicola Prezza. Sorting Finite Automata via Partition Refinement. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.15.

- 9 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de bruijn graphs. In Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*, pages 225–235, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 10 M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm, 1994.
- 11 Alessio Conte, Nicola Cotumaccio, Travis Gagie, Giovanni Manzini, Nicola Prezza, and Marinella Sciortino. Computing matching statistics on wheeler dfas. In *2023 Data Compression Conference (DCC)*, pages 150–159, 2023. doi:10.1109/DCC55655.2023.00023.
- 12 Nicola Cotumaccio. Graphs can be succinctly indexed for pattern matching in $o(|e|^2 + |v|^{5/2})$ time. In *2022 Data Compression Conference (DCC)*, pages 272–281, 2022. doi:10.1109/DCC52660.2022.00035.
- 13 Nicola Cotumaccio. Prefix Sorting DFAs: A Recursive Algorithm. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2023.22.
- 14 Nicola Cotumaccio. A myhill-nerode theorem for generalized automata, with applications to pattern matching and compression, 2024. arXiv:2302.06506.
- 15 Nicola Cotumaccio, Giovanna D’Agostino, Alberto Policriti, and Nicola Prezza. Co-lexicographically ordering automata and regular languages - part i. *J. ACM*, 70(4), August 2023. doi:10.1145/3607471.
- 16 Nicola Cotumaccio, Travis Gagie, Dominik Köppl, and Nicola Prezza. Space-time trade-offs for the lcp array of wheeler dfas. In Franco Maria Nardini, Nadia Pisanti, and Rossano Venturini, editors, *String Processing and Information Retrieval*, pages 143–156, Cham, 2023. Springer Nature Switzerland.
- 17 Nicola Cotumaccio and Nicola Prezza. On indexing and compressing finite automata. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’21*, pages 2585–2599, USA, 2021. Society for Industrial and Applied Mathematics.
- 18 Samuel Eilenberg. *Automata, Languages, and Machines*. Academic Press, Inc., USA, 1974.
- 19 Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of string matching for graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.55.
- 20 Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless seth fails. In Tomáš Bureš, Riccardo Dondi, Johann Gamper, Giovanna Guerrini, Tomasz Jurdziński, Claus Pahl, Florian Sikora, and Prudence W.H. Wong, editors, *SOFSEM 2021: Theory and Practice of Computer Science*, pages 608–622, Cham, 2021. Springer International Publishing.
- 21 Massimo Equi, Veli Mäkinen, Alexandru I. Tomescu, and Roberto Grossi. On the complexity of string matching for graphs. *ACM Trans. Algorithms*, 19(3), April 2023. doi:10.1145/3588334.
- 22 Massimo Equi, Tuukka Norri, Jarno Alanko, Bastien Cazaux, Alexandru I. Tomescu, and Veli Mäkinen. Algorithms and complexity on indexing elastic founder graphs. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.20.
- 23 Massimo Equi, Tuukka Norri, Jarno Alanko, Bastien Cazaux, Alexandru I. Tomescu, and Veli Mäkinen. Algorithms and complexity on indexing founder graphs. *Algorithmica*, 85(6):1586–1623, 2023. doi:10.1007/s00453-022-01007-w.

- 24 P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS'00)*, pages 390–398, 2000. doi:10.1109/SFCS.2000.892127.
- 25 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, July 2005. doi:10.1145/1082036.1082039.
- 26 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for bwt-based data structures. *Theoretical Computer Science*, 698:67–78, 2017. Algorithms, Strings and Theoretical Approaches in the Big Data Era (In Honor of the 60th Birthday of Professor Raffaele Giancarlo). doi:10.1016/j.tcs.2017.06.016.
- 27 Dora Giammarresi and Rosa Montalbano. Deterministic generalized automata. In Ernst W. Mayr and Claude Puech, editors, *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings*, volume 900 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 1995. doi:10.1007/3-540-59042-0_84.
- 28 Dora Giammarresi and Rosa Montalbano. Deterministic generalized automata. *Theor. Comput. Sci.*, 215(1-2):191–208, 1999. doi:10.1016/S0304-3975(97)00166-7.
- 29 Kosaburo Hashiguchi. Algorithms for determining the smallest number of nonterminals (states) sufficient for generating (accepting) a regular language. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez Artalejo, editors, *Automata, Languages and Programming*, pages 641–648, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.
- 30 John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971. doi:10.1016/B978-0-12-417750-5.50022-1.
- 31 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006.
- 32 Ramana M. Idury and Michael S. Waterman. A new algorithm for DNA sequence assembly. *Journal of computational biology : a journal of computational molecular cell biology*, 2 2:291–306, 1995.
- 33 Sung-Hwan Kim, Francisco Olivares, and Nicola Prezza. Faster prefix-sorting algorithms for deterministic finite automata. In Laurent Bulteau and Zsuzsanna Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023, June 26-28, 2023, Marne-la-Vallée, France*, volume 259 of *LIPICs*, pages 16:1–16:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.CPM.2023.16.
- 34 Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. doi:10.1137/0206024.
- 35 Udi Manber and Sun Wu. Approximate string matching with arbitrary costs for text and hypertext. In *Advances In Structural And Syntactic Pattern Recognition*, pages 22–33. World Scientific, 1992.
- 36 Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Bioinformatics in the Era of High-Throughput Sequencing*. Cambridge University Press, 2 edition, 2023.
- 37 Gonzalo Navarro. Improved approximate pattern matching on hypertext. *Theor. Comput. Sci.*, 237(1–2):455–463, April 2000. doi:10.1016/S0304-3975(99)00333-3.
- 38 Gonzalo Navarro. *Compact Data Structures: A Practical Approach*. Cambridge University Press, 2016. doi:10.1017/CB09781316588284.
- 39 Kunsoo Park and Dong Kyue Kim. String matching in hypertext. In Zvi Galil and Esko Ukkonen, editors, *Combinatorial Pattern Matching*, pages 318–329, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- 40 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001. doi:10.1073/pnas.171285098.

- 41 Mikko Rautiainen and Tobias Marschall. Aligning sequences to general graphs in $o(v + me)$ time. *bioRxiv*, 2017. doi:10.1101/216127.
- 42 Nicola Rizzo and Veli Mäkinen. Linear time construction of indexable elastic founder graphs. In Cristina Bazgan and Henning Fernau, editors, *Combinatorial Algorithms*, pages 480–493, Cham, 2022. Springer International Publishing.
- 43 Jared T. Simpson and Richard Durbin. Efficient construction of an assembly string graph using the fm-index. *Bioinform.*, 26(12):367–373, 2010. doi:10.1093/bioinformatics/btq217.

Nonnegativity Problems for Matrix Semigroups

Julian D’Costa  

Department of Computer Science, University of Oxford, UK

Joël Ouaknine  

Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

James Worrell  

Department of Computer Science, University of Oxford, UK

Abstract

The matrix semigroup membership problem asks, given square matrices M, M_1, \dots, M_k of the same dimension, whether M lies in the semigroup generated by M_1, \dots, M_k . It is classical that this problem is undecidable in general, but decidable in case M_1, \dots, M_k commute. In this paper we consider the problem of whether, given M_1, \dots, M_k , the semigroup generated by M_1, \dots, M_k contains a non-negative matrix. We show that in case M_1, \dots, M_k commute, this problem is decidable subject to Schanuel’s Conjecture. We show also that the problem is undecidable if the commutativity assumption is dropped. A key lemma in our decidability proof is a procedure to determine, given a matrix M , whether the sequence of matrices $(M^n)_{n=0}^\infty$ is ultimately nonnegative. This answers a problem posed by S. Akshay [1]. The latter result is in stark contrast to the notorious fact that it is not known how to determine, for any specific matrix index (i, j) , whether the sequence $(M^n)_{i,j}$ is ultimately nonnegative. Indeed the latter is equivalent to the Ultimate Positivity Problem for linear recurrence sequences, a longstanding open problem.

2012 ACM Subject Classification Computing methodologies \rightarrow Symbolic and algebraic manipulation; Theory of computation \rightarrow Formal languages and automata theory

Keywords and phrases Decidability, Linear Recurrence Sequences, Schanuel’s Conjecture

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.27

Related Version *Full Version:* <https://arxiv.org/abs/2311.06241>

Funding *Julian D’Costa:* emmy.network foundation under the aegis of the Fondation de Luxembourg, UKRI Frontier Research Grant EP/X033813/1.

Joël Ouaknine: DFG grant 389792660 as part of TRR 248 (see <https://perspicuous-computing.science>). Joël Ouaknine is also affiliated with Keble College, Oxford as emmy.network Fellow.

James Worrell: UKRI Frontier Research Grant EP/X033813/1.

1 Introduction

The *Membership Problem* for finitely generated matrix semigroups asks, given square matrices M, M_1, \dots, M_k of the same dimension and with rational entries, whether M lies in the semigroup generated by M_1, \dots, M_k . The problem was shown to be undecidable by Markov in the 1940s [15], thereby becoming one of the first instances of a natural undecidable mathematical problem. The problem, however, becomes decidable under the assumption that the matrices M_1, \dots, M_k commute [2].

There are many variants of the Membership Problem. In the *Mortality Problem* one asks whether the zero matrix lies in a finitely generated matrix semigroup. This problem is undecidable already for 3×3 matrices [21]. Meanwhile, the *Identity Problem* asks to determine whether the identity matrix lies in a given finitely generated matrix semigroup. The latter problem is undecidable in general but decidable for certain nilpotent and low-order matrix groups [3, 9, 10, 12].



© Julian D’Costa, Joël Ouaknine, and James Worrell;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 27; pp. 27:1–27:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



The present paper is concerned with the *Non-negative Membership Problem*, which asks to determine whether a given finitely generated matrix semigroup contains a non-negative matrix, i.e., a matrix all of whose entries are non-negative. We show that this problem is undecidable in general but is decidable in the commutative case subject to Schanuel's Conjecture, a well-known unifying conjecture in transcendence theory. Our reliance on Schanuel's Conjecture arises because we reduce the commutative case of the Non-negative Membership Problem to the decision problem for the first-order theory of real-closed fields with exponential. As shown by Macintyre and Wilkie [14], the latter theory is decidable if one assumes Schanuel's Conjecture for the real exponential.

A key lemma in our main decidability result involves determining, for a given matrix M , whether for all but finitely many $n \in \mathbb{N}$ the matrix power M^n is non-negative. In such a case we say that M is eventually non-negative. We give an effective characterisation of eventually non-negative matrices, answering a question posed by S. Akshay [1]. The characterisation is relatively straightforward and relies on classical results about rational sequences over the semi-ring of non-negative rational numbers. We note that the problem of determining whether, for some fixed index (i, j) , the sequence of scalars $\langle (M^n)_{i,j} : n \in \mathbb{N} \rangle$ is ultimately non-negative is equivalent to the Ultimate Positivity Problem for linear recurrence sequences, decidability of which is a longstanding open problem [20].

It is immediate that a matrix semigroup contains a non-negative matrix if and only if it contains an eventually non-negative matrix. Using a symbolic version of our criterion characterizing eventually non-negative matrices, we reduce the Non-negative Membership Problem to a version of integer programming with certain transcendental constants (namely logarithms of algebraic numbers). In turn we reduce the solution of such integer programs to the decision problem for the first-order theory of real closed fields with exponential.

A simpler variant of our main result concerns the problem of deciding whether a finitely generated matrix semigroup contains a positive matrix, i.e., a matrix all of whose entries are strictly positive. Here, to show decidability, we rely on a known characterisation of eventually positive matrices, due to Noutsos [18]. While we still need to invoke Schanuel's Conjecture in this case, we do so through the use of a procedure of Richardson [22] for deciding equality of elementary numbers (which is much more straightforward than the result of Macintyre and Wilkie mentioned above).

As far as we are aware, the Non-negative Membership Problem has not been directly addressed before. We note however that decidability of the version of this problem for sub-semigroups of the group $\mathbf{GL}(2, \mathbb{Z})$ of 2×2 invertible integer matrices follows directly from [7, Theorem 13].

2 Mathematical Background

Here we state some number-theoretic results that we will need in the sequel.

First stated in the 1960s, Schanuel's conjecture is a unifying conjecture in transcendental number theory that generalizes many of the classical results in the field.

► **Conjecture 1** (Schanuel's conjecture [13]). *If $\alpha_1, \dots, \alpha_k \in \mathbb{C}$ are rationally linearly independent, then some k -element subset of $\{\alpha_1, \dots, \alpha_k, e^{\alpha_1}, \dots, e^{\alpha_k}\}$ is algebraically independent.*

An elementary point is an element of \mathbb{C}^n that arises as an isolated, nonsingular solution of n equations in n variables x_1, \dots, x_n , with each equation being either of the form $P(x_1, \dots, x_n) = 0$, where $P \in \mathbb{Q}[x_1, \dots, x_n]$ is a polynomial with rational coefficients, or of the form $x_j - e^{x_i} = 0$ for $i, j \in \{1, \dots, n\}$. An elementary number is the polynomial image of an elementary point.

Roughly speaking, an elementary number is obtained by starting with the rationals and applying addition, subtraction, multiplication, division, exponentiation, and taking natural logarithms. Elementary numbers may be transcendental and, unsurprisingly, it is non-trivial to determine whether an elementary number is equal to zero.

► **Proposition 2** (Richardson [22]). *The problem of determining zeroness of an elementary number is semi-decidable. The problem is moreover decidable if one assumes Schanuel’s conjecture.*

We will also need the following theorem due to Masser.

► **Theorem 3** (Multiplicative relations among algebraic numbers [16]). *Let m be fixed, and let $\lambda_1, \dots, \lambda_m$ be complex algebraic numbers. Consider the free abelian group L under addition given by*

$$L = \{(v_1, \dots, v_m) \in \mathbb{Z}^m : \lambda_1^{v_1} \dots \lambda_m^{v_m} = 1\}.$$

Then L has a basis $\{w_1, \dots, w_p\} \subseteq \mathbb{Z}^m$ (with $p \leq m$), where the entries of each of the w_j are all polynomially bounded in the sum of the heights and degrees of the minimal polynomials of $\lambda_1, \dots, \lambda_m$.

3 Linear Recurrence Sequences

First, we recall some basic terminology and results about linear recurrence sequences.

A sequence $\mathbf{u} = (u_n)_{n=0}^\infty$ of elements of a semiring K is called *K -rational* if there exists $d \geq 1$, $v, w \in K^d$ and $M \in K^{d \times d}$ such that $u_n = v^\top M^n w$ for all n . When K is a field, a sequence is *K -rational* if and only if it satisfies a linear recurrence relation

$$u_n = a_1 u_{n-1} + \dots + a_d u_{n-d} \quad (n \geq d)$$

where $a_1, \dots, a_d \in K$. In this case we also call \mathbf{u} a *linear recurrence sequence (LRS)*.

With the unique minimal order recurrence satisfied by \mathbf{u} we associate the *characteristic polynomial*

$$P(X) = X^d - a_1 X^{d-1} - \dots - a_d.$$

The roots of $P(X)$ are called the *characteristic roots* of \mathbf{u} . Writing $\lambda_1, \dots, \lambda_m$ for the distinct characteristic roots, in non-increasing order of absolute value, the sequence \mathbf{u} admits a closed-form *exponential-polynomial* representation:

$$u_n = \sum_{i=1}^m P_i(n) \lambda_i^n,$$

where the P_i are univariate polynomials whose coefficients are algebraic over K . We say that \mathbf{u} is *non-degenerate* if no quotient of two distinct characteristic roots is a root of unity. We also say that a matrix $M \in \mathbb{Q}^{d \times d}$ is non-degenerate if no quotient of two distinct eigenvalues is a root of unity.

In this paper we exclusively consider sequences with rational entries. We say that an LRS \mathbf{u} is *dominated* if λ_1 is the unique characteristic root of maximum modulus. Note that in this case λ_1 is necessarily real. We have the following three straightforward propositions concerning dominated LRS.

27:4 Nonnegativity Problems for Matrix Semigroups

► **Proposition 4.** *If an LRS \mathbf{u} is dominated then $\{n \in \mathbb{N} : u_n \geq 0\}$ is an effectively computable ultimately periodic set.*

Proof. Consider the closed form representation $u_n = \sum_{i=1}^m P_i(n)\lambda_i^n$ with unique dominant root λ_1 , necessarily real. Suppose that P_1 has degree k and leading coefficient a . Then we have $\frac{u_n}{n^k|\lambda_1|^n} = a(\lambda_1/|\lambda_1|)^n + o(1)$. Hence for sufficiently large n the sign of u_n is determined by the sign of a and the parity of n . ◀

► **Proposition 5.** *If \mathbf{u} is a non-degenerate LRS such that some subsequence $(u_{cn+d})_{n=0}^\infty$ is dominated, where c is a positive integer and $d \in \{0, 1, \dots, c-1\}$, then \mathbf{u} itself is dominated.*

Proof. The sequence \mathbf{u} admits a closed-form representation $u_n = \sum_{i=1}^m P_i(n)\lambda_i^n$, where $\lambda_1, \dots, \lambda_m$ are the characteristic roots and P_1, \dots, P_m are polynomials. Then

$$\begin{aligned} u_{cn+d} &= \sum_{i=1}^m P_i(cn+d)\lambda_i^{cn+d} \\ &= \sum_{i=1}^m Q_i(n)(\lambda_i^c)^n \end{aligned}$$

where $Q_i(n) := P_i(cn+d)\lambda_i^d$ for $i \in \{1, \dots, m\}$.

Note that the polynomials Q_1, \dots, Q_m are non-zero and, by non-degeneracy of \mathbf{u} , the numbers $\lambda_1^c, \dots, \lambda_m^c$ are pairwise distinct. Since the sequence $(u_{cn+d})_{n=0}^\infty$ is dominated, we have that λ_1^c is its unique characteristic root of maximum modulus. But then λ_1 is the unique characteristic root of \mathbf{u} . ◀

► **Proposition 6.** *An LRS that is both non-degenerate and rational over the semiring \mathbb{Q}_+ of nonnegative rational numbers is dominated.*

Proof. Berstel [4] showed that if a sequence \mathbf{u} is \mathbb{Q}_+ -rational then its characteristic roots of maximum modulus all have the form $\rho\omega$ for some non-negative real number ρ and root of unity ω . Since \mathbf{u} is non-degenerate it follows that it has a unique dominant root. For an exposition, see [5, Chap. 8, Thm 1.1]. We provide an alternate proof based on the Perron-Frobenius theorem in Appendix B. ◀

► **Theorem 7.** *Given $M \in \mathbb{Q}^{d \times d}$ the set $S := \{n \in \mathbb{N} : M^n \geq 0\}$ is ultimately periodic and effectively computable.*

Proof. Recall that for some (effectively computable) strictly positive integer L the matrix M^L is non-degenerate. It will suffice to show that for each $l \in \{0, \dots, L-1\}$ we can compute the set $S_l := \{n \in S : n \equiv l \pmod{L}\}$. Our procedure to do this is as follows. First, check for every pair of indices $i, j \in \{1, \dots, d\}$, whether the sequence $(u_n^{(i,j)})_{n=0}^\infty$ given by $u_n^{(i,j)} = (M^{Ln+l})_{i,j}$, is dominated. If yes then by Proposition 4 we can compute S_l as the intersection over all pairs (i, j) of the sets $\{n \in \mathbb{N} : u_n^{(i,j)} \geq 0\}$. If no, then we claim that S_l is empty.

Indeed, suppose $n_0 \in S_l$. Then for each pair of indices $i, j \in \{1, \dots, d\}$, the LRS $(v_n^{(i,j)})_{n=0}^\infty$ defined by $v_n^{(i,j)} = (M^{n_0(1+Ln)})_{i,j} = e_i^\top M^{n_0} (M^{Ln_0})^n e_j^\top$ is both non-degenerate and \mathbb{Q}^+ -rational. By Proposition 6 each sequence $(v_n^{(i,j)})_{n=0}^\infty$ is dominated. Moreover, since $(v_n^{(i,j)})_{n=0}^\infty$ is a subsequence of $(u_n^{(i,j)})_{n=0}^\infty$, the latter is also dominated by Proposition 5. This proves (the contrapositive of) our claim. ◀

► **Remark 8.** We can extract from the proof of Theorem 7 an effective characterisation of those matrices M such that $M^n \geq 0$ for some positive integer n . Let L be the least positive integer such that M^L is non-degenerate. Then some positive power of M is a non-negative matrix iff some positive power of M^L is non-negative iff for all indices (i, j) the sequence $(u_n^{(i,j)})_{n=0}^\infty$ defined by $u_n^{(i,j)} := (M^{Ln})_{i,j}$ is dominated and not ultimately negative.

4 The Positive Membership Problem

4.1 Eventually Positive Matrices

Recall that a positive matrix is one whose entries are all strictly positive. In this section we show decidability of the following problem.

► **Problem 9** (Positive Membership for Commutative Semigroup). *Given a set of commuting $d \times d$ matrices $\{A_1, \dots, A_k\}$ with rational entries, decide whether the generated semigroup contains a positive matrix.*

We approach this problem through the notion of eventually positive matrix.

► **Definition 10** (Eventually Positive Matrix). *We call a matrix M eventually positive if there exists a natural number n_0 such that for all $n \geq n_0$, the matrix M^n is positive.*

We will need the following definition and result, adapted from Noutsos [18], which characterizes eventual positivity of a matrix by proving a converse of the Perron-Frobenius theorem.

► **Definition 11** (Strong Perron-Frobenius property). *A matrix $A \in \mathbb{R}^{n \times n}$ has the strong Perron-Frobenius property if there exists an eigenvalue λ with the following properties:*

1. λ is real and positive,
2. λ is the unique dominant eigenvalue,
3. λ is simple,
4. λ has a corresponding eigenvector, all of whose entries are positive.

We now have:

► **Theorem 12** (Characterizing eventual positivity [18]). *A matrix A is eventually positive iff A and A^\top both have the strong Perron-Frobenius property.*

Clearly a semigroup contains a positive matrix if and only if it contains an eventually positive matrix. By the theorem above, this holds if and only if the semigroup contains a matrix A such that both A and A^\top have the strong Perron-Frobenius property.

4.2 Reduction to Integer Programming

We consider a direct-sum decomposition of \mathbb{C}^d induced by a collection of commuting matrices $A_1, \dots, A_k \in \mathbb{C}^{d \times d}$. Let $\sigma(A_i)$ denote the set of eigenvalues of A_i and consider a tuple of eigenvalues

$$\lambda = (\lambda_1, \dots, \lambda_k) \in \sigma(A_1) \times \dots \times \sigma(A_k).$$

Recall that $\ker(A_i - \lambda_i I)^d$ is the generalized eigenspace of A_i corresponding to λ_i for $i \in \{1, \dots, k\}$. We say that the tuple λ is a *joint eigenvalue* of A_1, \dots, A_k if

$$V_\lambda := \bigcap_{i=1}^k \ker(A_i - \lambda_i I)^d$$

is non null. Intuitively, joint eigenvalues are tuples of eigenvalues of the A_i whose respective generalized eigenspaces have non-trivial intersection. The set of joint eigenvalues is called the *joint spectrum* of A_1, \dots, A_k , denoted Σ . Using the fact that commuting matrices preserve each other's generalized eigenspaces, it can be shown (see, e.g., [19, Theorem 2.4]) that

$$\mathbb{C}^d = \bigoplus_{\lambda \in \Sigma} V_\lambda,$$

and that, for all $i \in \{1, \dots, k\}$ and $\lambda \in \Sigma$, A_i preserves V_λ , and the restriction of A_i to V_λ has spectrum $\{\lambda_i\}$. The commuting matrices $A_1^\top, \dots, A_k^\top$ induce an analogous decomposition $\mathbb{C}^d = \bigoplus_{\lambda \in \Sigma} W_\lambda$, where $W_\lambda := \bigcap_{i=1}^k \ker(A_i^\top - \lambda_i I)$.

Consider a matrix $A := A_1^{m_1} \cdots A_k^{m_k}$ for $m_1, \dots, m_k \in \mathbb{N}$. Given $\lambda = (\lambda_1, \dots, \lambda_k) \in \Sigma$, by the Spectral Mapping Theorem the restriction of A to the subspace V_λ has a single eigenvalue $\lambda_1^{m_1} \cdots \lambda_k^{m_k}$. Thus all non-zero vectors in V_λ are generalized eigenvectors of A for this eigenvalue. It follows that if A is eventually positive, then Conditions 1 and 2 of Theorem 12 imply that there exists $\lambda \in \Sigma$ such that $\lambda_1^{m_1} \cdots \lambda_k^{m_k}$ is real positive and is the unique dominant eigenvalue of A . Meanwhile, by Conditions 3 and 4 we can furthermore choose λ such that the spaces V_λ and W_λ are both one dimensional and contain a positive vector. In such a situation we call λ a *dominant joint eigenvalue* for A , V_λ a *positive right eigenspace*, and W_λ a *positive left eigenspace*.

Conversely, suppose that there exist $m_1, \dots, m_k \in \mathbb{N}$ and $\lambda \in \Sigma$ such that λ is a dominant joint eigenvalue of $A := A_1^{m_1} \cdots A_k^{m_k}$, and V_λ and W_λ are positive right and left joint eigenspaces respectively. Then Theorem 12 implies that A is eventually positive.

In summary, the semigroup generated by A_1, \dots, A_k contains a positive matrix if and only if there exists $\lambda \in \Sigma$, such that V_λ and W_λ are positive joint eigenspaces, and there are $m_1, \dots, m_k \in \mathbb{N}$ with

1. $\prod_{i=1}^k \lambda_i^{m_i} > 0$,
2. $\forall \mu \in \Sigma \setminus \{\lambda\}, \prod_{i=1}^k \lambda_i^{m_i} > \left| \prod_{i=1}^k \mu_i^{m_i} \right|$.

It is clear that the Positive Membership Problem reduces to the restricted version of the problem in which one asks for the existence of a positive matrix formed by a product in which all generators of the semigroup are used at least once. Thus we can assume that the desired exponents m_1, \dots, m_k in Conditions 1 and 2 are all strictly positive. In this situation we may rewrite Condition 1 as $\sum_{i=1}^k m_i \arg(\lambda_i) = 0 \pmod{2\pi}$ and Condition 2 as either $\mu_1 \cdots \mu_k = 0$ or $\sum_{i=1}^k m_i \log |\mu_i / \lambda_i| < 0$ for all $\mu \in \Sigma \setminus \{\lambda\}$. Let $\mathbf{c} = (\arg(\lambda_1), \dots, \arg(\lambda_k)) \in \mathbb{R}^k$ and let A be the $(|\Sigma| - 1 \times k)$ -matrix defined by writing, for all $i \in \{1, \dots, k\}$ $\mu \in \Sigma \setminus \{\lambda\}$, $a_{\mu,i} := \log |\mu_i / \lambda_i|$ if $\mu_1 \cdots \mu_k \neq 0$ and $a_{\mu,i} = -1$ if $\mu_1 \cdots \mu_k = 0$. Then the two conditions above are equivalent to the existence of a solution $x \in \mathbb{N}^k$ of the following integer program:

$$(c^\top x = 0 \pmod{2\pi}) \wedge Ax < 0.$$

Note that by incorporating positivity constraints $-x_i < 0$ in A we can assume without loss of generality that x ranges over \mathbb{Z}^k . We now show that the satisfiability of such an integer program is decidable, subject to Schanuel's conjecture.

4.3 Integer Programming with Logs of Algebraic Numbers

► **Problem 13 (IP-log).** *An instance of the IP-log problem consists of a matrix $A \in \mathbb{R}^{m \times n}$ whose entries are sums of logarithms of non-zero real algebraic numbers and a vector $c \in \mathbb{R}^m$ whose entries are arguments of non-zero algebraic numbers. The problem asks to determine whether there exists a vector $x \in \mathbb{Z}^n$ such that $c^\top x = 0 \pmod{2\pi}$ and $Ax < 0$.*

Let us start by observing that one can eliminate the equation in the IP-log problem by an effective linear change of variables. Indeed, exponentiation turns the linear relation $c^\top x = 0 \pmod{2\pi}$ into a multiplicative relation among algebraic numbers. We can then use Theorem 3 to find a basis $\{v_1, \dots, v_l\} \subseteq \mathbb{Z}^n$ of the group of integer solutions of the equation $c^\top x = 0 \pmod{2\pi}$. Let $B \in \mathbb{Z}^{n \times l}$ be the matrix that has these basis vectors as columns. Then $c^\top x = 0 \pmod{2\pi}$ iff $x = By$ for some integer vector y . Hence the instance of IP-log has a solution iff there exists a vector $y \in \mathbb{Z}^l$ such that $ABy < 0$. In other words, we have reduced the initial instance of IP-log to another instance with a trivial linear constraint.

► **Theorem 14.** *The strict homogenous IP-log problem is decidable, assuming Schanuel's conjecture.*

Proof. An instance of the strict homogenous IP-log problem asks to determine the truth of the sentence:

$$\exists x \in \mathbb{Z}^n : (c^\top x = 0 \pmod{2\pi}) \wedge Ax < 0.$$

As described above, we may assume without loss of generality that the equality constraint is trivial (i.e., $c = 0$). It thus suffices to determine whether the system of inequalities $Ax < 0$ admits a solution $x \in \mathbb{Z}^n$. But, by scaling, this system of inequalities has a solution in \mathbb{Z}^n iff it has a solution in \mathbb{Q}^n . In turn, by strictness of the constraints, there is a solution in \mathbb{Q}^n iff there is a solution in \mathbb{R}^n . We are thus left with the task of determining whether there exists $x \in \mathbb{R}^n$ such that $Ax < 0$. Here we can apply Fourier-Motzkin elimination [8] (that is to say, quantifier elimination in linear real arithmetic).

Recall that that Fourier-Motzkin elimination solves a system of linear inequalities by eliminating the variables sequentially until one obtains an equisatisfiable variable-free system of inequalities between constants. In the case at hand these constants will be rational expressions in a fixed number of logarithms of real algebraic numbers. As such, they are elementary numbers and so, using Richardson's algorithm (Proposition 2), we can determine which coefficients are zero. For a coefficient which is not zero, we need merely compute it to sufficient precision to determine its sign. ◀

The following example illustrates the main points of the argument above: Let $\omega = e^{2\pi i/3}$ be a primitive cube root of unity and let $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ be real positive algebraic numbers. Consider the following system of inequations and an equation:

$$\begin{aligned} x_1 \arg(\omega) + x_2 \arg(\omega^2) &= 0 \pmod{2\pi} \\ x_1 \log(\lambda_1) + x_2 \log(\lambda_2) &< 0 \\ x_1 \log(\lambda_3) + x_2 \log(\lambda_4) &< 0 \end{aligned}$$

Clearly the equation above is satisfied when $x_1 = 3y_1 + 2y_2$ and $x_2 = 3y_1 - y_2$ for some $y_1, y_2 \in \mathbb{Z}$. Thus we eliminate the equation and obtain

$$\begin{aligned} (3y_1 + 2y_2) \log \lambda_1 + (3y_1 - y_2) \log \lambda_2 &< 0 \\ (3y_1 + 2y_2) \log \lambda_3 + (3y_1 - y_2) \log \lambda_4 &< 0, \end{aligned}$$

which is equivalent to

$$\begin{aligned} y_1 \log(\lambda_1^3 \lambda_2^3) + y_2 \log(\lambda_1^2 / \lambda_2) &< 0 \\ y_1 \log(\lambda_3^3 \lambda_4^3) + y_2 \log(\lambda_3^2 / \lambda_4) &< 0. \end{aligned}$$

27:8 Nonnegativity Problems for Matrix Semigroups

Assuming $\lambda_1\lambda_2 > 1 > \lambda_3\lambda_4$ we can isolate y_1 by dividing out the coefficients (with known signs) to get the following lower and upper bound on y_1 :

$$y_1 < -y_2 \frac{\log(\lambda_1^2/\lambda_2)}{\log(\lambda_1^3\lambda_2^3)} \wedge y_1 > -y_2 \frac{\log(\lambda_3^2/\lambda_4)}{\log(\lambda_3^3\lambda_4^3)}.$$

Evidently the system above has a solution iff

$$-y_2 \frac{\log(\lambda_3^2/\lambda_4)}{\log(\lambda_3^3\lambda_4^3)} < -y_2 \frac{\log(\lambda_1^2/\lambda_2)}{\log(\lambda_1^3\lambda_2^3)},$$

i.e., iff

$$\frac{\log(\lambda_1^2/\lambda_2)}{\log(\lambda_1^3\lambda_2^3)} < \frac{\log(\lambda_3^2/\lambda_4)}{\log(\lambda_3^3\lambda_4^3)}.$$

The truth of the latter inequality can be checked by first using Richardson's algorithm to verify that the left-hand and right-hand expressions are unequal and then calculating to sufficient precision to determine which of the two is greater. (In this case existence of a solution was equivalent to the matrix A having non-zero determinant, but this will not hold for general systems with more constraints.)

4.4 Algorithm

In summary, we have the following algorithm for the positive membership problem in the commutative case:

INPUT: A set of commuting rational matrices $\{A_1, \dots, A_k\}$.

1. Find all joint eigenvalues $\lambda \in \Sigma$ for which both the corresponding right eigenspace V_λ and the left eigenspace W_λ are one-dimensional and contain a positive vector. This can be done, e.g., using a decision procedure for the theory of real-closed fields.
2. For each joint eigenvalue λ identified in Step 1, compute the corresponding IP-log problem as in Section 4.2 to see if λ is dominant. If the IP-log problem is satisfiable then output "YES" and halt.
3. Output "NO".

We conclude:

► **Theorem 15.** *The positive membership problem is decidable for commutative semigroups, assuming Schanuel's conjecture is true.*

5 The Non-negative Membership Problem for Commutative Semigroups

We now combine the ideas in Sections 3 and 4 to solve the problem of deciding whether a semigroup of commuting matrices contains a non-negative matrix. For ease of exposition we will assume that the matrices are simultaneously diagonalizable. The general commuting case involves more complicated algebra and is proven in Appendix A.

► **Problem 16** (Non-negative Matrix in Commutative Simultaneously Diagonalizable Semigroup). *Given a set of commuting simultaneously diagonalizable $d \times d$ matrices $\{A_1, \dots, A_k\}$ with rational entries, decide whether the semigroup generated by multiplying these matrices together contains a matrix with all its entries greater than or equal to zero.*

First, we refine our notion of dominated recurrences.

► **Definition 17** (Positively dominated by p). *Let $\mathbf{u} = (u_n)_{n=0}^\infty$ be a linear recurrence sequence which is non-degenerate and does not have any polynomial terms in its exponential-polynomial form. Then $\mathbf{u} = \sum_{l=1}^d c_l \lambda_l^n$ for complex numbers $\lambda_1, \dots, \lambda_d$ and coefficients c_l . We say \mathbf{u} is positively dominated by term p (here p (for positive) refers to the index) if*

1. $c_p > 0$,
2. $\lambda_p > 0$,
3. $\forall l \neq p, |\lambda_p| > |\lambda_l|$.

Call this predicate $PD_p(\mathbf{u})$.

Given a single matrix M , we consider the d^2 recurrences \mathbf{u}^{ij} defined by $(u^{ij})_n := e_i^\top M^n e_j$.

We have shown in Section 3 that M being eventually non-negative is equivalent to the decidable condition

$$\forall i \forall j (\mathbf{u}^{ij} \text{ is ultimately zero} \vee \exists p PD_p(\mathbf{u}^{ij})).$$

We now show a similar construction for multiple matrices. Let $A_1, \dots, A_k \in \mathbb{Q}^{d \times d}$ be a set of commuting simultaneously diagonalizable matrices. The idea is to search for an eventually non-negative matrix $A_1^{m_1} \dots A_k^{m_k}$.

Let \mathbf{m} denote the tuple (m_1, \dots, m_k) . Define the integer parameterized matrix entry recurrence $\mathbf{u}^{ij}(\mathbf{m})$ by $[\mathbf{u}^{ij}(\mathbf{m})]_n := e_i^\top [A_1^{m_1} \dots A_k^{m_k}]^n e_j$.

The existence of an eventually non-negative matrix (and thus, a non-negative matrix) in the semigroup is equivalent to the decidable condition

$$ENN := \exists \mathbf{m} \forall i \forall j (\mathbf{u}^{ij}(\mathbf{m}) \text{ is ultimately zero} \vee \exists p PD_p(\mathbf{u}^{ij}(\mathbf{m}))).$$

Let S be a matrix that simultaneously diagonalizes the matrices A_1, \dots, A_k such that $A_r = S^{-1} D_r S = S^{-1} \text{diag}(\lambda_{r1}, \dots, \lambda_{rd}) S$. The notation λ_{rl} denotes the l th eigenvalue of A_r (with multiplicity).

Then

$$\begin{aligned} u_n^{ij}(m_1, \dots, m_k) &:= e_i^\top [A_1^{m_1} \dots A_k^{m_k}]^n e_j \\ &= e_i^\top S^{-1} [D_1^{m_1} \dots D_k^{m_k}]^n S e_j \\ &= e_i^\top S^{-1} [\text{diag}(\lambda_{11}, \dots, \lambda_{1d})^{m_1} \dots \text{diag}(\lambda_{k1}, \dots, \lambda_{kd})^{m_k}]^n S e_j \\ &= e_i^\top S^{-1} \left[\text{diag} \left(\prod_{r=1}^k \lambda_{r1}^{m_r}, \dots, \prod_{r=1}^k \lambda_{rd}^{m_r} \right) \right]^n S e_j \\ &= \sum_{l=1}^d (S^{-1})_{il} (S)_{lj} \left[\prod_{r=1}^k \lambda_{rl}^{m_r} \right]^n. \end{aligned}$$

Now we have an exponential-polynomial representation for $\mathbf{u}^{ij}(\mathbf{m})$ with coefficients $(S^{-1})_{il} (S)_{lj}$ and roots $\prod_{r=1}^k \lambda_{r1}^{m_r}, \dots, \prod_{r=1}^k \lambda_{rd}^{m_r}$.

We see that $\mathbf{u}^{ij}(\mathbf{m})$ is positively dominated by p if

1. $(S^{-1})_{ip} (S)_{pj} > 0$,
2. $\prod_{r=1}^k \lambda_{rp}^{m_r} > 0$,
3. $\forall l \neq p$ such that $(S^{-1})_{il} (S)_{lj} \neq 0$, $\left| \prod_{r=1}^k \lambda_{rp}^{m_r} \right| > \left| \prod_{r=1}^k \lambda_{rl}^{m_r} \right|$.

27:10 Nonnegativity Problems for Matrix Semigroups

Let $c(p) = (\arg(\lambda_{1p}), \dots, \arg(\lambda_{kp}))$ and let $A(p)$ be the (at most) $(d-1) \times k$ -matrix defined by $a_{lr} = \log |\lambda_{rl}/\lambda_{rp}|$. Here l runs over elements of $\{1, \dots, d\}$ apart from p such that $(S^{-1})_{il}(S)_{lj} \neq 0$. Let \mathbf{m} be the vector $(m_1, \dots, m_k) \in \mathbb{N}^k$.

Now Condition 2 is equivalent to $c(p)^\top \mathbf{m} = 0 \pmod{2\pi}$ and Condition 3 is equivalent to $A(p)\mathbf{m} < 0$.

For simplicity we introduce the following new notation:

$$Z(i, j, \mathbf{m}) := \mathbf{u}^{ij}(\mathbf{m}) \text{ is identically zero}$$

$$S(i, j, p) := (S^{-1})_{ip}(S)_{pj} > 0$$

$$C(p, \mathbf{m}) := c(p)^\top \mathbf{m} = 0 \pmod{2\pi}$$

$$A(i, j, p, \mathbf{m}) := A(p)\mathbf{m} < 0$$

$$NND(i, j, p, \mathbf{m}) := Z(i, j, \mathbf{m}) \vee (S(i, j, p) \wedge C(p, \mathbf{m}) \wedge A(i, j, p, \mathbf{m})).$$

Note that $A(i, j, p, \mathbf{m})$ depends on i and j because we only care about the eigenvalue blocks where the coefficient is non-zero.

Writing out the predicate ENN in full with new notation, we have

$$\begin{aligned} ENN &:= \exists \mathbf{m} \forall i \forall j (\mathbf{u}^{ij}(\mathbf{m}) \text{ is identically zero} \vee \exists p PD_p(\mathbf{u}^{ij}(\mathbf{m}))) \\ &\equiv \exists \mathbf{m} \forall i \forall j (Z(i, j, \mathbf{m}) \vee \exists p (S(i, j, p) \wedge C(p, \mathbf{m}) \wedge A(i, j, p, \mathbf{m}))) \\ &\equiv \exists \mathbf{m} \forall i \forall j \exists p NND(i, j, p, \mathbf{m}) \end{aligned}$$

Now observe that the quantifications over i, j and p are finite and range over $\{1, \dots, d\}$. Thus we can replace the quantifications with finite disjunctions and conjunctions.

Our goal is to move all disjunctions outside the existential quantifier on \mathbf{m} so that we can use the IP-log algorithm from Section 4.3 on conjunctions of terms of the form $A\mathbf{m} < 0$.

Let f range over functions assigning a particular choice of p to each sequence $\mathbf{u}^{ij}(\mathbf{m})$. There are $d^{(d \times d)}$ such functions.

$$\begin{aligned} ENN &\equiv \exists \mathbf{m} \forall i \forall j (\exists p NND(i, j, p, \mathbf{m})) \\ &\equiv \exists \mathbf{m} \wedge_{ij} (\vee_p NND(i, j, p, \mathbf{m})) \\ &\equiv \exists \mathbf{m} \vee_f (\wedge_{ij} NND(i, j, p = f(i, j), \mathbf{m})) \\ &\equiv \vee_f (\exists \mathbf{m} \wedge_{ij} NND(i, j, p = f(i, j), \mathbf{m})). \end{aligned}$$

Essentially this means we non-deterministically choose a p for each sequence $\mathbf{u}^{ij}(\mathbf{m})$ and then check if there is an \mathbf{m} that works for all of them – that makes all the selected p 's into real positively dominating terms.

Analysing the elements of $NND(i, j, f(i, j), \mathbf{m})$, we see that $Z(i, j, \mathbf{m})$ and $S(i, j, f(i, j))$ constraints are trivially checkable. Constraints $\wedge_{ij} C(f(i, j), \mathbf{m})$ can be removed iteratively using Masser's theorem. The remaining conjunctions are terms of the form $\wedge_{ij} A(i, j, f(i, j), \mathbf{m})$, but since these are linear programs $A(i, j, p)\mathbf{m} < \mathbf{0}$ we can simply concatenate the various matrices $A(i, j, p)$ together. We can then use the IP-log algorithm from Section 4.3 to check if there exists an integer solution to the conjunction of these terms.

Of course in practice we would only need to check d different matrices of size $d-1 \times k$ since the eigenvalues are the same for all sequences.

Thus we have reduced the diagonalizable case to a finite disjunction of predicates, each of which can be reduced to strict homogenous IP-log. More generally, we have:

► **Theorem 18.** *The non-negative membership problem is decidable for commutative semigroups, assuming Schanuel's conjecture is true.*

Proof. See Appendix A. ◀

6 Undecidability in the Non-commutative Case

To complement the decidability results for commuting matrices in the preceding sections, in this section we show undecidability of the full version of the membership problem, in which commutativity is not assumed:

► **Problem 19** (Non-negative Membership). *Given a set of $d \times d$ matrices with rational entries, decide whether the generated semigroup contains a non-negative matrix.*

The proof of undecidability is by reduction from the threshold problem for probabilistic automata, which is well-known to be undecidable [11].

► **Problem 20** (Threshold Problem for Probabilistic Automata). *Given vectors u and v in \mathbb{Q}^d and a matrix semigroup \mathcal{S} generated by stochastic matrices $\{A_1, \dots, A_k\} \in \mathbb{Q}^{d \times d}$, decide whether there exists a matrix $A \in \mathcal{S}$ such that $u^\top Av \geq 1/2$.*

► **Theorem 21.** *The Non-negative Membership Problem is undecidable.*

Proof. Given non-negative integers m, n , write $0^{m \times n}$ for the zero matrix of dimension $m \times n$. Suppose that we are given an instance of the threshold problem for probabilistic automata, defined by vectors $u, v \in \mathbb{Q}^d$ and a matrix semigroup \mathcal{S} generated by stochastic matrices $A_1, \dots, A_k \in \mathbb{Q}^{d \times d}$. Now consider the semigroup \mathcal{S}' generated by the following matrices of dimension $(d + 2) \times (d + 2)$:

$$\begin{aligned}
 U &:= \begin{bmatrix} 1 & -1/2 & \mathbf{u}^\top \\ 0 & 0 & 0^{1 \times d} \\ 0^{d \times 1} & 0^{d \times 1} & 0^{d \times d} \end{bmatrix}, \\
 A'_i &:= \begin{bmatrix} 1 & -1/2 & 0^{1 \times d} \\ 0 & 0 & 0^{1 \times d} \\ 0^{d \times 1} & 0^{d \times 1} & A_i \end{bmatrix} \quad (i = 1, \dots, k), \\
 \text{and } V &:= \begin{bmatrix} 1 & -1/2 & 0^{1 \times d} \\ 0 & 0 & 0^{1 \times d} \\ 0^{d \times 1} & \mathbf{v} & 0^{d \times d} \end{bmatrix}.
 \end{aligned}$$

Note that matrix A'_i incorporates A_i for $i = 1, \dots, k$, while the matrices U and V respectively incorporate the initial and final vectors u and v of the probabilistic automaton.

We claim that the semigroup \mathcal{S}' contains a non-negative matrix if and only if there exists a matrix $A \in \mathcal{S}$ such that $u^\top Av \geq \frac{1}{2}$. To this end, consider a string of matrices chosen from the set $\{U, V\} \cup \{A'_1, \dots, A'_k\}$. Any product B of such a string that does not end with a suffix $UA'_{i_1} \cdots A'_{i_s} V$, for some $i_1, \dots, i_s \in \{1, \dots, k\}$, has $B_{1,2} = -\frac{1}{2}$ and hence cannot be a non-negative matrix. It remains to consider products B of strings that do have such a suffix. In this case we have

$$B_{1,2} = (UA'_{i_1} \cdots A'_{i_s} V)_{1,2} = u^\top A_{i_1} \cdots A_{i_s} v - \frac{1}{2},$$

and hence B is only non-negative if $u^\top A_{i_1} \cdots A_{i_s} v \geq \frac{1}{2}$. Since it further holds that $B_{1,1} = 1$ and $B_{i,j} = 0$ for all other entries (i, j) , we conclude that there exists a non-negative matrix in the semigroup \mathcal{S}' if and only if there exists a matrix $A \in \mathcal{S}$ such that $u^\top Av \geq \frac{1}{2}$. ◀

7 Further work

We leave open the question of quantitative refinements of our decidability results. These include giving complexity upper bounds for the Non-negative and Positive Membership Problems as well as the related question of giving upper bounds on the length of the shortest string of generators that yields a non-negative or positive matrix in a given semigroup. Both questions would seem to be difficult owing to the use of Schanuel's Conjecture in our proofs. Characterising the complexity of determining whether a matrix is eventually non-negative would seem to be more straightforward. We claim that the decision procedure can be implemented in non-deterministic polynomial time. Note that the analogous Eventual Positivity problem is in **PTIME** [18].

References

- 1 S. Akshay, S. Chakraborty, and D. Pal. On eventual non-negativity and positivity for the weighted sum of powers of matrices. In *Automated Reasoning - 11th International Joint Conference, IJCAR, 2022*. doi:10.1007/978-3-031-10769-6_39.
- 2 L. Babai, R. Beals, J.-Y. Cai, G. Ivanyos, and E. M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 498–507, 1996.
- 3 P. Bell, M. Hirvensalo, and I. Potapov. The identity problem for matrix semigroups in $SL(2, \mathbb{Z})$ is NP-complete. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 2017*.
- 4 J. Berstel. Sur les pôles et le quotient de hadamard de séries n-rationnelles. *C. R. Acad. Sci. Paris Sér. A-B*, 272:A1079-A1081, 1971.
- 5 Jean Berstel and Christophe Reutenauer. *Noncommutative rational series with applications*. Cambridge University Press, 2011.
- 6 J.-Y. Cai. Computing Jordan normal forms exactly for commuting matrices in polynomial time. *Int. J. Found. Comput. Sci.*, 5(3/4):293–302, 1994. doi:10.1142/S0129054194000165.
- 7 T. Colcombet, J. Ouaknine, P. Semukhin, and J. Worrell. On Reachability Problems for Low-Dimensional Matrix Semigroups. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:15, 2019.
- 8 George Dantzig. *Linear programming and extensions*. Princeton University Press, 1963.
- 9 R. Dong. On the identity problem for unitriangular matrices of dimension four. In *47th International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 241 of *LIPIcs*, pages 43:1–43:14, 2022.
- 10 R. Dong. The Identity Problem in $\mathbb{Z} \wr \mathbb{Z}$ Is Decidable. In *50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 124:1–124:20, 2023.
- 11 N. Fijalkow. Undecidability results for probabilistic automata. *ACM SIGLOG News*, 4(4):10–17, 2017.
- 12 S.-K. Ko, R. Niskanen, and I. Potapov. On the identity problem for the special linear group and the Heisenberg group. In *45th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 132:1–132:15, 2018.
- 13 S. Lang. Introduction to transcendental numbers. Addison-Wesley Series in Mathematics. Reading, Mass. etc.: Addison-Wesley Publishing Company. VI, 105 p. (1966)., 1966.
- 14 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In Piergiorgio Odifreddi, editor, *Kreiseliana. About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.
- 15 A. Markov. On certain insoluble problems concerning matrices. *Doklady Akad. Nauk SSSR*, 57(6):539–542, June 1947.

- 16 D. W. Masser. Linear relations on algebraic groups. In *New Advances in Transcendence Theory*. Camb. Univ. Press, 1988.
- 17 Carl D. Meyer. *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2023.
- 18 D. Noutsos. On perron–frobenius property of matrices having some negative entries. *Linear Algebra and its Applications*, 412(2-3):132–153, 2006.
- 19 J. Ouaknine, A. Pouly, J. Sousa-Pinto, and J. Worrell. Solvability of matrix-exponential equations. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 798–806, 2016.
- 20 J. Ouaknine and J. Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of SODA’14*. ACM-SIAM, 2014. doi:10.1137/1.9781611973402.27.
- 21 M. S. Paterson. Undecidability in 3 by 3 matrices. *J. of Math. and Physics*, 1970.
- 22 D. Richardson. How to recognize zero. *Journal of Symbolic Computation*, 24(6):627–645, 1997. doi:10.1006/jscs.1997.0157.

A Non-diagonalizable Case

► **Problem 22** (Non-negative Membership for Commutative Semigroup). *Given a set of commuting $d \times d$ matrices $\{A_1, \dots, A_k\}$ with rational entries, decide whether the semigroup generated by multiplying these matrices together contains a matrix with all its entries greater than or equal to zero.*

It is known that unfortunately simultaneous Jordanization of commuting matrices is not always possible [6]. However, a slightly weaker block diagonal form [19, Thm 12] is possible. Here we put the matrices into block diagonal form, where each block is of the form $\lambda_i I + N$ where N is strictly upper triangular and thus nilpotent.

Let $A_1, \dots, A_k \in \mathbb{Q}^{d \times d}$ be a set of commuting matrices.

As in Section 5, define the integer parameterized matrix entry recurrence $u^{ij}(\mathbf{m})$ by $u_n^{ij}(m_1, \dots, m_k) := e_i^\top [A_1^{m_1} \dots A_k^{m_k}]^n e_j$.

Let S be a matrix that simultaneously block-diagonalizes the matrices such that $A_r = S^{-1} B D_r S = S^{-1} \text{diag}(B_{r1}, \dots, B_{rb}) S$. Here $B_{rl} = \lambda_{rl} I + N_{rl}$ denotes the l th block of A_r , where N_{rl} is strictly upper triangular. Note that for fixed l the various N_{rl} inherit commutativity from the original matrices. Then we have that

$$\begin{aligned} u_n^{ij}(m_1, \dots, m_k) &:= e_i^\top [A_1^{m_1} \dots A_k^{m_k}]^n e_j \\ &= e_i^\top S^{-1} [B D_1^{m_1} \dots B D_k^{m_k}]^n S e_j \\ &= e_i^\top S^{-1} [\text{diag}(B_{11}, \dots, B_{1b})^{m_1} \dots \text{diag}(B_{k1}, \dots, B_{kb})^{m_k}]^n S e_j \\ &= e_i^\top S^{-1} \left[\text{diag} \left(\prod_{r=1}^k B_{r1}^{m_r}, \dots, \prod_{r=1}^k B_{rb}^{m_r} \right) \right]^n S e_j \\ &= [s_{i1}^{-1}, \dots, s_{id}^{-1}] \text{diag} \left(\left[\prod_{r=1}^k B_{r1}^{m_r} \right]^n, \dots, \left[\prod_{r=1}^k B_{rb}^{m_r} \right]^n \right) [s_{1j}, \dots, s_{dj}]^\top. \end{aligned}$$

Let us examine the structure of the submatrix $\left[\prod_{r=1}^k B_{rb}^{m_r} \right]^n$. Recall that $B_{rl} = \lambda_{rl} I + N_{rl}$. Note that any power or product of powers of the nilpotents can be non-zero only upto total degree at most d . We adopt the convention that a zero power indicates the identity matrix of appropriate size. For ease of notation we drop the block subscript and expand out

27:14 Nonnegativity Problems for Matrix Semigroups

$$\begin{aligned}
\left[\prod_{r=1}^k B_{rb}^{m_r} \right]^n &= \left[\prod_{r=1}^k (\lambda_r I + N_r)^{m_r} \right]^n \\
&= \prod_{r=1}^k \lambda_r^{nm_r} (I + N_r/\lambda_r)^{nm_r} \\
&= \prod_{r=1}^k \lambda_r^{nm_r} \cdot \prod_{r=1}^k \left(\sum_{i_r=0}^d \binom{nm_r}{i_r} (N_r/\lambda_r)^{i_r} \right) \\
&= \prod_{r=1}^k \lambda_r^{nm_r} \cdot \left[\sum_{(i_1, \dots, i_k) = (0, \dots, 0)}^{i_1 + \dots + i_k = d} \left(\prod_{r=1}^k \binom{nm_r}{i_r} (N_r/\lambda_r)^{i_r} \right) \right] \\
&= \text{MPoly}(n\mathbf{m}) \prod_{r=1}^k \lambda_r^{nm_r}.
\end{aligned}$$

Here $\text{MPoly}(n\mathbf{m})$ is shorthand for the block-matrix with entries which are polynomial of degree at most d in the variables (m_1, \dots, m_k) (scaled by n) with coefficients arising from the nilpotent submatrices.

Substituting this back into our original expression for $u_n^{ij}(m_1, \dots, m_k)$ we get

$$\begin{aligned}
u_n^{ij}(m_1, \dots, m_k) &:= e_i^\top [A_1^{m_1} \dots A_k^{m_k}]^n e_j \\
&= [s_{i1}^{-1}, \dots, s_{id}^{-1}] \text{diag} \left(\left[\prod_{r=1}^k B_{r1}^{m_r} \right]^n, \dots, \left[\prod_{r=1}^k B_{rb}^{m_r} \right]^n \right) [s_{1j}, \dots, s_{dj}]^\top \\
&= [s_{i1}^{-1}, \dots, s_{id}^{-1}] \text{diag} \left(\text{MPoly}_1(n\mathbf{m}) \prod_{r=1}^k \lambda_{r1}^{nm_r}, \dots, \text{MPoly}_b(n\mathbf{m}_r) \prod_{r=1}^k \lambda_{rd}^{nm_r} \right) [s_{1j}, \dots, s_{dj}]^\top \\
&= \sum_{l=1}^d \left(\text{poly}_l^{ij}(n\mathbf{m}) \prod_{r=1}^k \lambda_{rl}^{nm_r} \right) \text{ (after folding in constants)}.
\end{aligned}$$

Notice that the asymptotic top term in n in the polynomial $\text{poly}_l^{ij}(n\mathbf{m})$ is the homogenous subpolynomial of highest degree in (m_1, \dots, m_k) - call it $h_l^{ij}(\mathbf{m})$. Once we pick a particular p to be our positively dominating term, we only need the following three conditions for the recurrence to be positively dominated by p :

1. $h_p^{ij}(\mathbf{m}) > 0$,
2. $\prod_{r=1}^k \lambda_{rp}^{m_r} > 0$,
3. $\forall l \neq p$ such that $\text{poly}_l^{ij}(n\mathbf{m})$ is not the zero polynomial, $\left| \prod_{r=1}^k \lambda_{rp}^{m_r} \right| > \left| \prod_{r=1}^k \lambda_{rl}^{m_r} \right|$.

Via the same algebraic manipulations as in the diagonalizable case, checking

$$ENN := \exists m \forall i \forall j (\mathbf{u}^{ij}(\mathbf{m}) \text{ is identically zero} \vee \exists p \text{ PD}_p(\mathbf{u}^{ij}(\mathbf{m}))),$$

reduces to solving conjunctions of the form

$$\wedge_{ij} (h_l^{ij}(\mathbf{m}) > 0 \wedge \mathbf{c}(p)^\top \mathbf{m} = 0 \bmod 2\pi \wedge \mathbf{A}(p)\mathbf{m} < 0)$$

over the integers.

We can eliminate the second conjunct using Masser's theorem. Now suppose there exists a real solution on the unit sphere to $h_l^{ij}(\mathbf{m}) > 0 \wedge \mathbf{A}(p)\mathbf{m} < 0$. By openness, there exists a rational solution nearby. Since both these conjuncts are homogenous, \mathbf{m} is a solution iff $n\mathbf{m}$ is a solution, for all real $n > 0$. Thus we may clear denominators from the rational solution to obtain an integer solution \mathbf{m} to ENN .

The sentence

$$\exists \mathbf{m} \in \mathbb{R}^k : h_i^{ij}(\mathbf{m}) > 0 \wedge \mathbf{A}(p)\mathbf{m} < 0$$

can be written in the first order theory of the reals with exponentiation, which is decidable assuming Schanuel’s conjecture as shown by Wilkie and Macintyre [14].

We now need to prove that failing to find such a real solution implies that the semigroup does not contain a non-negative matrix. Suppose that there exists some $m = (m_1, \dots, m_k)$ such that $A_1^{m_1} \dots A_k^{m_k}$ is non-negative. Then $(A_1^{m_1} \dots A_k^{m_k})^n$ is non-negative for all n . By Proposition 6, each individual recurrence is ultimately zero or must have a strictly dominant top term. Thus m satisfies $Am < 0$ and $c^\top m = 0 \pmod{2\pi}$ for the appropriate A and c . The top term (as a function of n) has a polynomial coefficient which is the homogenous polynomial we identified above. Thus the homogenous polynomial is non-negative for m , completing the requirements necessary for the sentence $\exists \mathbf{m} \in \mathbb{R}^k : h_i^{ij}(\mathbf{m}) > 0 \wedge \mathbf{A}(p)\mathbf{m} < 0$ to be true.

We conclude:

► **Theorem 23.** *The non-negative membership problem is decidable for commutative semi-groups, assuming Schanuel’s conjecture is true.*

B Positive Rational Sequences are Dominated

We need the following classical results from Perron-Frobenius theory (see, e.g., [17, Chap. 8]).

► **Theorem 24.**

1. If $A \geq 0$ is irreducible then it has cyclic peripheral spectrum, i.e., its eigenvalues of maximum modulus have the form $\{\rho, \rho\omega, \dots, \rho\omega^{k-1}\}$, where $\rho > 0$, k is a positive integer, and ω is a primitive k -th root of unity.
2. If a non-negative irreducible matrix A has only one eigenvalue on the spectral circle it is called a primitive matrix. If A is primitive, the pointwise limit $\lim_{n \rightarrow \infty} (A/\rho(A))^n$ exists and is a strictly positive matrix.

We now prove Berstel’s result for matrix entry recurrences.

► **Proposition 25.** *Let $M \in \mathbb{Q}^{d \times d}$ be a non-negative non-degenerate matrix. Then for all $i, j \in \{1, \dots, d\}$ the LRS $u_n = (M^n)_{i,j}$ is dominated.*

Proof. Since $M \geq 0$ there exists a permutation matrix P such that M can be written in the form $M = PUP^{-1}$, where $U \geq 0$ is block upper triangular. It follows that there exist $i', j' \in \{1, \dots, d\}$ such that

$$(M^n)_{i,j} = (PU^n P^{-1})_{i,j} = (U^n)_{i',j'}$$

for all $n \in \mathbb{N}$. Now write

$$U = \begin{pmatrix} B_{1,1} & B_{1,2} & \dots & B_{1,e} \\ 0 & B_{2,2} & \dots & B_{2,e} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & B_{e,e} \end{pmatrix},$$

where all the blocks in U are non-negative and the diagonal blocks $B_{1,1}, B_{2,2}, \dots, B_{e,e}$ are irreducible. Then

$$(U^n)_{i',j'} = \sum_{\substack{l_1 < l_2 < \dots < l_m \\ n_1 + n_2 + \dots + n_m = n - (m-1)}} e_{i'}^\top B_{l_1, l_1}^{n_1} B_{l_1, l_2} B_{l_2, l_2}^{n_2} \dots B_{l_{m-1}, l_m} B_{l_m, l_m}^{n_m} e_{j'}, \quad (1)$$

27:16 Nonnegativity Problems for Matrix Semigroups

where the sum runs over all positive integers m and strictly increasing sequences of block indices $l_1 < \dots < l_m$.

Consider a single block $B_{l,l}$ along the diagonal. Since it is irreducible and non-negative, it has cyclic peripheral spectrum. By our assumption of non-degeneracy, $r_l := \rho(B_{l,l}) \geq 0$ is the only eigenvalue on the spectral circle. Thus $B_{l,l}$ is primitive and by our second Perron-Frobenius result above, asymptotically $B_{l,l}^n/r_l^n \sim C_l$ where C_l is a positive matrix and the asymptotic equivalence relation \sim applies entry-wise. Let r_{max} be the maximum spectral radius of a block $B_{l,l}$ lying on a path from i' to j' .

We now analyse the asymptotic behavior of the normalized recurrence $(U^n)_{i',j'}/r_{max}^n$. Consider a summand S_n in $(U^n)_{i',j'}/r_{max}^n$. Replacing the diagonal blocks with their asymptotic limits,

$$S_n \sim \left(\frac{r_{l_1}}{r_{max}}\right)^{n_1} \left(\frac{r_{l_2}}{r_{max}}\right)^{n_2} \dots \left(\frac{r_{l_m}}{r_{max}}\right)^{n_m} e_{i'}^\top C_{l_1} B_{l_1,l_2} C_{l_2} \dots B_{l_{m-1},l_m} C_{l_m} e_{j'}.$$

Although the number of terms in the sum grows polynomially in n , we see that each term with some $r_{l_k} < r_{max}$ that does not have n_k constant tends to zero exponentially quickly. The remaining summands in $(U^n)_{i',j'}/r_{max}^n$ are thus those where n_k is non-constant only for blocks with $\rho(B_{l_k,l_k}) = r_{max}$. Let K be the sum of the constant powers for non-maximal blocks in such a summand Q_n , and P be the product of the powers of non-maximal r_{l_k} . Then

$$Q_n \sim r_{max}^n \cdot (P/r_{max}^{K+m-1}) \cdot e_{i'}^\top C_{l_1} B_{l_1,l_2} C_{l_2} \dots B_{l_{m-1},l_m} C_{l_m} e_{j'}.$$

Observe that the coefficient of r_{max}^n is a product of spectral radii and non-negative matrices, and is thus non-negative. This implies different such terms containing r_{max}^n cannot cancel out. So $e_{i'}^\top U^n e_{j'} \sim A(n) \cdot r_{max}^n$ for some polynomial A with positive coefficients depending on the non-diagonal blocks and the spectral radii of the diagonal blocks. Thus $(M^n)_{i,j}$ is either dominated by r_{max} or ultimately zero (the latter in case $r_{max} = 0$ or the sum in (1) is empty). \blacktriangleleft

One n Remains to Settle the Tree Conjecture

Jack Dippel   

McGill University, Montreal, Canada

Adrian Vetta   

McGill University, Montreal, Canada

Abstract

In the famous *network creation game* of Fabrikant et al. [11] a set of agents play a game to build a connected graph. The n agents form the vertex set V of the graph and each vertex $v \in V$ buys a set E_v of edges inducing a graph $G = (V, \bigcup_{v \in V} E_v)$. The private objective of each vertex is to minimize the sum of its building cost (the cost of the edges it buys) plus its connection cost (the total distance from itself to every other vertex). Given a cost of α for each individual edge, a long-standing conjecture, called the *tree conjecture*, states that if $\alpha > n$ then every Nash equilibrium graph in the game is a spanning tree. After a plethora of work, it is known that the conjecture holds for any $\alpha > 3n - 3$. In this paper we prove the tree conjecture holds for $\alpha > 2n$. This reduces by half the open range for α with only $(n - 3, 2n)$ remaining in order to settle the conjecture.

2012 ACM Subject Classification Theory of computation \rightarrow Algorithmic game theory; Theory of computation \rightarrow Social networks

Keywords and phrases Algorithmic Game Theory, Network Creation Games, Tree Conjecture

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.28

Related Version *Previous Version*: <https://arxiv.org/abs/2310.08663>

1 Introduction

A foundational motivation for the field of algorithmic game theory was to understand the evolution and functionality of networks, specifically, the Internet; see Papadimitriou [15]. Of particular fascination concerned how the actions of self-motivated agents affected the structure of the World Wide Web and social networks more generally. An early attempt to study this conundrum was undertaken by Fabrikant et al. [11] with their now classical *network creation game*. Despite its extreme simplicity, their model (detailed below) has become highly influential for two reasons. First, it inspired the development of a wide range of network formation models, [2],[5]. Second, it has lead to one of the longest-standing open problems in algorithmic game theory, namely, the tree conjecture. The latter motivates this research. So let us begin by describing the model and the conjecture.

1.1 The Network Creation Game

Consider a set of vertices, $V = \{1, 2, \dots, n\}$, who attempt to construct a connected graph between themselves. To do this, each vertex (agent) can purchase individual edges for a fixed cost of α each. Consequently, a strategy for vertex $v \in V$ is a set of (incident) edges E_v . Together the strategies of the agents forms a graph $G = (V, E)$, where $E = E_1 \cup E_2 \cup \dots \cup E_n$. In the *network creation game*, the objective of each vertex is to minimize the sum of its building and its connection cost. The *building cost* for vertex v is $\alpha \cdot |E_v|$, the cost of all the edges it buys. The *connection cost* is $D(v) = \sum_{u:u \neq v} d_G(u, v)$, the sum of the distances in G of v to every other vertex, where $d_G(u, v) = \infty$ if there is no path between u and v . That is, the total cost to the vertex is $c_v(E) = \alpha \cdot |E_v| + D(v)$.



© Jack Dippel and Adrian Vetta;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 28; pp. 28:1–28:16



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Given the different objectives of the agents, we study Nash Equilibria (NE) in the network creation game. A Nash equilibrium graph is a graph $G = (V, E)$ in which no vertex v can reduce its total cost by changing its strategy, that is, by altering the set of edges it personally buys, given the strategies of the other vertices remain fixed. Thus E_v is a best response to $(E_u)_{u \neq v}$, for every vertex v . Observe that every Nash equilibrium graph must be a connected graph. Attention in the literature has focused on whether or not every Nash equilibrium graph is minimal, that is, a spanning tree.

1.2 The Tree Conjecture

The network creation game was designed by Fabrikant et al. [11]. They proved that any network equilibrium graph which forms a tree costs at most 5 times that of a star, i.e. the optimal network. They proposed that for α greater than some constant, every Nash equilibrium graph is a spanning tree. This was the original *tree conjecture* for network creation games.

In the subsequent twenty years, incremental progress has been made in determining the exact range of α for which all Nash equilibria are trees. Albers et al. [1] demonstrated that the conjecture holds for $\alpha \geq 12n \log n$. However, they also provided a counterexample to the original tree conjecture. Moreover, Mamageishvili et al. [13] proved the conjecture is false for $\alpha \leq n - 3$.

As a result, a revised conjecture took on the mantle of the *tree conjecture*, namely that every Nash equilibrium is a tree if $\alpha > n$. Mihalák and Schlegel [14] were the first to show that this tree conjecture holds for $\alpha \geq cn$ for a large enough constant c , specifically, $c > 273$. Since then, the constant has been improved repeatedly, by Mamageishvili et al. [13], then Álvarez and Messegué [3], followed by Bilò and Lenzner [6], and finally by Dippel and Vetta [10] who proved the result for $\alpha > 3n - 3$.

We remark that extensions and variations of the network creation game have also been studied; we refer the interested reader to [4, 7, 8, 9, 16, 17].

1.3 Our Contribution

In this paper we improve the range in which the tree conjecture is known to hold from $\alpha > 3n - 3$ [10] to $\alpha > 2n$:

► **Theorem 1.** *If G is a Nash equilibrium graph for the network creation game (n, α) and $\alpha > 2n$, then G is a tree.*

Our high-level strategy to prove Theorem 1 is straightforward: we assume the existence of a biconnected component H in a Nash equilibrium and then prove, via consideration of a collection of strategy deviations, that some vertex has a better strategy, provided $\alpha > 2n$. This contradicts the best response conditions and proves that every Nash equilibrium graph is a spanning tree.

Our approach applies some prior methodologies and combines them with some original tools. One important technique we exploit is that of *min-cycles*, introduced by Lenzner [12]. A min-cycle is a cycle in a graph with the property that, for every pair of vertices in the cycle, the cycle contains a shortest path between the pair. For certain values of α , currently $\alpha > 2n - 3$, min-cycles are known to have the nice property that each vertex in the cycle buys exactly one edge of the cycle. This property has been leveraged in various ways [3, 6, 10] to show that no min-cycles can exist in a Nash equilibrium graph for large

enough α . Importantly, the smallest cycle in any graph must be a min-cycle. Consequently, for large enough α , there can be no smallest cycle and, thus, no cycle at all. This implies that, for such an α , all Nash equilibrium graphs are trees.

The techniques we develop concern the analysis of the presupposed biconnected component H in the Nash equilibrium. The existence of H implies the existence of a special vertex, called r , which has the lowest connection cost amongst all vertices in H . That is, $D(r) \leq D(v)$ for all $v \in H$. Given r , we take a shortest path tree T in G rooted at r . The key is to exploit the structural properties inherent in T . To derive these properties, we design a new class of strategy deviations available to vertices in H . Because these deviations must be non-improving responses, they impose numerous beneficial restrictions on the edges in H . In particular, we can then show that the sum of degrees of vertices within H exceeds 2 times the number of edges within H . This absurdity proves the Nash equilibrium graph is a tree.

1.4 Overview of Paper

This paper has three main sections. Section 2 consists of preliminaries and contains three things. First, we examine the structure of Nash equilibrium graphs which contain a hypothesized biconnected component H . Second, we discuss min-cycles, which appear in several previous papers, each of which contains a useful lemma that we take advantage of in this paper. Third, we introduce some new lemmas pertaining to T , the shortest path tree mentioned above.

Section 3 has two parts. The first presents a set of three deviation strategies. That is, three specific ways a vertex in a biconnected component might consider changing which edges it buys in order to decrease its personal cost. For a Nash equilibrium graph these alternate strategies cannot reduce the personal cost, and this allows us to derive deviation bounds for that vertex and the edges it buys. The second part uses these deviation bounds to prove claims about the structure of the graph. In particular, we show that vertices which buy edges outside T cannot be too close to one another.

Finally, Section 4 presents two main lemmas. The first shows that, in a set U of three vertices adjacent in T , all at different depths, where the lowest buys an edge outside T , only one vertex in U can have degree two in H . The second lemma bounds the number of times such sets can intersect. These two lemmas are the core of the proof of Theorem 1. After proving the main result, the paper culminates with concluding remarks on the future of the tree conjecture.

2 Preliminaries

Recall our basic approach is to prove by contradiction the non-existence of a biconnected component in a Nash equilibrium graph. Accordingly, we begin in Section 2.1 by introducing the notation necessary to analyse biconnected components in network creation game graphs. To disprove the existence of a biconnected component it suffices to disprove the existence of a cycle. Of particular importance here is the concept of a min-cycle. So in Section 2.2 we present a review of min-cycles along with a corresponding set of fundamental lemmas which appear in [12, 6, 13]. Finally, in Section 2.3, we prove a collection of technical results concerning the shortest path tree T that we will utilize throughout the rest of the paper. Lemma 9 is an especially useful tool for this method, and could prove valuable in future work on this problem.

2.1 Biconnected Components

Given a subgraph W of a graph G , we let $d_W(v, u)$ denote the distance between v and u in W . In particular, $d(u, v) = d_G(u, v)$ is the distance from v to u in the whole graph. Let $D(v) = \sum_{u:u \neq v} d_G(u, v)$ be the *connection cost* for vertex v , the sum of the distances from v to every other vertex.

In a Nash equilibrium graph G containing biconnected components, we will refer to the largest biconnected component as H . For a vertex $v \in H$, we denote by $C(v)$ a smallest cycle containing v , breaking ties arbitrarily. Let $r \in H$ be a vertex whose connection cost is smallest amongst all the vertices in H . Once built, an edge uv of G can be traversed in either direction. Thus G is an undirected graph. However, it will often be useful to view G as a directed graph. Specifically, we may orient uv is from u to v if the edge was bought by u and orient it from v to u if it was bought by v .

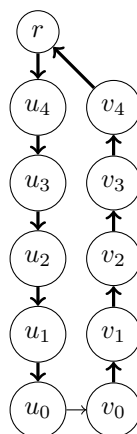
Given r , we will make heavy use of the shortest path tree T rooted at r . There may be multiple options for the choice of T . So we insist our choice of T has the following property: For all shortest paths P between r and any vertex u which is directed from r to u , we have $P \subseteq T$. The proof of Lemma 3 shows that we will never have two shortest paths directed from r to the same vertex v , so this choice of T is well-defined. Next, given an edge uv where u is the parent of v in T , we say that uv is a *down-edge* of T if the edge was bought by u ; otherwise we say it is an *up-edge* of T . We denote by T^\downarrow the set of down-edges and by T^\uparrow the set of up-edges of T . Let $T(v)$ be the subtree rooted at v in the tree T rooted at r . If $uv \in T^\downarrow$ then we define $T(uv) = T(v)$. Otherwise, if $uv \notin T^\downarrow$ then we define $T(uv) = \emptyset$.

Finally, there remain two types of sets we need to define. The first type is the “ S -sets”, which are sets of vertices. In a graph G , with a subgraph W , $S_W(v)$ is the set of vertices w with the following property: a shortest path in G from w to W contains v . Therefore $S_W(v)$ will always contain v , and it will contain no vertices of $W \setminus v$. W can even be a single vertex u , i.e. $S_u(v)$ is the set of all vertices in G with a shortest path to u containing v . The second type is the “ X -sets”, which are sets of edges. For largest biconnected component H of G and the shortest path tree T , the set X_0 is the set of all edges in $H \setminus T$, or *out-edges*. For all integers $i \geq 1$, X_i contains the set X_{i-1} as well as all edges $uv \in H \cap T^\downarrow$ bought by the parent u and where the child v buys an edge of X_{i-1} . The X_i sets can be thought of as the set of all out-edges and all down-edges in directed paths of length $\leq i$ to a vertex which buys an out-edge. Sometimes it will be useful to include in the X -sets all the edges $vu \in H \cap T^\uparrow$, bought by child v to parent u . To do this, we add superscript $+$ to the X set. For instance $X_0^+ = X_0 \cup T^\uparrow$ is the set of all up-edges and out-edges, which happens to be the set of all edges e with $T(e) = \emptyset$. Similarly, $X_i^+ = X_i \cup T^\uparrow$.

2.2 Min-Cycles

Recall, a *min-cycle* is a cycle in a graph with the property that, for every pair of vertices in the cycle, the cycle contains a shortest path between the pair. This concept, introduced by Lenzner [12], has proved very fruitful in the study of Nash equilibria in network creation games. In particular, we now present three useful lemmas concerning min-cycles that appear in [12, 6, 13], respectively. For completeness and because some of the ideas used are informative, we include short proofs of these three lemmas; similar proofs appeared in [10]

► **Lemma 2** ([12]). *The smallest cycle C containing an edge e is a min-cycle.*



■ **Figure 1** In the diagram, edges of T are bold. Vertex u_0 buys an edge $u_0v_0 \in X_j$ for all $j \geq 0$. Similarly, each vertex u_i buys edge $u_iu_{i-1} \in X_j$ for all $4 \geq i \geq 1, j \geq i$. Each vertex v_i buys $v_iv_{i+1} \in X_j^+$ for all $3 \geq i \geq 0, j \geq 0$. v_4 buys $v_4r \in X_j^+$ and r buys $ru_4 \in X_k$ for $j \geq 0, k \geq 5$.

Proof. Consider the smallest cycle C containing an edge e . Suppose for the sake of contradiction that there are two vertices $u, v \in C$ such $d_G(u, v) < d_C(u, v)$. Without loss of generality, suppose the shortest path between u and v , labelled P , lies entirely outside C . Note that C contains two paths between u and v . Let Q be the path of C from u to v that contains e . Then $P \cup Q$ is a cycle containing e that is strictly smaller than C , a contradiction. ◀

For an edge e or vertex u in a biconnected component H , we may write $C(e), C(u)$ to denote the smallest cycle containing edge e , vertex u respectively, breaking ties arbitrarily. A nice property of min-cycles is that every vertex in the min-cycle buys exactly one edge of the cycle.

► **Lemma 3** ([6]). *Let $\alpha > 2(n - 1)$. Every min-cycle in a Nash equilibrium graph G is directed.*

Proof. Let C be a min-cycle that is *not* directed. Then there is a vertex v that buys two edges of the cycle, say vx and vy . Let $uw \in C$ be an edge furthest from v (if $|C|$ is odd, there is a unique choice, otherwise choose one of two furthest edges). Let u buy uw . We've chosen uw so that both u and w have a shortest path to v which does not contain uw , therefore every vertex in G has a shortest path to v without uw . Without loss of generality, let $d(u, y) < d(u, v)$. Then because of how we chose uw , both v and x have a shortest path to u which does not contain vx , therefore every vertex in G has a shortest path to u without vx .

First, consider that if u sells uw and buys uv , no vertices become farther from v . It follows, by the Nash equilibrium conditions, that

$$D(u) \leq D(v) + n - 1 \tag{1}$$

because $D(u)$ must be less than the cost to swap uw for uv and use the edge uv in followed by the shortest path from v to each of the $n - 1$ vertices.

On the other hand, v can sell *both* vx and vy and instead buy the edge vu without increasing the distance from u to any other vertex. All of the vertices which used vy in their shortest paths to u become closer to u . It follows, by the Nash equilibrium conditions, that

$$D(v) \leq D(u) + n - 1 - \alpha \tag{2}$$

Together (1) and (2) give $D(v) \leq D(v) + 2(n-1) - \alpha < D(v)$. This contradiction implies that C must be a directed cycle. ◀

Furthermore, we may now obtain a lower bound on the girth of a Nash equilibrium graph in terms of the cost α .

► **Lemma 4** ([13]). *Any cycle C in a Nash equilibrium graph has $|C| \geq \frac{2\alpha}{n} + 2$.*

Proof. Take a minimum length cycle $C = v_0, v_1, \dots, v_{k-1}, v_k = v_0$ in G . First assume $e_i = v_i v_{i+1}$ is bought by v_i , for each $0 \leq i \leq k-1$. Now, for each vertex $u \in V$, we define a set $L_u \subseteq \{0, 1, \dots, k-1\}$ as follows. We have $i \in L_u$ if and only if every shortest path from $v_i \in C$ to u uses the edge e_i .

We claim $|L_u| \leq \frac{|C|-1}{2}$ for every vertex u . If not, take a vertex u with $|L_u| > \frac{|C|-1}{2}$. Let $d(v_i, u)$ be the shortest distance between u and $v_i \in C$. Next give v_i a label $\ell_i = d(v_i, u) - d(v_{i+1}, u)$. Observe that $\ell_i \in \{-1, 0, 1\}$. Furthermore, the labels sum to zero as

$$\sum_{i=0}^{|C|-1} \ell_i = \sum_{i=0}^{|C|-1} (d(v_i, u) - d(v_{i+1}, u)) = \sum_{i=0}^{|C|-1} d(v_i, u) - \sum_{i=1}^{|C|} d(v_i, u) = 0$$

Now take a vertex v_i in C that uses e_i in every shortest path to u ; that is, $i \in L_u$. Then $\ell_i = 1$ and $\ell_{i-1} \geq 0$. In particular, if $|L_u| > \frac{|C|-1}{2}$ then there are $> \frac{|C|-1}{2}$ positive labels and $> 1 + \frac{|C|-1}{2}$ non-negative labels. Hence, there are $< |C| - \left(1 + \frac{|C|-1}{2}\right) = \frac{|C|-1}{2}$ negative labels. But then the sum of the labels is strictly positive, a contradiction.

Now, as $|L_u| \leq \frac{|C|-1}{2}$ for every vertex u , there must exist a v_i, e_i pair where v_i needs e_i for its shortest paths to $\leq \frac{n}{2}$ vertices. For the vertices that do require e_i in their shortest paths, deleting e_i increases their distance by $\leq |C| - 2$, as we can replace e_i with $C \setminus e_i$ in those paths. Therefore, if v_i sells e_i its cost increases by $\leq (|C| - 2) \cdot \frac{n}{2} - \alpha$. This must be non-negative by the Nash equilibrium conditions. Rearranging, we have $\frac{2\alpha}{n} + 2 \leq |C|$.

Now consider the other case, where some vertex v_i buys $e_1 = v_i v_{i+1}$ and $e_2 = v_i v_{i-1}$. Without loss of generality, v_i needs e_1 for its shortest paths to $\leq \frac{n}{2}$ vertices. Therefore, just as in the previous case, if v_i sells e_1 its cost increases by $\leq (|C| - 2) \cdot \frac{n}{2} - \alpha$. This must be non-negative by the Nash equilibrium conditions. Again, rearranging, we have $\frac{2\alpha}{n} + 2 \leq |C|$. ◀

An important immediate consequence of Lemma 4 is that we may assume that the girth of any Nash equilibrium graph of interest here is at least 7.

► **Corollary 5.** *Let $\alpha > 2n$. All cycles in a Nash equilibrium graph have length at least 7.*

2.3 The Shortest Path Tree T

Let us now consider the shortest path tree T rooted at the vertex r with smallest connection cost amongst the vertices in the biconnected component H . In this section, we present some technical lemmas that give useful insights into the structure of this tree T . We begin by upper bounding the size of any subtree in T .

► **Lemma 6.** *If $v \neq r$ then $|T(v)| \leq \frac{n}{2}$.*

Proof. Recall T is the shortest path tree of the vertex r with the smallest connection cost and $T(v)$ is the subtree of T rooted at v . For any vertex $x \in T(v)$, it immediately follows that $d(v, x) = d(r, x) - d(v, r)$. On the other hand, for any vertex $x \notin T(v)$, the triangle inequality implies that $d(v, x) \leq d(r, x) + d(v, r)$. Thus

$$\begin{aligned} D(v) &\leq D(r) - d(v, r) \cdot |T(v)| + d(v, r) \cdot (n - |T(v)|) \\ &= D(r) + d(v, r) \cdot (n - 2 \cdot |T(v)|) \end{aligned}$$

But $D(r) \leq D(v)$. Therefore $d(v, r) \cdot (n - 2 \cdot |T(v)|) \geq 0$ and rearranging gives $|T(v)| \leq \frac{n}{2}$. ◀

The next two lemmas concern the properties of paths related to any vertex u that buys an edge $uv \in X_i$ where $i \leq 2$.

► **Lemma 7.** *Let $\alpha > 2n$. If u buys $uv \in X_i$, for some $i \leq 2$, then all $w \in T(uv)$ have a path to r of length $\leq d(w, r) + 2i$ which does not contain u .*

Proof. If $uv \in X_0$ then, as $uv \notin T^\perp$, we have $|T(uv)| = \emptyset$. Thus the claim is trivially true.

So assume $uv \notin X_0$. Then uv is the first edge in a path of length i in T from u to a vertex x which buys $xy \in H \setminus T$. Thus, there is a path P_0 in T of length $\ell(P_0) = i - 1$ from v to x . Now let P_1 be the path from y to r in T . Observe that P_1 does not contain u ; otherwise xy and T would define a cycle of length ≤ 6 (because $i \leq 2$), contradicting Corollary 5. Note also that P_1 has length $\ell(P_1) \leq d(r, v) + i$ because T is a shortest path tree.

Next, let P_2 be the path from w to v . Note P_2 has length $d(w, r) - d(v, r)$ and also does not contain u . Consequently P_2, P_0, xy, P_1 is a path from w to r . This path has length at most

$$\begin{aligned} \ell(P_2) + \ell(P_0) + 1 + \ell(P_1) &\leq (d(w, r) - d(v, r)) + (i - 1) + 1 + (d(v, r) + i) \\ &= d(w, r) + 2i \end{aligned}$$

Furthermore this path does not contain u . ◀

► **Lemma 8.** *Let $\alpha > 2n$. If u buys $uv \in X_i$, for some $i \leq 2$, then $d(u, r) \geq \lfloor \frac{|C(u)|}{2} \rfloor - i$.*

Proof. So $uv \in X_i$ is associated with an edge $xy \in X_0$, where $xy = uv$ if $uv \in X_0$. Let C be the cycle defined by $T + xy$. Because T is a shortest path tree it cannot be the case that y is an ancestor of x in T . Neither is $y \in T(u)$; if so, $d(u, y) \leq 3$ and therefore $\ell(C) \leq 6$, contradicting Corollary 5. Thus $y \notin T(u)$ and, consequently, $uv \in C$.

Let z be the lowest common ancestor of x and y . Then C is a path from z to x , plus the edge xy , plus a path from y to z . So it has length

$$d(x, z) + 1 + d(y, z) \leq 2d(x, r) + 2 = 2d(u, r) + 2i + 2$$

Observe that to achieve this maximum length, we must have $d(x, z) = d(x, r) = d(y, r) - 1$, and C must consist of two shortest paths from y to r , meaning C will also contain r .

Next consider the smallest cycle $C(u)$ containing u . Suppose for the sake of contradiction that $d(u, r) \leq \lfloor \frac{|C(u)|}{2} \rfloor - i - 1$. If $|C(u)|$ is odd, then $2d(u, r) + 2i + 3 \leq |C(u)|$, which contradicts the choice of $C(u)$. Therefore, $|C(u)|$ is even and $2d(u, r) + 2 + 2i \leq |C(u)|$. However, $|C| \leq 2d(u, r) + 2i + 2$. Thus $|C| = |C(u)|$ and therefore C a min cycle, by Lemma 2. Furthermore, C is directed by Lemma 3. This is a contradiction because then xy is in a directed shortest path, and must be part of T and not an edge in X_0 . Hence $d(u, r) \geq \lfloor \frac{|C(u)|}{2} \rfloor - i$, as desired. ◀

We present one last technical lemma in this section. We remark that this lemma is of critical value in our analysis and, we believe, may be of importance in achieving future improvements.

► **Lemma 9.** *If uvw is a directed path in H and $\deg_H(v) = 2$, then $|S_H(v)| \geq \frac{\alpha}{2(|C(v)|-3)}$. If, in addition, uvw is a directed path of down-edges in T , then $|S_H(v)| \geq |T(w)|$.*

Proof. Consider the strategy change of u selling uv and buying uw . As $\deg_H(v) = 2$, the only vertices which become farther from u are those of $S_H(v)$. They all become farther from u by 1. All the vertices in $S_u(w)$ conversely, become closer to u by 1. Therefore, by the equilibrium conditions, we must have $|S_H(v)| \geq |S_u(w)|$, otherwise u has an incentive to switch strategies. Further, if uvw is a path of down-edges in T , then because u is the ancestor of w and T is a shortest path tree, $T(w) \subseteq S_u(w)$. Thus $|S_H(v)| \geq |T(w)|$, as desired.

Now consider a second strategy change in which u deletes uv . There are two sets of vertices whose distance to u increases. Shortest paths from u to $S_H(v)$ go from using uv to using $C(v) \setminus uv$; note that as $\deg_H(v) = 2$ it must be that case that $uv, vw \in C(v)$. The distance of these vertices from u then increase by $|C(v)| - 2$. Meanwhile, paths from u to $S_u(w)$ go from using uvw to using at most $C(v) \setminus \{uv, vw\}$, an increase in distance of at most $|C(v)| - 4$. No other vertices have increased distance to u . Therefore u 's cost as a result changes by at most

$$(|C(v)| - 2) \cdot |S_H(v)| + (|C(v)| - 4) \cdot |S_u(w)| - \alpha \leq (2|C(v)| - 6) \cdot |S_H(v)| - \alpha$$

Here the inequality holds as $|S_H(v)| \geq |S_u(w)|$. This change must non-negative by the equilibrium condition. This implies $|S_H(v)| \geq \frac{\alpha}{2(|C(v)|-3)}$, as desired. \blacktriangleleft

3 A Class of Deviation Strategies

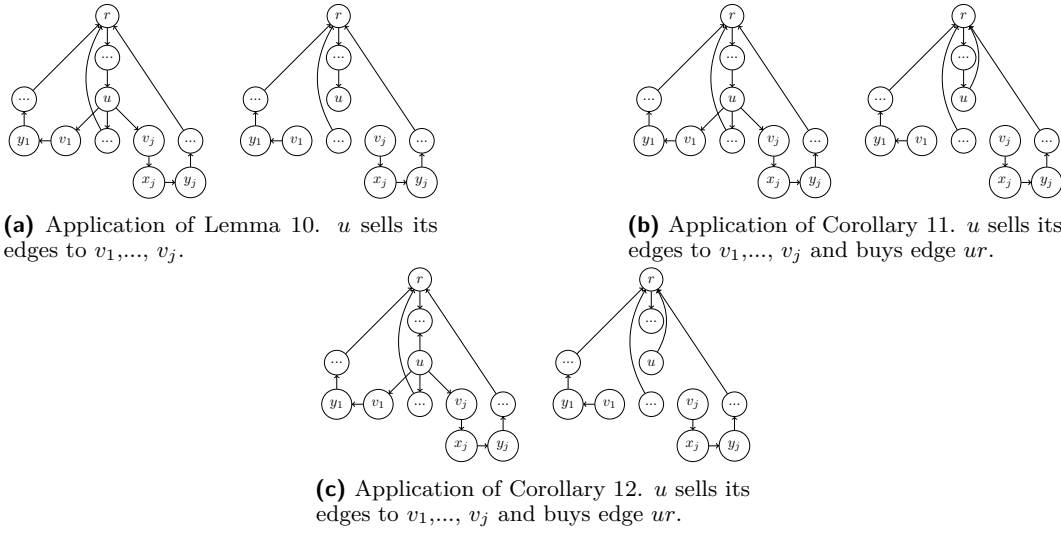
We now present a new class of deviation strategies. Specifically, in Section 3.1 we study three related strategies that a vertex in the biconnected component may ponder. Concretely, we derive bounds on the change in cost to the vertex resulting from using these strategy changes. In particular, we will use the fact that, at a Nash equilibrium, these cost changes must be non-negative. Then, in Section 3.2, we apply these deviation strategies to make three substantial observations concerning vertices that buy multiple edges of specific types in H .

3.1 Three Deviation Strategies

As stated, we now introduce a class of deviation strategies. The first, shown in Figure 2a, involves a vertex u selling edges of X_2 . Lemma 7 guarantees that the graph is still connected without these edges. Now u 's cost will change: it saves α for each edge sold, but its distance to many vertices may increase by varying amounts, requiring us to bound the new total distance to all vertices. The second deviation, depicted in Figure 2b, is very similar to the first. The only change is that u buys the edge ur . This reduces the saving by α for the extra edge bought, but it also reduces the bound on the increased distance to the other vertices considerably. Finally, the third deviation, pictured in Figure 2c, is slightly different than the second. The bound given is weaker, because u may sell edges of X_2^+ , rather than just X_2 . Selling the edge to u 's parent reduces some of the savings from the previous strategies.

► Lemma 10. *If $\alpha > 2n$, u buys $uv_1 \in X_{i_1}, \dots, uv_j \in X_{i_j}$ for $j \geq 1$, $i_k \leq 2$, for all k , $1 \leq k \leq j$, $u = u_0, u_1, \dots, u_{d(u,r)-1}, u_{d(u,r)} = r$ is the path from u to r in T , then the change of u selling uv_1, \dots, uv_j results in a change of cost of at most*

$$d(u, r) \cdot n - \sum_{l=0}^{d(u,r)-1} 2 \cdot |T(u_l)| - j \cdot \alpha + \sum_{k=1}^j (2i_k + 2d(u, r)) \cdot |T(uv_{i_k})| \quad (3)$$



■ **Figure 2** The three main strategy changes for vertex u .

Proof. We commence by bounding $D(u)$ before the changes, by comparing it to $D(r)$. By the triangle inequality, u 's distance to any vertex v is at most $d(r, u) + d(r, v)$ because there is a path from u to r and a path from r to v . This is the bound we use for vertices in $T(r) \setminus T(u_{d(u,r)-1})$. For other vertices this is obviously not the shortest path. For instance, for any vertex $v \in T(u)$, $d(u, v) = d(r, v) - d(r, u)$. Furthermore, for l in $1 \leq l \leq d(u, r)$, the length of u 's shortest path to $v \in |T(u_l) \setminus T(u_{l-1})|$ differs from r 's by $(d(u, r) - 2l)$. Therefore we have the bound:

$$D(u) \leq D(r) - d(u, r) \cdot |T(u)| - \sum_{l=1}^{d(u,r)} (d(u, r) - 2l) \cdot |T(u_l) \setminus T(u_{l-1})|$$

Now $D(r) \leq D(u)$. Moreover, if we rearrange, we can more concisely write this as

$$0 \leq d(u, r) \cdot n - \sum_{l=0}^{d(u,r)-1} 2 \cdot |T(u_l)|$$

To complete the proof, consider what happens when u sells uv_1, \dots, uv_j . First, u obviously saves $j\alpha$ from its edge costs. Second, by Lemma 7, we know that all $w \in T(uv_i)$ have a path of length $\leq 2i_k + d(w, r)$ to r that does not contain u . This leads to an increase in distance from u to w of at most

$$(d(u, r) + 2i_k + d(w, r)) - (d(w, r) - d(u, r)) = 2d(u, r) + 2i_k$$

This yields our desired bound on the change of cost for u :

$$0 \leq d(u, r) \cdot n - \sum_{l=0}^{d(u,r)-1} 2 \cdot |T(u_l)| - j \cdot \alpha + \sum_{k=1}^j (2i_k + 2d(u, r)) \cdot |T(uv_{i_k})| \quad \blacktriangleleft$$

► **Corollary 11.** *If $\alpha > 2n$, u buys $uv_1 \in X_{i_1}, \dots, uv_j \in X_{i_j}$ for $j \geq 1$, $i_k \leq 2$, for all k , $1 \leq k \leq j$, $u = u_0, u_1, \dots, u_{d(u,r)-1}, u_{d(u,r)} = r$ is the path from u to r in T , then the strategy change of u selling uv_1, \dots, uv_j and buying ur results in a change of cost of at most*

$$n - |T_{u_{\frac{d(u,r)}{2}}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| - (j-1) \cdot \alpha + \sum_{k=1}^j (2i_k + d(u, r) + 1) \cdot |T(uv_{i_k})|$$

where the second term is removed if $\frac{d(u,r)}{2}$ is not integral.

28:10 One n Remains to Settle the Tree Conjecture

Proof. This strategy change is identical that of to Lemma 10 except u also buys ur . This means that the new path from u to r is now length 1, not length $d(u, r)$. Hence we replace $d(u, r) \cdot n$ with n in Equation (3). There are still vertices which are as close or closer to u than to r . The vertices of $|T_{u \frac{d(u,r)}{2}}|$, which only exists if $\frac{d(u,r)}{2}$ is integral, are at-least as close to u as to r , therefore u does not use the edge ur on the path to any of the vertices in $|T_{u \frac{d(u,r)}{2}}|$, thus we subtract $|T_{u \frac{d(u,r)}{2}}|$ from Equation (3). For the remaining sets $T(u_l)$ for $l < \frac{d(u,r)}{2}$, each time l decreases by 1, the vertices in the set are 1 closer to u and 1 farther from r .

Finally, the last two terms of Equation (3) are affected as well. We now buy an additional edge, so we add α , resulting in the $(j-1) \cdot \alpha$ term. Furthermore, as previously stated. The path from u to r now has length 1, meaning for all $w \in T(uv_i)$ the increase in distance from u to w is at most

$$(2d(u, r) + 2i_k) - (d(u, r) - 1) = 1 + d(u, r) + 2i_k$$

This yields our desired bound on the change of cost for u :

$$n - |T_{u \frac{d(u,r)}{2}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| - (j-1) \cdot \alpha + \sum_{k=1}^j (2i_k + d(u, r) + 1) \cdot |T(uv_{i_k})| \quad \blacktriangleleft$$

► **Corollary 12.** *If $\alpha > 2n$, u buys $uv_1 \in X_{i_1}^+, \dots, uv_j \in X_{i_j}^+$ for $j \geq 1$, $i_k \leq 2$, for all $k, 1 \leq k \leq j$, then the strategy change of u selling uv_1, \dots, uv_j and buying ur results in a change of cost of at most*

$$n - (j-1)\alpha - (d(u, r) + 1) \cdot |T(u)| + \sum_{k=1}^j (2i_k + d(u, r) + 1) \cdot |T(uv_{i_k})|$$

Proof. This strategy change is nearly identical to Corollary 11, except now u is potentially selling the edge to its parent in T . We are no longer certain that u is closer to its former parent than r is, nor any vertices along the original path from u to r . Thus we lose the savings of all $T(u_l)$ except for $l = 0$, as $T(u_0) = T(u)$. ◀

3.2 Observations arising from the Deviation Strategies

We now apply these three deviation strategies to make three observations concerning vertices that buy multiple edges of specific types in H . In turn these observations will be instrumental in proving the main result in Section 4. The first observation simply states that no vertex can buy two edges in X_1^+ .

► **Observation 13.** *No vertex $u \in H$ buys two edges $uv_1, uv_2 \in X_1^+$.*

Proof. Suppose $u \in H$ buys $uv_1, uv_2 \in X_1^+$. Observe r cannot buy an edge in X_1^+ , as this would imply the existence of a short cycle, contradicting Corollary 5; therefore $u \neq r$. Hence, Lemma 6 implies $|T(u)| \leq \frac{n}{2}$. We now apply Corollary 12 to the strategy change where u sells $uv_1, uv_2 \in X_1^+$ and buys ur . The change in cost for u is at most

$$\begin{aligned} n - \alpha - (d(u, r) + 1) \cdot |T(u)| + \sum_{k=1}^2 (2i_k + d(u, r) + 1) \cdot |T(uv_{i_k})| \\ = n - \alpha - (d(r, u) + 1) \cdot (|T(u)| - |T(uv_1)| - |T(uv_2)|) + 2(|T(uv_1)| + |T(uv_2)|) \\ \leq n - \alpha + 2 \cdot |T(u)| \end{aligned}$$

$$\begin{aligned}
 &\leq n - \alpha + 2\frac{n}{2} \\
 &\leq 2n - \alpha \\
 &< 0
 \end{aligned}$$

This contradicts the Nash equilibrium conditions. \blacktriangleleft

Next we observe that no vertex can buy three edges in X_2^+ . Furthermore any vertex that buys two edges in X_2^+ must be the parent of a child at the root of a large subtree in T .

► **Observation 14.** *No vertex $u \in H$ buys three edges $uv_1, uv_2, uv_3 \in X_2^+$, and if u buys $uv_1, uv_2 \in X_2^+$, then $|T(uv_1) \cup T(uv_2)| > \frac{n}{4}$.*

Proof. Suppose $u \in H$ buys $uv_1, uv_2, uv_3 \in X_2^+$. Again, r cannot buy edges in X_2 , as this would contradict Corollary 5; so $u \neq r$. Thus Lemma 6 implies $|T(u)| \leq \frac{n}{2}$. Now we apply Corollary 12 to the strategy change where v sells $vv_1, vv_2, vv_3 \in X_2^+$ and buys vr . The change in cost for v is:

$$\begin{aligned}
 &n - 2\alpha - (d(u, r) + 1) \cdot |T(u)| + \sum_{k=1}^2 (2i_k + d(u, r) + 1) \cdot |T(uv_{i_k})| \\
 &= n - 2\alpha - (d(r, u) + 1) \cdot (|T(u)| - |T(uv_1)| - |T(uv_2)| - |T(uv_3)|) \\
 &\quad + 4(|T(uv_1)| + |T(uv_2)| + |T(uv_3)|) \\
 &\leq n - 2\alpha + 4 \cdot |T(u)| \\
 &\leq n - 2\alpha + 4\frac{n}{2} \\
 &\leq 3n - 2\alpha \\
 &< 0
 \end{aligned}$$

This contradicts the Nash equilibrium conditions. Now suppose $u \in H$ buys $uv_1, uv_2 \in X_2$. We can apply Corollary 12 to the strategy change where v sells $uv_1, uv_2 \in X_2^+$ and buys ur . The change in cost for u is at most

$$\begin{aligned}
 &n - \alpha - (d(u, r) + 1) \cdot (|T(v)| - |T(uv_1)| - |T(uv_2)|) + 4 \cdot (|T(uv_1)| + |T(uv_2)|) \\
 &\leq n - \alpha + 4 \cdot |T(uv_1) \cup T(uv_2)| \\
 &< 4 \cdot |T(uv_1) \cup T(uv_2)| - n
 \end{aligned}$$

This is non-negative by the Nash equilibrium conditions. Therefore $|T(vv_1) \cup T(vv_2)| > \frac{n}{4}$, as desired. \blacktriangleleft

Finally, we observe some properties that follow when a vertex buys two edges in X_2 .

► **Observation 15.** *If u buys two edges $uv_1, uv_2 \in X_2$, then $d(r, u) \geq 3$. If, in addition, $|T(uv_1) \cup T(uv_2)| > \frac{n}{4}$ then $d(r, u) = 3$.*

Proof. By Corollary 5, r cannot buy edges of X_2 , therefore $u \neq r$. Further, by Lemma 6, $|T(u)| \leq \frac{n}{2}$.

We now apply Lemma 10 with the strategy change where u sells uv_1, uv_2 . Its cost changes by at most

$$\begin{aligned}
 &d(r, u) \cdot n - 2\alpha - 2d(r, u) \cdot |T(u)| + (2d(r, u) + 4) \cdot |T(uv_1) \cup T(uv_2)| \\
 &\leq d(r, u) \cdot n - 2\alpha + 4 \cdot |T(uv_1) \cup T(uv_2)| \\
 &\leq d(r, u) \cdot n - 2\alpha + 4\frac{n}{2}
 \end{aligned}$$

28:12 One n Remains to Settle the Tree Conjecture

$$\begin{aligned} &< d(r, u) \cdot n - 2(2n) + 2n \\ &= (d(r, u) - 2) \cdot n \end{aligned}$$

If $d(r, u) \leq 2$ then this cost change is negative, a contradiction. Thus $d(r, u) \geq 3$.

Now suppose $|T(uv_1) \cup T(uv_2)| > \frac{n}{4}$. Without loss of generality, let $|T(uv_1)| \geq |T(uv_2)|$. Therefore $|T(uv_1)| > \frac{n}{8}$, meaning $T(uv_1) = T(v_1)$. As $uv_1 \in X_2$, we know v_1 buys an edge in X_0 or $X_1 \setminus X_0$. Suppose v_1 buys $v_1y \in X_0$. By Corollary 11, v selling v_1y and buying v_1r changes v_1 's cost by

$$\begin{aligned} &\leq n - (d(v_1, r) + 1) \cdot |T(v_1)| - (d(v_1, r) - 1) \cdot |T(u) \setminus T(v_1)| \\ &\leq n - 2 \cdot |T(v_1)| - (d(v, r) - 1) \cdot |T(u)| \\ &< n - 2 \frac{n}{8} - (d(v_1, r) - 1) \frac{n}{4} \\ &= n - d(v_1, r) \frac{n}{4} \\ &= n - (d(u, r) + 1) \frac{n}{4} \\ &\leq n - (3 + 1) \frac{n}{4} \\ &= 0 \end{aligned}$$

This contradicts the equilibrium condition, therefore v_1 buys $v_1w \in X_1 \setminus X_0$. w buys $wy \in X_0$. By Corollary 11, w selling wy and buying wr changes w 's cost by

$$\begin{aligned} &\leq n - (d(w, r) + 1)|T(w)| - (d(w, r) - 1)|T(v_1) \setminus T(w)| - (d(w, r) - 3)|T(u) \setminus T(v_1)| \\ &\leq n - 2 \cdot |T(v_1)| - (d(w, r) - 3) \cdot |T(u)| \\ &< n - 2 \frac{n}{8} - (d(w, r) - 3) \frac{n}{4} \\ &= \frac{3}{2}n - d(w, r) \frac{n}{4} \\ &= \frac{3}{2}n - (d(u, r) + 2) \frac{n}{4} \\ &= n - d(u, r) \frac{n}{4} \end{aligned}$$

If $d(r, u) \geq 4$, this leads to a contradiction. Thus $d(r, u) = 3$, as desired. \blacktriangleleft

4 The Tree Conjecture holds for $\alpha > 2n$

We now have all the resources necessary to prove the tree conjecture holds if $\alpha > 2n$. To do this, we prove two final lemmas which together give the main result. The first of these lemmas investigates the path to the root r from a vertex $u \in H$ that buys an out-edge.

► **Lemma 16.** *Let $\alpha > 2n$. If u_0 buys $u_0v \in X_0$ and $u_0, u_1, \dots, u_{d(u,r)-1}, u_{d(u,r)} = r$ is the path from u_0 to r in T then at most one of u_0, u_1, u_2 has $\deg_H = 2$*

Proof. By Observation 13, no vertex buys two edges in X_1^+ . Thus u_0 cannot buy u_0u_1 as well as u_0v . Therefore u_1 buys u_1u_0 . Similarly, by Observation 13, u_1 cannot buy u_1u_2 as well as u_1u_0 . Therefore u_2 buys u_2u_1 .

Now if $\deg_H(u_0) = 2$ then $S_H(u_0) = T(u_0)$, because u_1 is u_0 's parent and $v \notin T(u_0)$, thus $T(u_0) \cap H = \{u_0\}$. Similarly, if $\deg_H(u_1) = 2$ then $S_H(u_1) = T(u_1) \setminus T(u_0)$, and if $\deg_H(u_2) = 2$ then $S_H(u_2) = T(u_2) \setminus T(u_1)$.

Next by Lemma 8, we know that $d(r, u_2) \geq \lfloor \frac{|C(u_2)|}{2} \rfloor - 2$, that $d(r, u_1) \geq \lfloor \frac{|C(u_1)|}{2} \rfloor - 1$ and that $d(r, u_0) \geq \lfloor \frac{|C(u_0)|}{2} \rfloor$.

Furthermore, if $\deg_H(u_1) = 2$ then, by Lemma 9, $|S_H(u_1)| \geq |T(u_0)|$. If $\deg_H(u_2) = 2$ then, by Lemma 3, $u_2 u_1$ is in a directed cycle, which means u_3 buys $u_3 u_2$. Thus, by Lemma 9, we have $|S_H(u_2)| \geq |T(u_1)|$

We proceed by case analysis. First, assume that $\deg_H(u_2) = \deg_H(u_1) = 2$. Then, by Lemma 9, we have $|S_H(u_1)| \geq \frac{\alpha}{2(|C(u_1)|-3)}$ and $|S_H(u_2)| \geq |T(u_1)|$. Thus $|T(u_2)| = |S_H(u_2)| + |T(u_1)| \geq 2|T(u_1)| \geq 2|S_H(u_1)| \geq \frac{2\alpha}{2(|C(u_1)|-3)}$. Now, by Lemma 6, $|T(u_2)| \leq \frac{n}{2}$. Therefore $\frac{n}{2} \geq \frac{2\alpha}{2(|C(u_1)|-3)}$. Rearranging gives $|C(u_1)| \geq 8$. By Observation 15, $d(u_1, r) \geq 3$.

Applying Corollary 11 when u_1 sells $u_1 u_0$ and buys $u_1 r$, the change in cost to u_1 is at most

$$\begin{aligned}
& n - |T_{u_{\frac{d(u,r)}{2}}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| - (j-1) \cdot \alpha + \sum_{k=1}^j (2i_k + d(u, r) + 1) \cdot |T(uv_{i_k})| \\
&= n - |T_{u_{\frac{d(u,r)}{2}}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| + (d(u_1, r) + 1 + 2) \cdot |T(u_1 u_0)| \\
&= n - |T_{u_{\frac{d(u,r)}{2}}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| + (d(u_1, r) + 3) \cdot |T(u_0)| \\
&\leq n - |T_{u_{\frac{d(u,r)}{2}}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| + (2d(u_1, r)) \cdot |T(u_0)| \\
&\leq n - 2|T(u_1)| - (d(u_1, r) - 1) \cdot |T(u_2)| + (2d(u_1, r)) \cdot |T(u_0)| \\
&= n - (d(u_1, r) + 1) \cdot |T(u_1)| - (d(u_1, r) - 1) \cdot (|T(u_2) \setminus |T(u_1)||) + (2d(u_1, r)) \cdot |T(u_0)| \\
&= n - (d(r, u_1) + 1) \cdot |T(u_1)| - (d(u_1, r) - 1) \cdot |S_H(u_2)| + (d(u_1, r) + 3) \cdot |T(u_0)| \\
&\leq n - (d(r, u_1) + 1) \cdot |T(u_1)| - (d(u_1, r) - 1) \cdot |T(u_1)| + (2d(u_1, r)) \cdot |T(u_0)| \\
&= n - 2(d(r, u_1)) \cdot |T(u_1)| + 2(d(u_1, r)) \cdot |T(u_0)| \\
&= n - 2(d(r, u_1)) \cdot |T(u_1) \setminus T(u_0)| \\
&= n - 2(d(r, u_1)) \cdot |S_H(u_1)| \\
&\leq n - 2(d(r, u_1)) \cdot \frac{\alpha}{2(|C(u_1)|-3)} \\
&\leq n - 2(\lfloor \frac{|C(u_1)|}{2} \rfloor - 1) \cdot \frac{\alpha}{2(|C(u_1)|-3)} \\
&\leq n - (|C(u_1)| - 3) \cdot \frac{\alpha}{2(|C(u_1)|-3)} \\
&= n - \frac{\alpha}{2} \\
&< 0
\end{aligned}$$

Therefore it cannot be that $\deg_H(u_2) = \deg_H(u_1) = 2$.

Second, suppose $\deg_H(u_0) = \deg_H(u_2) = 2$. By Lemma 9, we have $|S_H(u_0)| \geq \frac{\alpha}{2(|C(u_1)|-3)}$ and $|S_H(u_2)| \geq |T(u_1)|$. Because $u_0 v \in X_0$, it follows that $T(v) \not\subseteq T(u_0)$. Thus $T(u_0 v) = \emptyset$. Applying Corollary 11 when u_0 sells $u_0 v$ and buys $u_0 r$, the change in cost to u_0 is at most

$$\begin{aligned}
& n - |T_{u_{\frac{d(u,r)}{2}}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| - (j-1) \cdot \alpha + \sum_{k=1}^j (2i_k + d(u, r) + 1) \cdot |T(uv_{i_k})| \\
&= n - |T_{u_{\frac{d(u,r)}{2}}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| \\
&\leq n - 2|T(u_0)| - 2|T(u_1)| - (d(u_0, r) - 3) \cdot |T(u_2)|
\end{aligned}$$

28:14 One n Remains to Settle the Tree Conjecture

$$\begin{aligned}
&= n - 2|T(u_0)| - (d(u_0, r) - 1) \cdot |T(u_1)| - (d(u_0, r) - 3) \cdot |T(u_2) \setminus T(u_1)| \\
&= n - 2|T(u_0)| - (d(u_0, r) - 1) \cdot |T(u_1)| - (d(u_0, r) - 3) \cdot |S_H(u_2)| \\
&\leq n - 2|T(u_0)| - (d(u_0, r) - 1) \cdot |T(u_1)| - (d(u_0, r) - 3) \cdot |T(u_1)| \\
&= n - 2|T(u_0)| - (2d(u_0, r) - 4) \cdot |T(u_1)| \\
&\leq n - (2d(u_0, r) - 2)|T(u_0)| \\
&= n - (2d(u_0, r) - 2)|S_H(u_0)| \\
&\leq n - (2d(r, u_0) - 2) \cdot \frac{\alpha}{2(|C(u_0)|-3)} \\
&\leq n - (2\lfloor \frac{|C(u_0)|}{2} \rfloor - 2) \cdot \frac{\alpha}{2(|C(u_0)|-3)} \\
&\leq n - (|C(u_0)| - 3) \cdot \frac{\alpha}{2(|C(u_0)|-3)} \\
&= n - \frac{\alpha}{2} \\
&< 0
\end{aligned}$$

Therefore it is not true that $\deg_H(u_2) = \deg_H(u_0) = 2$.

Third, suppose $\deg_H(u_0) = \deg_H(u_1) = 2$. By Lemma 9, we have $|S_H(u_0)| \geq \frac{\alpha}{2(|C(u_1)|-3)}$ and $|S_H(u_1)| \geq |T(u_0)|$. Because $u_0v \in X_0$, it follows that $T(v) \not\subseteq T(u_0)$. Thus $T(u_0v) = \emptyset$. Applying Corollary 11 when u_0 sells u_0v and buys u_0r , the change in cost to u_0 is at most

$$\begin{aligned}
&n - |T_{u_{\frac{d(u,r)}{2}}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| - (j-1) \cdot \alpha + \sum_{k=1}^j (2i_k + d(u, r) + 1) \cdot |T(uv_{i_k})| \\
&= n - |T_{u_{\frac{d(u,r)}{2}}}| - \sum_{l < \frac{d(u,r)}{2}} 2 \cdot |T(u_l)| \\
&\leq n - 2|T(u_0)| - (d(u_0, r) - 1) \cdot |T(u_1)| \\
&= n - (d(u_0, r) + 1) \cdot |T(u_0)| - (d(u_0, r) - 1) \cdot |T(u_1) \setminus T(u_0)| \\
&= n - (d(u_0, r) + 1) \cdot |T(u_0)| - (d(u_0, r) - 1) \cdot |S_H(u_1)| \\
&\leq n - (d(u_0, r) + 1) \cdot |T(u_0)| - (d(u_0, r) - 1) \cdot |T(u_0)| \\
&\leq n - (2d(u_0, r)) \cdot |T(u_0)| \\
&= n - (2d(u_0, r)) \cdot |S_H(u_0)| \\
&\leq n - (2d(r, u_0)) \cdot \frac{\alpha}{2(|C(u_0)|-3)} \\
&\leq n - (2\lfloor \frac{|C(u_0)|}{2} \rfloor) \cdot \frac{\alpha}{2(|C(u_0)|-3)} \\
&\leq n - (|C(u_0)| - 3) \cdot \frac{\alpha}{2(|C(u_0)|-3)} \\
&= n - \frac{\alpha}{2} \\
&< 0
\end{aligned}$$

Therefore it is not true that $\deg_H(u_1) = \deg_H(u_0) = 2$. Thus, at most one of u_0, u_1, u_2 has $\deg_H = 2$, as desired. \blacktriangleleft

We now apply a counting argument to upper bound the number of vertices in H that buy two edges in X_2 .

► **Lemma 17.** *The number of vertices $v \in H$ that buy two edges in X_2 is $< \deg_H(r)$.*

Proof. The only vertices that can buy two edges e_1, e_2 in X_2 are those with $|T(e_1) \cup T(e_2)| > \frac{n}{4}$ by Observation 14. All of these vertices are distance 3 from r , by Observation 15. The common ancestor in T of any pair of these vertices must be r , otherwise there is a vertex

$u \neq r$ with $|T(u)| > \frac{n}{2}$, contradicting Lemma 6. Furthermore, r cannot buy any edge e such that $T(e)$ contains such a vertex v , otherwise r could sell e and buy rv , reducing its distance to $T(v)$ by 2 and increasing its distance to $T(e) \setminus T(v)$ by ≤ 2 . This gives a net increase in cost of $\leq 2|T(v)| - 2|T(e) \setminus T(v)| \leq 4|T(v)| - 2|T(e)| < 4\frac{n}{4} - 2\frac{n}{2} = 0$, a contradiction. Therefore the number of vertices which buy two edges in X_2 is $\leq \deg_H^-(r)$. By Lemma 3, we must have $\deg_H^+(r) \geq 1$, implying $\deg_H^-(r) < \deg_H(r)$ and completing the proof. \blacktriangleleft

Putting everything together we can now prove the main result.

Proof of Theorem 1. Here we use combine all to prove that if G is a Nash equilibrium graph for the network creation game (n, α) and $\alpha > 2n$, G is a tree.

This is a proof by contradiction based on the assumption that there exists a biconnected component H in G containing n_H vertices. Within H , the sum of degrees of all vertices in H equals $2(n_H - 1) + 2|X_0|$. That is twice the number of edges in the spanning tree on H induced by T plus twice the number of out-edges in H .

Let's instead count the degrees in the following way:

$$\begin{aligned}
 & \sum_{i=2}^{n_H} \sum_{v \in H: \deg_H(v)=i} i \\
 &= 2n_H + \sum_{i=3}^{n_H} \sum_{v \in H: \deg_H(v)=i} i - 2 \\
 &= 2n_H + \deg_H(r) - 2 + \sum_{i=3}^{n_H} \sum_{v \in H-r: \deg_H(v)=i} i - 2 \\
 &\geq 2n_H + \deg_H(r) - 2 + 2|X_0| - \sum_{i=2}^{|X_2|} \sum_{v \in H: v \text{ is in } i \text{ paths of } X_2 \text{ edges}} i - 1 \quad [\text{by Lemma 16}] \\
 &= 2n_H + \deg_H(r) - 2 + 2|X_0| - \sum_{v \in H: v \text{ is in 2 paths of } X_2 \text{ edges}} 1 \quad [\text{by Observation 14}] \\
 &> 2n_H + \deg_H(r) - 2 + 2|X_0| - \deg_H(r) \quad [\text{by Lemma 17}] \\
 &= 2(n_H - 1) + 2|X_0|
 \end{aligned}$$

This is a contradiction, implying that H does not exist for $\alpha > 2n$. \blacktriangleleft

5 Conclusion

In this paper we proved the revised tree conjecture holds for $\alpha > 2n$. Moreover, we have reached a natural limit in the quest to settle this conjecture. Specifically, for $\alpha \in (n - 3, 2n]$, the range in which the conjecture is unsettled, we are no longer certain that directed cycles must be present in non-tree equilibrium graphs. To close this remaining gap, we believe min-cycles still have an important role to play but, to allow their usage, more precise analyses will be necessary. A potentially useful intermediate step would be to determine conditions that allow for an undirected min-cycle.

References

- 1 Susanne Albers, Stefan Eilts, Eyal Even-Dar, Yishay Mansour, and Liam Roditty. On nash equilibria for a network creation game. *ACM Transactions on Economics and Computation*, 2(1):1–27, 2014. doi:10.1145/2560767.
- 2 N. Alon, E. Demaine, M. Hajiaghayi, and T. Leighton. Basic network creation games. *SIAM Journal on Discrete Mathematics*, 27(2):656–668, 2013. doi:10.1137/090771478.
- 3 C. Àlvarez and A. Messegué. Network creation games: Structure vs anarchy. arXiv:1706.09132.
- 4 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Locality-based network creation games. *ACM Transactions on Parallel Computing*, 3(1):1–26, 2016. doi:10.1145/2938426.
- 5 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Network creation games with traceroute-based strategies. *Algorithms*, 14(2):35, 2021. doi:10.3390/a14020035.
- 6 Davide Bilò and Pascal Lenzner. On the tree conjecture for the network creation game. *Theory of Computing Systems*, 64(3):422–443, 2019. doi:10.1007/s00224-019-09945-9.
- 7 A. Chauhan, P. Lenzner, A. Melnichenko, and L. Molitor. Selfish network creation with non-uniform edge cost. In *Proceedings of 10th Symposium on Algorithmic Game Theory (SAGT)*, pages 160–172, 2017. doi:10.1007/978-3-319-66700-3_13.
- 8 A. Cord-Landwehr and P. Lenzner. Network creation games: Think global - act local. In *Proceedings of 40th Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 248–260, 2015. doi:10.1007/978-3-662-48054-0_21.
- 9 E. Demaine, M. Hajiaghayi, H. Mahini, and M. Zadimoghaddam. The price of anarchy in cooperative network creation games. *ACM Transactions on Economics and Computation*, 8(2), 2012.
- 10 J. Dippel and A. Vetta. An improved bound for the tree conjecture in network creation games. In *Proceedings of 15th Symposium on Algorithmic Game Theory (SAGT)*, pages 241–257, 2022.
- 11 Alex Fabrikant, Ankur Luthra, Elitza Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, 2003. doi:10.1145/872035.872088.
- 12 P. Lenzner. *On Selfish Network Creation*. PhD thesis, Humboldt-Universität zu Berlin, 2014.
- 13 Akaki Mamageishvili, Matúš Mihalák, and Dominik Müller. Tree nash equilibria in the network creation game. *Internet Mathematics*, 11(4–5):472–486, 2015. doi:10.1080/15427951.2015.1016248.
- 14 Matúš Mihalák and Jan Christoph Schlegel. The price of anarchy in network creation games is (mostly) constant. *Theory of Computing Systems*, 53(1):53–72, 2013. doi:10.1007/s00224-013-9459-y.
- 15 C. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the 33rd ACM Symposium on the Theory of Computing (STOC)*, pages 749–753, 2001.
- 16 Q. Wang. On tree equilibria in max-distance network creation games. In *Proceedings of 15th Symposium on Algorithmic Game Theory (SAGT)*, pages 293–310, 2022.
- 17 Carme Àlvarez and Arnau Messegué Buisan. On the poa conjecture: Trees versus biconnected components. *SIAM Journal on Discrete Mathematics*, 37(2):1030–1052, 2023. doi:10.1137/21m1466426.

Semënov Arithmetic, Affine VASS, and String Constraints

Andrei Draghici ✉ 

Department of Computer Science, University of Oxford, UK

Christoph Haase ✉ 

Department of Computer Science, University of Oxford, UK

Florin Manea ✉ 

Computer Science Department and Campus-Institut Data Science, Göttingen University, Germany

Abstract

We study extensions of Semënov arithmetic, the first-order theory of the structure $\langle \mathbb{N}, +, 2^x \rangle$. It is well-known that this theory becomes undecidable when extended with regular predicates over tuples of number strings, such as the Büchi V_2 -predicate. We therefore restrict ourselves to the existential theory of Semënov arithmetic and show that this theory is decidable in EXPSPACE when extended with arbitrary regular predicates over tuples of number strings. Our approach relies on a reduction to the language emptiness problem for a restricted class of affine vector addition systems with states, which we show decidable in EXPSPACE. As an application of our result, we settle an open problem from the literature and show decidability of a class of string constraints involving length constraints.

2012 ACM Subject Classification Theory of computation \rightarrow Logic

Keywords and phrases arithmetic theories, Büchi arithmetic, exponentiation, vector addition systems with states, string constraints

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.29

Funding This work is part of a project that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT). Florin Manea’s work was supported by a Heisenberg Grant, funded by the Deutsche Forschungsgemeinschaft (DFG, project 466789228).



1 Introduction

This paper studies the decidability and complexity of the existential theory of an extension of the structure $\langle \mathbb{N}, 0, 1, +, 2^x \rangle$, where 2^x is the function mapping a natural number n to 2^n . Decidability of the first-order theory of this structure was first shown by Semënov in a more general framework using an automata-theoretic approach [13], and we henceforth call this theory *Semënov arithmetic*. As shown by Cherlin and Point [7], see also [11], Semënov arithmetic admits quantifier elimination and has a quantifier-elimination procedure that runs in non-elementary time, and this upper bound is tight [11]. The existential fragment of Semënov arithmetic has recently been shown decidable in NEXP [2] by giving a more elaborate quantifier elimination procedure. Unlike its substructure Presburger arithmetic (obtained by dropping the function 2^x), Semënov arithmetic is not automatic in the sense of the theory of automatic structures [5, 10]. Consider¹ the family of formulas $\Phi_n \equiv x_1 = 1 \wedge \bigwedge_{1 \leq i \leq n} x_{i+1} = 2^{x_i} \wedge y < x_n$. Then the (finite) number of solutions of Φ_n is a tower of height n . Suppose Semënov arithmetic (viewed as a relational structure) was automatic. Then each of its relations is definable by a deterministic finite automaton (DFA)

¹ We thank an anonymous reviewer for suggesting this argument.



© Andrei Draghici, Christoph Haase, and Florin Manea;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 29; pp. 29:1–29:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



of size at most m for some $m \in \mathbb{N}$ over some alphabet Σ . But then the DFA for Φ_n is acyclic and has at most m^{n+2} many states, meaning that Φ_n has at most $(m^{(n+2)}|\Sigma|)^{(m^{n+2})}$ different solutions, a contradiction.

The decidability of Semënov arithmetic is fragile with respect to extensions of the structure. For instance, it is not difficult to see that extending Semënov arithmetic with the Büchi predicate $V_2(x, y)$, where $V_2(x, y)$ holds whenever x is the largest power of two dividing y without remainder, results in an undecidable first-order theory, see e.g. [11]. However, this undecidability result requires an $\exists^*\forall^*$ -quantifier prefix and does not rule out decidability of the existential fragment. The main result of this paper is to show that the existential theory of *generalised Semënov arithmetic*, i.e., the existential theory of $\langle \mathbb{N}, 0, 1, +, 2^x, \{R_i\}_{i \geq 0} \rangle$, where R_0, R_1, \dots is an enumeration of all regular languages over the alphabets $\{0, 1\}^d$, $d \geq 1$, is decidable in EXPSPACE. Non-automaticity of Semënov arithmetic and undecidability of $\langle \mathbb{N}, 0, 1, +, 2^x, V_2 \rangle$ rule out the possibility of approaching this existential theory via automatic structures based on finite-state automata or via quantifier-elimination *à la* Cherlin and Point, since V_2 is definable as a regular language over pairs of number strings. Instead, our decidability result is based on a reduction to the language non-emptiness problem of a special class of *affine vector addition systems with states (affine VASS)*.

A VASS comprises a finite-state controller with a finite number of counters ranging over the natural numbers. In affine VASS, counters can be updated by affine functions $x \mapsto ax + b$ when taking a transition, provided that the resulting counter is non-negative. While reachability in affine VASS is decidable for a single counter [8], already in the presence of two counters reachability becomes undecidable [12]. Our reduction consequently requires a restricted class of affine VASS to obtain decidability. We call this class *restricted labelled affine VASS (restricted LAVASS)*. A restricted LAVASS is an affine VASS with d pairs of counters and hence $2d$ counters in total. For every pair, once the first counter achieves a non-zero value along a run, it keeps getting incremented at every transition; the second counter is only updated via affine functions $x \mapsto 2x$ and $x \mapsto 2x + 1$. A configuration consisting of a control state and $2d$ counter values is accepting whenever the control state is accepting and for every pair of counters, the first counter has the same value as the second counter. We give an EXPSPACE procedure for deciding emptiness of restricted LAVASS whose correctness proof is based on a counter elimination procedure in which we successively encode counters into a finite state space while preserving equi-non-emptiness. The EXPSPACE upper bound for existential generalised Semënov arithmetic follows from a reduction to language non-emptiness of a restricted LAVASS whose language encodes all solutions of the given formula. Note that obtaining an elementary upper bound is non-trivial since, e.g., the family of formulas Φ_n above shows that smallest solution of an existential formula can be non-elementary in bit-length.

As an application of our EXPSPACE upper bound for existential generalised Semënov arithmetic, we show that a certain class of string constraints with length constraints is decidable in EXPSPACE. This class allows for existentially quantifying over bit-strings, and to assert that the value of a string variable lies in a regular language, as well as Presburger-definable constraints over the lengths of the bit-strings stored in string variables and the numerical values of those variables (when viewed as encoding a number in binary). Decidability of this class was left open in [3]. We settle this open problem by showing that it can be reduced to the existential fragment of generalised Semënov arithmetic. Formulas of this class of string constraints appear widely in practice – in fact, all formulas in the extensive collection of standard real-world benchmark sets featured in [3, 4] lie in this class or can be reduced to formulas of this class by standard methods.

2 Preliminaries

2.1 Basic notation

By \mathbb{Z} and \mathbb{N} we denote the integers and non-negative integers, respectively. Given an $m \times n$ integer matrix A , we denote by $\|A\|_{1,\infty}$ the $(1, \infty)$ -norm of A , which is the maximum over the sum of the absolute values of the elements of the rows in A . For $\mathbf{b} \in \mathbb{Z}^m$, $\|\mathbf{b}\|_\infty$ is the largest absolute value of the numbers occurring in \mathbf{b} .

2.2 Numbers as strings and strings as numbers

Here and below, let $\Sigma = \{0, 1\}$ be a binary alphabet. Any string from Σ^* has an interpretation as a binary encoding of a natural number, possibly with an arbitrary number of leading zeros. Conversely, any natural number in \mathbb{N} can be converted into its bit representation as a string in Σ^* . Finally, by considering strings over $(\Sigma^k)^*$ for $k \geq 1$, we can represent k -tuples of natural numbers as strings over Σ^k , and *vice versa*. Formally, given $u = \mathbf{u}_n \mathbf{u}_{n-1} \dots \mathbf{u}_0 \in (\Sigma^k)^*$, we define the tuple of natural numbers corresponding to u in *most-significant digit first (msd)* notation as

$$\llbracket u \rrbracket := \sum_{i=0}^n 2^i \cdot \mathbf{u}_i.$$

Note that $\llbracket \cdot \rrbracket$ is surjective but not injective. We lift the definition of $\llbracket \cdot \rrbracket$ to sets in the natural way.

2.3 Generalised Semënov arithmetic

For technical convenience, the structures we consider in this paper are relational. We refer to the first-order theory of $\langle \mathbb{N}, 0, 1, +, 2^{(\cdot)} \rangle$ as *Semënov arithmetic*, where $+$ is the natural ternary addition relation, and $2^{(\cdot)}$ is the power relation of base two, consisting of all tuples $(a, b) \in \mathbb{N}^2$ such that $b = 2^a$. Semënov arithmetic is an extension of Presburger arithmetic, which is the first-order theory of the structure $\langle \mathbb{N}, 0, 1, + \rangle$. It is known that Semënov arithmetic is decidable and admits quantifier elimination [2, 7, 13].

For presentational convenience, atomic formulas of Semënov arithmetic are one of the following:

- linear equations of the form $a_1 \cdot x_1 + \dots + a_d \cdot x_d = b$, $a_i, b \in \mathbb{Z}$, and
- exponential equations of the form $x = 2^y$.

Here, x_1, \dots, x_d, y are arbitrary first-order variables. Clearly, richer atomic formulas such as $x + 2^{2^y} + y = z + 5$ can be defined from those basic class of atomic formulas, since, in this example, $x + 2^{2^y} + y = z + 5 \equiv \exists u \exists v u = 2^v \wedge v = 2^y \wedge x + u + y - z = 5$. Moreover, since we are interpreting numbers over non-negative integers, we can define the order relation in existential Semënov arithmetic. This enables us to without loss of generality assume that existential formulas of Semënov arithmetic are positive, since $\neg(x = y) \equiv x < y \vee y < x$ and $\neg(x = 2^y) \equiv \exists z z = 2^y \wedge \neg(x = z)$.

The main contribution of this paper is to show that the existential fragment of a generalisation of Semënov arithmetic is decidable. Subsequently, we write $\mathbf{0}$ to denote a tuple of zeros in any arbitrary but fixed dimension. *Generalised Semënov arithmetic* additionally allows for non-negated atomic formulas $R(x_1, \dots, x_k)$, where $R = \mathbf{0}^* \cdot L$ for some regular language $L \subseteq (\Sigma^k)^*$. We interpret R as $\llbracket R \rrbracket \subseteq \mathbb{N}^k$, and the additional leading zeros we

require ensure that $R = \llbracket R \rrbracket^{-1}$. Subsequently, we call a language $L \subseteq (\Sigma^k)^*$ *zero closed* if $L = \mathbf{0}^* \cdot L$. Given a formula $\Phi(x_1, \dots, x_n)$ of generalised Semënov arithmetic, we define $\llbracket \Phi \rrbracket \subseteq \mathbb{N}^d$ as the set of all satisfying assignments of Φ .

The size of an atomic formula $R(x_1, \dots, x_k)$ is defined as the number of states of the canonical minimal DFA defining R . For all other atomic formulas φ , we define their sizes $|\varphi|$ as the number of symbols required to write down φ , assuming binary encoding of numbers. The size $|\Phi|$ of an arbitrary existential formula Φ of generalised Semënov arithmetic is the sum of the sizes of all atomic formulas of Φ .

The full first-order theory of generalised Semënov arithmetic is known to be undecidable [11]. This follows from the undecidability of $\langle \mathbb{N}, 0, 1, +, 2^{(\cdot)}, V_2 \rangle$, where V_2 is the binary predicate such that $V_2(x, y)$ holds if and only if x is the largest power of 2 dividing y without remainder. Note that V_2 can be defined in terms of a regular language, cf. [6]. The central result of this paper is the following:

► **Theorem 1.** *The existential fragment of generalised Semënov arithmetic is decidable in EXPSPACE.*

2.4 Affine vector addition systems with states

A technical tool for our decidability results is a tailor-made class of *labelled affine vector addition systems with states (LAVASS)*. Formally, an LAVASS is a tuple $V = \langle Q, d, \Sigma, \Delta, \lambda, q_0, F, \Phi \rangle$, where Q is a finite set of *control states*, $d \geq 0$ is the *dimension of V* , Σ is a *finite alphabet*, $\Delta \subseteq Q \times \mathcal{P}(\Sigma) \times Q$ is a finite set of *transitions*, $\lambda: \Delta \rightarrow \text{Ops}^d$ is the *update function*, where $\text{Ops} \subseteq \mathbb{Z}[x]$ is the set of all affine functions over a single variable, $q_0 \in Q$ is the *initial control state*, $F \subseteq Q$ is the set of *final control states*, and Φ is a quantifier-free formula of Presburger arithmetic $\Phi(x_1, \dots, x_d)$ that specifies a infinite set $\llbracket \Phi \rrbracket \subseteq \mathbb{N}^d$ of *final counter values*. Note that when $d = 0$ then V is essentially a non-deterministic finite automaton.

The set of *configurations* of V is $C(V) := Q \times \mathbb{N}^d$. The *initial configuration* of V is $c_0 = (q_0, 0, \dots, 0)$, and the set of *final configurations* is $C_f(V) := \{(q_f, \mathbf{v}) : q_f \in F, \mathbf{v} \in \llbracket \Phi \rrbracket\}$. For an update function $\lambda: \Delta \rightarrow \text{Ops}^d$, we define

$$\|\lambda\| := \max\{|a| + |b| : \lambda(t) = (f_1, \dots, f_d), f_i = ax + b, 1 \leq i \leq d, t \in \Delta\}.$$

We define the size $|V|$ of an LAVASS $V = \langle Q, d, \Sigma, \Delta, \lambda, q_0, F, S_f \rangle$ as

$$|V| := |Q| + |\Delta| \cdot (d + 1) \cdot \log(\|\lambda\| + 1) + |\Phi|.$$

An LAVASS induces an (infinite) labelled directed *configuration graph* $G = (C(V), \rightarrow)$, where $\rightarrow \subseteq C(V) \times \Sigma \times C(V)$ such that $c \xrightarrow{a} c'$ if and only if $c = (q, m_1, \dots, m_d)$ and $c' = (q', m'_1, \dots, m'_d)$ and there is $t = (q, A, q') \in \Delta$ such that $a \in A$, $\lambda(t) = (f_1, \dots, f_d)$, and $m'_i = f_i(m_i)$ for all $1 \leq i \leq d$. We lift the definition of \rightarrow to words $w = a_1 \cdots a_n \in \Sigma^*$ in the natural way, and thus write $c \xrightarrow{w} c'$ whenever $c \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} c_{n-1} \xrightarrow{a_n} c'$ for some $c_1, \dots, c_{n-1} \in C$. The *language* $L(V) \subseteq \Sigma^*$ of V is defined as

$$L(V) := \{w \in \Sigma^* : c_0 \xrightarrow{w} c_f, c_f \in C_f(V)\}.$$

If we are interested in runs of LAVASS, we write $\pi = c_1 \xrightarrow{t_1} c_2 \xrightarrow{t_2} \cdots \xrightarrow{t_{n-2}} c_{n-1} \xrightarrow{t_{n-1}} c_n$ to emphasise the sequence of configurations and transitions taken. For $1 \leq i \leq j \leq n$, we denote by $\pi[i, j]$ the subsequence $c_i \xrightarrow{t_i} c_{i+1} \xrightarrow{t_{i+1}} \cdots \xrightarrow{t_{j-1}} c_j$. We denote by $\text{val}(\pi, x_i)$ the value m_i of the i -th counter in the last configuration of π .

The *non-emptiness problem* for an LAVASS is to decide whether $L(V) \neq \emptyset$. Affine VASS are a powerful class of infinite state systems, and even in the presence of only two counters and $\Phi(x_1, x_2) \equiv x_1 = x_2$, the emptiness problem is undecidable [12]. In Section 4, we identify a syntactic fragment of LAVASS for which non-emptiness can be decided in EXPSPACE.

We briefly discuss closure properties of LAVASS and show that they are closed under union and intersection, and restricted kinds of homomorphisms and inverse homomorphisms, using essentially the standard constructions known for finite-state automata. Let $V_i = \langle Q_i, d_i, \Sigma, \Delta_i, \lambda_i, q_0^{(i)}, F_i, \Phi_i \rangle$, $i \in \{1, 2\}$, be two LAVASS.

► **Proposition 2.** *The languages of LAVASS are closed under union and intersection. Moreover, for V such that $L(V) = L(V_1) \cap L(V_2)$, we have $|V| \leq |V_1| \cdot |V_2|$.*

Proof. This result can be obtained by generalising the standard constructions known from non-deterministic finite-state automata. The set of control states of the LAVASS V accepting the intersection of LAVASS V_1 and V_2 is $Q_1 \times Q_2$. The dimension of V is the sum of the dimensions of V_1 and V_2 , and the counters of V_1 and V_2 get independently simulated in the counters of V . Upon reading an alphabet symbol a , the LAVASS V then simultaneously simulates the respective transitions of V_1 and V_2 for a ; further details are relegated to Appendix A.1. ◀

Note that since LAVASS languages contain regular languages, Proposition 2 in particular enables us to intersect LAVASS languages with regular languages.

Let Σ, Γ be two finite alphabets. Recall that a homomorphism $h: \Gamma^* \rightarrow \Sigma^*$ is fully defined by specifying $h(a)$ for all $a \in \Gamma$. We call h a *projection* if $|h(a)| = 1$ for all $a \in \Gamma$.

► **Proposition 3.** *The languages of LAVASS are closed under projections and inverses of projections.*

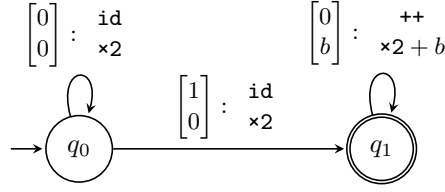
Proof. Let $h: \Gamma^* \rightarrow \Sigma^*$ be a projection. Given an LAVASS $V = \langle Q, d, \Sigma, \Delta, \lambda, q_0, F, S_f \rangle$, to obtain closure under projections replace any $t = (q, A, q') \in \Delta$ with $t' = (q, h(A), q')$, and set $\lambda(t') := \lambda(t)$. To obtain closure under inverse projections, replace any $t = (q, A, q') \in \Delta$ with $t' = (q, h^{-1}(A), q')$ and set $\lambda(t') := \lambda(t)$. ◀

3 Reducing Semënov arithmetic to restricted LAVASS

Let $\Sigma = \{0, 1\}$. In this section, we show how given a quantifier-free formula $\Phi(x_1, \dots, x_d)$ of Semënov arithmetic, we can construct an LAVASS V over the alphabet $\Sigma_d := \{0, 1\}^d$ such that $\llbracket L(V) \rrbracket = \{\mathbf{x} \in \mathbb{N}^d : \Phi(\mathbf{x})\}$. We will subsequently observe that the resulting LAVASS enjoy strong structural restrictions, giving rise to the fragment of restricted LAVASS that we then formally define. For our purposes, it will be sufficient to primarily focus on formulas Φ of Semënov arithmetic which are conjunctions of atomic formulas.

In the previous section, we stated that for presentational convenience, the atomic formulas of Semënov arithmetic are either linear equations or exponential equations of the type $x = 2^y$. It is well known that the sets of solutions of systems of linear equations $A \cdot \mathbf{x} = \mathbf{b}$, where $\mathbf{x}, \mathbf{b} \in \mathbb{Z}$ can be represented by a regular language and are hence definable via an LAVASS.

► **Lemma 4** ([14], see also [9, Eqn. (1)]). *Given a system of equations $\Phi \equiv A \cdot \mathbf{x} = \mathbf{b}$ with $A \in \mathbb{Z}^{m \times d}$ and $\mathbf{b} \in \mathbb{Z}^m$, there is a DFA V with at most $2^m \cdot \max\{\|A\|_{1,\infty}, \|\mathbf{b}\|_\infty\}^m$ states such that $L(V)$ is zero-closed and $\llbracket L(V) \rrbracket = \llbracket \Phi \rrbracket$.*



■ **Figure 1** A gadget with two counters for the exponential equation $x = 2^y$, where $b \in \{0, 1\}$.

The crucial part, which requires the power of LAVASS, are exponential equations $x = 2^y$. An LAVASS V with two counters and $\llbracket L(V) \rrbracket = \llbracket x = 2^y \rrbracket$ is depicted in Figure 1. Control-states are depicted as circles and transitions as arrows between them. The vector before the colon is the alphabet symbol read. For instance, the transition from q_0 to q_1 reads the alphabet symbol $(1, 0) \in \{0, 1\}^2$. After a colon, we display the counter operations when reading the alphabet symbol, the operation on the first counter is displayed on the top and the operation on the second counter on the bottom. Here and subsequently, for presentational convenience, **id** is the identity function $x \mapsto x$, and $\times 2$ and $\times 2 + 1$ are the functions $x \mapsto 2x$ and $x \mapsto 2x + 1$, respectively. Thus, the transition from q_0 to q_1 applies the identity function on the first counter, and the function $x \mapsto 2x$ on the second counter.

The idea behind the gadget in Figure 1 is as follows. For an example, suppose that $y = 5$, then $x = 32$, and in binary the sequence of digits of x and y looks as follows:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \cdots \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Since $x = 2^y$, we have that $x \in 0^*10^*$, and the number of trailing zeros of x is equal to the value of y . Thus, once a 1 in the binary representation of x has been read, the first counter in the gadget of Figure 1 keeps incrementing and counts the number of trailing zeros of x . At the same time, the second counter in the gadget of Figure 1 keeps the value 0 until it reads the first 1 of the binary expansion of y , since $2 \cdot 0 = 0$. It then computes the value of y in binary on the second counter by multiplying the value of the second counter by 2 when reading a zero, and multiplying by two and adding one when reading a one. The LAVASS in Figure 1 only accepts when the first and the second counter have the same value, i.e., when the number of trailing zeros of the binary expansion of x equals the value of y , as required.

A closer look at the gadget constructed in Figure 1 reveals a number of important structural properties:

- (i) all operations performed on the first counter are either the identity map **id** or increments **++**;
- (ii) all operations performed on the second counter are affine updates $\times 2$ and $\times 2 + 1$;
- (iii) once the first counter gets incremented on a run, it gets incremented at every subsequent transition; and
- (iv) only counter configurations in which the value of the first counter equals the value of the second counter are accepted.

Those properties are crucial to obtain decidability of (generalised) existential Semënov arithmetic.

► **Definition 5.** An LAVASS is restricted if and only if it has an even number of $2d$ counters called x_i, y_i , $1 \leq i \leq d$, such that every counter pair (x_i, y_i) adheres to the above Properties (i)–(iv), where x_i is regarded as the first counter and y_i as the second counter and the set of final counter values is defined by $\Phi \equiv \bigwedge_{1 \leq i \leq d} x_i = y_i$.

For convenience, when referring to the counters in a pair, we subsequently refer to the first counter as its *x-counter* and to the second counter as its *y-counter*. We will usually write m for the value of the *x-counter* and n for the value of the *y-counter*. The following lemma is an immediate result of Figure 1 together with the previous discussion.

► **Lemma 6.** *There is a fixed restricted LAVASS V of dimension two such that $L(V)$ is zero closed and $\llbracket L(V) \rrbracket = \llbracket x = 2^y \rrbracket$.*

Next, by following the constructions in Section 2.4 and noting that the counters of different LAVASS are simulated independently we obtain the following result.

► **Proposition 7.** *The languages of restricted LAVASS are closed under union, intersection, projection, and inverse projections.*

Thus, we are ready to present the central lemma of this section.

► **Lemma 8.** *Consider a positive conjunctive formula of Semënov arithmetic*

$$\Phi(\mathbf{x}) \equiv A \cdot \mathbf{x} = \mathbf{b} \wedge \bigwedge_{i \in I} x_i = 2^{y_i},$$

where $A \in \mathbb{Z}^{m \times n}$, $\mathbf{b} \in \mathbb{Z}^m$, I is a finite index set, and x_i and y_i are variables from \mathbf{x} . There is a restricted LAVASS V of dimension $2|I|$ and of size $(\|A\|_{1,\infty} + \|\mathbf{b}\|_\infty + 2)^{O(m+|I|)}$ such that $\llbracket L(V) \rrbracket = \llbracket \Phi \rrbracket$.

This lemma follows from the combination of Lemma 4, Lemma 6, and Proposition 7.

Finally, for technical convenience, we assume that for a restricted LAVASS, we have $|Q| \geq 2$. This is with no loss of generality, since if $|Q| = 1$ then deciding non-emptiness is trivial (the restricted LAVASS has non-empty language if and only if the only control state is accepting).

The next sections will be devoted to the proof of the main result of this paper on restricted LAVASS.

► **Proposition 9.** *Language emptiness of a restricted LAVASS V with $2d$ counters is decidable in $\text{NSPACE}(|V| \cdot 2^{O(d)})$.*

This proposition enables us to prove Theorem 1, by appealing to Lemma 8. Further details are relegated to Appendix A.2.

4 Certificates witnessing non-emptiness of restricted LAVASS

We now show that language emptiness for restricted LAVASS is decidable in exponential space. Clearly, this problem reduces to deciding whether a given restricted LAVASS has an accepting run, but witnessing runs may be of non-elementary length. To overcome this problem, we define an abstraction for configurations of restricted LAVASS. Abstract configurations store residue classes of counter values, as well as some further information that is required to witness the existence of concrete accepting runs. Before giving the formal definition, we provide some high level intuition that leads to our definition of abstract configurations. Next, we introduce reachability certificates, which are abstract runs with certain further properties. We argue that the existence of *witnessing certificates*, which are special kinds of reachability certificates witnessing that the language of an LAVASS is non-empty, are decidable in EXPSPACE. The last two sections then establish that witnessing certificates actually witness non-emptiness of restricted LAVASS.

4.1 Key observations

Given a *restricted* LAVASS V in dimension d , assuming that $L(V) \neq \emptyset$, there is a run π from an initial configuration c to a final configuration c' . With no loss of generality, throughout this section, we assume that $\text{val}(c', x_i) \geq \text{val}(c', x_{i+1}) > 0$ for all $1 \leq i < d$. In particular, this implies that every counter gets incremented at least once along a path witnessing non-emptiness.

Our first observation is that if, along π , a counter y_i achieves the first time a non-zero value by taking a $\times 2+1$ labeled transition, then the length of the remaining segment of π is bounded by $O(\log(m_i + 1))$, where m_i is the value of counter x_i before the transition is taken. The reason is that, once y_i has non-zero value, its value is at least doubling for every following transition taken. Hence if π is “long” then along π there will be loops incrementing a counter x_i before the corresponding y_i achieves non-zero value.

In the latter scenario, we may actually, subject to some bookkeeping, discard concrete values of x_i and y_i and only store their residue classes modulo ℓ_i , where ℓ_i is the length of the first loop incrementing x_i along π . In particular, if we are given a non-accepting run π' such that $\text{val}(\pi', x_i) \equiv \text{val}(\pi', y_i) \pmod{\ell_i}$ and $\text{val}(\pi', x_i) < \text{val}(\pi', y_i)$ then π' can be turned into a run π'' where $\text{val}(\pi'', x_i) = \text{val}(\pi'', y_i)$ by iterating the loop of length ℓ_i .

There are, however, some further subtleties that need to be taken care of. Consider the segment π' of π between the first transition labeled by $++$ on x_i and the first transition labeled by $++$ on x_{i+1} . If π' contains no loop then we are in a situation where the first loop incrementing x_i is also the first loop incrementing x_{i+1} . This means that the values of x_i and x_{i+1} get paired together, and hence, for an accepting run, also the values of y_i and y_{i+1} are paired together. In our approach, we deal with such circumstances by introducing so-called *y-constraints*. A *y-constraint* of the form $y_i - y_{i+1} = \delta_i$ for some constant $\delta_i \in \mathbb{N}$ asserts that the counters y_{i+1} and y_i must eventually have constant difference δ_i along a run.

Otherwise, if π' above contains a loop, the difference between the values of x_i and x_{i+1} is not necessarily constant, but lower-bounded by the length δ_i of the loop-free sub path of π' . Thus, in an accepting run, the difference between y_i and y_{i+1} must also be at least δ_i , which is asserted by a *y-constraint* of the form $y_i - y_{i+1} \geq \delta_i$.

4.2 Abstract configurations for restricted LAVASS

Our decision procedure for non-emptiness of restricted LAVASS is based on reducing this problem to a reachability problem in a carefully designed finite-state abstraction of the state-space of LAVASS. Throughout this section, let $V = \langle Q, 2d, \Sigma, \Delta, \lambda, q_0, F, \Phi \rangle$ be a restricted LAVASS. We first define the state space of the abstracted LAVASS.

► **Definition 10.** *An abstract configuration is a tuple*

$$\alpha = (q, m_1, n_1, \dots, m_d, n_d, u_1, u_2, \dots, u_{d-1}, \ell_1, \dots, \ell_d) \\ \in Q \times (\mathbb{N} \cup \{\perp\})^{2d} \times \mathbb{N}^{d-1} \times (\mathbb{N} \cup \{\top\})^d,$$

such that $m_i, n_i \in [0, 2dM_i] \cup \{\perp\}$ and $u_i \in [0, U_i]$ and $\ell_i \in [0, M_i - 1] \cup \{\top\}$, where

- $M_i := \lfloor |Q|^{((1/8) \cdot 32^{i-1} + 1)} \rfloor$; and
- $U_i := |Q|^{(32^{i-1} + 4)}$.

The idea is that m_i, n_i store the residue classes modulo ℓ_i of the counter pair x_i, y_i respectively, where the value \top for ℓ_i acts as an indicator that we are storing actual values and not residue classes. The value \perp for some x_i or y_i indicates that the counter has not yet been initialised. If for an update function f , $f = ++$ or $f = \times 2+1$ then $f(\perp) := 1$; otherwise $f(\perp) := \perp$, and we stipulate that $\perp \pmod n = \perp$. The value of u_i in an abstract configuration carries the

current difference between the value of the counters y_i and y_{i+1} . This difference is potentially unbounded, however for our purposes it suffices to only store its value if it is less than U_i , and to indicate the fact that it is at least U_i by the value $u_i = U_i$.

We denote the (finite) set of all abstract configurations of V by $A(V)$. Let us now define a transition relation $\xrightarrow{t} \subseteq A(V) \times \Delta \times A(V)$ such that $\alpha \xrightarrow{t} \alpha'$, $t = (q, a, q') \in \Delta$ if and only if:

- $\alpha = (q, m_1, n_1, \dots, u_{d-1}, \ell_1, \dots, \ell_d)$ and $\alpha' = (q', m'_1, n'_1, \dots, u'_{d-1}, \ell_1, \dots, \ell_d)$;
- $\lambda(t) = (f_{x_1}, f_{y_1}, \dots, f_{x_d}, f_{y_d})$;
- $f_{x_i} = ++$ for all i such that $m_i \neq \perp$;
- if $\ell_i \neq \top$, $m'_i = f_{x_i}(m_i) \bmod \ell_i$ and $n'_i = f_{y_i}(n_i) \bmod \ell_i$;
- if $\ell_i = \top$, $m'_i = f_{x_i}(m_i)$ and $n'_i = f_{y_i}(n_i)$; and
- for all $i \in \{1, \dots, d-1\}$,

$$u'_i = \begin{cases} \min(2u_i + 1, U_i) & \text{if } f_{y_i} = \times 2+1, f_{y_{i+1}} = \times 2 \\ \min(2u_i, U_i) & \text{if } f_{y_i} = f_{y_{i+1}} \\ \min(2u_i - 1, U_i) & \text{if } f_{y_i} = \times 2, f_{y_{i+1}} = \times 2+1. \end{cases}$$

Assuming that the value of y_i is at least the value of y_{i+1} , which we will always ensure, the definition of how to update u_i ensures that it exactly stores the difference $y_i - y_{i+1}$ unless the difference becomes too large, in which case it is levelled off at U_i .

An abstract configuration path is a sequence of abstract configurations and transitions of the form $R = \alpha_1 \xrightarrow{t_1} \alpha_2 \xrightarrow{t_2} \dots \xrightarrow{t_{n-1}} \alpha_n$.

Given two consecutive y -counters y_i, y_{i+1} and $\delta_i \in \mathbb{N}$, we say that $y_i - y_{i+1} = \delta_i$ and $y_i - y_{i+1} \geq \delta_i$ are y -constraints. Let Y be a set of y -constraints, an abstract configuration $\alpha = (q, m_1, n_1, \dots, u_1, \dots, u_{d-1}, \ell_1, \dots, \ell_d)$ respects Y whenever

- $u_i \geq \delta_i$ for all constraints of type $y_i - y_{i+1} \geq \delta_i$ in Y ,
- and $u_i < U_i$ and $u_i = \delta_i$ for all constraints $y_i - y_{i+1} = \delta_i$ in Y .

We say that α_f is a *final abstract configuration respecting* Y whenever $q \in F$, $m_i = n_i$ for all $1 \leq i \leq d$, and α_f respects Y .

4.3 Witnessing certificates

While any concrete accepting run of an LAVASS gives rise to an abstract configuration path ending in an accepting abstract configuration, the converse does not hold. This motivates the introduction of reachability and witnessing certificates, which are special abstract configuration paths that carry further information that eventually enables us to derive from a witnessing certificate a concrete accepting run of an LAVASS.

A *reachability certificate* is a tuple (R, X, Y, L) such that $R = \alpha_1 \xrightarrow{t_1} \alpha_2 \xrightarrow{t_2} \dots \xrightarrow{t_{n-1}} \alpha_n$ is an abstract configuration path, and $X, Y, L: \{1, \dots, d\} \rightarrow \{1, \dots, n\}$. Here, $X(i)$ and $Y(i)$ indicate the position where the x_i -counter and y_i -counter obtain a value different from \perp for the first time. Moreover, $L(i)$ is the position where a loop of length ℓ_i can be found. Formally, (R, X, Y, L) is required to have the following properties:

- (a) $\alpha_1 = (q_0, \perp, \dots, \perp, 0, \dots, 0, \ell_1, \dots, \ell_d)$ and if $\ell_i = \top$ then $\ell_j = \top$ for all $i < j \leq d$;
- (b) $\lambda(x_i, t_{X(i)-1}) = ++$ and $\lambda(y_i, t_{Y(i)-1}) = \times 2+1$ for all $1 \leq i \leq d$;
- (c) $\lambda(x_i, t_j) = \text{id}$ for all $1 \leq j < X(i) - 1$;
- (d) $\lambda(y_i, t_j) = \times 2$ for all $1 \leq j < Y(i) - 1$;
- (e) X, Y, L are monotonic; and
- (f) for all $1 \leq i \leq d$, if $\ell_i \neq \top$ then
 - $X(i) \leq L(i) < Y(i)$, and
 - there is a simple $\alpha_{L(i)}$ -loop $\alpha_{L(i)} \xrightarrow{t'_1} \alpha'_2 \xrightarrow{t'_2} \dots \xrightarrow{t'_{\ell_i-1}} \alpha'_{\ell_i-1} \xrightarrow{t'_{\ell_i}} \alpha_{L(i)}$ of length ℓ_i .

Those conditions can be interpreted as follows. Condition (a) asserts that the certificate starts in an initial abstract configuration. We require that \top monotonically propagates since the absence of a loop for counter x_i implies that the remainder of a path is short, hence we can afford to subsequently store actual counter values and not residue classes. Conditions (b), (c) and (d) assert that $X(i)$ and $Y(i)$ are the first position where the counters x_i, y_i hold a value different from \perp . Condition (e) states that the counters x_{i+1}, y_{i+1} do not carry a value different from \perp before the counters x_i and y_i , respectively. Condition (f) implies that, if $\ell_i \neq \top$ then between the first update for counter x_i and the first update for counter y_i there is a position $L(i)$ where we can find a loop in the abstract configurations of length ℓ_i . Notice that if $x_j = \perp$ or $y_j = \perp$ in $\alpha_{L(i)}$ then x_j and y_j remain to hold \perp along this loop, i.e., this loop does not update counters that have not been initialised already.

Given R , the set of y -constraints induced by R is the smallest set containing

- $y_i - y_{i+1} \geq \delta_i$, where $\delta_i := X(i+1) - X(i)$ if there is a j such that $X(i) \leq L(j) < X(i+1)$; and
 - otherwise $y_i - y_{i+1} = \delta_i$, where $\delta_i := X(i+1) - X(i)$,
- for all $1 \leq i < d$ such that $\ell_i \neq \top$.

We introduce some further notation. Given a reachability certificate (R, X, Y, L) , we denote by $\pi(R)$ the run corresponding to R in the configuration graph of V , with the initial configuration $(q_0, 0, 0, \dots, 0, 0)$. Given indices $1 \leq i \leq j \leq n$, we denote by $R[i, j]$ the segment $\alpha_i \xrightarrow{t_i} \alpha_{i+1} \cdots \xrightarrow{t_{j-1}} \alpha_j$ of R , and by $R[i] := \alpha_i$. We say that R is a *witnessing certificate* if, for $a \leq d$ being the largest index such that $\ell_a \neq \top$:

- $R[1, Y(a)]$ is a simple path and $n - Y(a) \leq 2dM_{d+1}$;
- α_n is a final abstract configuration respecting the set of induced y -constraints; and
- $val(\pi(R), x_a) \leq val(\pi(R), y_a)$.

Sometimes we will speak of witnessing certificates *restricted* to a set of counters. By that we mean a witnessing certificates where the relevant Conditions (a)–(f) are only required for that set of counters.

In the next section, we prove the following theorem that will enable us to give a proof for Proposition 9.

► **Theorem 11.** *The language of a restricted LAVASS V is non-empty if and only if there exists a witnessing certificate for V .*

5 Witnessing certificates witness non-emptiness of restricted LAVASS

In this section we prove Theorem 11. The proof is split into the two directions. First, we argue that the existence of a witnessing certificate for an LAVASS implies that the language of the LAVASS is non-empty. Subsequently, we show the converse direction.

► **Proposition 12.** *If there exists a witnessing certificate for a restricted LAVASS V then $L(V) \neq \emptyset$.*

The idea behind the proof of Proposition 12 is that, from a witnessing certificate (R, X, Y, L) of an LAVASS V , we obtain a sequence of runs of V such that the final run in that sequence is an accepting run of V . Initially, we obtain a run that ends in a configuration where the counters are in a congruence relation. We then carefully pump the simple loops pointed to by L , beginning from the last counter and working towards the first. The formal proof is presented in Appendix A.3.

We now turn towards the converse direction and show that we can obtain a witnessing certificate from a run witnessing non-emptiness.

► **Proposition 13.** *If $L(V) \neq \emptyset$ for a restricted LAVASS V then there exists a witnessing certificate for V .*

We begin by defining a function that turns a configuration from $C(V)$ into an abstract configuration. This function is parameterised by $\ell_1, \dots, \ell_d \in \mathbb{N}_+ \cup \{\top\}$:

$$f_V((q, m_1, n_1, \dots, m_d, n_d), \ell_1, \dots, \ell_d) := (q, m_1 \circ \ell_1, n_1 \circ \ell_1, \dots, m_d \circ \ell_d, n_d \circ \ell_d, \min(n_1 - n_2, U_1), \dots, \min(n_{d-1} - n_d, U_{d-1}), \ell_1, \dots, \ell_d).$$

Here, $m \circ \ell := \perp$ if $m = 0$; $m \circ \ell := m \bmod \ell$ if $\ell \in \mathbb{N}_+$; and $m \circ \ell := m$ if $\ell = \top$. We lift the definition of f_V to concrete runs π in the natural way, and write $f_V(\pi, \ell_1, \dots, \ell_d)$ for the resulting sequence of abstract configurations. Let $\pi = c_1 \xrightarrow{t_1} c_2 \cdots \xrightarrow{t_{n-1}} c_n$ be a run witnessing $L(V) \neq \emptyset$. We show how to obtain a witnessing certificate R from π . Without loss of generality, in c_n we have $m_1 \geq m_2 \geq \dots \geq m_d > 0$.

To this end, we show how from the accepting run π we can iteratively define a sequence $R_0, R_1, R_2, \dots, R_d$ of abstract runs and identify the required $\ell_1, \dots, \ell_d \in \mathbb{N}_+ \cup \{\top\}$ and X, Y, L such that (R_d, X, Y, L) is a reachability certificate. Let $X(i) := j$ such that j is the first position in π where the value of counter x_i is non-zero; analogously define $Y(i)$ to be the first position where the value of y_i is non-zero. Clearly, X, Y are monotonic and $X(i) \leq Y(i)$, for all $1 \leq i \leq d$. Otherwise, if a counter y_i gets initialised before the counter x_i in π , it must be the case that $n_i > m_i$ in c_n and therefore c_n cannot be an accepting configuration.

Recall that π has length n . In our proof, the subsequent technical lemma will allow us to conclude that, if for a counter pair x_i, y_i the y_i counter gets updated shortly after the x_i counter then the run will end shortly after and counter pairs x_j, y_j for $j \geq i$ will consequently have small values.

► **Lemma 14.** *If $Y(i) - X(i) \leq dM_i$ for some $1 \leq i \leq d$ then $n - Y(i) < dM_i$, so $m_j, n_j \leq 2dM_i$ in c_n for all $i \leq j \leq d$.*

Proof. We have that $Y(i) - X(i) \leq dM_i$ implies that $\text{val}(\pi[1, Y(i)], x_i) \leq dM_i + 1$, and since $\text{val}(\pi[1, Y(i) + k], y_i) \geq 2^k$ we get that:

- $\text{val}(\pi, y_i) \geq 2^{n-Y(i)}$; and
- $\text{val}(\pi, x_i) \leq dM_i + n - Y(i) + 1$.

Assume that $n - Y(i) \geq dM_i$. Then, $2^{n-Y(i)} - (dM_i + n - Y(i) + 1) > 2^{n-Y(i)} - (2n - 2Y(i) + 1) > 0$, if $n - Y(i) \geq 3$. However, π is an accepting path, so $\text{val}(\pi, x_i) = \text{val}(\pi, y_i)$, and we get a contradiction. Thus, we must have that $n - Y(i) < dM_i$ which implies that $\text{val}(\pi, x_i) \leq 2dM_i$, so $m_i = n_i \leq 2dM_i$ and for any $j, i < j \leq d$, $m_j \leq m_i$ and $n_j \leq n_i$, so $m_j, n_j < 2M_i$ in c_n , for all $i \leq j \leq d$ since π is an accepting path. ◀

Let $R_0 := f_V(\pi, 1, 1, \dots, 1)$. Note that R_0 together with X and Y as defined above adhere to Conditions (a)–(e) of reachability certificates.

Suppose R_{i-1} and $\ell_1, \dots, \ell_{i-1}$ have been constructed. If $i > 1$, and $L(i-1) \geq X(i)$ or $\ell_{i-1} = \top$ then we choose $\ell_i := \ell_{i-1}$, $L(i) = L(i-1)$ and $R_i := f_V(\pi, \ell_1, \dots, \ell_i, 1, \dots, 1)$. Otherwise, we distinguish two cases.

- $Y(i) - X(i) < dM_i$: we choose $\ell_i := \top$ and $L(i) := X(i)$.
- $Y(i) - X(i) \geq dM_i$: then there is a segment in $R_{i-1}[X(i), Y(i)]$ of length greater than M_i on which no x -counter has its first ++ transition. Let N_i be the number of different abstract configurations on this segment. Since $\ell_{i-1} \neq \top$ we know that m_j, n_j can take at most M_j different values for all $1 \leq j < i$, as they can either be \perp or a residue class

29:12 Semënov Arithmetic, Affine VASS, and String Constraints

modulo M_j . Also, for all $i \leq j \leq d$ the values of m_j, n_j have a constant value, either 0 or \perp , on this segment, and $u_i = \dots = u_d = 0$ in all abstract configurations of this segment. So

$$\begin{aligned}
N_i &\leq |Q| \prod_{1 \leq j < i} M_j^2 \cdot U_j \\
&\leq |Q| \prod_{1 \leq j < i} |Q|^{(1/4) \cdot 32^{j-1} + 2 + 32^{j-1} + 4} \\
&\leq |Q|^{(1/3968)(5 \cdot 32^i - 23968) + 6i + 1} \\
&< |Q|^{(1/8) \cdot 32^{i-1} + 1} \\
&= M_i
\end{aligned}$$

By the pigeonhole principle, there is a smallest k , $X(i) \leq k < Y(i)$, $\ell < M_i$, and a simple loop $\alpha_k \xrightarrow{t_k} \dots \xrightarrow{t_{k+\ell}} \alpha_{k+\ell+1} = \alpha_k$ in R_{i-1} . We choose $L(i) := k$, $\ell_i := \ell$ and let $R_i := f_V(\pi, \ell_1, \dots, \ell_i, 1, \dots, 1)$.

By construction, (R_d, X, Y, L) is a reachability certificate. It remains to turn it into a witnessing certificate. In particular, this requires to remove loops from R_d , to ensure that the final segment of R_d is short, and to establish that R_d is consistent with the implied y -constraints.

Let $R := R_d = f_V(\pi, \ell_1, \dots, \ell_d)$ and a be the largest index such that $\ell_a \neq \top$. In order to make R loop-free, we iterate the following process:

- Identify the first simple loop $\alpha_k \xrightarrow{t_k} \dots \xrightarrow{t_{k+\ell}} \alpha_{k+\ell+1}$ in $R[1, Y(a)]$ and replace it by α_k ; observe that for $I := \{k+1, \dots, k+\ell\}$, we have $I \cap \{X(i), Y(i), L(i) : 1 \leq i \leq d\} = \emptyset$ since $\alpha_{X(i)-1} \xrightarrow{t_{X(i)-1}} \alpha_{X(i)}$ occurring in R_d means that x_i has value \perp in $\alpha_{X(i)-1}$ and a value different from \perp in $\alpha_{X(i)}$, and thus $\alpha_{X(i)}$ cannot be part of a loop; the same argument applies to any $Y(i)$. Finally, since $L(i)$ was chosen as the index of the first configuration of the first cycle appearing after $X(i)$, we have $L(i) \notin I$ for all $1 \leq i \leq d$ as well.
- Update X, Y, L such that for all i such that $X(i) > k$, $X(i) := X(i) - \ell$, and analogously $Y(i) := Y(i) - \ell$ and $L(i) := L(i) - \ell$ for the respective i .

This process guarantees that $R[1, Y(a)]$ is loop-free. It is easy to verify that (R, X, Y, L) obtained in this way is a reachability certificate and that the last abstract configuration of R is accepting.

We now show that the y -constraints induced by R are valid in the final configuration of R . To this end, we first show that for all $1 \leq i \leq d$ such that $\ell_i \neq \top$, $X(i+1) - X(i) < U_i$. Consider the simple path $\alpha_{X(i)} \xrightarrow{t_{X(i)}} \alpha_{X(i)+1} \xrightarrow{t_{X(i)+1}} \dots \xrightarrow{t_{X(i+1)-1}} \alpha_{X(i+1)}$. If $Y(i) \geq X(i+1)$ then clearly $X(i+1) - X(i) \leq N_i < U_i$, where N_i is defined as above. Otherwise, there is a $k \in \mathbb{N}$ such that the path decomposes as

$$\alpha_{X(i)} \xrightarrow{t_{X(i)}} \dots \alpha_{Y(i)} \xrightarrow{t_{Y(i)}} \dots \alpha_{Y(i)+k} \xrightarrow{t_{Y(i)+k}} \dots \xrightarrow{t_{X(i+1)-1}} \alpha_{X(i+1)}$$

and

- $u_i = 0$ in all abstract states α_j with $X(i) \leq j \leq Y(i)$;
- $u_i = U_i$ in all abstract states α_j with $Y(i) + k \leq j \leq X(i+1)$; and
- $k \leq \log U_i$.

Thus, the maximum length of $R[X(i), X(i+1)]$ is bounded by:

$$\begin{aligned}
& N_i \cdot M_i + \log U_i + N_i \cdot 2M_i \\
& \leq 2 \cdot M_i^3 + \log U_i \\
& \leq |Q|^{(3/8) \cdot 32^{i-1} + 4} + |Q|^{5(i-1)+1} + 4|Q| \\
& < |Q|^{32^{i-1} + 4} \\
& = U_i
\end{aligned}$$

We can now show that R respects the induced y -constraints. Fix some $1 \leq i \leq d$ such that $\ell_i \neq \top$. We distinguish two cases.

- There is no $1 \leq j \leq a$ such that $X(i) \leq L(j) \leq X(i+1)$. Thus, we know that $y_i - y_{i+1} = \delta_i$ is in the set of induced y -constraints. Also, $\text{val}(\pi, y_i) - \text{val}(\pi, y_{i+1}) = \text{val}(\pi, x_i) - \text{val}(\pi, x_{i+1}) = X(i+1) - X(i) = \delta_i$ since we did not remove any abstract loops on the segment of R_d between the first $++$ update for x_i and the first $++$ update for x_{i+1} . Finally, since $\delta_i < U_i$ by the above argument, we conclude that $u_i = \delta_i$ in the last abstract configuration $R[n]$ of R .
- Otherwise, $X(i) \leq L(i) \leq Y(i)$, so $y_i - y_{i+1} \geq \delta_i$ is in the set of induced y -constraints. However, $\text{val}(\pi, y_i) - \text{val}(\pi, y_{i+1}) = \text{val}(\pi, x_i) - \text{val}(\pi, x_{i+1}) \geq X(i+1) - X(i) = \delta_i$ and again because $\delta_i < U_i$ we can conclude that $u_i \geq \delta_i$ in $R[n]$.

This establishes that the y -constraints are satisfied. Let n be the index of the last abstract configuration of R . For the final step, we now argue that $\text{val}(\pi(R), x_a) \leq \text{val}(\pi(R), y_a)$ and $n - Y(a) \leq 2dM_{d+1}$. We make a case distinction:

- $a = d$: Note that $\text{val}(\pi, x_d) = \text{val}(\pi, y_d)$. Since we only remove loops from $R_d[1, Y(d)]$, we have that $\text{val}(\pi(R), x_d) \leq \text{val}(\pi(R), y_d)$. If $n - Y(d) \leq 2dM_{d+1}$ we are done with (R, X, Y, L) as a witnessing certificate. Otherwise, assume $n - Y(d) > 2dM_{d+1}$. This implies that the path $R[Y(d), n]$ must contain at least one simple loop. Consider iterating the following process:
 - remove the first simple loop from $R[Y(d), n]$ and update $n := n - \ell$, where ℓ is the length of the simple loop that was removed; and
 - stop if $n - Y(d) \leq 2dM_{d+1}$.
We argue that, $n - Y(d) \geq M_d^2$. Let R' and n' be the previous values of R, n before the last iteration. It must be that $n' > 2dM_{d+1}$ and since the length of any simple loop of $R'[Y(d)+1, n']$ is bounded by M_{d+1} , we get that $n - Y(a) \geq M_{d+1} \geq M_d^2$. Note that $Y(d) - X(d) \leq M_d \cdot |Q| \cdot \prod_{1 \leq j < d} M_j^2 \cdot U_j \leq M_d^2$, so $\text{val}(\pi(R[1, Y(d)]), x_d) \leq M_d^2$. It must be then the case that $\text{val}(\pi(R), x_d) \leq \text{val}(\pi(R), y_d)$.
- $a < d$: we know $n - Y(a) \leq 2dM_{d+1}$ by Lemma 14. Moreover, we must have that $\text{val}(\pi(R), x_a) \leq \text{val}(\pi(R), y_a)$ since $\text{val}(\pi, x_a) = \text{val}(\pi, y_a)$ and we do not remove loops after the counter y_a is incremented.

We are now ready to prove Proposition 9, i.e., show that language emptiness for restricted LAVASS can be decided in $\text{NSPACE}(|V| \cdot 2^{O(d)})$.

Proof of Proposition 9. Clearly, an abstract configuration can be stored in space $|V| \cdot 2^{O(d)}$. An NEXPSpace algorithm can hence non-deterministically choose an initial configuration and non-deterministically verify that it leads to a final abstract configuration along a path that is a witnessing certificate. To this end, the algorithm computes the set of induced y -constraints on-the-fly while guessing the reachability certificate, and verifies them in the

last configuration. Note that the y -constraints can be stored in space $|V| \cdot 2^{O(d)}$. Finally, the requirement $val(\pi(R), x_a) \leq val(\pi(R), y_a)$ can also be verified in exponential space since we require that $R[1, Y(a)]$ is a simple path and $n - Y(a) \leq 2M_{d+1}$. ◀

6 A decidable class of string constraints

In this section, we show that a certain fragment of string constraints, whose decidability status has been left open in the literature, can be reduced in logarithmic space to generalised Semënov arithmetic, and is hence decidable in EXPSPACE. This demonstrates an important application of our results on generalised Semënov arithmetic, with deep connections to solving string constraints in practice, which has been one of the motivations for our work.

Let $\Sigma = \{0, 1\}$. The *theory of enriched string constraints* T_{inc} is the first-order theory of the two-sorted structure

$$\langle \Sigma^*, \mathbb{N}; \{w\}_{w \in \Sigma^*}, \cdot, len, sn, \{R_i\}_{i \in \mathbb{N}}, 0, 1, + \rangle,$$

where

- the binary function \cdot over Σ^* is the string concatenation operator,
- the unary function $len: \Sigma^* \rightarrow \mathbb{N}$ returns on input w the length $|w|$ of w ,
- the unary function $sn: \Sigma^* \rightarrow \mathbb{N}$ on input u returns $\llbracket u \rrbracket$, and
- $R_0, R_1, \dots \subseteq \Sigma^*$ is an enumeration of all regular languages.

The remaining predicates, constant and function symbols are defined in their standard semantics.

The above theory was introduced in [4], where an SMT solver addressing some fragments of this theory was defined, implemented, and compared to other state of the art solvers which can handle such string constraints. Extending [4], [3] presents in more details the motivation behind considering this theory and its fragments. More precisely, the authors of [3] analysed an extensive collection of standard real-world benchmarks of string constraints and extracted the functions and predicates occurring in them. The works [3, 4] focused on benchmarks that do not contain word equations, and the result of the aforementioned benchmark-analysis produced exactly the four functions and predicates mentioned above: len , sn , regular language membership, and concatenation of strings.

Complementing the practical results of [4], [3] showed a series of theoretical results regarding fragments of T_{inc} . In particular, the existential theory of T_{inc} is shown to be undecidable. Moreover, [3] leaves as an open problem the question whether the existential theories of T_{RELn} and T_{REnc} , which drop the concatenation operator and length function, respectively, are decidable. From these two, the existential theories of T_{RELn} seems particularly interesting, as all instances from the benchmarks considered in the analysis [3] can be easily translated into a formula from this particular fragment of T_{RELn} . Indeed, by the results reported in Table 1.b from [3], no instance contains both concatenation of strings and the sn function; moreover, the concatenation of strings, which appears only in formulas involving regular membership predicates and, in some cases, length function, can be easily removed in all cases by a folklore technique called automata splitting (see, e.g., [1]). Therefore, showing that the existential fragment of T_{RELn} is decidable actually shows that one can decide the satisfiability of all the instances from the standard benchmarks analysed in [3].

In this paper, we solve this open problem. By a reduction to generalised Semënov arithmetic, we can settle the decidability status of T_{RELn} :

► **Theorem 15.** *The existential fragment of T_{RELn} is decidable in EXPSPACE.*

The idea underlying our proof is that we map a string s to the number $\llbracket 1s \rrbracket$. Note that we cannot directly treat strings in Σ^* as natural numbers due to the possibility of leading zeros. This encoding enables us to treat strings as numbers and to implement the functions sn and len in generalised Semënov arithmetic as $sn(s, x) \iff \exists y. 2^y \leq s \wedge s < 2^{y+1} \wedge x = s - 2^y$ and $len(s, x) \iff 2^x \leq s \wedge s < 2^{x+1}$ respectively. A full proof is presented in Appendix A.4.

7 Conclusion

The main result of this article has been to show that the existential theory of generalised Semënov arithmetic is decidable in EXPSPACE. On a technical level, this result was obtained by showing that a restricted class of labelled affine VASS has an EXPSPACE-decidable language emptiness problem. The structural restrictions imposed on those restricted LAVASS are rather strong, though necessary to obtain a decidable class of LAVASS.

As an application of this main result, we showed that a highly relevant class of string constraints with length constraints is also decidable in EXPSPACE; the decidability of this class was the main problem left open in [3]. Furthermore, our decision procedure has a better complexity than the one reported in [15] for a fragment of generalised Semënov arithmetic.

An interesting aspect of our approach is that it establishes automaticity of the existential fragment of a logical theory that is different from traditional notions of automaticity, which are based on finite-state automata or tree automata over finite or infinite words and trees [5, 10], respectively. It would be interesting to better understand whether there are natural logical theories whose (existential) fragments are, say, Petri-net or visibly-pushdown automatic.

We have ignored algorithmic lower bounds throughout this article, but it would, of course, be interesting to see whether the upper bounds of the decision problems we considered in this article are tight. It is clear that generalised Semënov arithmetic is PSPACE-hard since it can readily express the DFA intersection non-emptiness problem, but this still leaves a considerable gap with respect to the EXPSPACE upper bound we established. In particular, the recent results of [2] showing an NEXP upper bound for the existential fragment of Semënov arithmetic suggest that, if an EXPSPACE lower bound for existential generalised Semënov arithmetic is possible, it will require the use of regular predicates.

References

- 1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukás Holík, Ahmed Rezzine, Philipp Rümmer, and Jari Stenman. Norn: An SMT solver for string constraints. In *Proc. Computer Aided Verification, CAV*, volume 9206 of *Lect. Notes Comp. Sci.*, pages 462–469, 2015. doi:10.1007/978-3-319-21690-4_29.
- 2 Michael Benedikt, Dmitry Chistikov, and Alessio Mansutti. The Complexity of Presburger Arithmetic with Power or Powers. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 112:1–112:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.112.
- 3 Murphy Berzish, Joel D. Day, Vijay Ganesh, Mitja Kulczynski, Florin Manea, Federico Mora, and Dirk Nowotka. Towards more efficient methods for solving regular-expression heavy string constraints. *Theor. Comput. Sci.*, 943:50–72, 2023. doi:10.1016/j.tcs.2022.12.009.
- 4 Murphy Berzish, Mitja Kulczynski, Federico Mora, Florin Manea, Joel D. Day, Dirk Nowotka, and Vijay Ganesh. An SMT solver for regular expressions and linear arithmetic over string length. In *Computer Aided Verification, CAV*, volume 12760 of *Lect. Notes Comp. Sci.*, pages 289–312, 2021. doi:10.1007/978-3-030-81688-9_14.

- 5 Achim Blumensath and Erich Grädel. Automatic structures. In *Logic in Computer Science, LICS*, pages 51–62. IEEE Computer Society, 2000. doi:10.1109/LICS.2000.855755.
- 6 Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p -recognizable sets of integers. *Bull. Belg. Math. Soc. Simon Stevin*, 1(2):191–238, 1994. doi:10.36045/bbms/1103408547.
- 7 Gregory Cherlin and Françoise Point. On extensions of Presburger arithmetic. In *Proc. 4th Easter Model Theory conference, Gross Kőrös*, pages 17–34, 1986.
- 8 Alain Finkel, Stefan Göller, and Christoph Haase. Reachability in register machines with polynomial updates. In *Proc. Mathematical Foundations of Computer Science, MFCS*, volume 8087 of *Lect. Notes Comp. Sci.*, pages 409–420. Springer, 2013. doi:10.1007/978-3-642-40313-2_37.
- 9 Florent Guépin, Christoph Haase, and James Worrell. On the existential theories of Büchi arithmetic and linear p -adic fields. In *Proc. Symposium on Logic in Computer Science, LICS*, pages 1–10, 2019. doi:10.1109/LICS.2019.8785681.
- 10 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *Logical and Computational Complexity, LCC*, volume 960 of *Lect. Notes Comp. Sci.*, pages 367–392. Springer, 1995. doi:10.1007/3-540-60178-3_93.
- 11 Françoise Point. On the expansion of $(\mathbb{N}, +, 2^x)$ of Presburger arithmetic. Unpublished manuscript, 2010. URL: <https://u.osu.edu/friedman.8/files/2014/01/0AppB072710-1mnlwyn.pdf>.
- 12 Julien Reichert. *Reachability games with counters: decidability and algorithms*. PhD thesis, École normale supérieure de Cachan-ENS Cachan, 2015.
- 13 A. L. Semënov. On certain extensions of the arithmetic of addition of natural numbers. *Mathematics of the USSR-Izvestiya*, 15(2):401, 1980. doi:10.1070/IM1980v015n02ABEH001252.
- 14 Pierre Wolper and Bernard Boigelot. On the construction of automata from linear arithmetic constraints. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 1–19, 2000. doi:10.1007/3-540-46419-0_1.
- 15 Hao Wu, Yu-Fang Chen, Zhilin Wu, Bican Xia, and Naijun Zhan. A decision procedure for string constraints with string/integer conversion and flat regular constraints. *Acta Informatica*, October 2023. doi:10.1007/s00236-023-00446-4.

A Appendix

A.1 Closure properties of LAVASS languages

Let $V_i = \langle Q_i, d_i, \Sigma, \Delta_i, \lambda_i, q_0^{(i)}, F_i, \phi_i \rangle$, $i \in \{1, 2\}$, be two LAVASS.

► **Proposition 16.** *The languages of LAVASS are closed under union and intersection.*

Closure under union is trivial since we allow for non-determinism. To show closure under intersection, we define the LAVASS $V := (Q', d_1 + d_2, \Sigma, \Delta', \lambda', q_0', F', \phi')$ such that

- $Q' := Q_1 \times Q_2$,
- $((q_1, q_2), a, (r_1, r_2)) \in \Delta'$ if and only if $(q_1, a, r_1) \in \Delta_1$ and $(q_2, a, r_2) \in \Delta_2$,
- $\lambda'((q_1, q_2), a, (r_1, r_2)) := (\lambda_1(q_1, a, r_1), \lambda_2(q_2, a, r_2))$,
- $q_0' := (q_0^{(1)}, q_0^{(2)})$,
- $F' := F_1 \times F_2$, and
- ϕ is the conjunction of ϕ_1 and ϕ_2 , with counters renamed accordingly.

► **Lemma 17.** *For any $w \in \Sigma^*$, $q^1, r^{(1)} \in Q_1$ and $q^2, r^{(2)} \in Q_2$, the following are equivalent:*

- (i) $((q^{(1)}, q^{(2)}), m_1, \dots, m_{d_1+d_2}) \xrightarrow{w}_V ((r^{(1)}, r^{(2)}), m'_1, \dots, m'_{d_1+d_2})$
- (ii) $(q^{(1)}, m_1, \dots, m_{d_1}) \xrightarrow{w}_{V_1} (r^{(1)}, m'_1, \dots, m'_{d_1})$ and $(q^{(2)}, m_{d_1+1}, m_{d_1+1} \dots, m_{d_1+d_2}) \xrightarrow{w}_{V_2} (r^{(2)}, m'_{d_1+1}, \dots, m'_{d_1+d_2})$.

Proof. Let $w \in \Sigma^*$, we prove the statement by induction on $|w|$. The base case $w = \epsilon$ is immediate by the definition of V .

For the induction step, let $w = u \cdot a$ for some $a \in \Sigma$. The induction hypothesis yields

$$\begin{aligned} & ((q^{(1)}, q^{(2)}), m_1, \dots, m_{d_1+d_2}) \xrightarrow{u}_V ((s^{(1)}, s^{(2)}), m''_1, \dots, m''_{d_1+d_2}) \\ \iff & (q^{(1)}, m_1, \dots, m_{d_1}) \xrightarrow{u}_{V_1} (s^{(1)}, m''_1, \dots, m''_{d_1}) \text{ and} \\ & (q^{(2)}, m_{d_1+1}, \dots, m_{d_1+d_2}) \xrightarrow{u}_{V_2} (s^{(2)}, m''_{d_1+1}, \dots, m''_{d_1+d_2}). \end{aligned}$$

Again, by definition of V we furthermore have

$$\begin{aligned} & ((s^{(1)}, s^{(2)}), m''_1, \dots, m''_{d_1+d_2}) \xrightarrow{a}_V ((r^{(1)}, r^{(2)}), m'_1, \dots, m'_{d_1+d_2}) \\ \iff & (s^{(1)}, m_1, \dots, m_{d_1}) \xrightarrow{a}_{V_1} (r^{(1)}, m'_1, \dots, m'_{d_1}) \text{ and} \\ & (s^{(2)}, m_{d_1+1}, \dots, m_{d_1+d_2}) \xrightarrow{a}_{V_2} (r^{(2)}, m'_{d_1+1}, \dots, m'_{d_1+d_2}). \end{aligned}$$

This concludes the proof of the statement. \blacktriangleleft

► **Corollary 18.** *Let V_1, V_2 be LAVASS. Then $L(V_1) \cap L(V_2) = L(V_1 \cap V_2)$.*

Proof. For any $w \in \Sigma^*$, by Lemma 17, we have:

$$\begin{aligned} & w \in L(V_1 \cap V_2) \\ \iff & (q'_0, 0, \dots, 0) \xrightarrow{w}_V ((r_1, r_2), m_1, \dots, m_{d_1+d_2}) \\ & \text{for some } (r_1, r_2) \in F', (m_1, \dots, m_{d_1+d_2}) \in S'_f \\ \iff & (q_0^{(1)}, 0, \dots, 0) \xrightarrow{w}_{V_1} (r_1, m_1, \dots, m_{d_1}) \text{ for some } r_1 \in F_1, (m_1, \dots, m_{d_1}) \in S_f^{(1)}, \text{ and} \\ & (q_0^{(2)}, 0, \dots, 0) \xrightarrow{w}_{V_2} (r_2, m_{d_1+1}, \dots, m_{d_1+d_2}) \\ & \text{for some } r_2 \in F_2, (m_{d_1+1}, \dots, m_{d_1+d_2}) \in S_f^{(2)} \\ \iff & w \in L(V_1) \text{ and } w \in L(V_2). \end{aligned} \quad \blacktriangleleft$$

From the construction, it is clear that for the size of the LAVASS for $L(V_1) \cap L(V_2)$, we have:

- $|Q| = |Q_1| \cdot |Q_2|$,
- $|\Delta| \leq |\Delta_1| \cdot |\Delta_2|$,
- $|\lambda| = \max(|\lambda_1|, |\lambda_2|)$, and
- $d = d_1 + d_2$.

A.2 Proof of Theorem 1

We show how Theorem 1 follows from Proposition 9. Given a formula Φ of generalised Semënov arithmetic, we can in space $2^{O(|\Phi|)}$ construct the disjunctive normal form of Φ . Every disjunct can be assumed to be of the form

$$A \cdot \mathbf{x} = \mathbf{b} \wedge \bigwedge_{i \in I} x_i = 2^{y_i} \wedge \bigwedge_{j \in J} R_j(\mathbf{x}),$$

where the R_j are predicates over regular languages. By Lemma 8, there is a restricted LAVASS for Φ of dimension $2|I|$ with a number of states bounded by $(\|A\|_{1,\infty} + \|\mathbf{b}\|_{\infty} + 2)^{O(m+|I|)} = 2^{p(|\Phi|)}$ for some polynomial p and whose language represents the set of solutions to $A \cdot \mathbf{x} = \mathbf{b} \wedge \bigwedge_{i \in I} x_i = 2^{y_i}$. Intersecting with the DFA for the R_j results in a restricted LAVASS V with $2|I| = O(|\Phi|)$ counters such that $|V| = 2^{p(|\Phi|)}$ for some polynomial p . By Proposition 9, it follows that emptiness of V is decidable in $\text{NSPACE}(2^{p(|\Phi|)} \cdot 2^{2|I|})$. We conclude the argument by recalling that $\text{NEXPSPACE} = \text{EXPSPACE}$ by Savitch's theorem.

A.3 Witnessing certificates imply language non-emptiness

To formally prove Proposition 12, let (R, X, Y, L) be a witnessing certificate, and let $\pi(R)$ be the run in the configuration graph of V induced by R . Let $a \leq d$ be maximal such that $\ell_a \neq \top$. We now define a sequence of runs π_0, \dots, π_a such that the following invariant holds. In the final configuration of π_i ,

- (i) $m_j \leq n_j$ and $m_j \equiv n_j \pmod{\ell_j}$ for the j -th counter pair, $1 \leq j \leq a - i$; and
- (ii) $m_j = n_j$ for the j -th counter pair, $a - i < j \leq d$.

It is clear that π_a then witnesses $L(V) \neq \emptyset$. We proceed by induction on i .

Base case $i = 0$. Let $\pi_0 = \pi(R)$. Since R is a witnessing certificate, $\text{val}(\pi(R), x_a) \leq \text{val}(\pi(R), y_a)$, and hence $m_a \leq n_a$ in the last configuration of π_0 . Moreover, R respects the set of induced y -constraints. Hence $n_{a-1} - n_a \geq \delta_{a-1}$, where δ_{a-1} is the length of the path from $R[X(a-1)]$ to $R[X(a)]$. Hence $n_{a-1} - n_a \geq m_{a-1} - m_a$ and thus $m_{a-1} \leq n_{a-1}$. Iterating this argument for the remaining counters, we get that (i) of the invariant is fulfilled for π_0 ; (ii) trivially holds since R ends in an accepting abstract configuration.

Induction step $i > 0$. Let π_{i-1} be the path that exists by the induction hypothesis. If $m_{a-i} = n_{a-i}$ in the last configuration of π_{i-1} then we are done and take $\pi_i = \pi_{i-1}$; otherwise $m_{a-i} < n_{a-i}$ and $m_{a-i} \equiv n_{a-i} \pmod{\ell_{a-i}}$. Hence, there is some $k \in \mathbb{N}$ such that $n_{a-i} = k \cdot \ell_{a-i}$. Since $\ell_i \neq \top$, let $\beta := \alpha_{L(a-i)} \xrightarrow{t_1} \alpha_2 \xrightarrow{t_2} \dots \alpha_{\ell_i-1} \xrightarrow{t_{\ell_i}} \alpha_{L(a-i)}$ be the simple α -loop at position $L(a-i)$ that is guaranteed to exist since R is a witnessing certificate. We insert the transitions of β^k and the induced updated configurations into π_{i-1} at position $L(a-i)$. Notice that $L(a-i) < X(a-i+1)$. Otherwise, by the definition of the induced y -constraints, $y_{a-i} - y_{a-i+1} = \delta_{a-i}$ is in the set of induced y -constraints, where $\delta_i = X(a-i+1) - X(a-i)$. Since the last abstract configuration of R respects the set of y -constraints, it must be the case that in the last configuration of π_{i-1} , $n_{a-i} - n_{a-i+1} = \delta_{a-i}$ and $m_{a-i} - m_{a-i+1} = \delta_{a-i}$, so $m_{a-i} = n_{a-i}$, because after the position $X(a-i+1) - 1$ in R and thus π_{i-1} , the counters x_{a-i}, x_{a-i+1} get incremented simultaneously. This contradicts our assumption that $m_{a-i} \neq n_{a-i}$. Thus, the counters $x_{a-i+1}, y_{a-i+1}, \dots, x_d, y_d$ remain unchanged by the insertion of β^k , so (ii) and consequently (i) continues to hold in the last configuration of π_i for those counters. Moreover, due to the ordering conditions imposed on witnessing certificates, the value of y_{a-i} does not change either, and hence $m_{a-i} = n_{a-i}$ in the last configuration of π_i . Since β is a loop in the abstract configuration space, we have $m_j \equiv n_j \pmod{\ell_j}$ for all $1 \leq j < a - i$ and the values of u_j , for all $1 \leq j < a$ are preserved.

A.4 From string constraints to Semënov arithmetic

Again, we treat T_{RELIn} as a relational structure. Without loss of generality, we may assume that atomic formulas of T_{RELIn} are one of the following:

- $R(s)$ for some string variable s and a regular language R ;
- $s = t$ for some string variables s and t ;
- $\text{len}(s, x)$ or $\text{sn}(s, x)$ for some string variable s and integer variable x ; or
- $\mathbf{a} \cdot \mathbf{x} \geq b$ for a vector of integer variables \mathbf{x} .

The size of a formula of T_{RELIn} is defined in the standard way as the number of symbols required to write it down, assuming binary encoding of numbers, and where the size of some R is the size of the smallest DFA accepting R . Furthermore, in a quantifier-free formula φ of T_{RELIn} , we may without loss of generality assume that all atomic formulas occur positive, except for atomic formulas $s = t$.

We now describe the reduction to existential Semënov arithmetic. As explained, the idea underlying our proof is that we map a string s to the number $\llbracket 1s \rrbracket$. Given a quantifier-free formula φ of T_{REIn} , we define by structural induction on φ a function σ that maps φ to an equi-satisfiable formula of generalised Semënov arithmetic:

- Case $\varphi \equiv R(s)$: $\sigma(\varphi) := (0^*1R)(s)$;
- Case $\varphi \equiv s = t$ or $\varphi \equiv \neg(s = t)$: $\sigma(\varphi) := s = t$ or $\sigma(\varphi) := \neg s = t$, respectively;
- Case $\varphi \equiv sn(s, x)$: $\sigma(\varphi) := \exists y. 2^y \leq s \wedge s < 2^{y+1} \wedge x = s - 2^y$;
- Case $\varphi \equiv len(s, x)$: $\sigma(\varphi) := 2^x \leq s \wedge s < 2^{x+1}$;
- Case $\varphi \equiv \mathbf{a} \cdot \mathbf{x} \geq b$: $\sigma(\varphi) := \mathbf{a} \cdot \mathbf{x} \geq b$; and
- Case $\varphi \equiv \varphi_1 \sim \varphi_2$, $\sim \in \{\wedge, \vee\}$: $\sigma(\varphi) := \sigma(\varphi_1) \sim \sigma(\varphi_2)$.

► **Lemma 19.** *Let φ be a quantifier-free formula of T_{REIn} and S be the set of string variables occurring in S . Then φ is satisfiable if and only if $\sigma(\varphi) \wedge \bigwedge_{s \in S} s > 0$ is satisfiable.*

Proof. Observe that the variables occurring in φ are the same variables as those occurring in $\sigma(\varphi)$. Let S be the set of string variables in φ and X be the set of integer-valued variables in φ . Given an assignment $\mathcal{I}_S: S \rightarrow \{0, 1\}^*$, we define $\tilde{\mathcal{I}}_S := S \rightarrow \mathbb{N}$ such that $\tilde{\mathcal{I}}_S(s) := \llbracket 1\mathcal{I}_S(s) \rrbracket$. Subsequently, denote by $\mathcal{I}_X: X \rightarrow \mathbb{N}$ an assignment to the integer-valued variables. We show by structural induction on φ that $(\mathcal{I}_S, \mathcal{I}_X) \models \varphi$ if and only if $(\tilde{\mathcal{I}}_S, \mathcal{I}_X) \models \sigma(\varphi) \wedge \bigwedge_{s \in S} s > 0$:

- Case $\varphi \equiv R(s)$: Let $\mathcal{I}_S(s) = b_{n-1} \cdots b_0$, we have $\mathcal{I}_S(s) \in R$ if and only if $2^n + \sum_{i=0}^{n-1} 2^i b_i \in \llbracket 0^*1R \rrbracket$, noting that $2^n + \sum_{i=0}^{n-1} 2^i b_i = \llbracket 1b_{n-1} \cdots b_0 \rrbracket = \tilde{\mathcal{I}}_S(s)$.
- Case $\varphi \equiv sn(s, x)$: Let $\mathcal{I}_S(s) = b_{n-1} \cdots b_0$ and $\mathcal{I}_X(x) = m$. We have that $m = \sum_{i=0}^{n-1} 2^i b_i$ if and only if $m = \tilde{\mathcal{I}}_S(s) - 2^n$ if and only if $(\tilde{\mathcal{I}}_S, \mathcal{I}_X) \models \sigma(\varphi) \wedge \bigwedge_{s \in S} s > 0$.
- Case $\varphi \equiv len(s, x)$: Let $\mathcal{I}_S(s) = b_{n-1} \cdots b_0$ and $\mathcal{I}_X(x) = m$. We have that $m = n$ if and only if $2^m \leq \llbracket 1b_{n-1} \cdots b_0 \rrbracket < 2^{m+1}$ if and only if $(\tilde{\mathcal{I}}_S, \mathcal{I}_X) \models \sigma(\varphi) \wedge \bigwedge_{s \in S} s > 0$.

The remaining cases follow obviously. ◀

Fixed-Parameter Debordering of Waring Rank

Pranjal Dutta   

School of Computing, National University of Singapore (NUS), Singapore

Fulvio Gesmundo   

Institut de Mathématiques de Toulouse, Université Paul Sabatier, Toulouse, France

Christian Ikenmeyer   

University of Warwick, UK

Gorav Jindal   

Max Planck Institute for Software Systems, Saarbrücken, Germany

Vladimir Lysikov   

Ruhr-Universität Bochum, Germany

Abstract

Border complexity measures are defined via limits (or topological closures), so that any function which can be approximated arbitrarily closely by low complexity functions itself has low border complexity. Debordering is the task of proving an upper bound on some non-border complexity measure in terms of a border complexity measure, thus getting rid of limits.

Debordering is at the heart of understanding the difference between Valiant’s determinant vs permanent conjecture, and Mulmuley and Sohoni’s variation which uses border determinantal complexity. The debordering of matrix multiplication tensors by Bini played a pivotal role in the development of efficient matrix multiplication algorithms. Consequently, debordering finds applications in both establishing computational complexity lower bounds and facilitating algorithm design. Currently, very few debordering results are known.

In this work, we study the question of debordering the border Waring rank of polynomials. Waring and border Waring rank are very well studied measures in the context of invariant theory, algebraic geometry, and matrix multiplication algorithms. For the first time, we obtain a Waring rank upper bound that is exponential in the border Waring rank and only *linear* in the degree. All previous known results were exponential in the degree. For polynomials with constant border Waring rank, our results imply an upper bound on the Waring rank linear in degree, which previously was only known for polynomials with border Waring rank at most 5.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases border complexity, Waring rank, debordering, apolarity

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.30

Related Version *Extended version with appendices*: [arXiv:2401.07631](https://arxiv.org/abs/2401.07631) [27]

Funding *Pranjal Dutta*: Funded under the project “Foundation of Lattice-based Cryptography”, by NUS-NCS Joint Laboratory for Cyber Security.

Christian Ikenmeyer: Supported by EPSRC grant EP/W014882/1.

Vladimir Lysikov: Part of the work was done while V.L. was affiliated with the QMATH Centre, University of Copenhagen. V.L. acknowledges financial support from VILLUM FONDEN via the QMATH Centre of Excellence (Grant No. 10059) and the European Union (ERC Grant Agreements 818761 and 101040907). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.



© Pranjal Dutta, Fulvio Gesmundo, Christian Ikenmeyer, Gorav Jindal, and Vladimir Lysikov;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 30; pp. 30:1–30:15



Leibniz International Proceedings in Informatics

LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

Given a homogeneous polynomial f of degree d over \mathbb{C} , its *Waring rank* $\text{WR}(f)$ is defined as the smallest number r such that there exist homogeneous linear forms ℓ_1, \dots, ℓ_r with

$$f = \sum_{i=1}^r \ell_i^d.$$

Equivalently, $\text{WR}(f)$ is the minimal top fanin of a homogeneous $\Sigma \wedge \Sigma$ circuit computing f . In the case of quadratic forms (polynomials of degree 2), this notion is equivalent to the rank of the symmetric matrix associated with a quadratic form; hence Waring rank can be regarded as a generalization of the rank of a symmetric matrix. Unlike the case of matrices, when $d \geq 3$, Waring rank is in general not lower semicontinuous¹, that is, a limit of a family of polynomials with low Waring rank can have higher Waring rank. The simplest example is given by the polynomial $x^{d-1}y$, which has Waring rank d (this is a classical result [47]), but can be presented as a limit

$$x^{d-1}y = \lim_{\epsilon \rightarrow 0} \frac{1}{d\epsilon} [(x + \epsilon y)^d - x^d]$$

of a family of Waring rank 2 polynomials (note that we work over \mathbb{C} , so this expression can be rearranged into a sum of two powers by moving constants inside the parentheses). The *border Waring rank* is a semicontinuous variation of Waring rank defined as follows: the border Waring rank of f , denoted $\underline{\text{WR}}(f)$, is the smallest r such that f can be written as a limit of a sequence of polynomials of Waring rank at most r . We have $\underline{\text{WR}}(x^{d-1}y) = 2$.

Waring rank was studied already in the eighteenth century [23, 50, 25] in the context of invariant theory, with the aim to determine normal forms for homogeneous polynomials. We mention the famous Sylvester Pentahedral Theorem, stating that a generic cubic form in four variables can be written uniquely as a sum of five cubes. At the beginning of the twentieth century, the early work on secant varieties in classical algebraic geometry [48, 51] implicitly commenced the study of border Waring rank. The notion of border rank for tensors was introduced in [11] to construct faster-than-Strassen matrix multiplication algorithms. In [10], Bini proved that tensor border rank and tensor rank define the same matrix multiplication exponent. Today this theory is deeply related to the study of Gorenstein algebras [35, 15], the Hilbert scheme of points [38], and deformation theory [18, 39].

In the context of algebraic complexity theory, Waring rank defines a model of computation known as the homogeneous diagonal depth 3 circuits or homogeneous $\Sigma \wedge \Sigma$ circuits, see e.g. [49]. This is a very weak computational model (determinants have provably exponential Waring rank [33]). Nevertheless, it is important as one of the simplest nontrivial computational models and has many unresolved open problems associated with it. The Waring rank of a generic homogeneous polynomial of degree $d \geq 3$ in n variables is $\lceil \frac{1}{n} \cdot \binom{n+d-1}{d} \rceil = \Omega(\frac{n^{d-1}}{d!})$ with finite number of exceptional values of (n, d) [2], but the best lower bounds obtained are of order $2n^{\lfloor d/2 \rfloor}$ (from tensor rank lower bounds in [3]). For a large class of lower bound methods, so called rank methods, there are barrier results showing that cannot give bounds significantly larger than $n^{\lfloor d/2 \rfloor}$ [28, 31, 30]. Waring rank can be useful when the degree of the polynomials considered is constant. For example, the results of [24] guarantee that the

¹ A function f is lower semicontinuous at a if $\liminf_{x \rightarrow a} f(x) \geq f(a)$.

matrix multiplication exponent is controlled by the Waring rank or border Waring rank of the polynomial $\text{Tr}(X^3)$ with $X \in \mathbb{C}^{n \times n}$, which is a symmetrized version of the matrix multiplication tensor.

The lack of semicontinuity is a common phenomenon in algebraic complexity not specific to Waring rank. Most complexity measures defined in terms of discrete structures (such as circuits or formulas) or in terms of decompositions (such as Waring rank or tensor rank) are not lower semicontinuous. To any algebraic complexity measure one can define the corresponding *border complexity measure* in the same way as border Waring rank arises from Waring rank: the border complexity of f is the smallest number s such that f can be approximated arbitrarily closely by polynomials of complexity at most s . Border tensor rank appears in the study of the computational complexity of matrix multiplication [11, 10], border complexity for algebraic circuits was first discussed in [45] and [20].

Replacing a complexity measure by its border measure in a complexity class, we obtain the *closure* of this class. For example, $\overline{\text{VP}}$ is the class of all polynomial sequences with polynomially bounded degree and border circuit size, and $\overline{\text{VF}}$ is defined analogously using formula size. Formally, the closure $\overline{\mathcal{C}}$ of a complexity class \mathcal{C} consists of all polynomial sequences $(f_n)_{n \in \mathbb{N}}$ such that there exists a bivariate sequence $(g_{n,m})_{n,m \in \mathbb{N}}$ with the property that $(g_{n,m})_{n \in \mathbb{N}}$ lies in \mathcal{C} for every fixed m , and $f_n = \lim_{m \rightarrow \infty} g_{n,m}$. The operation of going to the closure is indeed a closure operator in the sense of topology, see [36].

The relationship between border and non-border complexity is far from straightforward. In some contexts taking a limit can be a very strong operation, which sometimes turns non-universal computational models into universal ones. For example, there are polynomials which cannot be computed by width 2 algebraic branching programs [4], but the corresponding border measure is related to border formula size [14], so every polynomial is a limit of width 2 ABPs. Kumar [42] gives an even easier example: every polynomial can be presented as a limit of a sum of 2 products of affine linear forms. On the other hand, there are examples of complexity measures which are lower semicontinuous, so that there is no difference between border and non-border complexity measures. A simplest example is the number of monomials in a polynomial (equivalently, top fanin of a $\Sigma\Pi$ circuit). Other examples are noncommutative ABP width (implicit in [46]) and read-once ABP width [26].

Semicontinuous complexity measures and closed complexity classes are easier to work with using geometric methods. Because of this, the geometric complexity theory program [45] proposes to study conjectures $\text{VNP} \not\subseteq \overline{\text{VBP}}$ and $\text{VNP} \not\subseteq \overline{\text{VP}}$ instead of Valiant's conjectures $\text{VNP} \neq \text{VBP}$ and $\text{VNP} \neq \text{VP}$. The $\text{VNP} \not\subseteq \overline{\text{VBP}}$ conjecture was also proposed in [20]. These border variants of Valiant's conjecture are now usually referred to as the Mulmuley–Sohoni conjectures. Mulmuley–Sohoni conjectures are stronger than Valiant's conjectures, but it is not clear how much stronger, as most questions about the relations between complexity classes and their closures are wide open. It is unknown even whether or not $\overline{\text{VF}} \subseteq \text{VNP}$. Theorems of the form $\overline{\mathcal{C}} \subseteq \mathcal{D}$ for algebraic complexity classes \mathcal{C} and \mathcal{D} are called *debordering* results. These kind of results can also be proven directly on the complexity measures, by giving an upper bound on a non-border complexity in terms of border complexity. For example, $\text{abpw}(f) \leq \overline{\text{WR}}(f)$, where $\text{abpw}(f)$ is the algebraic branching program width of f . This is proven using semicontinuity of noncommutative ABP width, see [12, Thm 4.2] and [29]. In terms of complexity classes, this means $\overline{\text{VWaring}} \subseteq \text{VBP}$, where VWaring is the class of p -families that have polynomially bounded Waring rank.

Forbes [52] conjectures that $\overline{\text{VWaring}} = \text{VWaring}$. Since this puts $\overline{\text{VWaring}}$ in VF , a proof of this conjecture will also improve the results of Dutta, Dwivedi and Saxena [26] from $\overline{\Sigma^{[r]}\Pi\Sigma} \subseteq \text{VBP}$ to $\overline{\Sigma^{[r]}\Pi\Sigma} \subseteq \text{VF}$. Ballico and Bernardi [7] propose an even stronger conjecture

stating that $\text{WR}(f) \leq (\text{WR}(f) - 1) \cdot \deg f$. This was proven by case analysis for small values of border Waring rank: for $\text{WR}(f) \leq 3$ in [43], for $\text{WR}(f) = 4$ in [6], and for $\text{WR}(f) = 5$ and $\deg f \geq 9$ in [5].

Main result

We prove the following improved debordering theorem for border Waring rank.

► **Theorem 1** (Fixed-parameter debordering). *Let f be a homogeneous polynomial with $\deg f = d$ and $\text{WR}(f) = r$. Then $\text{WR}(f) \leq 4^r \cdot d$.*

Note that the example of the polynomial $x^{d-1}y$ with $\text{WR}(x^{d-1}y) = 2$ and $\text{WR}(x^{d-1}y) = d$ shows that any debordering bound must necessarily depend on both border Waring rank r and the degree d . We call our result a *fixed-parameter debordering* because the bound is polynomial (in this case even linear) in d , but exponential in the complexity parameter r . In the case of a fixed border Waring rank this gives a bound linear in the degree. This was previously known only for $\text{WR}(f) \leq 5$. Even for $r = O(\log d)$ we obtain an upper bound polynomial in d .

To the best of our knowledge, this is the first fixed-parameter debordering result. Previous methods applied to border Waring rank only allow upper bounds of the order d^r or r^d . To get $\text{WR}(f) \leq O(d^r)$, note that a polynomial with border Waring rank r can be transformed into a polynomial in only r variables using a linear change of variables (see Lemma 4), and then take the maximal possible Waring rank of an r -variate polynomial of degree d as an upper bound. Alternatively, an upper bound $\text{WR}(f) \leq 2^{d-1}r^d$ can be obtained by using the previously mentioned debordering into an ABP ($\text{abpw}(f) \leq \text{WR}(f)$) and writing the ABP as a sum of at most r^d products, one for each path. Other known debordering techniques, such as the interpolation technique using the bound on the degree of ϵ in the approximation from the work of Lehmkuhl and Lickteig [44] (which is exponential in the degree of the polynomial), or the DiDIL technique from [26] can be applied in the border Waring rank setting, but do not improve over the simpler results discussed above.

Proof ideas

The main ideas for the proof come from *apolarity theory* and the study of 0-dimensional schemes in projective space (we discuss these ideas in Appendix A of the extended version of the paper [27]). We managed to simplify the proof so that it is elementary and *does not use* the language of algebraic geometry and is based on partial derivative techniques (see Section 2.3).

To prove the debordering, we transform a border Waring rank decomposition for f into a *generalized additive decomposition* [34, 8, 9] of the form $f = \sum_{k=1}^m \ell_k^{d-r_k+1} g_k$, where ℓ_k are linear forms, and g_k are homogeneous polynomials of degrees $r_k - 1$. We then obtain an upper bound on the Waring rank, by first decomposing each g_k with respect to a basis consisting of powers of linear forms, and then using the classical fact (see also [19]) that $\text{WR}(\ell_1^a \ell_2^b) \leq \max(a+1, b+1)$.

To construct a generalized additive decomposition, we divide the summands of a border rank decomposition into several parts such that cancellations happen *only* between summands belonging to the same part; see Lemma 10. The key insight is that if the degree of polynomials involved is high enough, namely when $\deg f \geq \text{WR}(f) - 1$, then all parts of the decomposition are “local” in the sense that the lowest order term in each summand is a multiple of the same linear form. Each local part gives one term of the form $\ell^{d-r+1} g$, where r is the number of rank one summands in the part and ℓ is the common lowest order linear form; see Lemma 7.

For example, consider the family of polynomials $f_d = x_0^{d-1}y_0 + x_1^{d-1}y_1 + 2(x_0 + x_1)^{d-1}y_2$, adapted from [16]. If $d > 3$, then the border Waring rank of f is at most 6, as evidenced by the decomposition

$$f_d = \lim_{\epsilon \rightarrow 0} \frac{1}{d\epsilon} [(x_0 + \epsilon y_0)^d - x_0^d + (x_1 + \epsilon y_1)^d - x_1^d + 2(x_0 + x_1 + \epsilon y_2)^d - 2(x_0 + x_1)^d], \quad (1)$$

and a matching lower bound is obtained by considering the dimension of the space of second order partial derivatives. The summands of the decomposition (1) can be divided into three pairs. The lowest order term of the first pair is x_0^d , the one of the second pair is x_1^d and the one of the third pair is $(x_0 + x_1)^d$. For each pair, the sum of the two powers individually converges to a limit as $\epsilon \rightarrow 0$; these three limits are, respectively, $x_0^{d-1}y_0$, $x_1^{d-1}y_1$, and $2(x_0 + x_1)^{d-1}y_2$, which are the summands of a generalized additive decomposition for f_d .

When $d = 3$, the polynomial f_d is an example of a “wild form” [16]. It has border Waring rank 5 given for example by the decomposition

$$f_3 = \lim_{\epsilon \rightarrow 0} \frac{1}{9\epsilon} [3(x_0 + \epsilon y_0)^3 + 3(x_1 + \epsilon y_1)^3 + 6(x_0 + x_1 + \epsilon y_2)^3 - (x_0 + 2x_1)^3 - (2x_0 + x_3)^3]. \quad (2)$$

Unlike the previous decomposition, this one cannot be divided into parts that have limits individually, and is not local – all summands have different lowest order terms. This is only possible if the degree is low.

The condition on the degree is related to algebro-geometric questions about regularity of 0-dimensional schemes [35, Thm. 1.69], but for the schemes arising from border rank decompositions, this is ultimately a consequence of the fact that r distinct linear forms have linearly independent d -th powers when $d \geq r - 1$.

2 Debordering border Waring rank

The goal of this section is to prove Theorem 1. Given a homogeneous degree d polynomial f , we provide upper bounds for $\text{WR}(f)$ in terms of $\underline{\text{WR}}(f)$ and d .

2.1 Definitions

In this section we introduce some notation and give a formal definition of Waring rank and border Waring rank. We work over the field \mathbb{C} of complex numbers. The space of homogeneous polynomials of degree d in variables $\mathbf{x} = (x_1, \dots, x_n)$ is denoted by $\mathbb{C}[\mathbf{x}]_d$. We write $f \simeq g$ for $f, g \in \mathbb{C}(\epsilon)[\mathbf{x}]$ if $\lim_{\epsilon \rightarrow 0} f = \lim_{\epsilon \rightarrow 0} g$ (in particular, both limits must exist). Recall that the projective space $\mathbb{P}V$ is defined as the set of lines through the origin in V , that is, for each nonzero $v \in V$ we have a corresponding line $[v] \in \mathbb{P}V$, and $[v] = [w]$ if and only if $v = \alpha w$ for some α .

► **Definition 2.** A Waring rank decomposition of a homogeneous polynomial $f \in \mathbb{C}[\mathbf{x}]_d$ is a decomposition of the form

$$f = \sum_{k=1}^r \ell_k^d$$

for some linear forms $\ell_1, \dots, \ell_r \in \mathbb{C}[\mathbf{x}]_1$. The minimal number of summands in a Waring rank decomposition is called the Waring rank of f and is denoted by $\text{WR}(f)$.

It is known that every homogeneous polynomial over \mathbb{C} has finite Waring rank [47].

► **Definition 3.** A border Waring rank decomposition of a homogeneous polynomial $f \in \mathbb{C}[\mathbf{x}]_d$ is an expression of the form

$$f = \lim_{\epsilon \rightarrow 0} \sum_{k=1}^r \ell_k^d$$

where $\ell_1, \dots, \ell_r \in \mathbb{C}(\epsilon)[\mathbf{x}]_1$, that is, ℓ_i are linear forms in \mathbf{x} with coefficients rationally dependent on ϵ . The border Waring rank $\underline{\text{WR}}(f)$ is the minimal number of summands in a border Waring rank decomposition.

Equivalently, the border Waring rank of $f \in \mathbb{C}[\mathbf{x}]_d$ can be defined as the minimal number r such that f lies in the closure of the set $W_{d,r} = \{g \in \mathbb{C}[\mathbf{x}]_d \mid \text{WR}(g) \leq r\}$ of all polynomials with Waring rank at most r . The set $W_{d,r}$ is constructible, so its Zariski and Euclidean closures coincide, see e.g. [41, Anh.I.7.2 Folgerung]. The equivalence to the definition given above was established by Alder [1] (cited by [21, Ch.20]) for a similar notion of tensor rank, the proof remains essentially the same for Waring rank of polynomials.

2.2 Orbit closure and essential variables

The number of essential variables of a homogeneous polynomial f is the minimum integer m such that there is a linear change of coordinates after which f can be written as a polynomial in m variables. Denote the number of essential variables of f by $N_{\text{ess}}(f)$. It is a classical fact, which already appears in [50], that the number of essential variables of f equals the dimension of the linear span of its first order partial derivatives, or equivalently the rank of the first partial derivative map. In particular $N_{\text{ess}}(-)$ is a lower semicontinuous function. We refer to [22] and [40, Lemma B.1] for modern proofs of this result.

An immediate consequence of the semicontinuity of the number of essential variables is the following result.

► **Lemma 4.** For a homogeneous polynomial $f \in \mathbb{C}[\mathbf{x}]_d$ we have $N_{\text{ess}}(f) \leq \underline{\text{WR}}(f)$.

Proof. We first prove $N_{\text{ess}}(f) \leq \underline{\text{WR}}(f)$. Let p be the dimension of the linear space spanned by the linear forms ℓ_k in the decomposition $f = \sum_{k=1}^r \ell_k^d$. Without loss of generality the linear forms ℓ_1, \dots, ℓ_p are linearly independent, and $\ell_{p+1}, \dots, \ell_r$ are linear combinations of ℓ_1, \dots, ℓ_p . After applying a change of variables such that $y_k = \ell_k(\mathbf{x})$ for $k = 1, \dots, p$ we see that $N_{\text{ess}}(f) \leq p \leq r$.

The inequality $N_{\text{ess}}(f) \leq \underline{\text{WR}}(f)$ now follows from the semicontinuity of N_{ess} : if

$$f = \lim_{\epsilon \rightarrow 0} \sum_{k=1}^r \ell_k^d,$$

with $\ell_k \in \mathbb{C}(\epsilon)[\mathbf{x}]$, then $N_{\text{ess}}(f) \leq \lim_{\epsilon \rightarrow 0} N_{\text{ess}}(\sum_{k=1}^r \ell_k^d(\epsilon)) \leq r$. ◀

2.3 Fixed-parameter debordering

The proof of Theorem 1 is based on generalized additive decompositions of polynomials, in the sense of [34]. These decompositions were studied in algebraic geometry, usually in connection to 0-dimensional schemes and the notion of cactus rank. We defer the discussion of connections to algebraic geometry to the next section. Here we provide elementary proofs of some statements on generalized additive decompositions based on partial derivatives techniques, without using the language of 0-dimensional schemes. We bring from geometry a key insight: a border rank decomposition can be separated into *local* parts if the degree of the polynomial is large enough.

To define formally what it means for a border rank decomposition to be local, note that a rational family of linear forms $\ell \in \mathbb{C}(\epsilon)[\mathbf{x}]_1$ always has a limit when viewed projectively. Specifically, expanding $\ell(\epsilon)$ as a Laurent series $\ell(\epsilon) = \sum_{i=q}^{\infty} \epsilon^i \ell_i$ with $\ell_q \neq 0$, we have $\lim_{\epsilon \rightarrow 0} [\ell(\epsilon)] = \lim_{\epsilon \rightarrow 0} [\sum_{i=0}^{\infty} \epsilon^i \ell_{q+i}] = [\ell_q]$. A border Waring rank decomposition is called local if for all summands in the decomposition this limit is the same. More precisely, we give the following definition.

► **Definition 5.** Let $f \in \mathbb{C}[\mathbf{x}]_d$ be a homogeneous polynomial. A border Waring rank decomposition

$$f = \lim_{\epsilon \rightarrow 0} \sum_{k=1}^r \ell_k^d,$$

with $\ell_k \in \mathbb{C}(\epsilon)[\mathbf{x}]_1$ is called a local border decomposition if there exists a linear form $\ell \in \mathbb{C}[\mathbf{x}]_1$ such that $\lim_{\epsilon \rightarrow 0} [\ell_k(\epsilon)] = [\ell]$ for all $k \in \{1, \dots, r\}$. We call the point $[\ell] \in \mathbb{P}\mathbb{C}[\mathbf{x}]_1$ the base of the decomposition. A local decomposition is called standard if $\ell_1 = \epsilon^q \gamma \ell$ for some $q \in \mathbb{Z}$ and $\gamma \in \mathbb{C}$.

► **Lemma 6.** If f has a local border decomposition, then it has a standard local border decomposition with the same base and the same number of summands.

Proof. After applying a linear change of variables, we may assume that the base of the local decomposition for f is $[x_1]$. This means

$$f = \lim_{\epsilon \rightarrow 0} \sum_{k=1}^r \ell_k^d$$

with $\ell_k = \epsilon^{q_k} \cdot \gamma_k x_1 + \sum_{j=q_k+1}^{\infty} \epsilon^j \ell_{k,j}$.

Write $\ell_1 = \epsilon^{q_1} (\sum_{i=1}^n \alpha_i x_i)$ where $\alpha_i \in \mathbb{C}(\epsilon)$. Let $\hat{x}_1 = \frac{\gamma_1}{\alpha_1} x_1 - \sum_{i=2}^n \frac{\alpha_n}{\alpha_1} x_i$. Note that $\alpha_1 \simeq \gamma_1$ and $\alpha_i \simeq 0$ for $i > 1$, hence $\hat{x}_1 \simeq x_1$ and

$$f \simeq f(\hat{x}_1, \dots, x_n) \simeq \ell_1(\hat{x}_1, x_2, \dots, x_n)^d + \sum_{k=2}^r \ell_k(\hat{x}_1, x_2, \dots, x_n)^d = (\epsilon^{q_1} \gamma_1 x_1)^d + \sum_{k=2}^r \hat{\ell}_k^d.$$

where $\hat{\ell}_k(x_1, \dots, x_n) = \ell_k(\hat{x}_1, x_2, \dots, x_n)$. This defines a new border rank decomposition of f . Moreover, notice that $\lim_{\epsilon \rightarrow 0} [\hat{\ell}_k] = [x_1]$ for every k , so the new decomposition is again local with base $[x_1]$. Since the first summand is $\epsilon^{q_1} \gamma_1 x_1$, this is the desired standard local border decomposition. ◀

► **Lemma 7.** Suppose $f \in \mathbb{C}[\mathbf{x}]_d$ has a local border decomposition with r summands based at $[\ell]$. If $d \geq r - 1$, then $f = \ell^{d-r+1} g$ for some homogeneous polynomial g of degree $r - 1$.

Proof. After applying a linear change of variables we may assume $\ell = x_1$. We prove the statement by induction on r and the difference $d - (r - 1)$.

The cases $r = 1$ and $d = r - 1$ are trivial.

If $d > r - 1$, then by the previous Lemma there exists a standard local border decomposition

$$f = \lim_{\epsilon \rightarrow 0} \sum_{k=1}^r \ell_k(\epsilon)^d.$$

Write $\ell_k = \sum_{i=1}^n \alpha_{ki} x_i$ for some $\alpha_{ki} \in \mathbb{C}(\epsilon)$. Since the decomposition is standard, $\alpha_{1i} = 0$ for $i > 1$. For the derivatives of f we have the following border decompositions

$$\frac{\partial f}{\partial x_1} = \lim_{\epsilon \rightarrow 0} \sum_{k=1}^r d \cdot \alpha_{k1}(\epsilon) \ell_k(\epsilon)^{d-1},$$

30:8 Fixed-Parameter Debordering of Waring Rank

and

$$\frac{\partial f}{\partial x_i} = \lim_{\epsilon \rightarrow 0} \sum_{k=2}^r d \cdot \alpha_{ki}(\epsilon) \ell_k(\epsilon)^{d-1}.$$

for $i \neq 1$. These decompositions involve the same linear forms ℓ_k with multiplicative coefficients, so they are local with the same base $[x_1]$. By inductive hypothesis $\frac{\partial f}{\partial x_1} = x_1^{d-r} g_1$ and $\frac{\partial f}{\partial x_i} = x_1^{d-r+1} g_i$ for some homogeneous polynomials g_1, \dots, g_n of appropriate degrees. To get an analogous expression for f , combine these expressions using Euler's formula for homogeneous polynomials as follows

$$f = \frac{1}{d} \sum_{i=1}^n x_i \frac{\partial f}{\partial x_i} = \frac{1}{d} \left(x_1 \cdot x_1^{d-r} g_1 + \sum_{i=2}^n x_i x_1^{d-r+1} g_i \right) = \frac{1}{d} x_1^{d-r+1} \left(g_1 + \sum_{i=2}^n x_i g_i \right). \quad \blacktriangleleft$$

We will now extend this result to non-local border Waring rank decompositions. As long as the degree of the approximated polynomial is high enough, every border rank decomposition can be divided into local parts and transformed into a sum of terms of the form $\ell^{d-r+1} g$.

► **Definition 8.** A generalized additive decomposition of f is a decomposition of the form

$$f = \sum_{k=1}^m \ell_k^{d-r_k+1} g_k,$$

where ℓ_k are linear forms such that ℓ_i is not proportional to ℓ_j when $i \neq j$, and g_k are homogeneous polynomials of degrees $\deg g_k = r_k - 1$.

To show that there are no cancellations between different local parts, we need the following lemma, which in the case of 2 variables goes back to Jordan [35, Lem. 1.35]. This lemma can be seen as a generalization of a well-known fact that m pairwise non-proportional linear forms ℓ_1, \dots, ℓ_m have linearly independent powers $\ell_1^d, \dots, \ell_m^d$ for $d \geq m - 1$.

► **Lemma 9.** Let $\ell_1, \dots, \ell_m \in \mathbb{C}[\mathbf{x}]_1$ be linear forms such that ℓ_i is not proportional to ℓ_j when $i \neq j$. Let g_1, \dots, g_m be homogeneous polynomials of degrees $r_1 - 1, \dots, r_m - 1$ respectively. If

$$\sum_{k=1}^m \ell_k^{d-r_k+1} g_k = 0,$$

and $d \geq \sum_{k=1}^m r_k - 1$, then all g_k are zero.

Proof. We first prove the statement for polynomials in 2 variables y_1, y_2 by induction on the number of summands m ; this part of the proof closely follows [32, Appx.III].

The case $m = 1$ with one summand is clear. Consider the case $m > 2$. We can assume $\ell_1 = y_1$ by applying a linear change of variables if required. Note two simple facts about partial derivatives. First, for a homogeneous polynomial $f \in \mathbb{C}[y_1, y_2]_d$ we have $\partial_2^s f = 0$ if and only if $f = y_1^{d-r+1} g$ (here $\partial_2 := \frac{\partial}{\partial y_2}$). Second, differentiating r times a homogeneous polynomial of the form $\ell^{d-s+1} g$, we obtain a polynomial of the form $\ell^{d-r-s+1} h$.

Suppose

$$y_1^{d-r_1+1} g_1 + \sum_{k=2}^m \ell_k^{d-r_k+1} g_k = 0.$$

Differentiating r_1 times with respect to y_2 , we obtain

$$\sum_{k=2}^m \ell_k^{d-r_1-r_k+1} h_k = 0,$$

where $\ell_k^{d-r_1-r_k+1} h_k = \partial_2^{r_1}(\ell_k^{d-r_k+1} g_k)$. The degree condition $d - r_1 \geq \sum_{k=2}^m r_k - 1$ holds for this new expression. Therefore, by induction hypothesis we have $h_k = 0$ and thus $\partial_2^{r_1}(\ell_k^{d-r_k+1} g_k) = 0$. It follows that $\ell_k^{d-r_k+1} g_k = y_1^{d-r_1+1} \widehat{g}_k$ for some homogeneous polynomial \widehat{g}_k . This implies that $y_1^{d-r_1+1}$ divides g_k , which is impossible since $d - r_1 + 1 \geq \sum_{k=2}^m r_k \geq r_k > \deg g_k$.

Consider now the general case where the number of variables $n \geq 2$ (the case $n = 1$ is trivial). Suppose $\sum_{k=1}^m \ell_k^{d-r_k+1} g_k = 0$. The set of linear maps $A: (y_1, y_2) \mapsto (x_1, \dots, x_n)$ such that $\ell_i \circ A$ and $\ell_j \circ A$ are not proportional to each other is a nonempty Zariski open set given by the condition $\text{rank}(\ell_i \circ A, \ell_j \circ A) > 1$. Hence for a nonempty Zariski open (and therefore dense) set of linear maps A the linear forms $\ell_k \circ A$ are pairwise non-proportional. From the binary case above we have $g_k \circ A = 0$ if A lies in this open set. By continuity this implies $g_k \circ A = 0$ for all A . Since every point lies in the image of some linear map A we have $g_k = 0$. ◀

► **Lemma 10.** *Let $f \in \mathbb{C}[\mathbf{x}]_d$ be such that $\text{WR}(f) = r$. If $d \geq r - 1$, then there exists a partition $r = r_1 + \dots + r_m$ such that f has a generalized additive decomposition*

$$f = \sum_{k=1}^m \ell_k^{d-r_k+1} g_k,$$

and moreover $\text{WR}(\ell_k^{d-r_k+1} g_k) \leq r_k$.

Proof. Consider a border Waring rank decomposition

$$f = \lim_{\epsilon \rightarrow 0} \sum_{k=1}^r \ell_k^d$$

Divide the summands between several local decompositions as follows. Define an equivalence relation \sim on the set of indices $\{1, 2, \dots, r\}$ as $i \sim j \Leftrightarrow \lim_{\epsilon \rightarrow 0} [\ell_i] = \lim_{\epsilon \rightarrow 0} [\ell_j]$ and let I_1, \dots, I_m be the equivalence classes with respect to this relation. Further, let $r_k = |I_k|$ and let $[L_k] = \lim_{\epsilon \rightarrow 0} [\ell_i]$ for $i \in I_k$.

Consider the sum of all summands with indices in I_k . Let q_k be the power of ϵ in the lowest order term, that is,

$$\sum_{i \in I_k} \ell_i^d = \epsilon^{q_k} f_k + \sum_{j=q_k+1}^{\infty} \epsilon^j f_{k,j},$$

with $f_k \in \mathbb{C}[\mathbf{x}]_d$ nonzero. This expression can be transformed into a local border decomposition

$$f_k = \lim_{\epsilon \rightarrow 0} \sum_{i \in I_k} \left(\frac{\ell_i(\epsilon^d)}{\epsilon^{q_k}} \right)^d.$$

based at $[L_k]$. By Lemma 7 we have $f_k = L_k^{d-r_k+1} g_k$ for some homogeneous polynomial g_k of degree $r_k - 1$. The decomposition also gives $\text{WR}(f_k) \leq r_k$.

30:10 Fixed-Parameter Debordering of Waring Rank

Note that $q_k \leq 0$ since otherwise the summands ℓ_i with $i \in I_k$ can be removed from the original border rank decomposition of f without changing the limit. Let $q = \min\{q_1, \dots, q_m\}$. Note that if $q < 0$, then, comparing the terms before ϵ^q in the left and right hand sides of the equality

$$f + O(\epsilon) = \sum_{k=1}^m \sum_{i \in I_k} \ell_i^d$$

we get

$$0 = \sum_{k: q_k=q} f_k = \sum_{k: q_k=q} L_k^{d-r_k+1} g_k.$$

From Lemma 9 we obtain $g_k = 0$ and $f_k = 0$, in contradiction with the definition of f_k .

We conclude that $q = 0$ and

$$f = \sum_{k=1}^m f_k = \sum_{k=1}^m L_k^{d-r_k+1} g_k,$$

obtaining the required generalized additive decomposition. \blacktriangleleft

We will now take a brief detour to define a function $M(r)$ which we use to upper bound the Waring rank of generalized additive decomposition.

► **Definition 11.** Let $\max\mathbf{R}(n, d)$ denote the maximum Waring rank of a degree d homogeneous polynomial in n variables, that is $\max\mathbf{R}(n, d) = \max\{\mathbf{WR}(f) \mid f \in \mathbb{C}[x_1, \dots, x_n]_d\}$. Define the partition-maxrank function as

$$M(r) = \max_{r_1 + \dots + r_m = r} \sum_{k=1}^m \max\mathbf{R}(r_k, r_k - 1).$$

Since every homogeneous polynomial has finite Waring rank, the space $\mathbb{C}[x_1, \dots, x_n]_d$ is spanned by powers of linear forms. This implies a trivial upper bound on the maximum Waring rank: $\max\mathbf{R}(n, d) \leq \dim \mathbb{C}[x_1, \dots, x_n]_d = \binom{n+d-1}{d}$. Improved upper bounds were proven in [13, 37].

► **Proposition 12.** $\max\mathbf{R}(n, d_1) \leq \max\mathbf{R}(n, d_2)$ when $d_1 \leq d_2$.

Proof. Every form f of degree d_1 can be represented as a partial derivative of some form g of degree d_2 . By differentiating a Waring rank decomposition of g we obtain a Waring rank decomposition of f , thus $\mathbf{WR}(f) \leq \mathbf{WR}(g) \leq \max\mathbf{R}(n, d_2)$. Since f is arbitrary, $\max\mathbf{R}(n, d_1) \leq \max\mathbf{R}(n, d_2)$. \blacktriangleleft

We are now ready to prove a debordering theorem for Waring rank.

► **Theorem 13.** Let $f \in \mathbb{C}[\mathbf{x}]_d$ be such that $\underline{\mathbf{WR}}(f) = r$. Then

$$\mathbf{WR}(f) \leq M(r) \cdot d.$$

Proof. We consider two cases depending on relation of degree d and border Waring rank r .

Case $d < r - 1$. Since $\underline{\mathbf{WR}}(f) = r$, the number of essential variables of f is at most r . Taking the maximum Waring rank as an upper bound, we obtain

$$\mathbf{WR}(f) \leq \max\mathbf{R}(r, d) < \max\mathbf{R}(r, r - 1) \leq M(r) \leq M(r) \cdot d.$$

Case $d \geq r - 1$. By Lemma 10 f has a generalized additive decomposition

$$f = \sum_{k=1}^m \ell_k^{d-r_k+1} g_k$$

with $r_1 + \dots + r_m = r$, $\deg g_k = r_k - 1$ and $\underline{\text{WR}}(\ell_k^{d-r_k+1} g_k) \leq r_k$. Since $\underline{\text{WR}}(\ell_k^{d-r_k+1} g_k) \leq r_k$, the number of essential variables $N_{\text{ess}}(g_k) \leq r_k$. If $r_k = 1$, then

$$\text{WR}(\ell_k^{d-r_k+1} g_k) = \text{WR}(\ell_k^d) = 1 \leq d.$$

If $r_k \geq 2$, then we upper bound $\text{WR}(g_k)$ by $\max\text{R}(N_{\text{ess}}(g_k), \deg g_k) = \max\text{R}(r_k, r_k - 1)$. Taking a Waring rank decomposition $g_k = \sum_{i=1}^{\text{WR}(g_k)} L_i^{r_k-1}$ and multiplying it by $\ell_k^{d-r_k+1}$, we obtain a decomposition

$$\ell_k^{d-r_k+1} g_k = \sum_{i=1}^{\text{WR}(g_k)} \ell_k^{d-r_k+1} \cdot L_i^{r_k-1}.$$

It is known that $\text{WR}(y_1^a y_2^b) = \max\{a, b\} + 1$ (this is a classical fact known at least to Oldenburger [47], see also [19])². It follows that

$$\text{WR}(\ell_k^{d-r_k+1} L_i^{r_k-1}) \leq \text{WR}(y_1^{d-r_k+1} y_2^{r_k-1}) = \max\{d - r_k + 2, r_k\} \leq d.$$

Hence we have $\text{WR}(\ell_k^{d-r_k+1} g_k) \leq d \cdot \text{WR}(g_k) \leq d \cdot \max\text{R}(r_k - 1, r_k)$.

Combining all parts of the decomposition together, we get

$$\text{WR}(f) \leq d \sum_{k=1}^m \max\text{R}(r - k - 1, r_k) \leq M(r) \cdot d. \quad \blacktriangleleft$$

A more explicit upper bound is provided by the following immediate corollary.

► **Theorem 14.** *Let $f \in \mathbb{C}[x_1, \dots, x_n]_d$ and let $\underline{\text{WR}}(f) = r$. Then*

$$\text{WR}(f) \leq \binom{2r-2}{r-1} \cdot d.$$

Proof. The space of homogeneous polynomials of degree $r - 1$ in r variables has dimension $\binom{2r-2}{r-1}$ and is spanned by powers of linear forms. Therefore, $\max\text{R}(r - 1, r) \leq \binom{2r-2}{r-1}$. Note that if $r = p + q$ with $p, q \neq 0$, then the space $\mathbb{C}[x_1, \dots, x_r]_{r-1}$ contains a direct sum of $x_1^q \cdot \mathbb{C}[x_1, \dots, x_p]_{p-1}$ and $x_1^{p+1} \cdot \mathbb{C}[x_{p+1}, \dots, x_r]_{q-1}$. Taking the dimensions of these spaces, we obtain $\binom{2r-2}{r-1} \geq \binom{2p-2}{p-1} + \binom{2q-2}{q-1}$. It follows that $M(r) \leq \binom{2r-2}{r-1}$. ◀

Using the Blekherman–Teitler bound on the maximum rank [13], we can get a slightly better bound. The proof is essentially the same as for the previous theorem.

► **Corollary 15.** *Let $f \in \mathbb{C}[x_1, \dots, x_n]_d$ and let $\underline{\text{WR}}(f) = r$. Then*

$$\text{WR}(f) \leq 2 \left\lceil \frac{1}{r} \binom{2r-2}{r-1} \right\rceil \cdot d.$$

² it is easy to see that for $a \geq b$ the monomial $y_1^a y_2^b$ is proportional to $\sum_{k=0}^a \zeta^k (\zeta^k y_1 + y_2)^{a+b}$ where ζ is a primitive root of unity of order $a + 1$.

2.4 Scheme-theoretic proof

In this section we give a proof of Lemma 10 based on the theory of 0-dimensional schemes and apolarity. This short section assumes familiarity with these topics, we review them in more details in Appendix A of the extended version of the paper [27].

► **Lemma 10.** *Let $f \in \mathbb{C}[\mathbf{x}]_d$ be such that $\underline{\text{WR}}(f) = r$. If $d \geq r - 1$, then there exists a partition $r = r_1 + \dots + r_m$ such that f has a generalized additive decomposition*

$$f = \sum_{k=1}^m \ell_k^{d-r_k+1} g_k,$$

and moreover $\underline{\text{WR}}(\ell_k^{d-r_k+1} g_k) \leq r_k$.

Alternative proof. Denote by V the space of linear forms $\mathbb{C}[\mathbf{x}]_1$.

Since $d \geq r - 1$, the border Waring rank of f is equal to its *smoothable rank* $\text{SR}(f)$ [16], that is, there exists a 0-dimensional scheme $Z \subset \mathbb{P}V$ of degree r which is smoothable (obtained as a flat limit of the family of r -point subsets of $\mathbb{P}V$) and f is apolar to Z . Let I be the ideal of Z and let $I = I^{(1)} \cap \dots \cap I^{(m)}$ be the primary decomposition of this ideal. The primary ideals $I^{(j)}$ correspond to irreducible components Z_j of the scheme Z .

Since f is apolar to I , we have $f \in I_d^\perp = (I_d^{(1)})^\perp + \dots + (I_d^{(m)})^\perp$. In particular, there exist $f_j \in (I_d^{(j)})^\perp$ such that $f = f_1 + \dots + f_m$. Let r_j be the degree of Z_j . By the definition of degree, $r = r_1 + \dots + r_m$. If Z_j is supported at the point $[\ell_j] \in \mathbb{P}V$, then for the ideal $I^{(j)}$ we have $(\ell_j^\perp)^{r_j} \subset I^{(j)} \subset \ell_j^\perp$ and $(I_d^{(j)})^\perp \subset \ell_j^{d-r_j+1} \cdot \mathbb{C}[\mathbf{x}]_{r_j-1}$. Therefore the polynomials f_j have the form $\ell_j^{d-r_j+1} g_j$ for some g_j of degree $\deg g_j = r_j - 1$.

Additionally, all irreducible components of a smoothable scheme Z are smoothable [17, Thm. 1.1], and since f_j is apolar to Z_j , we have $\underline{\text{WR}}(f_j) \leq \text{SR}(f_j) \leq r_j$. ◀

References

- 1 Alexander Alder. *Grenzzrang und Grenzkomplexität aus algebraischer und topologischer Sicht*. PhD thesis, Universität Zürich, 1984.
- 2 J. Alexander and A. Hirschowitz. Polynomial interpolation in several variables. *J. Alg. Geom.*, 4(2):201–222, 1995.
- 3 Boris Alexeev, Michael A. Forbes, and Jacob Tsimerman. Tensor rank: Some lower and upper bounds. In *CCC 2011*, pages 283–291. IEEE Computer Society, 2011. doi:10.1109/CCC.2011.28.
- 4 Eric Allender and Fengming Wang. On the power of algebraic branching programs of width two. *Comput. Complex.*, 25(1):217–253, 2016. doi:10.1007/s00037-015-0114-7.
- 5 Edoardo Ballico. On the ranks of homogeneous polynomials of degree at least 9 and border rank 5. *Note di Matematica*, 38(2):55–92, 2019. doi:10.1285/i15900932v38n2p55.
- 6 Edoardo Ballico and Alessandra Bernardi. Stratification of the fourth secant variety of Veronese varieties via the symmetric rank. *Adv. Pure Appl. Math.*, 4(2):215–250, 2013. doi:10.1515/apam-2013-0015.
- 7 Edoardo Ballico and Alessandra Bernardi. Curvilinear schemes and maximum rank of forms. *Le Matematiche*, 72(1):137–144, 2017. doi:10.4418/2017.72.1.10.
- 8 Alessandra Bernardi, Jérôme Brachat, and Bernard Mourrain. A comparison of different notions of ranks of symmetric tensors. *Linear Algebra Appl.*, 460:205–230, 2014. doi:10.1016/j.laa.2014.07.036.
- 9 Alessandra Bernardi, Alessandro Oneto, and Daniele Tauffer. On schemes evinced by generalized additive decompositions and their regularity, 2023. arXiv:2309.12961.

- 10 Dario Bini. Relations between exact and approximate bilinear algorithms. Applications. *Calcolo*, 17(1):87–97, 1980. doi:10.1007/BF02575865.
- 11 Dario Bini, Milvio Capovani, Francesco Romani, and Grazia Lotti. $O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication. *Inf. Process. Lett.*, 8(5):234–235, 1979. doi:10.1016/0020-0190(79)90113-3.
- 12 Markus Bläser, Julian Dörfler, and Christian Ikenmeyer. On the complexity of evaluating highest weight vectors. In *36th Computational Complexity Conference (CCC 2021)*, volume 200 of *LIPICs*, pages 29:1–29:36. Dagstuhl, 2021. doi:10.4230/LIPICs.CCC.2021.29.
- 13 Grigoriy Blekherman and Zach Teitler. On maximum, typical and generic ranks. *Math. Ann.*, 362(3-4):1021–1031, 2015. doi:10.1007/s00208-014-1150-3.
- 14 Karl Bringmann, Christian Ikenmeyer, and Jeroen Zuiddam. On algebraic branching programs of small width. *J. ACM*, 65(5):32:1–32:29, 2018. doi:10.1145/3209663.
- 15 Weronika Buczyńska and Jarosław Buczyński. Secant varieties to high degree Veronese reembeddings, catalecticant matrices and smoothable Gorenstein schemes. *J. Algebraic Geom.*, 23(1):63–90, 2014. doi:10.1090/S1056-3911-2013-00595-0.
- 16 Weronika Buczyńska and Jarosław Buczyński. On differences between the border rank and the smoothable rank of a polynomial. *Glasg. Math. J.*, 57(2):401–413, 2015. doi:10.1017/S0017089514000378.
- 17 Jarosław Buczyński and Joachim Jelisiejew. Finite schemes and secant varieties over arbitrary characteristic. *Differential Geom. Appl.*, 55:13–67, 2017. doi:10.1016/j.difgeo.2017.08.004.
- 18 Weronika Buczyńska and Jarosław Buczyński. Apolarity, border rank, and multigraded Hilbert scheme. *Duke Math. J.*, 170(16):3659–3702, 2021. doi:10.1215/00127094-2021-0048.
- 19 Weronika Buczyńska, Jarosław Buczyński, and Zach Teitler. Waring decompositions of monomials. *Journal of Algebra*, 378:45–57, 2013. doi:10.1016/j.jalgebra.2012.12.011.
- 20 Peter Bürgisser. The complexity of factors of multivariate polynomials. *Found. Comput. Math.*, 4(4):369–396, 2004. doi:10.1007/s10208-002-0059-5.
- 21 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, 1997. doi:10.1007/978-3-662-03338-8.
- 22 Enrico Carlini. Reducing the number of variables of a polynomial. In *Algebraic geometry and geometric modeling*, pages 237–247. Springer, Berlin, 2006. doi:10.1007/978-3-540-33275-6_15.
- 23 Arthur Cayley. On the theory of linear transformations. *Cambridge Math. J.*, IV:193–209, 1845. Reprinted in A. Cayley, *The Collected Mathematical Papers I*, 80–94, Cambridge University Press, 1889. doi:10.1017/CB09780511703676.014.
- 24 Luca Chiantini, Jonathan D. Hauenstein, Christian Ikenmeyer, Joseph M. Landsberg, and Giorgio Ottaviani. Polynomials and the exponent of matrix multiplication. *Bull. LMS*, 50(3):369–389, 2018. doi:10.1112/blms.12147.
- 25 Alfred Clebsch. Zur Theorie der algebraischen Flächen. *J. Reine Angew. Math.*, 58:93–108, 1861. doi:10.1515/crll.1861.58.93.
- 26 Pranjal Dutta, Prateek Dwivedi, and Nitin Saxena. Demystifying the border of depth-3 algebraic circuits. In *FOCS 2021*, pages 92–103. IEEE, 2021. doi:10.1109/FOCS52979.2021.00018.
- 27 Pranjal Dutta, Fulvio Gesmundo, Christian Ikenmeyer, Gorav Jindal, and Vladimir Lysikov. Fixed-parameter debordering of Waring rank, 2024. (Extended version of this paper). arXiv:2401.07631.
- 28 Klim Efremenko, Ankit Garg, Rafael Mendes de Oliveira, and Avi Wigderson. Barriers for rank methods in arithmetic complexity. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *LIPICs*, pages 1:1–1:19. Dagstuhl, 2018. doi:10.4230/LIPICs.ITCS.2018.1.

- 29 Michael Forbes. Some concrete questions on the border complexity of polynomials. Presentation given at the Workshop on Algebraic Complexity Theory (WACT 2016) in Tel Aviv, 2016. URL: <https://www.youtube.com/watch?v=1HMogQIHT6Q>.
- 30 Maciej Gałazka. Vector bundles give equations of cactus varieties. *Lin. Alg. Appl.*, 521:254–262, 2017. doi:10.1016/j.laa.2016.12.005.
- 31 Ankit Garg, Visu Makam, Rafael Mendes de Oliveira, and Avi Wigderson. More barriers for rank methods, via a "numeric to symbolic" transfer. In *FOCS 2019*, pages 824–844. IEEE, 2019. doi:10.1109/FOCS.2019.00054.
- 32 John Hilton Grace and Alfred Young. *The Algebra of Invariants*. Cambridge Library Collection — Mathematics. Cambridge University Press, 2010. doi:10.1017/CB09780511708534.
- 33 Leonid Gurvits. Ryser (or polarization) formula for the permanent is essentially optimal: the Waring rank approach. Technical Report LA-UR08-06583, Los Alamos National Laboratory, 2008.
- 34 Anthony Iarrobino. Inverse system of a symbolic power II. the Waring problem for forms. *Journal of Algebra*, 174(3):1091–1110, 1995. doi:10.1006/jabr.1995.1169.
- 35 Anthony Iarrobino and Vassil Kanev. *Power sums, Gorenstein algebras, and determinantal loci*, volume 1721 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1999. doi:10.1007/BFb0093426.
- 36 Christian Ikenmeyer and Abhiroop Sanyal. A note on VNP-completeness and border complexity. *Inf. Process. Lett.*, 176:106243, 2022. doi:10.1016/j.ip1.2021.106243.
- 37 Joachim Jelisiejew. An upper bound for the Waring rank of a form. *Arch. Math. (Basel)*, 102(4):329–336, 2014. doi:10.1007/s00013-014-0632-6.
- 38 Joachim Jelisiejew. Pathologies on the Hilbert scheme of points. *Inventiones mathematicae*, 220(2):581–610, 2020. doi:10.1007/s00222-019-00939-5.
- 39 Joachim Jelisiejew and Tomasz Mańdziuk. Limits of saturated ideals, 2022. arXiv:2210.13579.
- 40 Neeraj Kayal and Nitin Saxena. Polynomial identity testing for depth 3 circuits. *comput. complex.*, 16(2):115–138, 2007. doi:10.1007/s00037-007-0226-9.
- 41 Hanspeter Kraft. *Geometrische Methoden in der Invariantentheorie*. Springer, Berlin, 2 edition, 1985. doi:10.1007/978-3-663-10143-7.
- 42 Mrinal Kumar. On the power of border of depth-3 arithmetic circuits. *ACM Trans. Comput. Theory*, 12(1):5:1–5:8, 2020. doi:10.1145/3371506.
- 43 Joseph M. Landsberg and Zach Teitler. On the ranks and border ranks of symmetric tensors. *Found. Comput. Math.*, 10(3):339–366, 2010. doi:10.1007/s10208-009-9055-3.
- 44 Thomas Lehmkuhl and Thomas Lickteig. On the order of approximation in approximative triadic decompositions of tensors. *Theor. Comput. Sci.*, 66(1):1–14, 1989. doi:10.1016/0304-3975(89)90141-2.
- 45 Ketan Mulmuley and Milind A. Sohoni. Geometric complexity theory I: an approach to the P vs. NP and related problems. *SIAM J. Comput.*, 31(2):496–526, 2001. doi:10.1137/S009753970038715X.
- 46 Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *STOC'91: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 410–418. ACM, 1991. doi:10.1145/103418.103462.
- 47 Rufus Oldenburger. Polynomials in several variables. *Annals of Mathematics*, 41(3):694–710, 1940. doi:10.2307/1968741.
- 48 Francesco Palatini. Sulle superficie algebriche i cui S_h ($h + 1$)-seganti non riempiono lo spazio ambiente. *Atti della R. Acc. delle Scienze di Torino*, 41:634–640, 1906. URL: <https://archive.org/details/attidellarealeac41real/page/634>.
- 49 Nitin Saxena. Diagonal circuit identity testing and lower bounds. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Part I*, volume 5125 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2008. doi:10.1007/978-3-540-70575-8_6.

- 50 James Joseph Sylvester. On the principles of the calculus of forms. *J. Cambridge and Dublin Math.*, 7:52–97, 1852. Reprinted in J. J. Sylvester, *The Collected Mathematical Papers I*, 284–327, Cambridge University Press, 1904.
- 51 Alessandro Terracini. Sulle V_k per cui la varietà degli S_h ($h + 1$)-seganti ha dimensione minore dell'ordinario. *Rend. Circ. Mat.*, 31:392–396, 1911. doi:10.1007/BF03018812.
- 52 WACT 2016. Some accessible open problems. URL: <https://www.cs.tau.ac.il/~shpilka/wact2016/concreteOpenProblems/openprobs.pdf>.

On the Power of Border Width-2 ABPs over Fields of Characteristic 2

Pranjal Dutta ✉🏠

National University of Singapore, Singapore

Christian Ikenmeyer ✉🏠

University of Warwick, UK

Balagopal Komarath ✉🏠

Indian Institute of Technology Gandhinagar, India

Harshil Mittal ✉

Indian Institute of Technology Gandhinagar, India

Saraswati Girish Nanoti ✉

Indian Institute of Technology Gandhinagar, India

Dhara Thakkar ✉🏠

Indian Institute of Technology Gandhinagar, India

Abstract

The celebrated result by Ben-Or and Cleve [SICOMP92] showed that algebraic formulas are polynomially equivalent to width-3 algebraic branching programs (ABP) for computing polynomials. i.e., $\mathbf{VF} = \mathbf{VBP}_3$. Further, there are simple polynomials, such as $\sum_{i=1}^8 x_i y_i$, that cannot be computed by width-2 ABPs [Allender and Wang, CC16]. Bringmann, Ikenmeyer and Zuiddam, [JACM18], on the other hand, studied these questions in the setting of approximate (i.e., border complexity) computation, and showed the universality of border width-2 ABPs, over fields of characteristic $\neq 2$. In particular, they showed that polynomials that can be approximated by formulas can also be approximated (with only a polynomial blowup in size) by width-2 ABPs, i.e., $\overline{\mathbf{VF}} = \overline{\mathbf{VBP}}_2$. The power of border width-2 algebraic branching programs when the characteristic of the field is 2 was left open.

In this paper, we show that width-2 ABPs can approximate every polynomial irrespective of the field characteristic. We show that any polynomial f with ℓ monomials and with at most t odd-power indeterminates per monomial can be approximated by $\mathcal{O}(\ell \cdot (\deg(f) + 2^t))$ -size width-2 ABPs. Since ℓ and t are finite, this proves universality of border width-2 ABPs. For univariate polynomials, we improve this upper-bound from $\mathcal{O}(\deg(f)^2)$ to $\mathcal{O}(\deg(f))$.

Moreover, we show that, if a polynomial f can be approximated by small formulas, then the polynomial f^d , for some small power d , can be approximated by small width-2 ABPs. Therefore, even over fields of characteristic two, border width-2 ABPs are a reasonably powerful computational model. Our construction works over any field.

2012 ACM Subject Classification Theory of computation \rightarrow Algebraic complexity theory

Keywords and phrases Algebraic branching programs, border complexity, characteristic 2

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.31

Funding *Pranjal Dutta*: Supported by the project “Foundation of Lattice-based Cryptography”, funded by NUS-NCS Joint Laboratory for Cyber Security, Singapore.

Christian Ikenmeyer: Supported by EPSRC grant EP/W014882/1.

Saraswati Girish Nanoti: Funded by CSIR NET JRF Fellowship.

Dhara Thakkar: Funded by CSIR-UGC NET JRF Fellowship.



© Pranjal Dutta, Christian Ikenmeyer, Balagopal Komarath, Harshil Mittal, Saraswati Girish Nanoti, and Dhara Thakkar; licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 31; pp. 31:1–31:16



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



1 Introduction

The fundamental aim in computational complexity theory is to separate computational complexity classes – classes of problems that can be solved using a bounded amount of computational resources (e.g., time, space). Despite a lot of research, separating classes has remained elusive because the general computational model, Turing machines, are surprisingly difficult to prove lower bounds against. Valiant [22] proposed a computational complexity theory for families of multivariate polynomials, now called *algebraic complexity*, where the computational models only use algebraic operations such as addition $+$, multiplication \times , etc. The central question in algebraic complexity is to compare the computational power of the permanent and determinant polynomials, for a symbolic matrix $\mathbf{X}_n = (x_{i,j})_{i,j \in [n]}$, defined as follows:

$$\begin{aligned} \text{per}_n &:= \text{per}_n(\mathbf{X}_n) = \sum_{\sigma \in S_n} \prod_{i=1}^n x_{i,\sigma(i)}, \\ \text{det}_n &:= \text{det}_n(\mathbf{X}_n) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^n x_{i,\sigma(i)}. \end{aligned}$$

The summations above are over all permutations on n elements. Efficient algorithms to compute the determinant of a matrix whose entries are from a suitable ring (e.g. integers) are known [3, 14]. However, efficient algorithms to compute the permanent would imply that $\#\text{P} = \text{FP}$, which is widely believed to be false.

A sequence $(c_n)_{n \in \mathbb{N}}$ of natural numbers is called *polynomially bounded* if there exists a polynomial q with $\forall n : c_n \leq q(n)$. A p -family is a sequence of polynomials whose degree and number of variables are polynomially bounded. Usually, algebraic complexity theorists are concerned with explicit p -families (e.g., $(\text{det}_n)_n, (\text{per}_n)_n$) because of its intimate connections to Boolean complexity.

One can define the *determinantal complexity* of a multivariate polynomial $f \in \mathbb{F}[\mathbf{x}]$ over a field \mathbb{F} , denoted $\text{dc}(f)$, to be the smallest n such that f can be written as the determinant of an $n \times n$ matrix with entries being affine linear forms (i.e. of the form $a_0 + a_1x_1 + \dots + a_nx_n$, where $a_i \in \mathbb{F}$). The class VBP consists of all p -families $(f_n)_{n \in \mathbb{N}}$ for which the determinantal complexity is polynomially bounded, see e.g. [13]. Interestingly, VBP can be captured by *algebraic branching programs* (ABPs) which can be thought of as a product of $w \times w$ matrices with affine linear entries, and w is called the *width* of the ABP.

The *permanental complexity* of a polynomial f , denoted $\text{pc}(f)$, is the smallest n such that f can be written as the permanent of an $n \times n$ matrix of affine linear forms. The class VNP consists of all p -families $(f_n)_{n \in \mathbb{N}}$ for which the permanental complexity is polynomially bounded.

It is known that $\text{VBP} \subseteq \text{VNP}$ [22, 21]. One of the central questions in algebraic complexity is Valiant's conjecture of $\text{VNP} \not\subseteq \text{VBP}$, or equivalently proving $\text{dc}(\text{per}_n) = n^{\omega(1)}$ [22]. This is often known as the *determinant vs permanent* problem. The best known bounds for $\text{dc}(\text{per}_n)$, over $\mathbb{F} = \mathbb{C}$ is: $n^2/2 \leq \text{dc}(\text{per}_n) \leq 2^n - 1$ [15, 10].

IMM-complexity. There are plausibly *weaker* classes than VBP , such as VF that tries to capture the *algebraic formula complexity* of polynomial families. An algebraic formula is a directed tree with a unique sink vertex. The source vertices are labelled by variables or constants from \mathbb{F} , and each internal node of the graph is labelled by either $+$ or \times . Nodes compute polynomials in the natural way by induction. The size of a formula is the number of its nodes. Finally, the *algebraic formula complexity* of a polynomial f is the minimum

size of a formula computing f . Ben-Or and Cleve [2] showed a surprising result that the polynomial family constructed using an iterated product of 3×3 symbolic matrices (formally it is called IMM_3 , see Definition 7) is computationally *equivalent* to algebraic formulas. And further, Valiant showed that any polynomial f with algebraic formula complexity s , has determinantal complexity at most $2s$ [22]. Therefore, separation questions like VF vs. VBP , and VF vs. VNP can be framed as whether $\text{immc}_3(\det_n) = n^{\omega(1)}$, and $\text{immc}_3(\text{per}_n) = n^{\omega(1)}$; for a formal definition of IMM-complexity for 3×3 matrices (immc_3), see Definition 9.

Universality vs. impossibility. It is noteworthy that all the above-mentioned complexity measures (dc , pc , immc_3) are *finite* for any polynomial $f \in \mathbb{F}[\mathbf{x}]$; in other words, the model of computation defined by these complexity measures are “*universal*”. Given the phenomenon of universality and the results of Ben-Or and Cleve and Valiant, it is natural to study the computational power of iterated multiplication of 2×2 matrices. Astonishingly, Allender and Wang [1] showed an *impossibility* result that the polynomial $\sum_{i=1}^8 x_i y_i$ *cannot* be computed using IMM_2 . In other words, the IMM_2 -complexity (Definition 9) of this polynomial is infinite! However, Bringmann, Ikenmeyer, and Zuiddam [4] showed that by allowing *approximations*, the polynomial family IMM_2 becomes universal! In fact, they proved a stronger statement that the IMM_2 -*approximation complexity*, which we denote by immc_2 , is polynomially related to approximate algebraic formula complexity. However, their proofs only work over fields \mathbb{F} when $\text{char}(\mathbb{F}) \neq 2$. They left open the following, which sets the fundamental basis for this work.

► **Question 1** ([4]). *Determine the computational power of IMM_2 with approximations over fields of characteristic 2.*

Border complexity & GCT. The study of *border complexity* measures, by allowing approximations in the algebraic model was first introduced in [17, 5]. Given $f \in \mathbb{F}[\mathbf{x}]$ and a suitable associated complexity measure Γ , the border- Γ complexity of f (denoted $\underline{\Gamma}(f)$) is the *smallest* n such that f can be approximated arbitrarily closely by polynomials of Γ -complexity at most n . Trivially, $\underline{\Gamma}(f) \leq \Gamma(f)$, for any f . By this definition, one can talk about the border-complexity measures such as $\underline{\text{immc}}$, $\underline{\text{dc}}$, $\underline{\text{pc}}$ etc. Replacing a complexity measure by its border measure in a complexity class, we obtain the *closure* of this class, such as $\overline{\text{VF}}$, $\overline{\text{VBP}}$, $\overline{\text{VNP}}$, and so on. The operation of going to the closure is indeed a closure operator in the sense of topology (See [11]). The original Geometric Complexity Theory (GCT) papers [17, 18] propose to use representation-theoretic techniques to separate VNP from $\overline{\text{VBP}}$ by studying the determinant orbit closure, but progress has been slow. Simpler models of computation are desirable to study the easier $\text{VNP} \not\subseteq \overline{\text{VF}}$ conjecture, for example immc_3 , or even the much simpler immc_2 . This was a main motivation for [4], but their result does not work in characteristic 2. This naturally leads to the following question.

► **Question 2.** *How is immc_2 related to immc_3 for fields of characteristic 2?*

Division and powering. Strassen [20] showed that we can eliminate divisions in algebraic circuits and formulas computing polynomials without loss of efficiency. The result relies on the ability to compute small powers of polynomials efficiently. This naturally leads to the following question.

► **Question 3.** *Given border width-2 computations for polynomials f and g , can we also compute $\frac{f}{g}$ (given g divides f) and f^r , for small r , efficiently?*

More generally, one can ask, given computations for f and g , what combinations of f and g are possible in the model? A known approach to produce such results is to use Waring decompositions (See [4, 12]). Given a homogeneous degree d polynomial f , the *Waring rank* of f , denoted $\text{WR}(f)$, is the smallest r such that there exist homogeneous linear polynomials ℓ_1, \dots, ℓ_r with $f = \sum_{i=1}^r \ell_i^d$. Border Waring rank, denoted $\text{WR}(f)$, can be defined analogously in the border setup. For example, a border Waring decomposition for xy would allow us to compute the product fg using only addition, scaling by constants, and squaring. Over fields of characteristic 2, the border Waring rank of xy is infinite and hence, this technique becomes infeasible.

1.1 Our Contributions

Our main theorem is to answer Question 1 by showing the universality of $\text{imm}_{\mathbb{C}_2}$:

► **Theorem 4** (Universality of $\text{imm}_{\mathbb{C}_2}$). *$\text{imm}_{\mathbb{C}_2}(f)$ is finite for every polynomial f , over all fields.*

This theorem over fields of characteristic other than two was proved by Bringmann, Ikenmeyer, and Zuiddam [4]. In fact, they prove the stronger statement that any polynomial family with small algebraic formulas approximating it can also be approximated with IMM_2 with only a polynomial blow-up in complexity. Unfortunately, our construction yields an *exponential* complexity for even simple polynomial families, such as $\prod_{i=1}^n x_i + \prod_{i=1}^n y_i + \prod_{i=1}^n z_i$ (see Theorem 19). However, the next theorem proves that for every polynomial with small formulas approximating them, we can approximate a small power of the polynomial using IMM_2 over any field. This partially answers Question 2.

► **Theorem 5** (Powering is powerful). *There exists a constant k such that for any polynomial f with a size- s formula approximating f , there is a $d \leq s^k + k$ such that $\text{imm}_{\mathbb{C}_2}(f^d) \leq s^k + k$.*

The above theorem shows that the border width-2 ABPs are a reasonably powerful computational model. Further, Theorem 4 and Theorem 5 can be seen as weak extensions of [4], over *any* field, *regardless* of its characteristic and size.

A natural question is which interesting classes of polynomials can be efficiently approximated using IMM_2 . In Theorem 19, we show that every sparse polynomial family (i.e., the number of monomials is $\text{poly}(n)$) where the monomials do not have a large number of variables with odd degree can be efficiently approximated. A particularly interesting subset of this class is the class of all univariate polynomials. Applying Theorem 19 to univariate polynomials, we obtain a computation of any degree- d univariate polynomial using $O(d^2)$ operations. But, Horner's rule gives a formula for any degree- d univariate polynomial that only uses $O(d)$ operations. The following theorem is a refinement of Theorem 19 to univariates where we show that every degree- d univariate can be approximated using $O(d)$ matrices. This construction is a consequence of our partial answers towards Question 3.

► **Theorem 6.** *For any degree- d univariate polynomial f , we have $\text{imm}_{\mathbb{C}_2}(f) \leq \frac{9d+4}{2}$.*

We leave open the question whether $\text{imm}_{\mathbb{C}_2}$ is polynomially related to approximate algebraic formula complexity over fields of characteristic 2.

1.2 Comparison with previous works

As mentioned before, [4] showed that any polynomial with small border algebraic formula complexity have small $\text{imm}_{\mathbb{C}_2}$ -complexity, when $\text{char}(\mathbb{F}) \neq 2$. Their proof was constructive, and *fundamentally* (& inductively) used the following identity: $x \cdot y = \frac{1}{2} \cdot ((x+y)^2 - x^2 - y^2)$. One

could also use even a smaller representation: $x \cdot y = (\frac{1}{2} \cdot (x + y))^2 - (\frac{1}{2} \cdot (x - y))^2$. However both representations use the constant $\frac{1}{2}$, and one can show that one *cannot* come up with an identity which does not use $\frac{1}{2^n}$, for some $n \in \mathbb{N}$. In other words, $\text{WR}(x \cdot y) = \underline{\text{WR}}(x \cdot y) = \infty$ over \mathbb{F} with $\text{char}(\mathbb{F}) = 2$. Therefore, their construction fails miserably over characteristic 2 fields.

On the other hand, Kumar [12] showed that for any $f \in \mathbb{C}[\mathbf{x}]$, a constant multiple of f can be approximated by $\prod_{i \in [m]} (1 + \ell_i) - 1$, where ℓ_i are linear polynomials in $\mathbb{C}(\epsilon)[\mathbf{x}]$. Note that, this implies that $\text{imm}_{\mathbb{C}_2}(f) \leq m$. The representation depends on the Waring decomposition of f , and further one can show that for the minimum m : $\underline{\text{WR}}(f) \leq m \leq \text{deg}(f) \cdot \underline{\text{WR}}(f)$ [8]. However, over \mathbb{F} of $\text{char}(\mathbb{F}) = 2$, for any $d \geq 2$, there are d -degree polynomials (e.g., $x_1 \cdots x_d$) which has infinite border Waring rank, and hence the above universality proof fails.

In this work, we come up with a *Waring-free* proof to show the universality over characteristic 2 fields, and therefore our proofs are very different (yet *simple*) from the known constructive proofs.

1.3 Proof ideas

The key building block in the proof of universality of border width-2 ABPs over fields characteristic $\neq 2$ in [4] is a Q matrix. For a polynomial f , they define $Q(f) = \begin{pmatrix} f & 1 \\ 1 & 0 \end{pmatrix}$. Given $Q(f)$ and $Q(g)$, $Q(f+g)$ can be computed as $Q(f)Q(0)Q(g)$. So, to prove universality, it suffices to show that $Q(fg)$ can also be computed from $Q(f)$ and $Q(g)$. Bringmann, Ikenmeyer and Zuiddam [4] showed that $Q(f^2)$ can be approximately computed using $Q(f)$, and then the identity $fg = (\frac{1}{2}(f+g))^2 - (\frac{1}{2}(f-g))^2$ can be used to compute the product using squaring, addition, and scaling by constants. As discussed before, such an identity does not exist over fields of characteristic two.

We overcome this block by not trying to compute the product of two arbitrary polynomials. We observe that for universality, it is enough to be able to compute $Q(fx)$ from $Q(f)$ for an arbitrary variable x . The advantage is that since x is a variable and not an arbitrary polynomial, we can use any 2×2 matrix that contains only constants and the variable x in the computation of $Q(fx)$, whereas for computing $Q(fg)$, both f and g are available to us only as Q matrices (or in any other form that have been proved inductively). This is the key idea in Lemma 15 (see Section 4).

The source of inefficiency of Lemma 15 is that $Q(f)$ is used twice to compute $Q(fx)$. Therefore, even computing a simple polynomial such as x^n using this lemma takes $\Omega(2^n)$ matrices. Compare this to the computation of $Q(fg)$ in [4] where they use $Q(f)$ and $Q(g)$ at most three times which is enough to stay within a polynomial factor of formula complexity. In Lemma 17, we show that we can compute $Q(fg^2)$ by using $Q(f)$ once and $Q(g)$ twice (see Section 4). This lemma enables efficient computation of powers of polynomials with small formulas (Theorem 20), sparse polynomials where each monomial only contains a few variables with odd power (Theorem 19), and univariate polynomials (Theorem 24). We also use this lemma to compute powers of polynomials efficiently. That is, given a computation of $Q(f)$ using s matrices, compute $Q(f^r)$ using $O(rs)$ matrices (see Section 7). We also observe that the division $Q(\frac{f}{g^2})$ from $Q(f)$ and $Q(g)$ can be performed by combining standard division elimination techniques [20] with Lemma 17 (see Section 8).

2 Preliminaries

We consider polynomial families $f = (f_n)_{n \geq 0}$ over an arbitrary field \mathbb{F} . The n^{th} polynomial in the family f_n is a polynomial in $\mathbb{F}[x_1, \dots, x_m]$ where $m = \text{poly}(n)$. The following polynomial family is particularly important in this paper.

► **Definition 7.** For any fixed, natural $k \geq 1$, we define the polynomial family $\text{IMM}_k = (\text{IMM}_{k,n})$ such that $\text{IMM}_{k,n}$ is the $(1,1)^{\text{th}}$ entry of the product of n matrices of order $k \times k$ where each entry of each matrix is a fresh variable, i.e., the $(i,j)^{\text{th}}$ entry of the m^{th} matrix is the variable $x_{i,j}^{(m)}$ for all $1 \leq i, j \leq k$ and $1 \leq m \leq n$.

► **Definition 8.** A weakest projection from a set of variables X to another set of variables Y is a mapping $X \mapsto Y \cup \mathbb{F}$. A weak projection is a mapping from X to affine linear forms in at most one variable in $\mathbb{F}[Y]$. For polynomials f and g , we say $f \leq^{\text{wst}} g$ ($f \leq^{\text{w}} g$), if there is a weakest projection (resp., weak projection) that maps g to f .

The notion of a projection is used to compare the number of algebraic operations required to compute polynomials. Note that if f_n is computable using s operations and if $g_m \leq^{\text{wst}} f_n$, then g_m is also computable using s operations. The weak variant \leq^{w} weakens this slightly since we can only conclude that g_m can be computed using at most $\text{poly}(s)$ operations.

► **Definition 9.** Let $f = (f_n)$ be a polynomial family. We define the f -complexity wrt \leq^{wst} (or \leq^{w}) of a polynomial g as the smallest m such that $g \leq^{\text{wst}} f_m$ (resp., \leq^{w}). If there is no such m , then the f -complexity of g is ∞ . We define the f -complexity of a polynomial family $g = (g_n)$ as the sequence $s = (s_n)$ where s_n is the f -complexity of the polynomial g_n .

We say that f computes a polynomial g wrt \leq^{wst} (or, \leq^{w}) if f -complexity of g wrt \leq^{wst} (resp., \leq^{w}) is finite.

For $f = (f_n)$, we denote f -complexity wrt \leq^{wst} (or, \leq^{w}) using fc^{wst} (resp., fc^{w}). We omit the projection from the notation if it is the weakest projection. For example, we denote det -complexity, IMM_3 -complexity, and IMM_2 -complexity under weakest projections by dc , immc_3 , and immc_2 respectively.

► **Definition 10.** A polynomial family $f = (f_n)_{n \geq 0}$ is called universal wrt \leq^{wst} (or \leq^{w}) if for any polynomial g , the f -complexity of g wrt \leq^{wst} (resp., \leq^{w}) is finite.

We can now define the approximation equivalent of \leq^{wst} and \leq^{w} .

► **Definition 11.** An approximate weakest projection is a map from X to $Y \cup \mathbb{F}(\epsilon)$. An approximate weak projection is a map from X to affine linear forms in at most one variable in $\mathbb{F}(\epsilon)[Y]$.

Given $f, g \in \mathbb{F}[X]$, we say $f \leq^{\text{wst}} g$ ($f \leq^{\text{w}} g$) if there is an approximate weakest projection (resp., approximate weak projection) that maps g to some polynomial that approximates f .

We can use these to define approximate f -complexity of polynomials.

► **Definition 12.** Let $f = (f_n)$ be a polynomial family. We define the approximate f -complexity of a polynomial g as the smallest m such that $g \leq^{\text{wst}} f_m$ (or $g \leq^{\text{w}} f_m$). If no such m exists, we define the f -complexity of g as ∞ . We define the f -complexity of a polynomial family $g = (g_n)$ as the sequence $s = (s_n)$ where s_n is the f -complexity of the polynomial g_n .

We say that f approximately computes a polynomial g wrt \leq^{wst} (or, \leq^{w}) if the approximate f -complexity of g wrt \leq^{wst} (resp., \leq^{w}) is finite.

We denote approximate f -complexity wrt \leq^{wst} (or, \leq^{w}) $\underline{fc}^{\text{wst}}$ (resp., $\underline{fc}^{\text{w}}$). As before, we omit the projection if it is the weakest projection.

We now introduce some additional definitions that are applicable when $f = \text{IMM}_2$. In this case, we can naturally consider computation of 2×2 matrices of polynomials by f .

► **Definition 13.** Let $A = \begin{pmatrix} g_1 & g_2 \\ g_3 & g_4 \end{pmatrix}$ where g_1, g_2, g_3, g_4 are polynomials. We say that A is computed wrt \leq^{wst} (or, \leq^{w}) by a sequence of m matrices if there is a sequence of m 2×2 matrices, where all $4m$ entries are variables or constants from \mathbb{F} (resp., affine linear forms in at most one variable), such that the product of those matrices is A .

The above definition can be naturally extended into the setting of approximate computation. Following [4], we use the notation $\mathcal{O}(\epsilon^k)$ to denote an arbitrary polynomial in the set $\epsilon^k \mathbb{F}[\epsilon, x_1, \dots, x_n]$.

► **Definition 14.** We say that A is approximately computed wrt \leq^{wst} (or, \leq^{w}) by a sequence of m matrices if there is a sequence of m 2×2 matrices, where all $4m$ entries are variables or constants from $\mathbb{F}(\epsilon)$ (resp., affine linear forms over $\mathbb{F}(\epsilon)$ in at most one variable), such that the product of those matrices is $\begin{pmatrix} g_1 + \mathcal{O}(\epsilon) & g_2 + \mathcal{O}(\epsilon) \\ g_3 + \mathcal{O}(\epsilon) & g_4 + \mathcal{O}(\epsilon) \end{pmatrix}$.

We omit the projection if it is the weakest projection. All results in this paper except Theorem 26 hold wrt weakest projections.

3 Approximately computing the Allender-Wang polynomial over fields of characteristic 2

Allender and Wang showed that $\text{imm}_2(\text{AW}) = \infty$ where $\text{AW} = \sum_{i=1}^8 x_i y_i$. Bringmann, Ikenmeyer, and Zuiddam (See Example 3.8 in [4]) constructed an approximation to the AW polynomial when $\text{char}(\mathbb{F}) \neq 2$ thereby showing that $\text{imm}_2(\text{AW})$ is finite when $\text{char}(\mathbb{F}) \neq 2$. Here, we show that it is finite when $\text{char}(\mathbb{F}) = 2$ as well.

We restate the definition of Q -matrix computing a polynomial f from [4].

$$Q(f) = \begin{pmatrix} f & 1 \\ 1 & 0 \end{pmatrix}$$

Observe that $Q(f + g) = Q(f)Q(0)Q(g)$. That is, if we can compute two polynomials as Q -matrices, then we can also compute their sum as a Q -matrix. Now, let

$$F(x, y) := \begin{pmatrix} \frac{1}{\epsilon} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\epsilon} & y \\ -1 & 1 \end{pmatrix} \begin{pmatrix} x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & -\epsilon \end{pmatrix}.$$

Note that $F(x, y)$ computes $\begin{pmatrix} xy & 1 \\ 1 + \epsilon y & 0 \end{pmatrix}$.

Finally, the following sequence approximately computes AW:

$$(1 \ 0) F(x_1, y_1) \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} F(x_2, y_2) \cdots \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} F(x_8, y_8) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \text{AW} + \mathcal{O}(\epsilon).$$

This shows that $\text{imm}_2(\text{AW}) \leq 55$. The above computation works over all fields, irrespective of the characteristic.

4 Universality of IMM₂ with approximations

The key idea in [4] that allows IMM₂ to efficiently simulate formulas is a way to compute $Q(f^2)$ from $Q(f)$ (squaring). Then, the identity $fg = ((f+g)^2 - f^2 - g^2)/2$ that is valid only when $\text{char}(\mathbb{F}) \neq 2$ is used to compute $Q(fg)$ from $Q(f)$ and $Q(g)$ using addition and squaring. The following lemma allows one to multiply an arbitrary polynomial with any indeterminate when $\text{char}(\mathbb{F}) = 2$.

► **Lemma 15.** *Let f be a polynomial. Suppose that there is a sequence, say σ , of N matrices that approximately computes $Q(f)$. Then, for any indeterminate x , there is a sequence of $2N + 4$ matrices that approximately computes $Q(fx)$.*

Proof. Consider the following sequence, say σ' , of $2N + 4$ matrices:

$$\begin{pmatrix} \frac{1}{\epsilon} & 0 \\ 0 & 1 \end{pmatrix} \sigma|_{\epsilon \rightarrow \epsilon^2} \begin{pmatrix} \epsilon & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\epsilon} & x \\ -1 & 1 \end{pmatrix} \sigma|_{\epsilon \rightarrow \epsilon^2} \begin{pmatrix} 1 & 0 \\ 1 & -\epsilon \end{pmatrix}$$

where $\sigma|_{\epsilon \rightarrow \epsilon^2}$ denotes the sequence obtained from σ by replacing ϵ with ϵ^2 .

Note that σ' computes

$$\begin{aligned} & \begin{pmatrix} \frac{1}{\epsilon} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} f + \mathcal{O}(\epsilon^2) & 1 + \mathcal{O}(\epsilon^2) \\ 1 + \mathcal{O}(\epsilon^2) & \mathcal{O}(\epsilon^2) \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\epsilon} & x \\ -1 & 1 \end{pmatrix} \begin{pmatrix} f + \mathcal{O}(\epsilon^2) & 1 + \mathcal{O}(\epsilon^2) \\ 1 + \mathcal{O}(\epsilon^2) & \mathcal{O}(\epsilon^2) \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & -\epsilon \end{pmatrix} \\ &= \begin{pmatrix} \frac{f}{\epsilon} + \mathcal{O}(\epsilon) & \frac{1}{\epsilon} + \mathcal{O}(\epsilon) \\ 1 + \mathcal{O}(\epsilon^2) & \mathcal{O}(\epsilon^2) \end{pmatrix} \begin{pmatrix} 0 & \epsilon x + 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} f + 1 + \mathcal{O}(\epsilon^2) & -\epsilon + \mathcal{O}(\epsilon^3) \\ 1 + \mathcal{O}(\epsilon^2) & \mathcal{O}(\epsilon^3) \end{pmatrix} \\ &= \begin{pmatrix} -\frac{1}{\epsilon} + \mathcal{O}(\epsilon) & fx + \frac{f+1}{\epsilon} + \mathcal{O}(\epsilon) \\ \mathcal{O}(\epsilon^2) & \epsilon x + 1 + \mathcal{O}(\epsilon^2) \end{pmatrix} \begin{pmatrix} f + 1 + \mathcal{O}(\epsilon^2) & -\epsilon + \mathcal{O}(\epsilon^3) \\ 1 + \mathcal{O}(\epsilon^2) & \mathcal{O}(\epsilon^3) \end{pmatrix} \\ &= \begin{pmatrix} fx + \mathcal{O}(\epsilon) & 1 + \mathcal{O}(\epsilon^2) \\ 1 + \epsilon x + \mathcal{O}(\epsilon^2) & \mathcal{O}(\epsilon^3) \end{pmatrix}. \quad \blacktriangleleft \end{aligned}$$

We also provide a Macaulay program in Appendix A to verify the construction described in the proof of Lemma 15. Although not as powerful as multiplying two arbitrary polynomials, Lemma 15 is sufficient to prove universality. Let p be a polynomial with ℓ monomials. Note that for any monomial, say m , of p , repeatedly applying Lemma 15 gives a sequence of $\mathcal{O}(2^{\deg(m)})$ matrices that approximately computes $Q(m)$. Thus, $Q(p)$ can be approximately computed using a sequence of $\mathcal{O}(\ell \cdot 2^{\deg(p)})$ matrices.

Although sufficient to show universality, this is *inefficient*. Even for simple polynomials such as x^n which can be computed using $n - 1$ operations, we require $\mathcal{O}(2^n)$ matrices. We can improve the efficiency by using the following lemma.

► **Remark 16.** For any degree- d monomial m , we have $\text{immc}_2(m) = d$. We can write $m = y_1 \cdots y_d$ where each y_i is a variable. Then, we set the $(1, 1)$ entry of the i^{th} matrix to y_i . All other entries are 0. The product now computes m at entry $(1, 1)$ and 0 elsewhere. Since this construction does not compute $Q(m)$, it is not possible to use this to compute, say $\prod_{i=1}^n x_i + \prod_{i=1}^n y_i + \prod_{i=1}^n z_i$ using $\text{poly}(n)$ operations.

► **Lemma 17.** *Let f and g be polynomials. Suppose that there is a sequence, say σ , of N matrices that approximately computes $Q(f)$, and a sequence, say π , of M matrices that approximately computes $Q(g)$. Then, there is a sequence of $N + 2M + 4$ matrices that approximately computes $Q(fg^2)$.*

Proof. Consider the following sequence, say σ' , of $N + 2M + 4$ matrices:

$$\begin{pmatrix} -\frac{1}{\epsilon} & 0 \\ 0 & \epsilon \end{pmatrix} \pi|_{\epsilon \rightarrow \epsilon^3} \begin{pmatrix} \epsilon & 0 \\ 0 & \frac{1}{\epsilon} \end{pmatrix} \sigma|_{\epsilon \rightarrow \epsilon^5} \begin{pmatrix} -\epsilon & 0 \\ 0 & \frac{1}{\epsilon} \end{pmatrix} \pi|_{\epsilon \rightarrow \epsilon^3} \begin{pmatrix} \frac{1}{\epsilon} & 0 \\ 0 & \epsilon \end{pmatrix}$$

where $\sigma|_{\epsilon \rightarrow \epsilon^5}$ denotes the sequence obtained from σ by replacing ϵ with ϵ^5 , and $\pi|_{\epsilon \rightarrow \epsilon^3}$ denotes the sequence obtained from π by replacing ϵ with ϵ^3 .

Note that σ' computes

$$\begin{aligned} & \begin{pmatrix} -\frac{1}{\epsilon} & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} g + \mathcal{O}(\epsilon^3) & 1 + \mathcal{O}(\epsilon^3) \\ 1 + \mathcal{O}(\epsilon^3) & \mathcal{O}(\epsilon^3) \end{pmatrix} \begin{pmatrix} \epsilon & 0 \\ 0 & \frac{1}{\epsilon} \end{pmatrix} \begin{pmatrix} f + \mathcal{O}(\epsilon^5) & 1 + \mathcal{O}(\epsilon^5) \\ 1 + \mathcal{O}(\epsilon^5) & \mathcal{O}(\epsilon^5) \end{pmatrix} \begin{pmatrix} -\epsilon & 0 \\ 0 & \frac{1}{\epsilon} \end{pmatrix} \\ & \begin{pmatrix} g + \mathcal{O}(\epsilon^3) & 1 + \mathcal{O}(\epsilon^3) \\ 1 + \mathcal{O}(\epsilon^3) & \mathcal{O}(\epsilon^3) \end{pmatrix} \begin{pmatrix} \frac{1}{\epsilon} & 0 \\ 0 & \epsilon \end{pmatrix} \\ & = \begin{pmatrix} -g + \mathcal{O}(\epsilon^3) & -\frac{1}{\epsilon^2} + \mathcal{O}(\epsilon) \\ \epsilon^2 + \mathcal{O}(\epsilon^5) & \mathcal{O}(\epsilon^3) \end{pmatrix} \begin{pmatrix} f + \mathcal{O}(\epsilon^5) & 1 + \mathcal{O}(\epsilon^5) \\ 1 + \mathcal{O}(\epsilon^5) & \mathcal{O}(\epsilon^5) \end{pmatrix} \begin{pmatrix} -g + \mathcal{O}(\epsilon^3) & -\epsilon^2 + \mathcal{O}(\epsilon^5) \\ \frac{1}{\epsilon^2} + \mathcal{O}(\epsilon) & \mathcal{O}(\epsilon^3) \end{pmatrix} \\ & = \begin{pmatrix} -fg - \frac{1}{\epsilon^2} + \mathcal{O}(\epsilon) & -g + \mathcal{O}(\epsilon^3) \\ \epsilon^2 f + \mathcal{O}(\epsilon^3) & \epsilon^2 + \mathcal{O}(\epsilon^5) \end{pmatrix} \begin{pmatrix} -g + \mathcal{O}(\epsilon^3) & -\epsilon^2 + \mathcal{O}(\epsilon^5) \\ \frac{1}{\epsilon^2} + \mathcal{O}(\epsilon) & \mathcal{O}(\epsilon^3) \end{pmatrix} \\ & = \begin{pmatrix} fg^2 + \mathcal{O}(\epsilon) & 1 + \epsilon^2 fg + \mathcal{O}(\epsilon^3) \\ 1 - \epsilon^2 fg + \mathcal{O}(\epsilon^3) & -\epsilon^4 f + \mathcal{O}(\epsilon^5) \end{pmatrix}. \end{aligned}$$

This proves Lemma 17. ◀

We also provide a Macaulay program in Appendix A to verify the construction described in the proof of Lemma 17. The key improvement here is that instead of using σ for $Q(f)$ two times as in Lemma 15, we can compute $Q(fx^2)$ using $Q(f)$ only once. Crucially, this allows certain monomials to be computed efficiently.

► **Lemma 18.** *Consider a monomial, say $m = c \cdot x_1^{k_1} \cdots x_n^{k_n}$. Let λ denote the number of odd k_i 's in k_1, \dots, k_n . Then, $Q(m)$ can be approximately computed using a sequence of $(5 \cdot 2^\lambda - 4) + 3 \cdot (\deg(m) - \lambda)$ matrices.*

Proof. Without loss of generality, assume that k_1, \dots, k_λ are the λ odd k_i 's. At a high level, we start with $Q(c)$, then repeatedly apply Lemma 15 to get $Q(c \cdot x_1 \cdots x_\lambda)$, then repeatedly apply Lemma 17 to get $Q(c \cdot x_1^{k_1} \cdots x_\lambda^{k_\lambda})$, and then repeatedly applying Lemma 17 to get $Q(c \cdot x_1^{k_1} \cdots x_\lambda^{k_\lambda} x_{\lambda+1}^{k_{\lambda+1}} \cdots x_n^{k_n})$. More precisely, our construction is as follows:

We begin with the sequence $Q(c)$. Using Lemma 15 (with indeterminate x_1), we get a sequence of $2 \cdot 1 + 4 = 6$ matrices that approximately computes $Q(c \cdot x_1)$. Next, using Lemma 15 (with indeterminate x_2), we get a sequence of $2 \cdot 6 + 4 = 16$ matrices that approximately computes $Q(c \cdot x_1 x_2)$. Again, using Lemma 15 (with indeterminate x_3), we get a sequence of $2 \cdot 16 + 4 = 36$ matrices that approximately computes $Q(c \cdot x_1 x_2 x_3)$. We continue this process until finally, using Lemma 15 (with indeterminate x_λ), we get a sequence of $2 \cdot (5 \cdot 2^{\lambda-1} - 4) + 4 = 5 \cdot 2^\lambda - 4$ matrices that approximately computes $Q(c \cdot x_1 x_2 x_3 \cdots x_\lambda)$.

Now, using Lemma 17 (with $g = x_1$) $\frac{k_1-1}{2}$ times, we get a sequence of $(5 \cdot 2^\lambda - 4) + (2 + 4) \cdot \frac{k_1-1}{2}$ matrices that approximately computes $Q(c \cdot x_1^{k_1} x_2 \cdots x_\lambda)$. Next, using Lemma 17 (with $g = x_2$) $\frac{k_2-1}{2}$ times, we get a sequence of $(5 \cdot 2^\lambda - 4) + (2 + 4) \cdot \frac{k_1-1}{2} + (2 + 4) \cdot \frac{k_2-1}{2}$ matrices that approximately computes $Q(c \cdot x_1^{k_1} x_2^{k_2} x_3 \cdots x_\lambda)$. We continue this process until

31:10 On the Power of Border Width-2 ABPs over Fields of Characteristic 2

finally, using Lemma 17 (with $g = x_\lambda$) $\frac{k_\lambda-1}{2}$ times, we get a sequence of $(5 \cdot 2^\lambda - 4) + (2+4) \cdot \binom{k_\lambda-1}{2} + (2+4) \cdot \binom{k_\lambda-1}{2} + \dots + (2+4) \cdot \binom{k_\lambda-1}{2} = (5 \cdot 2^\lambda - 4) + 3 \cdot \left(\sum_{i=1}^\lambda k_i - \lambda\right)$ matrices that approximately computes $Q(c \cdot x_1^{k_1} x_2^{k_2} x_3^{k_3} \dots x_\lambda^{k_\lambda})$.

Now, using Lemma 17 (with $g = x_{\lambda+1}$) $\frac{k_{\lambda+1}}{2}$ times, we get a sequence of $(5 \cdot 2^\lambda - 4) + 3 \cdot \left(\sum_{i=1}^\lambda k_i - \lambda\right) + (2+4) \cdot \binom{k_{\lambda+1}}{2}$ matrices that approximately computes $Q(c \cdot x_1^{k_1} \dots x_\lambda^{k_\lambda} x_{\lambda+1}^{k_{\lambda+1}})$. Next, using Lemma 17 (with $g = x_{\lambda+2}$) $\frac{k_{\lambda+2}}{2}$ times, we get a sequence of $(5 \cdot 2^\lambda - 4) + 3 \cdot \left(\sum_{i=1}^\lambda k_i - \lambda\right) + (2+4) \cdot \binom{k_{\lambda+1}}{2} + (2+4) \cdot \binom{k_{\lambda+2}}{2}$ matrices that approximately computes $Q(c \cdot x_1^{k_1} \dots x_\lambda^{k_\lambda} x_{\lambda+1}^{k_{\lambda+1}} x_{\lambda+2}^{k_{\lambda+2}})$. We continue this process until finally, using Lemma 17 (with $g = x_n$) $\frac{k_n}{2}$ times, we get a sequence of $(5 \cdot 2^\lambda - 4) + 3 \cdot \left(\sum_{i=1}^\lambda k_i - \lambda\right) + (2+4) \cdot \binom{k_{\lambda+1}}{2} + (2+4) \cdot \binom{k_{\lambda+2}}{2} + \dots + (2+4) \cdot \binom{k_n}{2} = (5 \cdot 2^\lambda - 4) + 3 \cdot \left(\sum_{i=1}^\lambda k_i - \lambda\right) + 3 \cdot \sum_{i=\lambda+1}^n k_i$ matrices that approximately computes $Q(c \cdot x_1^{k_1} \dots x_\lambda^{k_\lambda} x_{\lambda+1}^{k_{\lambda+1}} x_{\lambda+2}^{k_{\lambda+2}} \dots x_n^{k_n})$. That is, we get a sequence of $(5 \cdot 2^\lambda - 4) + 3 \cdot (\deg(m) - \lambda)$ matrices that approximately computes $Q(m)$.

This proves Lemma 18. ◀

Note that Lemma 18 allows us to compute x^n using $O(n)$ matrices.

► **Theorem 19.** *Let p be a polynomial with ℓ monomials, each containing at most t odd-power indeterminates. Then, $Q(p)$ can be approximately computed using a sequence of at most $\ell \cdot (5 \cdot 2^t + 3 \cdot \deg(p))$ matrices.*

Proof. Let m_1, \dots, m_ℓ denote the ℓ monomials of p . For each $1 \leq i \leq \ell$, we use Lemma 18 to get a sequence, say σ_i , of at most $(5 \cdot 2^t - 4) + 3 \cdot \deg(m_i)$ matrices that approximately computes $Q(m_i)$. Now, the following sequence approximately computes $Q(p)$:

$$\sigma_1 \cdot Q(0) \cdot \sigma_2 \cdot Q(0) \dots Q(0) \cdot \sigma_\ell$$

Note that the number of matrices in this sequence is at most

$$(\ell - 1) + \sum_{i=1}^{\ell} ((5 \cdot 2^t - 4) + 3 \cdot \deg(m_i)) \leq \ell \cdot (5 \cdot 2^t + 3 \cdot \deg(p))$$

This proves Theorem 19. ◀

5 Connections to Algebraic Formulas

In this section, we explore the relationship between the computational power of width-2 ABPs and algebraic formulas. Our main theorem in this section is:

► **Theorem 20.** *There exists a constant k such that for any polynomial f with a size- s formula approximating it, there is a $d \leq s^k + k$ such that $\text{imm}_{\mathbb{C}_2}(f^d) \leq s^k + k$.*

Proof. If the field has characteristic $\neq 2$, this can be done by using the methods in [4]. We consider fields of characteristic two. It is sufficient to consider $\text{IMM}_{3,n}$ for an arbitrary n as IMM_3 is a VF-complete family. We can consider without loss of generality that n is a power of two. These polynomials have polynomial-size algebraic formulas of depth $O(\log(n))$ where every path from root to leaf has the same number of product gates. We now construct a width-two algebraic branching program inductively from the formula as follows. For every polynomial p computed at a sub-formula with product depth d , we will compute $Q(p^{2^d})$. For input gates, this is trivial. Suppose f and g are sub-formulas that have product depth d .

For the formula $f + g$, notice that $(f + g)^{2^d} = f^{2^d} + g^{2^d}$ over fields of characteristic two. We can compute $Q(f^{2^d} + g^{2^d})$ from $Q(f^{2^d})$ and $Q(g^{2^d})$. For the formula $f \cdot g$, we compute $Q((f^{2^d})^2 (g^{2^d})^2) = Q((fg)^{2^{d+1}})$ using Lemma 17. Notice that since the product depth is the same on every root to leaf path, these cases are exhaustive. Since each step can at most double the size and depth is $O(\log(n))$, the size of the resulting width-two algebraic branching program is only $\text{poly}(n)$. ◀

The following remarks discuss two important consequences of this theorem. First, it allows us to extend the main result of [4] to more fields.

► **Remark 21.** Over characteristic 2, it is not clear whether one can compute f from f^d , for a polynomially-bounded d , which is a power of 2, using $\text{imm}_{\mathbb{C}_2}$. However, over large fields of characteristic $\neq 2$, one can follow the efficient *root-finding* procedure, for e.g., see [5, 6, 19], to conclude a small border width-2 complexity of f .

Second, it allows us to reduce border PIT for formulas to border PIT for width-2 ABPs.

► **Remark 22.** The border PIT problem (for definition and further connections with lower bounds, see [16, Section 2.6], [9], or [7, Section 7.1]) for a computational model is to check whether or not the polynomial computed by the given computation is approximately 0. Theorem 20 shows that border PIT for formulas reduces to border PIT for width-2 ABPs over all fields. For fields of characteristic $\neq 2$, this was already a consequence of the main result in [4]. Theorem 20 extends this to all fields. Notice that the proof of this theorem is constructive. That is, given a formula that approximately computes f , the proof of Theorem 20 can be easily modified to produce a polynomial-time algorithm to output a width-2 algebraic program approximating f^d . Now, over any field, f^d is approximately 0 if and only if f is approximately 0.

We say that a model supports efficient computation of square roots if any computation of f^2 in the model implies the existence of a computation for f where the size is polynomially related to the computation for f^2 . The following corollary establishes that if we can efficiently compute square roots approximately using width-two algebraic branching programs, then all polynomial families with constant-depth, polynomial-size circuits can be approximately computed using polynomial-size width-two algebraic branching programs.

► **Corollary 23.** *Suppose k is a universal constant such that given any width-two algebraic branching program of size s approximately computing a polynomial f^2 , we can approximately compute f using width-two algebraic branching programs of size at most $s^k + k$. Then, any polynomial family p that has constant depth algebraic circuits of size s can be approximately computed using width-two algebraic branching programs of size $\text{poly}(s)$.*

Proof. Since p has polynomial-size algebraic circuits of constant depth, it also has polynomial-size algebraic formulas of constant depth where all root to leaf paths have the same product depth. We then apply Theorem 20 to obtain a width-two algebraic branching program that computes f^{2^d} , where d is the product depth of the formula. Notice that the construction in Lemma 17 can obtain a width-two algebraic branching program that approximately computes $(f_1 \cdots f_k)^2$ in size $2 \sum_{i=1}^k s_i + O(k)$ from those of size s_i for f_i , where $1 \leq i \leq k$, even when k is unbounded. Finally, we apply the square root computation given by the hypothesis d times to obtain a width-two algebraic branching program that approximately computes f in size $O(s^{k^d})$. ◀

6 Improved bound for univariate polynomials

For univariate polynomials, a quadratic (in degree) upper bound on immc_2 over fields of characteristic 2 follows from Theorem 19. However, we can do better. In fact, we can make this asymptotically optimal by using a two-step Horner's method.

► **Theorem 24.** *Let p be a univariate polynomial in x . Then, $Q(p)$ can be approximately computed using a sequence of at most $\frac{9 \cdot \deg(p) + 4}{2}$ matrices.*

Proof. Let $d := \deg(p)$ if $\deg(p)$ is even, and $d := \deg(p) - 1$ otherwise.

If $\deg(p)$ is even, p is of the following form:

$$a_d x^d + a_{d-1} x^{d-1} + \dots + a_1 x + a_0.$$

Otherwise, p is of the following form:

$$a_{d+1} x^{d+1} + a_d x^d + a_{d-1} x^{d-1} + \dots + a_1 x + a_0.$$

Note that in both the cases, p can be expressed as follows:

$$\left(\dots \left((ax^2 + a_{d-1}x + a_{d-2})x^2 + a_{d-3}x + a_{d-4} \right) x^2 + \dots + a_3x + a_2 \right) x^2 + a_1x + a_0,$$

where $a := a_d$ if $\deg(p)$ is even, and $a := a_{d+1}x + a_d$ otherwise.

At a high level, our construction exploits the above expression by starting with $Q(a)$, then obtaining $Q(ax^2)$ using Lemma 17, then obtaining $Q(ax^2 + a_{d-1}x + a_{d-2})$ by appending a few matrices, then obtaining $Q\left((ax^2 + a_{d-1}x + a_{d-2})x^2\right)$ using Lemma 17, and so on, until we finally obtain $Q(p)$. More precisely, we construct the desired sequence as follows:

First, we compute $Q(a)$. When d is even, the matrix $Q(a_d)$ computes $Q(a)$. When d is odd, we could have taken $Q(a_{d+1}x)Q(0)Q(a_d)$ as a sequence of matrices computing $Q(a)$ if we were in the weak setting. However, since we are in the weakest setting, we instead use the length-2 sequence $\begin{pmatrix} a_{d+1} & a_d \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x & \frac{1}{a_{d+1}} \\ 1 & 0 \end{pmatrix}$ to compute $Q(a)$.

Next, using Lemma 17 (with $g = x$), we get a sequence of at most $2 + 2 + 4 = 8$ matrices that approximately computes $Q(ax^2)$. Again, if we were in the weak setting, we could have appended this sequence with $Q(0)Q(a_{d-1}x)Q(0)Q(a_{d-2})$ to get $Q(ax^2 + a_{d-1}x + a_{d-2})$. However, since we are in the weakest setting, we instead append this sequence with $Q(0) \begin{pmatrix} a_{d-1} & a_{d-2} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x & \frac{1}{a_{d-1}} \\ 1 & 0 \end{pmatrix}$ when $a_{d-1} \neq 0$, and $Q(0)Q(a_{d-2})$ when $a_{d-1} = 0$. This gives us a sequence of at most $8 + 3 = 11$ matrices that computes $Q(ax^2 + a_{d-1}x + a_{d-2})$.

Again, using Lemma 17 (with $g = x$), we get a sequence of at most $11 + 2 + 4 = 17$ matrices that approximately computes $Q\left((ax^2 + a_{d-1}x + a_{d-2})x^2\right)$. As before, we append it with $Q(0) \begin{pmatrix} a_{d-3} & a_{d-4} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x & \frac{1}{a_{d-3}} \\ 1 & 0 \end{pmatrix}$ when $a_{d-3} \neq 0$, and $Q(0)Q(a_{d-4})$ when $a_{d-3} = 0$. This gives us a sequence of at most $17 + 3 = 20$ matrices that approximately computes $Q\left((ax^2 + a_{d-1}x + a_{d-2})x^2 + a_{d-3}x + a_{d-4}\right)$.

We continue this process. Finally, we get a sequence of at most $\frac{9d + 4}{2} \leq \frac{9 \cdot \deg(p) + 4}{2}$ matrices that approximately computes $Q(p)$. This proves Theorem 24. ◀

7 Powering

Efficiently computing f^r from f , or powering, is an essential ingredient in many constructions, such as division elimination.

► **Lemma 25.** *Let p be a polynomial. Let $r \geq 1$ be an integer. Suppose that there is a sequence of M matrices that approximately computes $Q(p)$. Then, there is a sequence of at most $rM + 2r + 1$ matrices that approximately computes $Q(p^r)$.*

Proof. At a high level, we repeatedly use Lemma 17 to get $Q(p^2), Q(p^4), \dots, Q(p^r)$ when r is even, and $Q(p^3), Q(p^5), \dots, Q(p^r)$ when r is odd. More precisely, we construct the desired sequence as follows:

Case 1: r is even. Using Lemma 17 (with $f = 1$ and $g = p$), we get a sequence of $1 + 2M + 4 = 2M + 5$ matrices that approximately computes $Q(p^2)$. Next, using Lemma 17 (with $f = p^2$ and $g = p$), we get a sequence of $(2M + 5) + 2M + 4 = 4M + 9$ matrices that approximately computes $Q(p^4)$. Again, using Lemma 17 (with $f = p^4$ and $g = p$), we get a sequence of $(4M + 9) + 2M + 4 = 6M + 13$ matrices that approximately computes $Q(p^6)$. We continue this process until finally, using Lemma 17 (with $f = p^{r-2}$ and $g = p$), we get a sequence of $((r - 2)M + 2r - 3) + 2M + 4 = rM + 2r + 1$ matrices that approximately computes $Q(p^r)$.

Case 2: r is odd. Using Lemma 17 (with $f = p$ and $g = p$), we get a sequence of $M + 2M + 4 = 3M + 4$ matrices that approximately computes $Q(p^3)$. Next, using Lemma 17 (with $f = p^3$ and $g = p$), we get a sequence of $(3M + 4) + 2M + 4 = 5M + 8$ matrices that approximately computes $Q(p^5)$. Again, using Lemma 17 (with $f = p^5$ and $g = p$), we get a sequence of $(5M + 8) + 2M + 4 = 7M + 12$ matrices that approximately computes $Q(p^7)$. We continue this process until finally, using Lemma 17 (with $f = p^{r-2}$ and $g = p$), we get a sequence of $((r - 2)M + 2r - 6) + 2M + 4 = rM + 2r - 2$ matrices that approximately computes $Q(p^r)$. This proves Lemma 25. ◀

8 Division Elimination

We are now ready to prove a division elimination result. The usual division elimination computes f/g from f and g given that g divides f . Since we can compute $Q(fg^2)$ efficiently from $Q(f)$ and $Q(g)$. Efficient division elimination will imply that we can compute $Q(fg) = Q(fg^2/g)$ as well. In the following theorem, we prove a weaker version of division elimination, where we show how to compute f/g^2 from f and g given g^2 divides f . This is the only construction in this paper that relies on the additional power of weak projections over weakest projections.

► **Theorem 26.** *Let $f(\mathbf{x})$ and $g(\mathbf{x})$ be n -variate polynomials over a sufficiently large field of characteristic 2, where $\mathbf{x} = (x_1, \dots, x_n)$. Suppose that there are sequences, say σ and π , of N and M matrices that approximately compute $Q(f)$ and $Q(g)$ wrt weak projections respectively. Assume that g^2 divides f . Then, there is a sequence, say η , of $\mathcal{O}(N^4M(M + N))$ matrices that approximately computes $Q(\frac{f}{g^2})$ wrt weak projections.*

Proof. Define $h(\mathbf{x}) := \frac{f(\mathbf{x})}{g(\mathbf{x})^2}$. Let k be the degree of $h(\mathbf{x})$. If $g(\mathbf{0}) \neq 1$, then we find α such that $g(\mathbf{x} + \alpha) = 1 + g_1(\mathbf{x})$.

31:14 On the Power of Border Width-2 ABPs over Fields of Characteristic 2

Using the sequence π , we can get a new sequence of $\mathcal{O}(M)$ matrices that approximately computes $g_1(\mathbf{x})$. We have

$$h(\mathbf{x} + \alpha) = \frac{f(\mathbf{x} + \alpha)}{(g(\mathbf{x} + \alpha))^2} = \frac{f(\mathbf{x} + \alpha)}{(1 + (-1 + g(\mathbf{x} + \alpha)))^2} = \frac{f(\mathbf{x} + \alpha)}{(1 + g_1(x))^2} = \frac{f(\mathbf{x} + \alpha)}{1 + g_1^2(\mathbf{x})} = \sum_{i \geq 0} f \cdot (g_1^2)^i$$

For each $0 \leq i \leq k/2$, we get a sequence, say η_i , of $\mathcal{O}(k(M+N))$ matrices, that approximately computes $Q(f \cdot g_1^{2i})$ using Lemma 17.

Define $\mathcal{P}(\mathbf{x}) := \sum_{i=0}^{k/2} f \cdot (g_1^2)^i$. The following sequence, say λ , of $\mathcal{O}(k^2(M+N))$ matrices, computes $Q(\mathcal{P})$ approximately:

$$\eta_0 \cdot Q(0) \cdot \eta_1 \cdot Q(0) \cdots Q(0) \cdot \eta_{k/2}.$$

Let $\mathcal{R}(t) := \mathcal{P}(tx_1, \dots, tx_n)$. Note that $\mathcal{R}(t)$ is of the form, $\mathcal{R}(t) = b_0 + b_1 t + b_2 t^2 + \dots + b_\ell t^\ell$, where b_0, b_1, \dots, b_ℓ are polynomials in x_1, \dots, x_n over \mathbb{F} . Let $a_0, \dots, a_\ell \in \mathbb{F}$. Note that

$$A \cdot \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_\ell \end{bmatrix} = \begin{bmatrix} R(a_0) \\ R(a_1) \\ \vdots \\ R(a_\ell) \end{bmatrix}, \text{ where } A := \begin{bmatrix} 1 & a_0 & a_0^2 & \dots & a_0^\ell \\ 1 & a_1 & a_1^2 & \dots & a_1^\ell \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & a_\ell & a_\ell^2 & \dots & a_\ell^\ell \end{bmatrix}$$

For every $0 \leq i, j \leq \ell$, let $c_{i,j}$ denote the entry at the i^{th} row and the j^{th} column of A^{-1} . Then, we have

$$b_0 = c_{0,0} \cdot R(a_0) + c_{0,1} \cdot R(a_1) + \dots + c_{0,\ell} \cdot R(a_\ell)$$

$$b_1 = c_{1,0} \cdot R(a_0) + c_{1,1} \cdot R(a_1) + \dots + c_{1,\ell} \cdot R(a_\ell)$$

\vdots

$$b_\ell = c_{\ell,0} \cdot R(a_0) + c_{\ell,1} \cdot R(a_1) + \dots + c_{\ell,\ell} \cdot R(a_\ell)$$

For every $0 \leq i \leq \ell$, we obtain a sequence, say λ_i , from λ , by replacing x_r with $a_i \cdot x_r$ for every $1 \leq r \leq n$. Note that λ_i approximately computes $Q(R(a_i))$ using $\mathcal{O}(k^2(M+N))$ matrices.

Now, for every $0 \leq i \leq k$, the following sequence, say Γ_i , approximately computes $Q(b_i)$ using $\mathcal{O}(k^2 \ell(M+N))$ matrices:

$$\begin{bmatrix} c_{i,0} & 0 \\ 0 & 1 \end{bmatrix} \lambda_0 \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{c_{i,0}} \end{bmatrix} Q(0) \begin{bmatrix} c_{i,1} & 0 \\ 0 & 1 \end{bmatrix} \lambda_1 \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{c_{i,1}} \end{bmatrix} Q(0) \cdots Q(0) \begin{bmatrix} c_{i,\ell} & 0 \\ 0 & 1 \end{bmatrix} \lambda_\ell \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{c_{i,\ell}} \end{bmatrix}.$$

Also, we have

$$\begin{aligned} h(\mathbf{x} + \alpha) &= \text{hom}_0(\mathcal{P}(\mathbf{x})) + \text{hom}_1(\mathcal{P}(\mathbf{x})) + \dots + \text{hom}_k(\mathcal{P}(\mathbf{x})) \\ &= b_0 + b_1 + \dots + b_k \end{aligned}$$

Therefore, the following sequence of $\mathcal{O}(k^3 \ell(M+N))$ matrices approximately computes $Q(h(\mathbf{x} + \alpha))$:

$$\Gamma_0 \cdot Q(0) \cdot \Gamma_1 \cdot Q(0) \cdots Q(0) \cdot \Gamma_k$$

Finally, we replace \mathbf{x} by $\mathbf{x} + \alpha$ in the above sequence to get a sequence, say η , that approximately computes $Q(h(\mathbf{x}))$. Note that $k \leq \deg(f) \leq N$ and $\ell \leq \deg(f) + k \cdot \deg(g) \leq \mathcal{O}(MN)$. Thus, η has $\mathcal{O}(N^4 M(M+N))$ matrices. \blacktriangleleft

9 Conclusion

This work successfully establishes that width-2 ABPs can approximate any polynomial *regardless* of the characteristic of the field, thus resolving a weaker version of the open question from [4]. Here are some immediate questions which require rigorous investigation.

1. Let $f \in \mathbb{F}[\mathbf{x}]$, of degree d , where $\text{char}(\mathbb{F}) = 2$. Further, let $\text{immc}(f^2) = s$. Can we say that $\text{immc}_2(f) = \text{poly}(s, d)$?
2. Can we prove a subexponential upper bound on $\text{immc}_2(f)$, for any exponential-sparse polynomial f , of border formula-complexity $\text{poly}(n)$, over fields of characteristics 2? Of course, proving a polynomial upper bound would settle the open question of [4], proving that $\sqrt{V} = \sqrt{VBP_2}$, over fields of characteristics 2 (and hence, over any field!).

References

- 1 Eric Allender and Fengming Wang. On the power of algebraic branching programs of width two. *computational complexity*, 25(1):217–253, 2016.
- 2 Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992.
- 3 Stuart J Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information processing letters*, 18(3):147–150, 1984.
- 4 Karl Bringmann, Christian Ikenmeyer, and Jeroen Zuiddam. On algebraic branching programs of small width. *Journal of the ACM (JACM)*, 65(5):1–29, 2018.
- 5 Peter Bürgisser. The complexity of factors of multivariate polynomials. *Found. Comput. Math.*, 4(4):369–396, 2004. doi:10.1007/s10208-002-0059-5.
- 6 Pranjal Dutta. Discovering the roots: Unifying and extending results on multivariate polynomial factoring in algebraic complexity. *Master’s thesis*, 2018.
- 7 Pranjal Dutta. A tale of hardness, de-randomization and de-bordering in complexity theory. *PhD Thesis*, 2022.
- 8 Pranjal Dutta, Fulvio Gesmundo, Christian Ikenmeyer, Gorav Jindal, and Vladimir Lysikov. Border complexity via elementary symmetric polynomials. *arXiv preprint arXiv:2211.07055*, 2022.
- 9 Michael A Forbes and Amir Shpilka. A pspace construction of a hitting set for the closure of small algebraic circuits. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1180–1192, 2018.
- 10 Bruno Grenet. An upper bound for the permanent versus determinant problem. *Theory of Computing*, 2011.
- 11 Christian Ikenmeyer and Abhiroop Sanyal. A note on VNP-completeness and border complexity. *Information Processing Letters*, 176:106243, 2022.
- 12 Mrinal Kumar. On the power of border of depth-3 arithmetic circuits. *ACM Trans. Comput. Theory*, 12(1):5:1–5:8, 2020. doi:10.1145/3371506.
- 13 M. Mahajan. Algebraic complexity classes. *Perspectives in Comp. Compl.: The Somenath Biswas Ann. Vol.*, pages 51–75, 2014.
- 14 Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, 1997(5), 1997.
- 15 Thierry Mignon and Nicolas Ressayre. A quadratic bound for the determinant and permanent problem. *International Mathematics Research Notices*, 2004(79):4241–4253, 2004.
- 16 Ketan Mulmuley. Geometric complexity theory v: Efficient algorithms for noether normalization. *Journal of the American Mathematical Society*, 30(1):225–309, 2017.
- 17 Ketan Mulmuley and Milind A. Sohoni. Geometric complexity theory I: an approach to the P vs. NP and related problems. *SIAM J. Comput.*, 31(2):496–526, 2001. doi:10.1137/S009753970038715X.

31:16 On the Power of Border Width-2 ABPs over Fields of Characteristic 2

- 18 Ketan D Mulmuley and Milind Sohoni. Geometric complexity theory II: towards explicit obstructions for embeddings among class varieties. *SIAM Journal on Computing*, 38(3):1175–1206, 2008.
- 19 Amit Sinhababu and Thomas Thierauf. Factorization of polynomials given by arithmetic branching programs. *computational complexity*, 30:1–47, 2021.
- 20 Volker Strassen. Vermeidung von Divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973. URL: <http://eudml.org/doc/151394>.
- 21 Seinosuke Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Transactions on Information and Systems*, 75(1):116–124, 1992.
- 22 Leslie G Valiant. Completeness classes in algebra. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 249–261, 1979.

A Macaulay2 source code for main constructions

Listing 1 illustrates our construction of $Q(fx)$ from $Q(f)$. The code can be run using Macaulay2. The variables 01 through 08 in these programs represent (arbitrary) polynomials in the ring $\mathbb{Z}\mathbb{Z}/2[\text{eps}, x_1, \dots, x_n]$ that appear as a result of the approximation.

■ Listing 1 $Q(fx)$ from $Q(f)$.

```
R=ZZ/2[eps];
S=frac R;
S[f,x,01,02,03,04];
M1=matrix{{1/eps,0},{0,1}};
M2=matrix{{f+eps^2*01,1+eps^2*02},{1+eps^2*03,eps^2*04}};
M3=matrix{{eps,1},{0,1}};
M4=matrix{{1/eps,x},{-1,1}};
M5=matrix{{1,0},{1,-eps}};
print(M1*M2*M3*M4*M2*M5);
```

Listing 2 illustrates our construction of $Q(fg^2)$ from $Q(f)$ and $Q(g)$.

■ Listing 2 $Q(fg^2)$ from $Q(f)$ and $Q(g)$.

```
R=ZZ/2[eps];
S=frac R;
S[f,g,01,02,03,04,05,06,07,08];
M1=matrix{{-1/eps,0},{0,eps}};
M2=matrix{{g+eps^3*05,1+eps^3*06},{1+eps^3*07,eps^3*08}};
M3=matrix{{eps,0},{0,1/eps}};
M4=matrix{{f+eps^5*01,1+eps^5*02},{1+eps^5*03,eps^5*04}};
M5=matrix{{-eps,0},{0,1/eps}};
M6=matrix{{1/eps,0},{0,eps}};
print(M1*M2*M3*M4*M5*M2*M6);
```

$O(1/\varepsilon)$ Is the Answer in Online Weighted Throughput Maximization

Franziska Eberle  

Technische Universität Berlin, Germany

Abstract

We study a fundamental online scheduling problem where jobs with processing times, weights, and deadlines arrive online over time at their release dates. The task is to preemptively schedule these jobs on a single or multiple (possibly unrelated) machines with the objective to maximize the weighted throughput, the total weight of jobs that complete before their deadline. To overcome known lower bounds for the competitive analysis, we assume that each job arrives with some slack $\varepsilon > 0$; that is, the time window for processing job j on any machine i on which it can be executed has length at least $(1 + \varepsilon)$ times j 's processing time on machine i .

Our contribution is a best possible online algorithm for weighted throughput maximization on unrelated machines: Our algorithm is $\mathcal{O}(\frac{1}{\varepsilon})$ -competitive, which matches the lower bound for unweighted throughput maximization on a single machine. Even for a single machine, it was not known whether the problem with weighted jobs is “harder” than the problem with unweighted jobs. Thus, we answer this question and close weighted throughput maximization on a single machine with a best possible competitive ratio $\Theta(\frac{1}{\varepsilon})$.

While we focus on non-migratory schedules, on identical machines, our algorithm achieves the same (up to constants) performance guarantee when compared to an optimal migratory schedule.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases Deadline scheduling, weighted throughput, online algorithms, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.32

Related Version *Full Version:* <https://doi.org/10.48550/arXiv.2310.16697>

Funding Supported by the Dutch Research Council (NWO), Netherlands Vidi grant 016.Vidi.189.087.

1 Introduction

We consider an online scheduling problem with m parallel *unrelated* machines. Online over time, job j arrives at its *release date* r_j . Upon arrival of job j , its *processing time*, sometimes also referred to as *size*, $p_{ij} \in \mathbb{R}_{>0} \cup \{\infty\}$ on machine $i \in [m] := \{1, \dots, m\}$, its *weight* w_j , and its *deadline* d_j are revealed to the online algorithm. The *density* of j on machine i is given by $\rho_{ij} := \frac{w_j}{p_{ij}}$. A machine i is *eligible* for job j if $p_{ij} < \infty$. If $p_{ij} = p_j$ holds for all i and all j , we call the machines *identical* and omit the index.

The processing of a job is allowed to be interrupted, we say *preempted*, and resumed at any later point in time, also on a different machine; that is, we allow *migration*. A job j completes if $\sum_{i=1}^m \frac{q_{ij}}{p_{ij}} = 1$, where j receives a total of q_{ij} time units on machine i . In a feasible schedule, at any point in time, every machine can process at most one job, and every job can be processed by at most one machine. The objective is to find a feasible schedule that maximizes the *weighted throughput*, the total weight of jobs that meet their deadlines.

This model captures a resource allocation problem, e.g., encountered in public cloud computing environments or large internal clusters, where the available hardware needs to be allocated to (often) time-sensitive and mission-critical jobs [18]. Focusing on the objective



© Franziska Eberle;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 32; pp. 32:1–32:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



of maximizing weighted throughput allows us to factor in deadlines of time-sensitive jobs, account for the variety in the importance of jobs, and overall maximize total utility [8]. As explained in [6], allowing preemption in multiprocessor computer systems is cost-wise quite reasonable, while migration, which our algorithm will not use, can be quite expensive due to additional communication requirements.

Due to the lack of information because of the online nature of the problem, there are instances where an online algorithm cannot find the schedule with maximum throughput as shown in Section 3 of [3] for single-machine instances. Thus, we employ standard competitive analysis to measure the performance of an online algorithm, where we compare the weighted throughput of an online algorithm to the weighted throughput of an optimal offline algorithm, that has complete knowledge about the instance in advance and uses this information to make scheduling decisions.

It has been known for 30 years that in this general setting no deterministic algorithm can achieve a bounded competitive ratio on identical machines [4, 16]. In fact, even when allowing randomization on a single machine, the competitive ratio for weighted throughput maximization remains unbounded [7]. All of the aforementioned lower bounds heavily rely on “tight” jobs, that is, on jobs that need to be processed immediately and continuously upon release in order to finish before their deadlines.

To overcome these lower bounds, we make a standard *slackness* assumption that the window for processing job j has some slack: we assume that $d_j - r_j \geq (1 + \varepsilon)p_{ij}$ holds if machine i is eligible for j , i.e., if $p_{ij} < \infty$. We expect our algorithm to perform better with larger slack parameter $\varepsilon > 0$. This trade-off between slack and competitive ratio has successfully been studied before [1, 2, 8, 10–12, 18, 21]. The slackness assumption does not pose a real obstacle for applications since deadlines are typically not tight [13] and slack can even be introduced by lowering the operating level or increasing the speed [9, 20].

Recently, unweighted throughput maximization has been solved on a single machine with a competitive ratio of $\mathcal{O}(\frac{1}{\varepsilon})$ [8], on identical machines with a $\mathcal{O}(1)$ -competitive algorithm [19], and on unrelated machines with a competitive ratio of $\mathcal{O}(\frac{1}{\varepsilon})$ [10]. For weighted throughput maximization it is known that $\mathcal{O}(1)$ -competitive algorithms are not possible, independent of the machine environment, even when allowing randomization [16]. Even on a single machine, there remained a gap between the performance bound $\mathcal{O}(\frac{1}{\varepsilon^2})$ of the algorithm by [18] and the lower bound $\Omega(\frac{1}{\varepsilon})$ carried over from the unweighted setting [8].

In this work, we close this gap and essentially the line of research concerned with online weighted throughput maximization started in this form by [18]. We give an (up to constant factors) best possible online algorithm for weighted throughput maximization on unrelated machines with competitive ratio $\mathcal{O}(\frac{1}{\varepsilon})$. On identical machines, our non-migratory algorithm remains $\mathcal{O}(\frac{1}{\varepsilon})$ -competitive against the optimal schedule that is allowed to use migration. In particular, we solve the problem on a single machine, matching the known lower bound for unweighted throughput maximization [8].

Related work

Online throughput maximization gained a lot of interest during the last years [8, 10, 13, 19], but research has been active for decades [2–5].

For tight jobs with $d_j - r_j = p_j$, there are non-constant lower bounds on the competitive ratios of deterministic and randomized algorithms [4, 7], which justify our slackness assumption. Baruah et al. [4] give a lower bound of $(1 + \sqrt{\mu})^2$ on the competitive ratio of any deterministic single-machine algorithm, where $\mu = \frac{\max_j p_j}{\min_j p_j}$, while Koren and Shasha [17] give an algorithm with matching competitive ratio. On identical machines, their algorithm achieves the

best possible competitive ratio of $\Theta(\ln \mu)$ [16]. Canetti and Irani [7] consider randomized algorithms and show a lower bound of $\Theta\left(\frac{\log \nu}{\log \log \nu}\right)$, where $\nu = \min\left\{\frac{\max p_j}{\min p_j}, \frac{\max w_j}{\min w_j}\right\}$. They also give an almost matching upper bound.

Since both parameters, μ and ν , can become arbitrarily large, research started to investigate instances satisfying a slackness assumption [18]. Most relevant to our work is the $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ -competitive algorithm by Lucier et al. [18] for weighted throughput maximization on identical machines under the slackness assumption. Azar et al. [1] study the same problem and give a truthful mechanism with a competitive ratio of $2 + \Theta\left(\frac{1}{\sqrt[3]{1+\varepsilon}-1} + \frac{1}{(\sqrt[3]{1+\varepsilon}-1)^3}\right)$.

The special case of maximizing machine utilization, where the weight of each job equals its processing time, allows for $\mathcal{O}(1)$ -competitive algorithms, even in settings without slack. On a single machine, the algorithm by Baruah et al. [4, 5] is 4-competitive, and on identical machines, Koren and Shasha [16] claim an $\mathcal{O}(1)$ -competitive algorithm.

In the *unweighted* setting, Baruah et al. [3] show that non-trivial competitive ratios are impossible in the presence of tight jobs. However, randomization allows for a competitive ratio of $\mathcal{O}(1)$ [15]. If every job arrives with a slack of ε , the (deterministic) algorithm by Chen et al. [8] achieves the provably best competitive ratio of $\Theta\left(\frac{1}{\varepsilon}\right)$ on a single machine. On at least two machines, the algorithm by Moseley et al. [19] is $\mathcal{O}(1)$ -competitive, even for instances without slack. Eberle et al. [10] design a $\Theta\left(\frac{1}{\varepsilon}\right)$ -competitive algorithm for throughput maximization on unrelated machines when each job has ε -slack.

Our results

As our main result, we present an $\mathcal{O}\left(\frac{1}{\varepsilon}\right)$ -competitive algorithm for online weighted throughput maximization.

► **Theorem 1.** *For weighted throughput maximization on unrelated machines without migration, there is an $\mathcal{O}\left(\frac{1}{\varepsilon}\right)$ -competitive online algorithm.*

This generalizes and improves the $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ -competitive algorithm by [18]. It matches the known lower bound of $\Omega\left(\frac{1}{\varepsilon}\right)$ on a single machine [8] and, thus, closes the gap that remained.

During the analysis, we focus on comparing the non-migratory schedule obtained by our algorithm to an optimal, non-migratory schedule. On identical machines, it is known that the throughput achievable without migration is within a constant multiplicative factor of the throughput achievable using migration by Kalyanasundaram and Pruhs [14]. Thus, our result also holds in the migratory setting.

► **Theorem 2.** *For weighted throughput maximization on identical machines with migration, there is an $\mathcal{O}\left(\frac{1}{\varepsilon}\right)$ -competitive online algorithm.*

One threshold cannot beat $\Theta(1/\varepsilon^2)$

Previous results for throughput maximization use a threshold-based policy to decide about the admission of newly released and the preemption of currently running jobs [1, 8, 10, 18]. Crucially, these algorithms rely on a *single* density threshold $\gamma \in \Theta(\varepsilon)$ to determine if a currently running job is preempted in favor of a newly released job with higher density.

The following two examples give an intuition why a single-threshold algorithm cannot break the $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ -barrier. Let $\varepsilon < 1$ and suppose that $\gamma \in (0, 1]$ is the threshold which an algorithm uses to discard currently running jobs in favor of newly released jobs with density higher by a factor at least $\frac{1}{\gamma}$. In both examples, $\delta \ll 1$ is a small constant, for which we will eventually consider the limit $\delta \rightarrow 0$, and the jobs are *tight*, i.e., they satisfy $r_j + (1 + \varepsilon)p_j = d_j$.

► **Example 3.** There is a single machine and $n + 1$ tight jobs with the parameters $r_0 = 0$, $p_0 = w_0 = \rho_0 = 1$ and $r_j = r_{j-1} + (1 - \delta)p_{j-1}$, $p_j = (\varepsilon + \delta)p_{j-1}$ and $\rho_j = \frac{1+\delta}{\gamma}\rho_j$ for $j \in [n] := \{1, \dots, n\}$.

By our choice of parameters, job j interrupts the execution of job $j - 1$ immediately upon arrival. It is easy to calculate that $d_j \geq d_{j-1}$ and $C_j = d_{j-1}$ for $j \in [n]$ if the processing of job j is not interrupted. Combining these two observations implies that the algorithm can only complete the last job on time if the constant δ is chosen sufficiently small. Hence, the algorithm obtains a total weight of $\rho_n p_n = \left(\frac{(1+\delta)(\varepsilon+\delta)}{\gamma}\right)^n \rightarrow \left(\frac{\varepsilon}{\gamma}\right)^n$ as $\delta \rightarrow 0$. Conversely, by scheduling only job 1, one can obtain a total weight of 1, implying that the competitive ratio is at least $\left(\frac{\gamma}{\varepsilon}\right)^n$. Hence, $\gamma \leq \varepsilon$ is necessary to achieve a bounded competitive ratio.

► **Example 4.** We have a single-machine instance with two tight jobs 1 and 2. The parameters are $p_1 = w_1 = 2$, $r_1 = 0$ and $p_2 = \frac{1}{\varepsilon}$, $w_2 = \frac{1}{\varepsilon\gamma - \delta}$, $r_2 = \delta$. Since both jobs are tight, no feasible schedule can complete both jobs on time. Hence, it is optimal to schedule only the second job upon release and obtain a total weight of $\frac{1}{\varepsilon\gamma - \delta}$. However, the parameters are chosen such that an algorithm with threshold γ admits the first job upon release and cannot discard it in favor of the second job, implying a competitive ratio of $\Omega\left(\frac{1}{\varepsilon\gamma - \delta}\right)$, which goes to $\Omega\left(\frac{1}{\gamma\varepsilon}\right)$ as $\delta \rightarrow 0$.

What becomes apparent in the examples is that by relying on a single threshold to guide the admission decisions, an algorithm is both too careless (Example 3) and too conservative (Example 4) in admitting jobs. In fact, such an algorithm does not distinguish the reasons for a job having a relatively high density: it might be caused by a large weight or by a small processing time.

We show that, by using *two* distinct thresholds, a simple greedy algorithm achieves a competitive ratio of $\Theta\left(\frac{1}{\varepsilon}\right)$, which is optimal up to constants even in the unweighted setting [8]. Our algorithm compares the sizes of a newly released job j^* and a currently running job j , in order to decide whether to abandon the latter in favor of the former. In Example 3, we have already established that if j is preempted in favor of j^* with $p_{ij^*} \in \mathcal{O}(\varepsilon)p_{ij}$, then the density of j^* should be greater by a factor $\Omega\left(\frac{1}{\varepsilon}\right)$. Conversely, to avoid the issue present in Example 4, if the new job j^* is larger than the currently running job j , then j should be interrupted in favor of processing j^* if it has a similar density as j . For technical reasons, our algorithm employs a third admission rule that smoothly interpolates between the threshold $\Theta(\varepsilon)$ for jobs smaller by a factor $\mathcal{O}(\varepsilon)$ and a threshold $\Theta(1)$ for larger jobs.

In the following section, we formally describe our algorithm before analyzing its competitive ratio in the two subsequent sections.

2 The two-threshold algorithm

In this section, we design the two-threshold algorithm. We assume without loss of generality that $\varepsilon \leq 1$ as otherwise we can simply run the algorithm with $\varepsilon = 1$ and obtain a constant competitive ratio.

The two-threshold algorithm starts job j on machine i for the first time only before $d_j - \left(1 + \frac{\varepsilon}{2}\right)p_{ij}$. If machine i starts processing job j at time a_j , we say j is *admitted* to machine i at time a_j . For each machine i , the algorithm maintains the set of jobs that are active at time τ . A job j is *active* at time τ on machine i if it was admitted to i before time τ , is not yet completed and can still complete before time $a_j + \left(1 + \frac{\varepsilon}{2}\right)p_{ij}$, i.e., the remaining processing time of j on i is at most $a_j + \left(1 + \frac{\varepsilon}{2}\right)p_{ij} - \tau$.

On a high-level, our algorithm uses two independent subroutines: the scheduling routine and the admission routine. The admission routine merely assigns jobs to machines. Among the jobs assigned to a machine, the scheduling routine chooses which job to actually process.

Scheduling routine. At time τ and on each machine i , the algorithm simply processes the job j which is active for i at τ and has the highest density $\rho_{ij} = \frac{w_j}{p_{ij}}$ among all such jobs.

Admission routine. There are two events that trigger the admission routine at time τ : the release of a new job and the completion of a job. The admission routine loops over the machines and decides whether the currently running job j should be preempted for a job with higher density.

To this end, it considers the jobs that have been released, have not yet been admitted, and whose deadline is sufficiently far in the future, i.e., $d_{j^*} - \tau \geq (1 + \frac{\varepsilon}{2}) p_{ij^*}$, in decreasing order of machine-dependent density ρ_{ij^*} . Let j^* be the job with highest density that has not been considered for admission to machine i before. The algorithm compares j^* 's processing time with that of the job j that is currently processed by machine i .

If no such job exists, then j^* is admitted to machine i and starts executing immediately. If such a job j exists and its processing time is at least $\frac{2}{\varepsilon} p_{ij^*}$, then the first density-threshold $\frac{8}{\varepsilon}$ is invoked: if $\rho_{ij^*} \geq \frac{8}{\varepsilon} \rho_{ij}$, then j^* is admitted to i at time $a_{j^*} = \tau$. If j exists and $\frac{\varepsilon}{2} p_{ij} < p_{ij^*} \leq p_{ij}$, then we use a smooth transition between the two thresholds: if $w_{j^*} \geq 4w_j$, then j^* is admitted to i at time $a_{j^*} = \tau$. Otherwise, that is, j is currently running on machine i and its processing time is smaller than p_{ij^*} , then the second density-threshold 4 is invoked: if $\rho_{ij^*} \geq 4\rho_{ij}$, then j^* is admitted to i at time $a_{j^*} = \tau$.

If job j^* interrupts the execution of job j , we say that j is the *parent* $\pi(j^*)$ of j^* and j^* is a *child* of j . Note that by construction, a newly admitted job has highest density on its machine and starts processing immediately. We summarize our algorithm in Algorithm 1.

The following observation formalizes the “smooth transition” between the two density thresholds.

► **Observation 5.** Consider jobs j and k with $\frac{\varepsilon}{2} p_{ij} < p_{ik} \leq p_{ij}$ for some machine i . If $w_k \geq 4w_j$, then $\rho_{ik} = \frac{w_k}{p_{ik}} \geq \frac{4w_j}{p_{ij}} = 4\rho_{ij}$. If $w_k < 4w_j$, then $\rho_{ik} < \frac{4w_j}{\varepsilon/2 p_{ij}} = \frac{8}{\varepsilon} \rho_{ij}$. Further, if $p_{ik} > p_{ij}$ and $\rho_{ik} \geq 4\rho_{ij}$, then $w_k = \rho_{ik} p_{ik} \geq 4w_j$.

Roadmap of the analysis

The analysis of the two-threshold algorithm naturally splits into two parts. In Section 3, we show that the highest-density rule used for scheduling active jobs guarantees that the total weight of jobs completed before their deadlines is at least half of the total weight of jobs admitted by the admission routine. In Section 4, we compare the total weight of the jobs admitted by the two-threshold algorithm to the weighted throughput of an optimal solution before proving Theorem 1.

3 Weight of finished jobs vs. weight of admitted jobs

In this section, we show that the two-threshold algorithm obtains at least half of the total weight of the jobs that were admitted. We prove the following theorem where J denotes the set of jobs admitted by our algorithm and $F \subseteq J$ the set of jobs completed before their respective deadlines.

■ **Algorithm 1** Two-threshold algorithm.

Initialize: If $\varepsilon > 1$, then reset $\varepsilon \leftarrow 1$.

Scheduling Routine: At all times τ and on all machines i , run the job with highest density that is active for i .

Event: Release of a new job at time τ
Call Admission Routine

Event: Completion of a job at time τ
Call Admission Routine

Admission Routine:

```

for  $i = 1$  to  $m$  do
   $J^* \leftarrow \{j \mid r_j \leq \tau, d_j - \tau \geq (1 + \frac{\varepsilon}{2})p_{ij}, j \text{ not yet admitted}\}$  // eligible jobs
   $K \leftarrow \{k : k \text{ active on machine } i \text{ at time } \tau\}$  // jobs currently active for  $i$ 
  for  $j^* \in J^*$  in order of decreasing  $\rho_{ij^*}$  do // select highest-density job
    if  $K = \emptyset$  then
      admit  $j^*$  to  $i$  and  $a_{j^*} \leftarrow \tau$ 
       $\pi(j^*) \leftarrow \emptyset$  //  $j^*$  does not have a parent
      break for-loop
    else
       $j \leftarrow \arg \max\{\rho_{ik} \mid k \in K\}$  // currently running job
      if  $p_{ij^*} \leq \frac{\varepsilon}{2}p_{ij}$  and  $\rho_{ij^*} \geq \frac{8}{\varepsilon}\rho_{ij}$  then
        admit  $j^*$  to  $i$  and  $a_{j^*} \leftarrow \tau$ 
         $\pi(j^*) \leftarrow j$  // parent of  $j^*$ 
        break for-loop
      else if  $\frac{\varepsilon}{2}p_{ij} < p_{ij^*} \leq p_{ij}$  and  $w_{j^*} \geq 4w_j$  then
        admit  $j^*$  to  $i$  and  $a_{j^*} \leftarrow \tau$ 
         $\pi(j^*) \leftarrow j$  // parent of  $j^*$ 
        break for-loop
      else if  $p_{ij^*} > p_{ij}$  and  $\rho_{ij^*} \geq 4\rho_{ij}$  then
        admit  $j^*$  to  $i$  and  $a_{j^*} \leftarrow \tau$ 
         $\pi(j^*) \leftarrow j$  // parent of  $j^*$ 
        break for-loop

```

► **Theorem 6.** Let J and F be the set of jobs admitted and finished, respectively, by the two-threshold algorithm. Then,

$$\sum_{j \in F} w_j \geq \frac{1}{2} \sum_{j \in J} w_j.$$

For intuition, consider an instance that only consists of a job j and the set K of j 's children. Suppose that j does not finish on time as otherwise the theorem trivially holds. Recall that j was admitted at $a_j \leq d_j - (1 + \frac{\varepsilon}{2})p_{ij}$ to machine i . (Jobs that are not interrupted complete before $a_j + (1 + \frac{\varepsilon}{2})p_{ij} \leq d_j$.) This implies that the total processing time of jobs interrupting j is at least $\frac{\varepsilon}{2}p_{ij}$. If there is at least one job k with $p_{ik} > \frac{\varepsilon}{2}p_{ij}$, Observation 5 and the admission rule for jobs with $\frac{\varepsilon}{2}p_{ij} < p_{ik} \leq p_{ij}$ imply that $w_k \geq 4w_j$ showing the statement. If all jobs k have processing time at most $\frac{\varepsilon}{2}p_{ij}$, their densities are bounded from below by $\frac{8}{\varepsilon}\rho_{ij}$, and their total weight is again at least $4w_j$.

In the formal proof of Theorem 6, we assume the existence of an instance that violates the statement and restrict to one that is minimal with respect to the total number of jobs. The above intuition tells us that sub-instances consisting of a job and its children cannot cause the violation. In fact, we show that we can carefully merge such sub-instances into one job without changing the fact that the complete instance violates the theorem statement, which contradicts the minimality of the original instance.

Proof of Theorem 6. Let $U = J \setminus F$ be the set of jobs admitted by the two-threshold algorithm that were discarded, i.e., that did not complete by time $a_j + (1 + \frac{\varepsilon}{2})p_j$. In order to show the theorem, it suffices to prove $\sum_{j \in F} w_j \geq \sum_{j \in U} w_j$.

For the sake of contradiction, we assume that there is an instance with $\sum_{j \in F} w_j < \sum_{j \in U} w_j$. Among all such instances, we consider an instance with the smallest number of jobs. In particular, this implies that there are no jobs in the instance that were not admitted by the algorithm and there is only one machine in the instance. We show that there is another instance with strictly fewer jobs that still satisfies the above inequality, contradicting the choice of the instance.

Without loss of generality, for all jobs j , we can assume that $r_j = a_j$ and $d_j = r_j + (1 + \varepsilon)p_j$ holds. Indeed, since the first assumption does not change the availability of a job j at time a_j or the density, the two-threshold algorithm still admits j at time a_j . Further, as the algorithm discards jobs when they cannot be completed by time $a_j + (1 + \frac{\varepsilon}{2})p_j < d_j$, the second assumption does not change whether a job is completed on time by the algorithm.

Observe that a job that is not interrupted completes on time. Hence, the assumption $\sum_{j \in F} w_j < \sum_{j \in U} w_j$ implies that there are jobs whose processing is interrupted. Fix a job j that is preempted but whose children's processing is not interrupted. Let K be the set of children of j , and let $\pi = \pi(j)$ if it exists. Let $C'_{j'}$ be the last point in time that the two-threshold algorithm works on job j' , which is either the completion time of j' or the point when j' was discarded because of jobs with higher densities. Denote by $C' := \max\{\max_{k \in K} C'_k, C'_j\}$, the last point in time when j or one of its children were processed. Observe that during $[a_j, C')$ only j and j 's children are processed.

Our goal is to create a new instance where j and its children are replaced by a new job j^* . Let F' and U' denote the finished and unfinished jobs, respectively, after the replacement. We will show that the new instance satisfies the following properties:

- (i) Job j^* is admitted at a_j and completes at time C' .
- (ii) $\sum_{j' \in F'} w_{j'} < \sum_{j' \in U'} w_{j'}$.
- (iii) There are strictly fewer jobs.

By assumption j has at least one child. Hence, property (iii) follows trivially from our replacement. We do not make any changes to a job $j' \notin K \cup \{j\}$. Thus, property (i) and the assumptions on the instance imply that our changes will not influence whether such a job j' is discarded or completed by the algorithm.

We set $p_{j^*} = \bar{p}_j + \sum_{k \in K} p_k$, where $\bar{p}_j \leq p_j$ is the actual amount that the two-threshold algorithm processed j in the original instance. For ensuring that j^* is available at a_j , we set $r_{j^*} = a_j$ and $d_{j^*} = r_{j^*} + (1 + \varepsilon)p_{j^*}$. This choice of parameters implies that, if j^* is admitted at time r_{j^*} , it will complete at time

$$r_{j^*} + p_{j^*} = a_j + \bar{p}_j + \sum_{k \in K} p_k = C' < d_{j^*}$$

since no other job is released during the interval $[a_j, C')$ in the new instance. Thus, in order to show property (i), it suffices to show that j^* interrupts job π at r_{j^*} . Recall that the two-threshold algorithm compares p_{j^*} with p_π in order to decide upon admission of j^* . There are three possibilities: $p_{j^*} \leq \frac{\varepsilon}{2}p_\pi$, $p_{j^*} \in (\frac{\varepsilon}{2}p_\pi, p_\pi]$, and $p_{j^*} > p_\pi$. Depending on the interval, different admission rules apply.

For defining the weight of job j^* , we distinguish two cases based on job j .

Case I. If j completes on time, set $w_{j^*} = w_j + \sum_{k \in K} w_k \geq w_j$. We observe that $\sum_{k \in K} p_k \leq \frac{\varepsilon}{2}p_j$ as j completes on time. Further,

$$\rho_{j^*} = \frac{w_j + \sum_{k \in K} w_k}{p_j + \sum_{k \in K} p_k} \geq \frac{\rho_j p_j + \frac{8}{\varepsilon} \rho_j \sum_{k \in K} p_k}{p_j + \sum_{k \in K} p_k} \geq \rho_j.$$

Thus, if p_{j^*} and p_j belong to the same interval with respect to p_π , j^* is admitted upon release and property (i) is satisfied. If they belong to different intervals, we note that

$$\left(1 + \frac{\varepsilon}{2}\right) p_j \geq p_{j^*} = p_j + \sum_{k \in K} p_k \geq p_j \quad (1)$$

and distinguish two cases.

■ $p_j \leq \frac{\varepsilon}{2}p_\pi < p_{j^*}$: We note that (1) and $\varepsilon \leq 1$ imply $p_{j^*} \leq p_\pi$. Thus,

$$w_{j^*} = \rho_{j^*} p_{j^*} \geq \rho_j \frac{\varepsilon}{2} p_\pi \geq \frac{8}{\varepsilon} \rho_\pi \frac{\varepsilon}{2} p_\pi = 4w_\pi,$$

which guarantees property (i).

■ $p_j \leq p_\pi < p_{j^*}$: We have $p_{j^*} = (1 + \delta)p_j \leq (1 + \delta)p_\pi$ for some $\delta \in (0, \frac{\varepsilon}{2}]$. Further, $w_{j^*} \geq w_j + \delta \frac{8}{\varepsilon} w_j \geq 4(1 + \delta)w_\pi$. Thus, $\rho_{j^*} = \frac{w_{j^*}}{p_{j^*}} \geq \frac{4(1 + \delta)w_\pi}{(1 + \delta)p_\pi} \geq 4\rho_\pi$, and property (i) holds.

Hence, the total weight of the jobs completed by the two-threshold algorithm and the total weight of the discarded jobs does not change in all cases, which implies property (ii).

Case II. If j does not complete on time, we set $w_{j^*} = \frac{3}{4} \sum_{k \in K} w_k$. If some $k^* \in K$ satisfies $p_{k^*} > \frac{\varepsilon}{2}p_j$, then $\sum_{k \in K} w_k \geq w_{k^*} \geq 4w_j$ by definition of the two-threshold algorithm. Using that j does not finish on time, we know that $\sum_{k \in K} p_k > \frac{\varepsilon}{2}p_j$. Thus, if the processing times for all $k \in K$ are bounded from above by $\frac{\varepsilon}{2}p_j$, then $\sum_{k \in K} w_k \geq \frac{8}{\varepsilon} \rho_j \sum_{k \in K} p_k \geq 4w_j$.

For property (i), we start by bounding w_{j^*} and ρ_{j^*} . Using the observation above, $w_{j^*} = \frac{3}{4} \sum_{k \in K} w_k \geq 3w_j$. By Observation 5, $k \in K$ with $p_k \in (\frac{\varepsilon}{2}p_j, p_j]$ satisfy $\rho_k \geq 4\rho_j$. Hence, $\sum_{k \in K} w_k = \sum_{k \in K} \rho_k p_k \geq 4\rho_j \sum_{k \in K} p_k$. Using again that $\sum_{k \in K} w_k \geq 3w_j$, we have

$$\rho_{j^*} = \frac{w_{j^*}}{p_{j^*}} \geq \frac{1/2 \sum_{k \in K} w_k + w_j}{\bar{p}_j + \sum_{k \in K} p_k} \geq \frac{2\rho_j \sum_{k \in K} p_k + \rho_j p_j}{\sum_{k \in K} p_k + \bar{p}_j} \geq \rho_j.$$

As before, if p_j and p_{j^*} are in the same interval with respect to p_π , these observations guarantee that j^* interrupts π at a_j , which implies property (i). If they belong to different intervals, we distinguish five cases.

■ $p_j \leq \frac{\varepsilon}{2}p_\pi < p_{j^*} \leq p_\pi$: We have $w_{j^*} = \rho_{j^*} p_{j^*} \geq \rho_j \frac{\varepsilon}{2} p_\pi \geq \frac{8}{\varepsilon} \rho_\pi \frac{\varepsilon}{2} p_\pi = 4w_\pi$.

■ $p_j \leq \frac{\varepsilon}{2}p_\pi < p_\pi < p_{j^*}$: We have $\rho_{j^*} \geq \rho_j > 4\rho_\pi$.

■ $\frac{\varepsilon}{2}p_\pi < p_j \leq p_\pi < p_{j^*}$: We know that $\rho_{j^*} \geq \rho_j = \frac{w_j}{p_j} \geq \frac{4w_\pi}{p_\pi} = 4\rho_\pi$.

■ $p_{j^*} \leq \frac{\varepsilon}{2}p_\pi < p_j$: We note that $p_{j^*} \geq \sum_{k \in K} p_k > \frac{\varepsilon}{2}p_j$, which implies $p_j \leq p_\pi$. We have $\rho_{j^*} = \frac{w_{j^*}}{p_{j^*}} \geq \frac{3w_j}{\varepsilon/2p_\pi} \geq \frac{12w_\pi}{\varepsilon/2p_\pi} = \frac{24}{\varepsilon} \rho_\pi$.

■ $\frac{\varepsilon}{2}p_\pi < p_{j^*} \leq p_\pi < p_j$: We have $w_{j^*} \geq 3w_j = 3\rho_j p_j > 3 \cdot 4\rho_\pi p_\pi = 12w_\pi$.

Hence, in all cases, j^* satisfies the conditions of the two-threshold algorithm to interrupt the processing of π at r_{j^*} .

Recall that $\sum_{k \in K} w_k \geq 4w_j$. After replacing $K \cup \{j\}$ with j^* , it holds that

$$\sum_{j' \in F'} w_{j'} = \sum_{j' \in F} w_{j'} - \frac{1}{4} \sum_{k \in K} w_k < \sum_{j' \in F} w_{j'} - w_j < \sum_{j' \in U} w_{j'} - w_j = \sum_{j' \in U'} w_{j'},$$

which implies property (ii).

As argued above, this contradicts the choice of the instance, which concludes the proof. \blacktriangleleft

4 Weight of admitted jobs vs. weight of Opt

In this section, we show that the total weight of jobs finished by an optimal solution is up to a factor $\mathcal{O}(\frac{1}{\varepsilon})$ comparable to the total weight of jobs admitted by the two-threshold algorithm:

► **Theorem 7.** *Let OPT and J be the set of jobs admitted by an optimal, non-migratory solution and the two-threshold algorithm, respectively. Then, $\sum_{x \in OPT} w_x \leq \mathcal{O}(\frac{1}{\varepsilon}) \sum_{j \in J} w_j$.*

For proving this statement, it is sufficient to focus on X , the set of jobs scheduled by OPT that the two-threshold algorithm did not admit since $OPT \subseteq X \cup J$.

Fix a job $x \in X$ that OPT schedules on machine i . The two-threshold algorithm admits a job j during the interval $[r_j, d_j - (1 + \frac{\varepsilon}{2})p_{ij}]$ if it is sufficiently dense. Since x is not admitted by our algorithm, the algorithm is processing jobs J_x on machine i during the interval $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_{ix}]$ with densities that are large and prevent interruption by x . That is, for jobs $j \in J_x$ with $\frac{\varepsilon}{2}p_{ij} \geq p_{ix}$ it holds that $\rho_{ij} > \frac{\varepsilon}{8}\rho_{ix}$, for jobs $j \in J_x$ with $p_{ix} \in (\frac{\varepsilon}{2}p_{ij}, p_{ij}]$ Observation 5 implies $\rho_{ij} > \frac{\varepsilon}{8}\rho_{ix}$ and for jobs $p_{ij} < p_{ix}$ it holds that $\rho_{ij} > \frac{1}{4}\rho_{ix}$. We say that the jobs J_x *block* the admission of x . We will charge the weight w_x to the weight of the jobs in J_x . Exploiting the two thresholds which the algorithm uses to make admission decisions, we show that the algorithm “obtains” a weight from partially finished jobs of at least $\Omega(\varepsilon)w_x$ in the interval $[r_x, d_x]$.

Proof idea. We give an intuition by considering a single-machine instance where the two-threshold algorithm admits exactly one job j . Consider the jobs in X whose admission was blocked by j : We know that the interval $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$ is completely covered by the processing of job j , i.e., by the interval $[a_j, C_j]$.

Now consider a job x with $p_x \leq p_j$. Thus, the deadline of x is at most $C_j + (1 + \frac{\varepsilon}{2})p_x \leq a_j + (2 + \frac{\varepsilon}{2})p_j$. This implies that OPT can schedule such jobs only during $[a_j, a_j + (2 + \frac{\varepsilon}{2})p_j]$, an interval of length $(2 + \frac{\varepsilon}{2})p_j$. The admission rule for the case $p_x \leq \frac{\varepsilon}{2}p_j$ and Observation 5 for $p_x \in (\frac{\varepsilon}{2}p_j, p_j]$ imply $\rho_x < \frac{8}{\varepsilon}\rho_j$. Hence,

$$\sum_{\substack{x \in X \\ p_x \leq p_j}} w_x = \sum_{\substack{x \in X \\ p_x \leq p_j}} \rho_x p_x \leq \frac{8}{\varepsilon} \rho_j \sum_{\substack{x \in X \\ p_x \leq p_j}} p_x \leq \frac{8}{\varepsilon} \rho_j \left(2 + \frac{\varepsilon}{2}\right) p_j = \left(\frac{16}{\varepsilon} + 4\right) w_j.$$

For a job x with $p_x > p_j$, the slackness assumption guarantees that $r_x \leq d_x - (1 + \varepsilon)p_x$. Further, the interval $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$ is contained in $[a_j, C_j]$. This allows us to upper bound the processing time p_x by $\frac{2}{\varepsilon}p_j$. Thus, OPT can schedule such jobs only during $[a_j, a_j + (1 + \frac{2}{\varepsilon})p_j]$. Using the admission rule of our algorithm in this case gives

$$\sum_{\substack{x \in X \\ p_x > p_j}} w_x = \sum_{\substack{x \in X \\ p_x > p_j}} \rho_x p_x < 4\rho_j \sum_{\substack{x \in X \\ p_x > p_j}} p_x < 4\rho_j \left(1 + \frac{2}{\varepsilon}\right) p_j = \left(4 + \frac{8}{\varepsilon}\right) w_j.$$

Combining the above two calculations yields that $\sum_{x \in X} w_x \in \mathcal{O}(\frac{1}{\varepsilon}w_j)$.

Proof outline. In this particular instance, each job $x \in X$ is blocked by a job with either larger or smaller processing time. In general, this is not necessarily true. Hence, in order to extend this idea to arbitrary instances, we partition the jobs in X according to whether at least half of their *availability interval* $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$ is covered by jobs in J with smaller or larger processing times. We then show that by losing an additional factor 3, we can assume that only one type covers the availability interval of each job in X . This is done in Lemma 10.

An additional technical challenge poses the fact that, even after we assume that a job is blocked by either shorter or longer jobs, the densities of these jobs are still not uniform enough to directly generalize the above idea to arbitrary instances. In Lemma 8, we show that, at the loss of an additional factor 4, we can partition the jobs in X according to their densities and bound the weight for each density level separately. We then upper bound OPT's available time for scheduling jobs of a certain density level in Lemmas 13 and 14 depending on the size of the blocking jobs.

Proof of Theorem 7. Since OPT and the two-threshold algorithm are non-migratory, we fix one particular machine i and only consider jobs that either OPT or the two-threshold algorithm scheduled on machine i . For simplicity, we drop the index i for the remainder.

In order to partition jobs according to their densities, we geometrically partition the range of potential job densities, $(0, \max_j \rho_j]$, into intervals of the form $(2^{\ell-1}, 2^\ell]$ and call $\ell \in \mathbb{Z}$ a *density level*. We say that a job $j \in J$ has *density level* ℓ if $2^{\ell-1} < \rho_j \leq 2^\ell$. (For jobs $x \in X$ we are interested in the density levels of the jobs that block them, which we define later.)

For a job $j \in J$ with density level $\lceil \log_2 \rho_j \rceil$, we first argue that we can separately charge j at the levels $\ell \leq \lceil \log_2 \rho_j \rceil$ at the loss of a constant factor, formalized in the next lemma. This allows us to focus on one density level ℓ at a time.

► **Lemma 8.** *Suppose there is a scheme that charges job $j \in J$ a weight of at most $2^\ell c p_j$ at level $\ell \leq \lceil \log_2 \rho_j \rceil$ and no weight at level $\ell > \lceil \log_2 \rho_j \rceil$, where $c > 0$ is a constant. Then, the total weight charged to j is at most $4c w_j$.*

Proof. The total weight charged to j is upper bounded by

$$\sum_{-\infty}^{\lceil \log_2 \rho_j \rceil} 2^\ell c p_j = c \cdot p_j \left(1 + 2^{\lceil \log_2 \rho_j \rceil + 1} - 1 \right) \leq 4c \rho_j p_j = 4c w_j,$$

as desired. ◀

Having this lemma at hand, we now restrict to one density level $\ell \in \mathbb{Z}$ and define $J_\ell := \{j \in J : \rho_j \geq 2^\ell\}$. Next, we remove the technical challenge that a job $x \in X$ can be blocked by jobs with smaller and larger processing times. To this end, we carefully modify the intervals where jobs in J_ℓ are scheduled such that the availability interval of a job $x \in X$ blocked by jobs in J_ℓ is completely contained in the modified intervals. To this end, we fix a level ℓ and let \mathcal{S}_ℓ denote the set of processing intervals of the jobs in J_ℓ , that is, the intervals during which jobs in J_ℓ are processed.

The modification works as follows: We copy each interval in $I \in \mathcal{S}_\ell$ twice and call one copy the *early* and the other the *late* copy. For each original interval $I = [\alpha, \omega)$, we move the early copy earlier such that it ends at α and we move the late copy later such that it begins at ω . If a copy intersects with another original (even if only partially), by potentially splitting the copy, we shift the part that intersects further into the indicated direction; that is, for the

early copy, we move the part earlier and for the late copy, we move the part later. We treat the time points where multiple copies overlap similarly. More precisely, if the interval $[t, t']$ is currently contained in k different copies, we use a $\frac{1}{k}$ -fraction from every copy to cover $[t, t']$ and send the remaining $\frac{k-1}{k}$ -fraction into the directions indicated by their name.

We denote the resulting set of intervals (including the original ones) by \mathcal{I}_ℓ . Next, we prove some structural properties about \mathcal{I}_ℓ and, for each job $x \in X$, relate its availability interval $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$ to \mathcal{I}_{ℓ_x} for some carefully chosen $\ell_x \in \mathbb{Z}$.

► **Observation 9.** *Let \mathcal{S} and \mathcal{T} be two sets of intervals such that for each $S \in \mathcal{S}$ there is a $T \in \mathcal{T}$ with $S \subseteq T$. Then, the result of the modification of \mathcal{T} covers all time points covered by the result of the modification of \mathcal{S} .*

► **Lemma 10.** *For each job $x \in X$, let $\ell_{1x} = \frac{1}{4} \cdot 2^{\lceil \log_2 \rho_x \rceil}$ and $\ell_{2x} = \frac{\varepsilon}{8} \cdot 2^{\lceil \log_2 \rho_x \rceil}$. Then, $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x] \subseteq \bigcup_{I \in \mathcal{I}_{\ell_{1x}}} I$ or $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x] \subseteq \bigcup_{I \in \mathcal{I}_{\ell_{2x}}} I$.*

Proof. Fix a job $x \in X$ and the two levels ℓ_{1x} and ℓ_{2x} from the lemma statement. By assumption, x is blocked by jobs in J at all times in $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$.

Assume that x is blocked for at least half of the time by jobs j with $p_j < p_x$. By definition of Algorithm 1, this implies that $4\rho_j > \rho_x$ holds for these jobs j . Hence, $j \in J_{\ell_{1x}}$. We will show that in this case ℓ_{1x} satisfies the lemma; for simplicity, set $\ell = \ell_{1x}$.

Conversely, suppose that x is blocked for at least half of the time by jobs j with $p_x \leq p_j$. Observation 5 for $\frac{\varepsilon}{2}p_j < p_x \leq p_j$ and the admission threshold for $\frac{\varepsilon}{2}p_j \geq p_x$ guarantee $\rho_x < \frac{8}{\varepsilon}\rho_j$. Hence, $j \in J_{\ell_{2x}}$ holds if $p_x \leq p_j$ and j blocks x . For simplicity, set $\ell = \ell_{2x}$ in this case.

Using Observation 9, it suffices to focus on the set \mathcal{S} of intervals that actually cover the interval $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$ and correspond to scheduling times of jobs that block x at level ℓ . By truncating, we assume that the earliest interval in \mathcal{S} starts not earlier than r_x and that the latest interval ends no later than $d_x - (1 + \frac{\varepsilon}{2})p_x$. We index the intervals in \mathcal{S} by starting point and let $K = |\mathcal{S}|$. Denote by α_k and ω_k the start and end point, respectively, of the k th interval.

By assumption, \mathcal{S} covers at least half of $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$. Thus,

$$\omega_K - \alpha_1 \leq d_x - \left(1 + \frac{\varepsilon}{2}\right)p_x - r_x \leq 2 \sum_{k=1}^K (\omega_k - \alpha_k).$$

This implies that \mathcal{S} and its copies cover the intervals $[\alpha_1, \alpha_1 + 2 \sum_{k=1}^K (\omega_k - \alpha_k)]$ and $[\omega_K - 2 \sum_{k=1}^K (\omega_k - \alpha_k), \omega_K]$ because \mathcal{S} and the late copies would suffice to cover the former and \mathcal{S} and the early copies would suffice to cover the latter interval.

Hence, the lemma follows if we show that $r_x \geq \omega_K - 2 \sum_{k=1}^K (\omega_k - \alpha_k)$ and $d_x - (1 + \frac{\varepsilon}{2})p_x \leq \alpha_1 + 2 \sum_{k=1}^K (\omega_k - \alpha_k)$. To this end, we observe that

$$\begin{aligned} (\alpha_1 - r_x) + \left(\left(d_x - \left(1 + \frac{\varepsilon}{2}\right)p_x \right) - \omega_K \right) &= \left(\left(d_x - \left(1 + \frac{\varepsilon}{2}\right)p_x \right) - r_x \right) - (\omega_K - \alpha_1) \\ &\leq 2 \sum_{k=1}^K (\omega_k - \alpha_1) - (\omega_K - \alpha_1), \end{aligned}$$

where we used that $\alpha_1 \geq r_x$ and $\omega_K \leq d_x - (1 + \frac{\varepsilon}{2})p_x$ by assumption on \mathcal{S} . This implies that both summands on the left hand side are bounded by the term on the right hand side. Rearranging shows the above bounds on r_x and $d_x - (1 + \frac{\varepsilon}{2})p_x$ and proves the lemma. ◀

► **Lemma 11.** *For each job $j \in J_\ell$, j 's processing intervals are contained in one contiguous interval of $\bigcup_{I \in \mathcal{I}_\ell} I$.*

32:12 $O(1/\varepsilon)$ Is the Answer in Online Weighted Throughput Maximization

Proof. The statement holds trivially if j only has one processing interval as this interval is in \mathcal{I}_ℓ . If j is preempted at some time τ and resumed at some later time τ' , then the two-threshold algorithm processes higher-density jobs in the interval $[\tau, \tau']$. By definition, these higher-density jobs are in J_ℓ if $j \in J_\ell$. Hence, the processing intervals of j together with these higher-density jobs form a contiguous interval in $\bigcup_{I \in \mathcal{I}_\ell} I$. \blacktriangleleft

Consider subsets of jobs of J_ℓ that are inclusion-wise maximal with respect to the processing intervals of the corresponding jobs and their copies forming exactly one contiguous interval in $\bigcup_{I \in \mathcal{I}_\ell} I$. Let $J_{\ell,k}$ for $1 \leq k \leq K$ and $K \in \mathbb{N}$ be those maximal subsets of J_ℓ , i.e., for each k , the processing intervals of the jobs in $J_{\ell,k}$ form a contiguous interval and adding one more job to $J_{\ell,k}$ would create at least one more interval. Lemma 11 and Lemma 10 imply the following corollary.

► **Corollary 12.** *The above defined sets $J_{\ell,1}, \dots, J_{\ell,K}$ partition J_ℓ . If job $x \in X$ is blocked at level ℓ , then there exists exactly one index $k \in \{1, \dots, K\}$ such that $J_{\ell,k}$ blocks x .*

We now partition X as follows: Let $X_L \subseteq X$ and $X_S \subseteq X$ be the jobs in OPT that are, for at least half of their availability interval $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$ blocked by jobs with larger and smaller processing times, respectively. Let $X_{*\ell} \subset X_*$ for $* \in \{L, S\}$ be the jobs that are blocked at level ℓ . The previously proven structural properties allow us to upper bound the total time that OPT has available for processing jobs in $X_{S\ell}$ and $X_{L\ell}$ in the following two lemmas.

► **Lemma 13.** *For each level $\ell \in \mathbb{Z}$, the total time that OPT has available for processing jobs in $X_{L\ell}$ is at most $(4 + \frac{\varepsilon}{2}) \sum_{j \in J_\ell} p_j$.*

Proof. By Corollary 12 it suffices to separately show the lemma for each maximal subset J' .

Consider a job x that is blocked by a subset of J' for at least half of $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$, where the jobs in J' blocking x have a larger processing time than x . By the definition of the two-threshold algorithm, this implies that there is a job $j \in J'$ with $p_x \leq p_j$ that is processed during $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$. By Lemma 10, $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x] \subseteq \bigcup_{I \in \mathcal{I}_\ell} I$ and, therefore, $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$ is contained in the interval $I = [\alpha, \omega]$ associated with the jobs in J' . Combining these two observations implies that

$$[r_x, d_x] = \left[r_x, d_x - \left(1 + \frac{\varepsilon}{2}\right)p_x \right] \cup \left[d_x - \left(1 + \frac{\varepsilon}{2}\right)p_x, d_x \right] \subseteq I \cup \left[\omega, \omega + \left(1 + \frac{\varepsilon}{2}\right) \sum_{j \in J'} p_j \right].$$

Using that the length of I is at most $3 \sum_{j \in J'} p_j$ concludes the proof. \blacktriangleleft

► **Lemma 14.** *For each level $\ell \in \mathbb{Z}$, the total time that OPT has available for processing jobs in $X_{S\ell}$ is at most $(\frac{4}{\varepsilon} + 5) \sum_{j \in J_\ell} p_j$.*

Proof. By Corollary 12 it suffices to separately show the lemma for each maximal subset J' . Let $I = [\alpha, \omega]$ be the interval associated with J' .

Consider a job x that is blocked by a subset of J' for at least half of $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$, where the jobs in J' blocking x have a smaller processing time than x . By definition of the two-threshold algorithm, this implies that there is a job $j \in J'$ that is processed during $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$.

By our slackness assumption, it holds that $d_x - r_x \geq (1 + \varepsilon)p_x$ or equivalently, $p_x \leq \frac{2}{\varepsilon} (d_x - (1 + \frac{\varepsilon}{2})p_x - r_x)$. Since x is blocked for at least half of $[r_x, d_x - (1 + \frac{\varepsilon}{2})p_x]$ by jobs in J' , this implies $p_x \leq \frac{4}{\varepsilon} \sum_{j \in J'} p_j$. Thus,

$$\begin{aligned} [r_x, d_x] &= \left[r_x, d_x - \left(1 + \frac{\varepsilon}{2}\right)p_x \right) \cup \left[d_x - \left(1 + \frac{\varepsilon}{2}\right)p_x, d_x \right) \\ &\subseteq I \cup \left[\omega, \omega + \left(\frac{4}{\varepsilon} + 2\right) \sum_{j \in J'} p_j \right). \end{aligned}$$

Using again that the length of I is at most $3 \sum_{j \in J'} p_j$ concludes the proof. \blacktriangleleft

Proof of Theorem 7. We now bound the weight of the sets $X_{S\ell}$ and $X_{L\ell}$ separately for each $\ell \in \mathbb{Z}$.

By Lemma 14, the time available for processing jobs in $X_{S\ell}$ is bounded from above by $(\frac{4}{\varepsilon} + 5) \sum_{j \in J_\ell} p_j$. Being blocked at level ℓ by smaller jobs implies that $\rho_x \leq 4 \cdot 2 \cdot 2^\ell = 8 \cdot 2^\ell$. Hence,

$$\sum_{x \in X_{S\ell}} w_x = \sum_{x \in X_{S\ell}} \rho_x p_x \leq 8 \cdot 2^\ell \left(\frac{4}{\varepsilon} + 5\right) \sum_{j \in J_\ell} p_j.$$

Similarly, by Lemma 13, the time available for processing jobs in $X_{L\ell}$ is upper bounded by $(4 + \frac{\varepsilon}{2}) \sum_{j \in J_\ell} p_j$ and being blocked at level ℓ by larger jobs implies $\rho_x \leq \frac{8}{\varepsilon} \cdot 2 \cdot 2^\ell$, where we used Observation 5 for a blocking job j with $p_x \in (\frac{\varepsilon}{2}p_j, p_j]$. Hence,

$$\sum_{x \in X_{L\ell}} w_x = \sum_{x \in X_{L\ell}} \rho_x p_x \leq \frac{16}{\varepsilon} \cdot 2^\ell \left(4 + \frac{\varepsilon}{2}\right) \sum_{j \in J_\ell} p_j.$$

Combining the last two equations with Lemma 8 gives

$$\sum_{x \in X} w_x \leq 4 \left(8 \cdot \left(\frac{4}{\varepsilon} + 5\right) + \frac{16}{\varepsilon} \left(4 + \frac{\varepsilon}{2}\right)\right) \sum_{j \in J} w_j = 192 \left(\frac{2}{\varepsilon} + 1\right) \sum_{j \in J} w_j = \mathcal{O}\left(\frac{1}{\varepsilon}\right) \sum_{j \in J} w_j. \quad \blacktriangleleft$$

Proof of main result

Proof of Theorem 1. Recall that OPT is the set of jobs scheduled in an optimal non-migratory solution and that F is the set of jobs that the two-threshold algorithm completes on time. Combining Theorems 6 and 7, we obtain

$$\sum_{x \in \text{OPT}} w_x \leq \sum_{x \in X} w_x + \sum_{x \in J} w_x \leq \mathcal{O}\left(\frac{1}{\varepsilon}\right) \sum_{j \in J} w_j \leq \mathcal{O}\left(\frac{1}{\varepsilon}\right) \sum_{j \in F} w_j, \quad (2)$$

which concludes the proof of our main result. \blacktriangleleft

Proof of Theorem 2. On identical machines, it is known that the optimal throughput achievable without migration is within a constant multiplicative factor of the throughput achievable using migration by Kalyanasundaram and Pruhs [14]. More precisely, as before, let OPT be the subset of jobs finished by an optimal (offline) non-migratory schedule, and let OPT_{mig} be the subset of jobs finished by an optimal (offline) schedule that is allowed to use migration. Then, Theorem 1.1 in [14] shows that $\frac{1}{6} \sum_{j \in \text{OPT}_{\text{mig}}} w_j \leq \sum_{j \in \text{OPT}} w_j$. Combining this with (2), we immediately obtain

$$\sum_{j \in \text{OPT}_{\text{mig}}} w_j \leq \mathcal{O}(1) \sum_{j \in \text{OPT}} w_j \leq \mathcal{O}\left(\frac{1}{\varepsilon}\right) \sum_{j \in F} w_j,$$

which proves Theorem 2. \blacktriangleleft

5 Conclusion

We have presented a provably best possible non-migratory algorithm for online weighted throughput maximization on unrelated machines, that is $\mathcal{O}(\frac{1}{\epsilon})$ -competitive against an optimal non-migratory schedule. Even for a single machine, only an $\mathcal{O}(\frac{1}{\epsilon^2})$ -competitive algorithm was previously known [18] while the lower bound was $\Omega(\frac{1}{\epsilon})$ [8]. Our result closes this gap on a single machine.

In contrast to special cases such as maximizing throughput with unit weights [19] or maximizing machine utilization ($w_j = p_j$) [16], it is known that $\mathcal{O}(1)$ -competitive algorithms are not possible even on identical machines and even when using randomization [7]. It is conceivable that $o(\frac{1}{\epsilon})$ -competitive algorithms are possible for $m \geq 2$ identical machines, which we leave as an interesting open problem.

References



- 1 Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Truthful online scheduling with commitments. In *EC*, pages 715–732. ACM, 2015. doi:10.1145/2764468.2764535.
- 2 Sanjoy K. Baruah and Jayant R. Haritsa. Scheduling for overload in real-time systems. *IEEE Trans. Computers*, 46(9):1034–1039, 1997. doi:10.1109/12.620484.
- 3 Sanjoy K. Baruah, Jayant R. Haritsa, and Nitin Sharma. On-line scheduling to maximize task completions. In *RTSS*, pages 228–236. IEEE Computer Society, 1994. doi:10.1109/REAL.1994.342713.
- 4 Sanjoy K. Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis E. Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992. doi:10.1007/BF00365406.
- 5 Sanjoy K. Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, and Dennis E. Shasha. On-line scheduling in the presence of overload. In *FOCS*, pages 100–110. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185354.
- 6 Luca Becchetti, Stefano Leonardi, and S. Muthukrishnan. Scheduling to minimize average stretch without migration. In *SODA*, pages 548–557. ACM/SIAM, 2000.
- 7 Ran Canetti and Sandy Irani. Bounding the power of preemption in randomized scheduling. *SIAM J. Comput.*, 27(4):993–1015, 1998. doi:10.1137/S0097539795283292.
- 8 Lin Chen, Franziska Eberle, Nicole Megow, Kevin Schewior, and Cliff Stein. A general framework for handling commitment in online throughput maximization. *Math. Prog.*, 183:215–247, 2020. doi:10.1007/s10107-020-01469-2.
- 9 Bhaskar DasGupta and Michael A. Palis. Online real-time preemptive scheduling of jobs with deadlines. In *APPROX*, volume 1913 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 2000. doi:10.1007/3-540-44436-X_11.
- 10 Franziska Eberle, Nicole Megow, and Kevin Schewior. Online throughput maximization on unrelated machines: Commitment is no burden. *ACM Trans. Algorithms*, 19(1), February 2023. doi:10.1145/3569582.
- 11 Juan A. Garay, Joseph Naor, Bülent Yener, and Peng Zhao. On-line admission control and packet scheduling with interleaving. In *INFOCOM*, pages 94–103. IEEE Computer Society, 2002. doi:10.1109/INFOCOM.2002.1019250.
- 12 Michael H. Goldwasser. Patience is a virtue: The effect of slack on competitiveness for admission control. *J. Sched.*, 6(2):183–211, 2003. doi:10.1023/A:1022994010777.
- 13 Samin Jamalabadi, Chris Schwiegelshohn, and Uwe Schwiegelshohn. Commitment and slack for online load maximization. In *SPAA*, pages 339–348. ACM, 2020. doi:10.1145/3350755.3400271.
- 14 Bala Kalyanasundaram and Kirk Pruhs. Eliminating migration in multi-processor scheduling. *J. Algorithms*, 38(1):2–24, 2001. doi:10.1006/jagm.2000.1128.

- 15 Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. *J. Algorithms*, 49(1):63–85, 2003. doi:10.1016/S0196-6774(03)00074-9.
- 16 Gilad Koren and Dennis E. Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theor. Comput. Sci.*, 128(1&2):75–97, 1994. doi:10.1016/0304-3975(94)90165-1.
- 17 Gilad Koren and Dennis E. Shasha. D^{over} : An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM J. Comput.*, 24(2):318–339, 1995. doi:10.1137/S0097539792236882.
- 18 Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs: Extended abstract. In *SPAA*, pages 305–314. ACM, 2013. doi:10.1145/2486159.2486187.
- 19 Benjamin Moseley, Kirk Pruhs, Clifford Stein, and Rudy Zhou. A competitive algorithm for throughput maximization on identical machines. In *IPCO*, volume 13265 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 2022.
- 20 Kirk Pruhs and Clifford Stein. How to schedule when you have to buy your energy. In *APPROX*, volume 6302 of *Lecture Notes in Computer Science*, pages 352–365. Springer, 2010. doi:10.1007/978-3-642-15369-3_27.
- 21 Chris Schwiegelshohn and Uwe Schwiegelshohn. The power of migration for online slack scheduling. In *ESA*, volume 57 of *LIPICs*, pages 75:1–75:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ESA.2016.75.

On the Exact Matching Problem in Dense Graphs

Nicolas El Maalouly  

Department of Computer Science, ETH Zurich, Switzerland

Sebastian Haslebacher  

Department of Computer Science, ETH Zurich, Switzerland

Lasse Wulf  

Institute of Discrete Mathematics, TU Graz, Austria

Abstract

In the Exact Matching problem, we are given a graph whose edges are colored red or blue and the task is to decide for a given integer k , if there is a perfect matching with exactly k red edges. Since 1987 it is known that the Exact Matching Problem can be solved in randomized polynomial time. Despite numerous efforts, it is still not known today whether a deterministic polynomial-time algorithm exists as well. In this paper, we make substantial progress by solving the problem for a multitude of different classes of dense graphs. We solve the Exact Matching problem in deterministic polynomial time for complete r -partite graphs, for unit interval graphs, for bipartite unit interval graphs, for graphs of bounded neighborhood diversity, for chain graphs, and for graphs without a complete bipartite t -hole. We solve the problem in quasi-polynomial time for Erdős-Rényi random graphs $G(n, 1/2)$. We also reprove an earlier result for bounded independence number/bipartite independence number. We use two main tools to obtain these results: A local search algorithm as well as a generalization of an earlier result by Karzanov.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms; Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases Exact Matching, Perfect Matching, Red-Blue Matching, Bounded Color Matching, Local Search, Derandomization

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.33

Related Version *Full Version*: <https://doi.org/10.48550/arXiv.2401.03924>

Funding *Lasse Wulf*: Supported by the Austrian Science Fund (FWF): W1230.

1 Introduction

A fundamental problem in computer science is the question whether a randomized algorithm has any sort of advantage over a deterministic algorithm. In particular, theoretical computer scientists are concerned with the question: $P = BPP$? Here, P contains decision problems that can be solved deterministically in polynomial-time, while BPP contains decision problems that can be solved with randomized algorithms in polynomial-time (under a two-sided, bounded error probability [2]). One can also define the classes $RP \subseteq BPP$ and $CoRP \subseteq BPP$ of randomized polynomial-time algorithms with one-sided error probability (the difference between the two classes is the side of the error). Nowadays, experts in complexity theory believe that $P = RP = CoRP = BPP$, i.e. it is believed that randomness does not offer any sort of advantage for the task of solving a problem in polynomial time. The reason for this belief are deep connections between complexity theory, circuit lower bounds, and pseudorandom generators [2, 24, 26].

While it would be intriguing to attack the conjecture $P = BPP$ directly, it seems very hard to make direct progress in this way. In particular, $P = BPP$ would imply deterministic algorithms for *all* problems which can be solved with randomness. A more humble approach



© Nicolas El Maalouly, Sebastian Haslebacher, and Lasse Wulf;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 33; pp. 33:1–33:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



one can take is to look for one specific problem, where the research community knows a randomized, but no deterministic algorithm, and try to find a deterministic algorithm for this specific problem. Every single of these results can be seen as further evidence towards $P = BPP$. One famous example of such a “derandomization” is the deterministic algorithm for primality testing by Agrawal, Kayal and Saxena [1] from 2002.

Quite interestingly, we only know of a handful of problems where a randomized but no deterministic polynomial-time algorithm is known. This paper is concerned with one of these examples, the *Exact Matching problem* (EM). Given an integer k and a simple graph G together with a coloring of its edges in red or blue, EM is the problem of deciding whether G has a perfect matching with exactly k red edges. EM was introduced by Papadimitriou and Yannakakis [36] back in 1982. Not too long after its introduction, in 1987, Mulmuley et al. [33] showed that EM can be solved in randomized polynomial-time. Despite the original problem being from 1982, and in spite of multiple applications of EM in different areas (see next paragraph), it is still not known today, if a deterministic polynomial-time algorithm exists.

Another interesting aspect of EM is its connection to *polynomial identity testing* (PIT). PIT is another one of the rare problems in BPP for which we still do not know any deterministic polynomial-time algorithm. Given a multivariate polynomial described by an algebraic circuit, PIT is the problem of deciding whether the polynomial is identically equal to zero or not. Using the well-known Schwartz-Zippel Lemma (named after Schwartz [37] and Zippel [44] who discovered it in the eighties), it is clear that PIT belongs to CoRP. Therefore, under the conjecture $CoRP = P$, there should be a deterministic polynomial-time algorithm for PIT. However, Kabanets and Impagliazzo [26] provided strong evidence that derandomizing PIT might be notoriously hard, since it would imply proving circuit lower bounds. The known randomized algorithm for EM uses PIT as a subroutine on a slightly modified Tutte matrix of the given graph. Alternatively, one can substitute the use of PIT with the famous Isolation Lemma due to Mulmuley et al. [33]. Both approaches lead to randomized polynomial-time algorithms for EM and show that EM is contained in the class RP.

History of Exact Matching

We have already established that EM should belong to P if we believe the conjecture $P = RP = BPP$. However, the best deterministic algorithm to date takes exponential time. This is especially astonishing knowing that EM was introduced by Papadimitriou and Yannakakis [36] back in 1982. A few years later, in 1987, Mulmuley et al. [33] showed that EM can be solved in randomized polynomial-time in their famous paper that also introduced the Isolation Lemma. In fact, their algorithm additionally allows for a high degree of parallelism i.e. they proved that EM belongs to RNC (and hence also to RP and BPP). RNC is defined as the class of decision problems allowing an algorithm running in polylogarithmic time using polynomially many parallel processors, while having additional access to randomness (we refer the interested reader to [6, Chapter 12] for a formal definition). This means that if we allow for randomness, EM can be solved efficiently even in parallel, while the best known deterministic algorithm requires exponential time.

In the same year 1987, Karzanov [27] gave a precise characterization of the solution landscape of EM in complete and complete bipartite graphs. His characterization also implies deterministic polynomial-time algorithms for EM restricted to those graph classes. Several articles later appeared [19, 23, 42], simplifying and restructuring those results.

EM is known to admit efficient deterministic algorithms on some other restricted graph classes as well: With standard dynamic programming techniques, EM can be solved in polynomial-time on graphs of bounded tree-width [11, 40]. Moreover, derandomization results

exist for $K_{3,3}$ -minor free graphs [41, 43] and graphs of bounded genus [18]. These works make use of so-called Pfaffian orientations. Besides solving EM on restricted graph classes, some prior work has also focused on solving EM approximately. Yuster [43] proved that in a YES-instance, we can always find an almost exact matching in deterministic polynomial-time (an almost exact matching is a matching with exactly k red edges that fails to cover only two vertices).

This completes a summary of the history of the EM problem until two years ago. With so little progress, one might wonder if the community has lost interest in, or forgot about the problem. However, over the last decade alone, we have seen the problem appear in the literature from several areas. This includes budgeted, color bounded, or constrained matching [5, 28, 31, 32, 38], multicriteria optimization [21], matroid intersection for represented matroids [7], binary linear equation systems [4], recoverable robust assignment [17], or planarizing gadgets for perfect matchings [22]. In many of these papers, a full derandomization of EM would also derandomize some or all of the results of the paper. EM also appeared as an interesting open problem in the seminal work on the parallel computation complexity of the matching problem [39], which might be partly responsible for the increase in attention that the problem has received recently.

Recent progress

As mentioned above, until recently, EM was only solved for a handful of graph classes. This is even more extreme in the case of dense graphs where it was only solved on complete and complete bipartite graphs. In 2022, El Maalouly and Steiner [12] finally made progress on this side by showing that EM can be solved on graphs of bounded *independence number* and bipartite graphs of bounded *bipartite independence number*. Here, the independence number of a graph G is defined as the largest number α such that G contains an *independent set* of size α . The bipartite independence number of a bipartite graph G equipped with a bipartition of its vertices is defined as the largest number β such that G contains a *balanced independent set* of size 2β , i.e., an independent set using exactly β vertices from both color classes. This generalizes previous results for complete and complete bipartite graphs, which correspond to the special cases $\alpha = 1$ and $\beta = 0$. The authors also conjectured that counting perfect matchings is #P-hard for this class of graphs. This conjecture was later proven in [14] already for $\alpha = 2$ or $\beta = 3$. This makes them the first classes of graphs where EM can be solved, even though counting perfect matchings is #P-hard. This work was later extended to an FPT-algorithm on bipartite graphs parameterized by the bipartite independence number [12].

There has also been a recent interest in approximation algorithms for EM. Such approximation algorithms have been developed for the closely related *budgeted matching* problem, where sophisticated methods were used to achieve a PTAS [5] and, more recently, an efficient PTAS [8]. These methods however do not guarantee to return a perfect matching (but note that a deterministic FPTAS for budgeted matching would imply a deterministic polynomial-time algorithm for EM [5]). In [9, 11] it is argued that relaxing the perfect matching constraint takes away most of the difficulty of the problem. In contrast, the aim of the recent work has been to keep the perfect matching constraint and relax the requirement on the number of red edges. The first such result was given in [12], where it was shown that in a bipartite graph we can always find a perfect matching with at least $0.5k$ and at most $1.5k$ red edges in deterministic polynomial-time. This represents a two-sided approximation for the problem. Shortly after, [9] studied the surprisingly much more difficult problem of getting a one-sided approximation and presented a 3-approximation in that setting (i.e. an algorithm that outputs a perfect matching with at least $k/3$ and at most k red edges), relying on a newly defined concept of graph rigidity.

Another relaxation of the problem is to consider only modular constraints on the number of red edges, e.g., requiring the output perfect matching to have an odd number of red edges. In the case of bipartite graphs, the problem can be solved using the more general result of [3] on network matrices. This does not work for general graphs, for which the problem was solved in [13] with a different approach relying on a deep result by Lovász [30] on the *linear hull* of perfect matchings. The problem remains open for other congruency constraints, e.g., requiring the output to have $(r \bmod p)$ red edges for some integers r and p . The latter problem has been used by [34] as a building block in an algorithm for a special class of integer programs having a constraint matrix with bounded subdeterminants. This means that a deterministic algorithm for this special case of EM would also derandomize the algorithm of [34].

In [11], the Top- k Perfect Matching problem is introduced, where the input is a weighted graph and the goal is to find a PM that maximizes the weight of the k heaviest edges in the matching. In combination with the result from [15], the problem is shown to be polynomially equivalent to EM when the input weights are polynomially bounded. Several approximation and FPT algorithms were also developed.

Another recent line of work follows a polyhedral approach to understand the differences between finding a perfect matching and EM [25]. In particular, the authors show exponential extension complexity for the bipartite exact matching polytope. This stands in contrast with the bipartite perfect matching polytope whose vertices are all integral [10].

Finally, [40] studies some generalizations of EM to matching problems with vertex color constraints and shows an interesting connection to quantum computing.

Our contribution

In this paper, we study EM on dense graph classes. We are able to solve EM in deterministic polynomial time on many different classes of dense graphs, which before could only be handled by a randomized algorithm. In order to achieve this result, we use two key techniques: First, a local search algorithm, second, a generalization of Karzanov's [27] theorem. With the first technique, the local search algorithm, we obtain the following results: (For a formal definition of all the graph classes listed, as well as a motivation for why we consider exactly these classes, we refer the reader to Section 2.1.)

- There is a deterministic $n^{O(1)}$ time algorithm for EM on complete r -partite graphs for all $r \geq 1$. The constant in the exponent is independent of r . This is an extension of the special cases $r = n$ and $r = 2$, which correspond to the cases of complete and complete bipartite graphs [27] already known in 1987.
- There is a deterministic $n^{O(1)}$ time algorithm for EM on graphs of bounded neighborhood diversity $d = O(1)$. The neighborhood diversity is a parameter popular in the area of parameterized complexity [29].
- There is a deterministic $n^{O(1)}$ time algorithm for EM on graphs G which have no complete bipartite t -hole (i.e. $K_{t,t} \not\subseteq \overline{G}$) with $t = O(1)$.
- There is a deterministic $n^{O(\log^{12}(n)p^{-12})}$ time algorithm for EM on the random graph $G(n, p)$. By this, we mean the following: We say an algorithm is *correct* for a graph G , if for all possible red-blue edge colorings of G and all possible k , the algorithm correctly solves EM on that input. We show that there is a deterministic algorithm \mathcal{A} , which always halts in $n^{O(\log^{12}(n)p^{-12})}$ steps, and if G is sampled from the distribution $G(n, p)$, then with high probability \mathcal{A} is correct for G . As a special case, we obtain a quasi-polynomial algorithm for $G(n, 1/2)$. We are the first authors to consider EM from the perspective of random graphs.

- As a special case, our main theorem contains a re-proof of the two main results of [12], showing that there is a deterministic $n^{O(1)}$ algorithm for EM on graphs of bounded independence number/ bip. graphs of bounded bip. independence number. Our result is therefore a large generalization of this earlier result and puts it into the bigger context of local search.

We also identify a certain graph property, which we call the *path-shortening property*. Graphs which are very dense and structured are candidates to examine for this property. Our main theorem is that for every graph with the path-shortening property, a local search approach can be used to correctly solve the Exact Matching Problem. In fact, all the examples above follow from our main theorem. We remark that our local search algorithm is very simple, only the proof of its correctness is quite involved. The main idea of the observation is that in graphs with the path-shortening property, strong locality statements about the set of all perfect matchings can be made. Details are presented in Section 3.

While the local search approach allows us to tackle several new graph classes, we still notice that it fails even on some very dense and structured graph classes. In particular, we are interested in graph classes which are related to the problem of counting perfect matchings (for example, Okamoto et al. [35] list chordal, interval, unit interval, bipartite chordal, bipartite interval, and chain graphs among others). We highlight one example, the case of so-called chain graphs, where our local search fails.

This failure inspires us to seek other methods to understand the EM problem on dense graphs and leads us to consider our second key technique. We call this technique *Karzanov's property*, as it is a generalization of the result by Karzanov [27]. We show that several graph classes have Karzanov's property, including classes where our local search algorithm fails. We introduce a related property, which we call the *chord property*. We introduce a novel binary-search like procedure, which gives us both an efficient algorithm for EM on these graph classes, as well as a characterization of their solution landscape.

Finally, we are also able to identify some graph classes, where Karzanov's property almost holds. We call this the *weak Karzanov's property*. For those graph classes, we are not able to solve EM deterministically, but we are at least able to show that one can always find a PM with either k or $k - 1$ red edges. Hence we can come very close to solving the problem. These graph classes are therefore obvious candidates to attack next in the effort of derandomizing EM. In summary, we obtain the following.

- There is a deterministic $n^{O(1)}$ time algorithm for EM on chain graphs, unit interval graphs, bipartite unit interval graphs and complete r -partite graphs for all $r \geq 1$. The solution landscape for these graph classes can also be characterized by the perfect matchings of maximum and minimum number of red edges with a given parity.
- There is a deterministic $n^{O(1)}$ time algorithm on interval graphs, bipartite interval graphs, strongly chordal graphs and bipartite chordal graphs, that outputs a perfect matching with either $k - 1$ or k red edges or deduces that the answer of the given EM-instance is "No".

Organization of the paper

In the following we start with some preliminaries in Section 2. Then in Section 3 we introduce our local search algorithm, the main ideas behind its correctness and its limitations. In Section 4 we introduce Karzanov's property, as well as Karzanov's weak property and discuss the main ideas behind their utility and limitations.

In the appendix of the full version of this paper, you can find the detailed proofs missing from Section 3, proofs that the local search algorithm works for several graph classes, the detailed proofs missing from Section 4, and proofs that several graph classes satisfy Karzanov's property while some others only satisfy Karzanov's weak property.

2 Preliminaries and Problem Definition

All graphs in this paper are undirected and simple. For a graph $G = (V, E)$, we denote by $V(G) := V$ its vertex set and by $E(G) := E$ its edge set. We usually use the letters n, m to denote $n := |V|$ and $m := |E|$. In this paper, paths and cycles are always simple (i.e. no vertex is repeated). In order to simplify the notation, we identify paths and cycles with their edge sets. Any reference to their vertices will be made explicit. The *neighborhood* $N(v)$ of a vertex v is the set of all vertices adjacent to v . A *colored graph* in this paper is a graph where every edge has exactly one of two colors, i.e. a tuple (G, c) with $c : E(G) \rightarrow \{\text{red}, \text{blue}\}$. For a subset $F \subseteq E$ of edges, we denote by $R(F) := \{e \in F \mid e \text{ is red}\}$ its set of red edges and by $r(F) := |R(F)|$ its number of red edges. Analogously, we define $B(F)$ and $b(F)$ for blue edges. A *matching* of G is a set $M \subseteq E$ of edges, which touches every vertex at most once. A *perfect matching* (abbreviated PM) is a matching M which touches every vertex exactly once. An edge e is called *matching*, if $e \in M$ and *non-matching* otherwise. The *Exact Matching Problem* is formally defined as follows.

Problem EM

Input: Colored graph (G, c) , integer $k \geq 0$.

Question: Is there a perfect matching M in G such that $r(M) = k$?

The *symmetric difference* $A \triangle B$ of two sets A and B is $A \cup B \setminus (A \cap B)$. Let M be a PM. An *M -alternating cycle*, or simply an *alternating cycle* (if M is clear from context), is a cycle which alternates between edges in M and edges not in M . An *alternating path* is defined analogously. If M_1, M_2 are PMs, it is well known that $D := M_1 \triangle M_2$ is a vertex-disjoint union of alternating cycles. Since \triangle behaves like addition mod 2, we also have $M_2 = M_1 \triangle D$ and $M_1 = M_2 \triangle D$. In this paper, we try to follow the convention that the letter C denotes a single cycle and the letter D denotes a vertex-disjoint union of one or more cycles.

2.1 Definition of Graph Classes

Throughout the paper, we show how to solve EM on various classes of dense graphs. In this subsection, we properly define all graph classes used.

The motivation to consider exactly those classes comes from different sources. Some of the classes considered are direct generalizations of classes, where it was previously known that EM can be solved. Other classes are generally well-known. For the remaining classes, we regard them as interesting, because they appear in the context of counting the number of perfect matchings. (For example, in their paper about counting perfect matchings, Okamoto et al. [35] list chordal, interval, unit interval, bipartite chordal, bipartite interval, and chain graphs among others.) The reason for this is, that if it is #P-hard to count the number of perfect matchings, then the Pfaffian derandomization method used in [18, 41, 43] is unlikely to work (compare [15] for details). We also pay special attention to bipartite graphs, since we expect EM to be easier to tackle if the graph is bipartite.

A graph is *complete r -partite*, if the vertex set can be partitioned into r parts V_1, \dots, V_r such that inside each part there are no edges, and between two different parts, there are all the possible edges. The case $r = n$ corresponds to the complete graph, while $r = 2$ corresponds

to the complete bipartite graph. A generalization of complete r -partite graphs, if $r = O(1)$, are graphs of *bounded neighborhood diversity*, a parameter coming from parameterized complexity [29]. A graph has neighborhood diversity d , if $V(G)$ can be partitioned into d parts $V_1 \dots, V_d$, such that between every two parts V_i, V_j with $i \neq j$, there are either no edges, or all the possible edges, and every part itself induces either a complete or an empty graph.

The Erdős-Rényi random graph $G(n, p)$ is the random graph on n vertices, where every edge appears with probability p independently [16]. It is very well-studied and has a rich history.

The remaining definitions in this subsection are motivated by [35]. Furthermore, many of these graph classes are extensively studied in algorithmic graph theory [20]. A graph $G = (V, E)$ is an *interval graph* if there exists a mapping $I : V \rightarrow \{[a, b] \subseteq \mathbb{R} \mid a \leq b\}$ such that $\{u, v\} \in E \iff I(u) \cap I(v) \neq \emptyset$ holds for all distinct $u, v \in V$. If additionally $I(v)$ is a unit interval for all vertices v , then G is called a *unit interval graph*.

We may consider bipartite versions of interval graphs the following way: A bipartite graph $G = (X \dot{\cup} Y, E)$ is a *bipartite interval graph* if there exists a mapping $I : X \dot{\cup} Y \rightarrow \{[a, b] \subseteq \mathbb{R} \mid a \leq b\}$ such that $\{x, y\} \in E \iff I(x) \cap I(y) \neq \emptyset$ holds for all $x \in X, y \in Y$. If additionally $I(v)$ is a unit interval for all vertices v , then G is called a *bipartite unit interval graph*. Note that by this definition, if two vertices $x, y \in X$ are in the same color class of the bipartition, there is no edge between x and y even if the intervals $I(x)$ and $I(y)$ intersect.

Interval graphs are *strongly chordal*. A graph G is called strongly chordal if every cycle of length at least 4 admits a chord and every even cycle of length at least 6 admits an odd chord (i.e. a chord that splits the cycle into two odd length paths).

Bipartite interval graphs are *bipartite chordal*. A bipartite graph G is bipartite chordal if and only if every cycle of (necessarily even) length at least 6 admits a chord.

Finally, we consider a special case of bipartite interval graphs, so-called *chain graphs*. A bipartite graph $G = (X \dot{\cup} Y, E)$ is a chain graph if and only if its vertices can be relabeled as $x_1, \dots, x_{|X|} \in X$ and $y_1, \dots, y_{|Y|} \in Y$ such that $N(x_i) \subseteq N(x_{i+1})$ and $N(y_j) \subseteq N(y_{j+1})$ hold for all $1 \leq i < |X|$ and $1 \leq j < |Y|$.

3 Local Search

As of course is well known, the central idea behind a local search algorithm is to only examine solutions close to the current solution at every step. Hence we require a notion of distance. For our purpose, this notion is as follows.

► **Definition 1.** Let (G, c) be a colored graph and $M_1, M_2 \subseteq E(G)$ be two PMs. The distance between M_1, M_2 is

$$\text{dist}(M_1, M_2) := \min\{r(M_1 \triangle M_2), b(M_1 \triangle M_2)\}.$$

For an integer $s \geq 0$, the s -neighborhood of a PM M is

$$\mathcal{N}_s(M) := \{M' \subseteq E(G) \mid M' \text{ is a PM, } \text{dist}(M, M') \leq s\}.$$

Note that $\text{dist}(M_1, M_2) = \min\{|R(M_1) \triangle R(M_2)|, |B(M_1) \triangle B(M_2)|\}$. In other words, two PMs have small distance if and only if their two sets of red edges are almost the same, or their two sets of blue edges are almost the same. For example, if two PMs have the same set of red edges, i.e. $R(M_1) = R(M_2)$, then $\text{dist}(M_1, M_2) = 0$, even if their set of blue edges is completely different.

33:8 On the Exact Matching Problem in Dense Graphs

Observe that as a consequence of this definition, for a fixed PM M even the 0-neighborhood $\mathcal{N}_0(M)$ may have exponential size in n . This is a problem for us: how can we perform local search, if the size of the neighborhood is exponential? Fortunately, there is a fix: We do not need to know the complete neighborhood of M , all we need to know is which values of $r(M')$ are possible to achieve in the neighborhood, i.e. the set of all k' such that there exists a PM M' in the neighborhood with $r(M') = k'$. The following lemma states that this information can be computed efficiently. The idea is to guess either the set $R(M')$ or the set $B(M')$ and see if the guess can be completed to a PM using only edges of the opposite color.

► **Lemma 2.** *Assume we are given a PM M in a colored graph, and an integer $s \geq 0$. There is an algorithm which runs in $O(m^{s+3})$ time and computes the set $\{k' \in \mathbb{N} \mid \exists M' \in \mathcal{N}_s(M), r(M') = k'\}$ and for each k' in this set outputs at least one representative M'' with $r(M'') = k'$.*

Proof. Let (G, c) be the colored graph with $G = (V, E)$, and let $E_R := R(E)$ be the set of all red edges and $E_B := B(E)$ be the set of all blue edges. The algorithm works as follows:

1. Enumerate all (not necessarily perfect) matchings $X \subseteq E_R$ with $|X \triangle R(M)| \leq s$. For each such X , use a classical maximum matching algorithm on the blue edges to check whether there exists $Y \subseteq E_B$ such that $X \dot{\cup} Y =: M''$ is a PM. If the answer is affirmative, we add the number $k' := |X| = r(M'')$ to the output set (together with its representative M'').
2. After that, we repeat the same procedure with the colors switched: Enumerate all matchings $X \subseteq E_B$ with $|X \triangle B(M)| \leq s$. For each such X , check whether there exists $Y \subseteq E_R$ such that $X \dot{\cup} Y$ is a PM. If yes, we add the number $k' := n/2 - |X|$ to the output set (together with its representative M'').

The enumeration of sets X can be done in $O(m^s)$ time. Note that this algorithm is sound, in the sense that every PM M'' generated by it is indeed contained in $\mathcal{N}_s(M)$. On the other hand, the algorithm is also complete: If $M' \in \mathcal{N}_s(M)$, then either $R(M')$ or $B(M')$ appears in the enumeration. This means that not necessarily M' , but at least some M'' with $r(M') = r(M'')$ is found by the algorithm. The total runtime of the algorithm is $\Theta(m^s f_M)$, where f_M denotes the time it takes to solve the perfect matching problem deterministically. For simplification, we let $f_M = O(mn^2) = O(m^3)$ [10]. ◀

■ **Algorithm 1** A simple local search algorithm, LOCAL(s).

Input: Colored graph (G, c) , integer $k \geq 0$, local search parameter $s \geq 0$

Result: Either a PM M with $r(M) = k$, or the info that local search was unsuccessful.

$M_{\min} \leftarrow$ PM in G with minimum number of red edges among all PMs ;

$M \leftarrow M_{\min}$;

while $r(M) \neq k$ **do**

Try to find $M' \in \mathcal{N}_s(M)$ s.t. $r(M) < r(M') \leq k$ using Lemma 2;

if *successful* **then**

$M \leftarrow M'$;

else

return “local search failed.”;

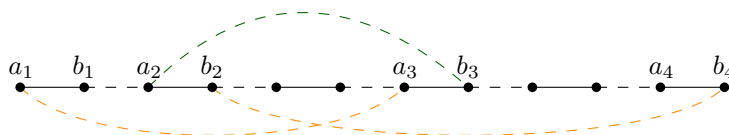
return M ;

With Lemma 2 in mind, we introduce Algorithm 1 as the most natural local search algorithm. It starts with a PM with the minimum number of red edges and iteratively tries to increase $r(M)$. Note that the PM M_{\min} in the first line of the algorithm can be computed in polynomial time (one can run a classical maximum weight perfect matching algorithm, where red edges receive weight -1, and blue edges receive weight 0). Algorithm 1 can return false negatives, in the sense that given a yes-instance of EM it is possible for the algorithm to get stuck in a local optimum and return “false”. Algorithm 1 can not return false positives. If we increase the search parameter s , we expect Algorithm 1 to be correct more often on average, but we also expect a longer runtime. We denote Algorithm 1 with parameter s by the name LOCAL(s). Since every successful iteration increases $r(M)$, the running time of LOCAL(s) is bounded by $O(m^{s+4})$. It is desirable to understand when LOCAL(s) correctly solves EM. This is partially answered in the next subsection.

3.1 A Sufficient Condition for Local Search

We present a sufficient condition for LOCAL(s) to correctly solve EM. Although the algorithm LOCAL(s) is quite simple, the proof that our condition suffices for correctness of the algorithm is involved. The main idea is the observation that in certain dense and highly structured graphs, it is possible to prove strong locality properties for the set of all perfect matchings. In particular, we consider graphs which have the following technical property:

► **Definition 3.** Let $t \geq 2$ be an integer. A graph G has the so-called path-shortening property PSHORT(t), if for all PMs $M \subseteq G$, and for all M -alternating paths P the following holds: If $F \subseteq P \cap M$ is a subset of matching edges of size $|F| = t$ and $F = \{\{a_1, b_1\}, \dots, \{a_t, b_t\}\}$, where the vertices $a_1, b_1, a_2, b_2, \dots, a_t, b_t$ appear in this order along the path, then the graph G contains an edge $\{a_i, b_j\}$ for some indices $1 \leq i < j \leq t$ or both the edges $\{a_{i_1}, a_{i_3}\}, \{b_{i_2}, b_{i_4}\}$ for some indices $1 \leq i_1 < i_2 < i_3 < i_4 \leq n$.



■ **Figure 1** An example of the property PSHORT(4) on a path of length 11. Matching edges are bold. Both possibilities of path shortening are highlighted.

An illustration of this property is provided in Figure 1. Note that the property is monotone in t , i.e. PSHORT(t) implies PSHORT(t') for all $t' > t$. Our main result is the insight that the property PSHORT is sufficient for local search to be correct.

► **Theorem 4.** If a graph G has property PSHORT(t), then the deterministic algorithm LOCAL($O(t^{12})$) solves EM on graph G in time $n^{O(t^{12})}$ (for all possible edge colorings $c : E(G) \rightarrow \{\text{red}, \text{blue}\}$ and all target values $k \in \mathbb{N}$).

In particular, if $t = O(1)$, then the algorithm above is polynomial-time. Such a theorem is only useful of course, if we can show that many different graphs have this mysterious property PSHORT. Indeed, we can show (proofs can be found in the appendix of the full version of this paper):

33:10 On the Exact Matching Problem in Dense Graphs

- Complete r -partite graphs have the property PSHORT(3) for all integers $r \geq 1$.
- Graphs of bounded neighborhood diversity d have the property PSHORT($d + 1$).
- Graphs of bounded independence number α have the property PSHORT($2 \text{Ram}(\alpha + 1)$), where $\text{Ram}(x) \leq 4^x$ is the diagonal Ramsey number.
- Graphs of bounded bipartite independence number β have property PSHORT($2\beta + 2$).
- If a graph G has no complete bipartite t -hole (i.e. the complement \overline{G} does not contain $K_{t,t}$ as subgraph), then G has the property PSHORT($2t$).
- The random graph $G(n, p)$ has property PSHORT($2 \log(n)/p$) with high probability.

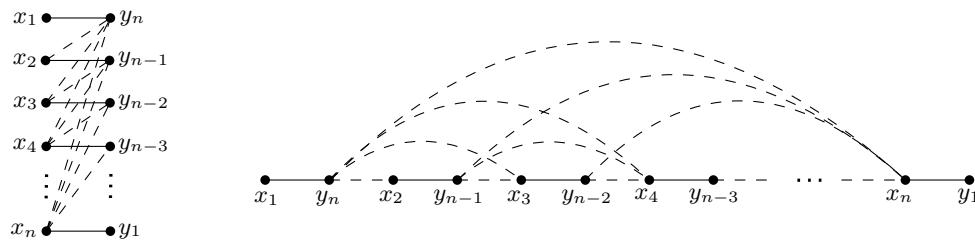
The proof of Theorem 4 is quite technical and requires many steps. The complete proof is given in the full version of this paper. The main insight is the observation that in graphs with property PSHORT, the set of all perfect matchings must obey strong locality guarantees. For the proof of this locality statement, we introduce the new idea of *local modifiers*. Each local modifier has a weight associated to it, and the goal becomes to combine the weights in such a way, that they cancel out to be 0. The proof uses ideas and tools from Combinatorics, like an argument similar to the Erdős-Szekeres theorem, and a helpful lemma from number theory about 0-sum subsequences.

3.2 Limitation of Local Search

A natural question is whether the approach we presented in this section extends to more graph classes, in particular to all dense graph classes. Here we show that our local search approach fails even for some very dense and very structured classes of graphs. We consider the case of chain graphs.

Recall the definition of a chain graph from Section 2.1. Note that chain graphs can be sparse (e.g. an empty graph is a chain graph), but when required to contain a perfect matching, the graph has to be dense. This can be seen by considering the vertex of highest label on one side and observing that it must be connected to all vertices on the other side. By recursively applying this observation, we can see that the number of edges in the graph must be at least $n^2/8$.

The following is an example of a chain graph that does not satisfy the property PSHORT(t) even for $t = n$. Let $G = (X \cup Y, E)$ be the chain graph defined by $|X| = |Y| = n$, $N(x_i) := \{y_{n-i+1}, \dots, y_n\}$ for all $1 \leq i \leq |X|$ (see Figure 2). Observe that it is a valid chain graph and that $M := \{\{x_i, y_{n-i+1}\}, \text{ for } 1 \leq i \leq |X|\}$ is a perfect matching. Also observe that there is no edge of the form $\{x_i, y_j\}$ for $1 \leq i \leq n - j \leq n$ as required for the property PSHORT(t) (since the graph is bipartite, no edges of the form $\{x_i, x_j\}$ or $\{y_i, y_j\}$ exist either).



■ **Figure 2** An example of a chain graph G and a perfect matching M in G (left figure), where there exists an M -alternating path with n edges from M (right figure). Edges in M are bold. The property PSHORT(n) is violated on this path.

4 Extending Karzanov's Characterization

In this section, we extend the characterization of exact matchings given by Karzanov [27] for complete and complete bipartite graphs to chain graphs, unit interval graphs, and complete r -partite graphs. Moreover, we exploit this to give deterministic polynomial-time algorithms for EM on those graph classes. This complements the results of Section 3, as chain graphs and unit interval graphs are not captured by the local search approach. On the other hand, complete r -partite graphs actually fit both frameworks. Note that we only provide a coarse outline here and defer many of the proofs and details to the appendix of the full version of this paper.

Given a colored graph (G, c) , we denote by $k_{\min}(G)$ the smallest integer k such that there is a PM M in G with $r(M) = k$, and by $k_{\max}(G)$ the largest integer k such that there is a PM M in G with $r(M) = k$. Assuming that G admits at least one PM, both $k_{\min}(G)$ and $k_{\max}(G)$ exist.

Karzanov [27] proved that unless a given colored complete or balanced complete bipartite graph (G, c) has a very specific structure, there must be a PM M in G with $r(M) = k$ for all $k_{\min}(G) \leq k \leq k_{\max}(G)$. Moreover, he also characterized the special cases where this is violated. In particular, even in those special cases the following property still holds. (We use the symbol \equiv_2 to denote equivalence modulo 2).

► **Definition 5** (Karzanov's Property). *A colored graph (G, c) satisfies Karzanov's property if for any two PMs M and M' with $r(M) \equiv_2 r(M')$ and any integer k with $r(M) \leq k \leq r(M')$ and $k \equiv_2 r(M) \equiv_2 r(M')$, G admits a PM M'' with $r(M'') = k$.*

In other words, if a graph has Karzanov's property, then if we go in steps of two we always find all possible values of red edges between two given PMs of the same parity. We extend this line of work by proving that all colored chain graphs, unit interval graphs, and complete r -partite graphs satisfy Karzanov's Property too. This allows us to decide EM on these graph classes by using an algorithm for Bounded Correct-Parity Perfect Matching (BCPM) introduced by [13].

Problem BCPM

Input: Colored graph (G, c) , integer $k \geq 0$.

Question: Is there a PM M in G with $r(M) \leq k$ and $r(M) \equiv_2 k$?

We claim that if a graph has Karzanov's property, then EM reduces to BCPM. Indeed, let \mathcal{A} be an algorithm for BCPM. Given a colored graph (G, c) and an integer k , we can call \mathcal{A} with input (G, c) and k to check whether there exists a PM M with $r(M) \leq k$ and $r(M) \equiv_2 k$. Next, assume we compute the inverse coloring \bar{c} with $\bar{c}(e) = \text{red}$ if $c(e) = \text{blue}$, and $\bar{c}(e) = \text{blue}$ if $c(e) = \text{red}$ for all $e \in E$. By calling \mathcal{A} with input (G, \bar{c}) and $k' = \frac{n}{2} - k$, we can check the existence of a PM M with $r(M) \geq k$ and $r(M) \equiv_2 k$ in (G, c) . If we know that Karzanov's property holds in (G, c) , these two pieces of information are sufficient to decide EM. (Note that instead of using the algorithm for BCPM twice, we can first use an algorithm for the simpler problem CPM which is defined similarly to BCPM but without the bound on the number of red edges. Depending on the number of red edges in the output matching we then set the BCPM input appropriately. CPM has been shown to be solvable in deterministic polynomial time on general graphs [13].)

► **Observation 6.** *EM reduces to BCPM in colored graphs that satisfy Karzanov's property.*

33:12 On the Exact Matching Problem in Dense Graphs

Recall that our goal is to give deterministic polynomial-time algorithms for EM in unit interval graphs, chain graphs, and complete r -partite graphs. Observation 6 provides a possible strategy to achieve this. In particular, we will now proceed to give a condition on graphs that implies Karzanov's property. As it turns out, the same condition is also sufficient to give deterministic polynomial-time algorithms for BCPM.

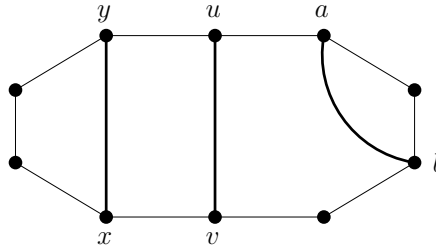
4.1 A Sufficient Condition for Karzanov's Property

We will now present a sufficient condition for Karzanov's property. As it turns out, the condition is also sufficient to give deterministic polynomial-time algorithms for BCPM. Our condition is based on the existence of certain chord structures in all even-length cycles of a given graph. To state it, we first need to introduce some terminology for chords.

► **Definition 7** (Odd Chord, Even Chord, Split of a Chord). *Let C be a cycle in a graph $G = (V, E)$. An edge $e \in E$ is a chord of C if and only if both endpoints of e are on C but $e \notin C$. Let now $e = \{u, v\}$ be a chord of C and consider the paths P_1, P_2 obtained by splitting C at u and v i.e. $C = P_1 \dot{\cup} P_2$. We call e an odd chord of C if and only if either P_1 or P_2 has odd length. Otherwise, e is called an even chord of C . The split of e is the minimum of the lengths of P_1 and P_2 .*

Note that the above definition technically allows C to have even or odd length, but in general we will only be interested in chords of even-length cycles here.

► **Definition 8** (Adjacent Chords). *Let C be a cycle in a graph G with chords $e = \{x, y\} \in E$ and $f = \{u, v\} \in E$ whose endpoints appear on C in the order u, v, x, y . Then e and f are said to be adjacent chords of C if additionally, we have $\{v, x\} \in C$ and $\{u, y\} \in C$.*



■ **Figure 3** An example of a 10-cycle with three chords. The chords $\{x, y\}$ and $\{u, v\}$ are adjacent and they are both odd chords. Conversely, $\{a, b\}$ is an even chord with split 2. The split of $\{x, y\}$ is 3 and the split of $\{u, v\}$ is 5.

An example of these definitions is found in Figure 3. Given these definitions, we are now ready to state our sufficient condition. Note that the condition is only about the graph structure, i.e. the coloring is irrelevant here.

► **Definition 9** (Chord Property). *A simple graph $G = (V, E)$ satisfies the chord property if*

1. *every even cycle C of length at least 6 either has an odd chord or all possible even chords, and*
2. *every even cycle C of length at least 8 either has two adjacent odd chords or all possible even chords with split at least 4.*

As it turns out (see appendix of the full version of this paper), chain graphs, unit interval graphs, and complete r -partite graphs all satisfy the chord property. In fact, the even chords are only needed in complete r -partite graphs. In other words, chain graphs and unit interval

graphs satisfy the chord property without making use of the parts about even chords. By our next lemma, this means that all three graph classes satisfy Karzanov's property for every possible coloring.

► **Lemma 10** (Chord Property is Sufficient). *Let G be an arbitrary graph satisfying the chord property and let c be an arbitrary coloring of G . Then the colored graph (G, c) satisfies Karzanov's property.*

Proof. Deferred to the appendix of the full version of this paper. ◀

Note that the reverse is not true, i.e. given a colored graph (G, c) with Karzanov's property, it is not necessarily the case that G has the chord property.

In order to solve EM on graphs satisfying the chord property, it remains to give a deterministic polynomial-time algorithm for BCPM on those graphs.

► **Lemma 11** (BCPM on Graphs with the Chord Property). *There is a deterministic polynomial-time algorithm that decides BCPM correctly for all inputs where the graph satisfies the chord property.*

Proof. Deferred to the appendix of the full version of this paper. ◀

Finally, we can combine Observation 6 with Lemma 10 and Lemma 11 to get the main result of this section.

► **Theorem 12.** *There is a deterministic polynomial-time algorithm that decides EM on all colored graphs (G, c) where G satisfies the chord property.*

Proof. The colored graph (G, c) satisfies Karzanov's property by Lemma 10. By Observation 6, this reduces deciding EM for (G, c) to deciding BCPM. Moreover, the graph G remains unaltered in this reduction. Hence, this can be achieved in deterministic polynomial-time with the algorithm from Lemma 11. ◀

We prove in the appendix of the full version of this paper that unit interval graphs, chain graphs, and complete r -partite graphs satisfy the chord property. We conclude that EM restricted to those graph classes can be decided in deterministic polynomial-time.

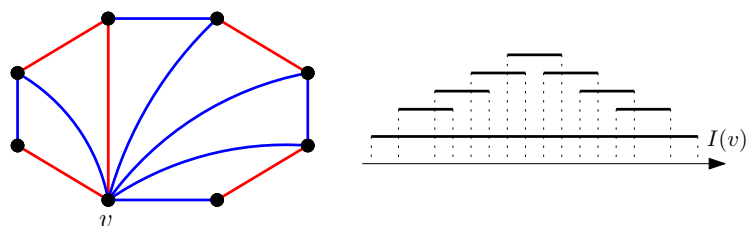
4.2 Limitation of Karzanov's Property

A natural question is whether the approach we presented in this section extends to more graph classes. In particular, interval graphs and bipartite interval graphs would be suitable candidates as they are superclasses of unit interval graphs and chain graphs, respectively.

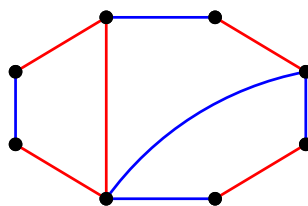
Unfortunately, it turns out that Karzanov's Property is violated on both graph classes. Concrete counterexamples are given in Figures 4 and 5.

While our approach fails to generalize to these graph classes, there still seems to be some hope. Consider the following property of colored graphs, which we coined *Karzanov's weak property*.

► **Definition 13** (Karzanov's Weak Property). *A colored graph (G, c) satisfies Karzanov's weak property if for any two PMs M and M' and integer k with $r(M) \leq k \leq r(M')$, there is a PM M'' with $r(M'') \in \{k, k + 1\}$.*



■ **Figure 4** On the left we have a colored interval graph on eight vertices which does not satisfy Karzanov's property. In particular, there are PMs with 0, 1, 3, and 4 red edges but there is no PM with exactly 2 red edges. The interval representation of the graph is given on the right. Interval $I(v)$ corresponds to vertex v from the left. Note that the vertical position of the intervals is irrelevant, only the relative horizontal position of the intervals matters.



■ **Figure 5** By deleting the even chords from the graph in Figure 4, we obtain a bipartite interval graph. It admits the same PMs as the interval graph in Figure 4. Hence, this colored graph also violates Karzanov's property.

The main difference to Karzanov's property is that we are missing the constraint on the parity of the number of red edges. Consider e.g. a graph with exactly two PMs with 0 and 3 red edges, respectively. Such a graph would satisfy Karzanov's property but violate Karzanov's weak property. In particular, Karzanov's property does not imply Karzanov's weak property. Still, compared to Karzanov's property, Karzanov's weak property gives us less structure to work with and typically holds on larger graph classes. Unfortunately, we have not yet been able to solve EM using Karzanov's weak property.

As it turns out, all colored bipartite chordal and strongly chordal graphs satisfy Karzanov's weak property.

► **Lemma 14.** *All colored bipartite chordal and strongly chordal graphs satisfy Karzanov's weak property.*

Proof. Deferred to the appendix of the full version of this paper. ◀

In particular, the same observation holds in all colored interval and bipartite interval graphs as they are subclasses of strongly chordal and bipartite chordal graphs, respectively.

5 Conclusion

In this paper we made substantial progress towards solving the notoriously difficult Exact Matching problem, in particular in the regime of dense graphs. We provide general frameworks that not only encompass all previously known results for these types of graphs, but also include a multitude of graph classes for which the problem is now solved. We remark that it is inherent to our techniques that they will fail on sparse graphs: It seems very unlikely that a local search on a sparse graph is successful (since changing one edge of a PM in a sparse graph often times requires changing many more edges). It is also unlikely that a sparse graph

has Karzanov's property: On sparse graphs, we do not expect the solution landscape to be dense. Still, we hope that our approach sheds further light onto these questions. One could imagine, for example, to split a graph into a dense and a sparse part, and apply different techniques to different parts.

In this paper, we also provided some open questions that are reasonable to attack next, since they seem to be in reach of current methods. In particular, is it possible to have a deterministic poly-time algorithm for $G(n, 1/2)$? Can one find a deterministic poly-time algorithm for those graph classes where the weak Karzanov property holds? (Interval graphs, bipartite interval graphs, strongly chordal graphs, bipartite chordal graphs.) Note that for graphs with the weak Karzanov property, we can always find a PM with either $k - 1$ or k red edges, but the final decision if k can be achieved still seems difficult.

References

- 1 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of mathematics*, pages 781–793, 2004.
- 2 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- 3 Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1206–1219, 2017.
- 4 Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, and Jacobo Torán. Solving linear equations parameterized by hamming weight. *Algorithmica*, 75(2):322–338, 2016.
- 5 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128(1):355–372, 2011.
- 6 Daniel P. Bovet and Pierluigi Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall International (UK) Ltd., GBR, 1994.
- 7 Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992.
- 8 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An EPTAS for Budgeted Matching and Budgeted Matroid Intersection via Representative Sets. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.
- 9 Anita Dürr, Nicolas El Maalouly, and Lasse Wulf. An approximation algorithm for the exact matching problem in bipartite graphs. *arXiv preprint arXiv:2307.02205*, 2023.
- 10 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.
- 11 Nicolas El Maalouly. Exact matching: Algorithms and related problems. *arXiv preprint arXiv:2203.13899*, 2022.
- 12 Nicolas El Maalouly and Raphael Steiner. Exact Matching in Graphs of Bounded Independence Number. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:14, 2022.
- 13 Nicolas El Maalouly, Raphael Steiner, and Lasse Wulf. Exact matching: Correct parity and FPT parameterized by independence number. *CoRR*, abs/2207.09797, 2022. doi: 10.48550/arXiv.2207.09797.
- 14 Nicolas El Maalouly and Yanheng Wang. Counting perfect matchings in dense graphs is hard. *arXiv preprint arXiv:2210.15014*, 2022.
- 15 Nicolas El Maalouly and Lasse Wulf. Exact matching and the top-k perfect matching problem. *arXiv preprint arXiv:2209.09661*, 2022.

33:16 On the Exact Matching Problem in Dense Graphs

- 16 Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. math. inst. hung. acad. sci.*, 5(1):17–60, 1960.
- 17 Dennis Fischer, Tim A Hartmann, Stefan Lendl, and Gerhard J Woeginger. An investigation of the recoverable robust assignment problem. *arXiv preprint arXiv:2010.11456*, 2020.
- 18 Anna Galluccio and Martin Loebl. On the theory of pfaffian orientations. I. Perfect matchings and permanents. *the electronic journal of combinatorics*, pages R6–R6, 1999.
- 19 Hans-Florian Geerdes and Jácint Szabó. A unified proof for Karzanov’s exact matching theorem. Technical Report QP-2011-02, Egerváry Research Group, Budapest, 2011.
- 20 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.
- 21 Fabrizio Grandoni and Rico Zenklusen. Optimization with more than one budget. *arXiv preprint arXiv:1002.2147*, 2010.
- 22 Rohit Gurjar, Arpita Korwar, Jochen Messner, Simon Straub, and Thomas Thierauf. Planarizing gadgets for perfect matching do not exist. In *International Symposium on Mathematical Foundations of Computer Science*, pages 478–490. Springer, 2012.
- 23 Rohit Gurjar, Arpita Korwar, Jochen Messner, and Thomas Thierauf. Exact perfect matching in complete graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(2):1–20, 2017.
- 24 Russell Impagliazzo and Avi Wigderson. P=BPP if e requires exponential circuits: Derandomizing the xor lemma. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 220–229, 1997.
- 25 Xinrui Jia, Ola Svensson, and Weiqiang Yuan. The exact bipartite matching polytope has exponential extension complexity. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1635–1654. SIAM, 2023.
- 26 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 355–364, 2003.
- 27 AV Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Cybernetics*, 23(1):8–13, 1987.
- 28 Steven Kelk and Georgios Stamoulis. Integrality gaps for colorful matchings. *Discrete Optimization*, 32:73–92, 2019.
- 29 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64:19–37, 2012.
- 30 László Lovász. Matching structure and the matching lattice. *Journal of Combinatorial Theory, Series B*, 43:187–222, 1987.
- 31 Monaldo Mastrolilli and Georgios Stamoulis. Constrained matching problems in bipartite graphs. In *International Symposium on Combinatorial Optimization*, pages 344–355. Springer, 2012.
- 32 Monaldo Mastrolilli and Georgios Stamoulis. Bi-criteria and approximation algorithms for restricted matchings. *Theoretical Computer Science*, 540:115–132, 2014.
- 33 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.
- 34 Martin Nägele, Christian Nöbel, Richard Santiago, and Rico Zenklusen. Advances on strictly δ -modular ips. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 393–407. Springer, 2023.
- 35 Yoshio Okamoto, Ryuhei Uehara, and Takeaki Uno. Counting the number of matchings in chordal and chordal bipartite graph classes. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 296–307. Springer, 2009.
- 36 Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM (JACM)*, 29(2):285–309, 1982.
- 37 Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

- 38 Georgios Stamoulis. Approximation algorithms for bounded color matchings via convex decompositions. In *International Symposium on Mathematical Foundations of Computer Science*, pages 625–636. Springer, 2014.
- 39 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-NC. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 696–707. Ieee, 2017.
- 40 Moshe Y Vardi and Zhiwei Zhang. Quantum-inspired perfect matching under vertex-color constraints. *arXiv preprint arXiv:2209.13063*, 2022.
- 41 Vijay V Vazirani. NC algorithms for computing the number of perfect matchings in k_3 , 3-free graphs and related problems. *Information and computation*, 80(2):152–164, 1989.
- 42 Tongnyoul Yi, Katta G Murty, and Cosimo Spera. Matchings in colored bipartite networks. *Discrete Applied Mathematics*, 121(1-3):261–277, 2002.
- 43 Raphael Yuster. Almost exact matchings. *Algorithmica*, 63(1):39–50, 2012.
- 44 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, pages 216–226. Springer, 1979.

Hardness of Linearly Ordered 4-Colouring of 3-Colourable 3-Uniform Hypergraphs

Marek Filakovský  

Masaryk University, Brno, Czech Republic

Tamio-Vesa Nakajima  

University of Oxford, UK

Jakub Opršal  

University of Birmingham, UK

Gianluca Tasinato  

ISTA, Klosterneuburg, Austria

Uli Wagner  

ISTA, Klosterneuburg, Austria

Abstract

A linearly ordered (LO) k -colouring of a hypergraph is a colouring of its vertices with colours $1, \dots, k$ such that each edge contains a unique maximal colour. Deciding whether an input hypergraph admits LO k -colouring with a fixed number of colours is NP-complete (and in the special case of graphs, LO colouring coincides with the usual graph colouring).

Here, we investigate the complexity of approximating the “linearly ordered chromatic number” of a hypergraph. We prove that the following promise problem is NP-complete: Given a 3-uniform hypergraph, distinguish between the case that it is LO 3-colourable, and the case that it is not even LO 4-colourable. We prove this result by a combination of algebraic, topological, and combinatorial methods, building on and extending a topological approach for studying approximate graph colouring introduced by Krokhin, Opršal, Wrochna, and Živný (2023).

2012 ACM Subject Classification Mathematics of computing \rightarrow Approximation algorithms; Mathematics of computing \rightarrow Algebraic topology; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases constraint satisfaction problem, hypergraph colouring, promise problem, topological methods

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.34

Related Version *Full Version:* <https://arxiv.org/abs/2312.12981> [14]

Funding *Marek Filakovský:* This research was supported by Charles University (project PRIMUS/21/SCI/014), the Austrian Science Fund (FWF project P31312-N35), and MSCAfellow5_MUNI (CZ.02.01.01/00/22_010/0003229).

Tamio-Vesa Nakajima: This research was funded by UKRI EP/X024431/1 and by a Clarendon Fund Scholarship. All data is provided in full in the results section of this paper.

Jakub Opršal: This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No 101034413.

Uli Wagner: This research was supported by the Austrian Science Fund (FWF project P31312-N35).



© Marek Filakovský, Tamio-Vesa Nakajima, Jakub Opršal, Gianluca Tasinato, and Uli Wagner;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 34; pp. 34:1–34:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

Deciding whether a given finite graph is 3-colourable (or, more generally, k -colourable, for a fixed $k \geq 3$) was one of the first problems shown to be NP-complete by Karp [17]. Since then, the complexity of *approximating* the chromatic number of a graph has been studied extensively. The best-known polynomial-time algorithm approximates the chromatic number of an n -vertex graph within a factor of $O(n^{\frac{(\log \log n)^2}{(\log n)^3}})$ (Halldórsson [16]); conversely, it is known that the chromatic number cannot be approximated in polynomial time within a factor of $n^{1-\varepsilon}$, for any fixed $\varepsilon > 0$, unless $P = NP$ (Zuckerman [31]). However, this hardness result only applies to graphs whose chromatic number grows with the number of vertices, and the case of graphs with *bounded* chromatic number is much less well understood.

Given an input graph G that is promised to be 3-colourable, what is the complexity of finding a colouring of G with some larger number $k > 3$ of colours? Khanna, Linial, and Safra [19] showed that this problem is NP-hard for $k = 4$, and it is generally believed that the problem is NP-hard for any constant k . However, surprisingly little is known, and the only improvement and best result to date, hardness for $k = 5$, was obtained only relatively recently by Bulín, Krokhn, and Opršal [9]. On the other hand, the best polynomial-time algorithm, due to Kawarabayashi and Thorup [18], uses a number of colours (slightly less than $n^{1/5}$) that depends on the number n of vertices of the input graph.

More generally, it is a long-standing conjecture that finding a k -colouring of a c -colourable graph is NP-hard for all constants $k \geq c \geq 3$, but the complexity of this *approximate graph colouring* problem remains wide open. The results from [9] generalise to give hardness for $k = 2c - 1$ and all $c \geq 3$. For $c \geq 6$, this was improved by Wrochna and Živný [29], who showed that it is hard to colour c -colourable graphs with $k = \binom{c}{\lfloor c/2 \rfloor}$ colours. We remark that conditional hardness (assuming different variants of Khot's *Unique Games Conjecture*) for approximate graph coloring for all $k \geq c \geq 3$ was obtained by Dinur, Mossel, and Regev [12], Guruswami and Sandeep [15], and Braverman, Khot, Lifshitz, and Mulzer [6].

Given the slow progress on approximate graph colouring, we believe there is substantial value in developing and extending the available methods for studying this problem and related questions, and we hope that the present paper contributes to this effort. As our main result (Theorem 1.1 below), we establish NP-hardness of a relevant hypergraph colouring problem that falls into a general scope of *promise constraint satisfaction problems*; in the process, we considerably extend a topological approach and toolkit for studying approximate colouring that was introduced by Krokhn, Opršal, Wrochna, and Živný [21, 29, 23].

Graph colouring is a special case of the *constraint satisfaction problem (CSP)*, which has several different, but equivalent formulations. For us, the most relevant formulation is in terms of homomorphisms between relational structures. The starting point is the observation that finding a k -colouring of a graph G is the same as finding a *graph homomorphism* (an edge-preserving map) $G \rightarrow K_k$ where K_k is the complete graph with k vertices. The general formulation of the constraint satisfaction problem is then as follows (see Section 2.1 below for more details): Fix a relational structure \mathbf{A} (e.g., a graph, or a uniform hypergraph), which parametrises the problem. $\text{CSP}(\mathbf{A})$ is then the problem of deciding whether a given structure \mathbf{X} allows a homomorphism $\mathbf{X} \rightarrow \mathbf{A}$. One of the celebrated results in the complexity theory of CSPs is the Dichotomy Theorem of Bulatov [8] and Zhuk [30], which asserts that for every finite relational structure \mathbf{A} , $\text{CSP}(\mathbf{A})$ is either NP-complete, or solvable in polynomial time.

The framework of CSPs can be extended to *promise constraint satisfaction problems (PCSPs)*, which include approximate graph colouring. PCSPs were first introduced by Austrin, Guruswami, and Håstad [1], and the general theory of these problems was further

developed by Brakensiek and Guruswami [5], and by Barto, Bulín, Krokhin, and Opršal [3]. Formally, a PCSP is parametrised by two relational structures \mathbf{A} and \mathbf{B} such that there exists a homomorphism $\mathbf{A} \rightarrow \mathbf{B}$. Given an input structure \mathbf{X} , the goal is then to distinguish between the case that there is a homomorphism $\mathbf{X} \rightarrow \mathbf{A}$, and the case that there does not even exist a homomorphism $\mathbf{X} \rightarrow \mathbf{B}$ (these cases are distinct but not necessarily complementary, and no output is required in case neither holds); we denote this decision problem by $\text{PCSP}(\mathbf{A}, \mathbf{B})$. For example, $\text{PCSP}(K_3, K_k)$ is the problem of distinguishing, given an input graph G , between the case that G is 3-colourable and the case that G is not k -colourable. This is the *decision version* of the approximate graph colouring problem whose *search version* we introduced above. We remark that the decision problem reduces to the search version, hence hardness of the former implies hardness of the latter.

PCSPs encapsulate a wide variety of problems, including versions of hypergraph colouring studied by Dinur, Regev, and Smyth [13] and Brakensiek and Guruswami [4]. A variant of hypergraph colouring that is closely connected to approximate graph colouring and generalises (monotone¹) *1-in-3SAT* is *linearly ordered (LO) hypergraph colouring*. A linearly ordered k -colouring of a hypergraph H is an assignment of the colours $[k] = \{1, \dots, k\}$ to the vertices of H such that, for every hyperedge, the maximal colour assigned to elements of that hyperedge occurs exactly once. Note that for graphs, linearly ordered colouring is the same as graph colouring. Moreover, LO 2-colouring of 3-uniform hypergraphs corresponds to (monotone) 1-in-3SAT (by viewing the edges of the hypergraph as clauses). In the present paper, we focus on 3-uniform hypergraphs; whether such a graph has an LO k -colouring can be expressed as $\text{CSP}(\mathbf{LO}_k)$ for a specific relational structure \mathbf{LO}_k with one ternary relation (see Section 2.1); in particular, 1-in-3SAT corresponds to $\text{CSP}(\mathbf{LO}_2)$.

The promise version of LO hypergraph colouring was introduced by Barto, Battistelli, and Berg [2], who studied the *promise 1-in-3SAT* problem. More precisely, let \mathbf{B} be a fixed ternary structure such that there is a homomorphism $\mathbf{LO}_2 \rightarrow \mathbf{B}$. Then $\text{PCSP}(\mathbf{LO}_2, \mathbf{B})$ is the following decision problem: Given an instance \mathbf{X} of 1-in-3SAT, distinguish between the case that \mathbf{X} is satisfiable, and the case that there is no homomorphism $\mathbf{X} \rightarrow \mathbf{B}$. For structures \mathbf{B} with three elements, Barto et al. [2] obtained an almost complete dichotomy; the only remaining unresolved case is $\mathbf{B} = \mathbf{LO}_3$, i.e., the complexity of $\text{PCSP}(\mathbf{LO}_2, \mathbf{LO}_3)$. They conjectured that this problem is NP-hard, and more generally that $\text{PCSP}(\mathbf{LO}_c, \mathbf{LO}_k)$ is NP-hard for all $k \geq c \geq 2$ [2, Conjecture 27]. Subsequently, the following conjecture emerged and circulated as folklore (first formally stated by Nakajima and Živný [27]): $\text{PCSP}(\mathbf{LO}_2, \mathbf{B})$ is either solved by the *affine integer programming relaxation*, or NP-hard (see Ciardo, Kozik, Krokhin, Nakajima, and Živný [10] for recent progress in this direction).

Promise LO hypergraph colouring was further studied by Nakajima and Živný [26], who found close connections between promise LO hypergraph colouring and approximate graph colouring. In particular, they provide a polynomial time algorithm for LO-colouring 2-colourable 3-uniform hypergraphs with a superconstant number of colours, by adapting methods used for similar algorithms for approximate graph colouring, e.g., [18]. In the other direction, NP-hardness of $\text{PCSP}(\mathbf{LO}_k, \mathbf{LO}_c)$ for $4 \leq k \leq c$ follows relatively easily from NP-hardness of the approximate graph colouring $\text{PCSP}(K_{k-1}, K_{c-1})$, as was observed by Nakajima and Živný and by Austrin (personal communications).²

¹ In the present paper, we will only consider the monotone version of 1-in-3SAT, i.e., the case where clauses contain no negated variable, and we will often omit the adjective “monotone” in what follows.

² To see why, observe that $(\mathbf{LO}_k, \mathbf{LO}_c)$ promise primitive-positive defines (K_{k-1}, K_{c-1}) ; in particular, we can define $x \neq y$ by $\exists z \cdot R(z, z, x) \wedge R(z, z, y) \wedge R(x, y, z)$. We then see that if R is interpreted in \mathbf{LO}_k , then the required z exists if and only if $x \neq y$, as required.

Our main result is the following, which cannot be obtained using these arguments.

► **Theorem 1.1.** $\text{PCSP}(\mathbf{LO}_3, \mathbf{LO}_4)$ is NP-complete.

Apart from the intrinsic interest of LO hypergraph colouring, we believe that the main contribution of this paper is on a technical level, by extending the topological approach of [23] and bringing to bear more advanced methods from algebraic topology, in particular *equivariant obstruction theory*. To our knowledge, this paper is the first that uses these methods in the PCSP context; we view this as a “proof of concept” and believe these tools will be useful to make further progress on approximate graph colouring and related problems.

The proof of Theorem 1.1 has two main parts. For a natural number n , let $(\mathbf{LO}_3)^n$ be the n -fold power of the relational structure \mathbf{LO}_3 (see Section 2.2). In the first part of the proof, we use topological methods to show (Lemma 3.2 below) that with every homomorphism $f: (\mathbf{LO}_3)^n \rightarrow \mathbf{LO}_4$, we can associate an *affine map* $\chi(f): \mathbb{Z}_3^n \rightarrow \mathbb{Z}_3$ (i.e., a map of the form $(x_1, \dots, x_n) \mapsto \sum_{i=1}^n \alpha_i x_i$, for some $\alpha_i \in \mathbb{Z}_3$ and $\sum_{i=1}^n \alpha_i \equiv 1 \pmod{3}$); moreover, the assignment $f \mapsto \chi(f)$ preserves natural *minor relations* that arise from maps $\pi: [n] \rightarrow [m]$, i.e., χ is a *minion homomorphism* (see Section 2.2 for the precise definitions).

In the second part of the proof, we show by combinatorial arguments that the maps $\chi(f): \mathbb{Z}_3^n \rightarrow \mathbb{Z}_3$ form a very restricted subclass of affine maps: They are projections $\mathbb{Z}_3^n \rightarrow \mathbb{Z}_3$, $(x_1, \dots, x_n) \mapsto x_i$ (Corollary 3.4). Theorem 1.1 then follows from a hardness criterion (Theorem 2.6) obtained as part of a general algebraic theory of PCSPs [3].

In a nutshell, topology enters in the first part of the proof as follows. First, with every homomorphism $f: (\mathbf{LO}_3)^n \rightarrow \mathbf{LO}_4$ we associate a continuous map $f_*: T^n \rightarrow P^2$, where T^n is the n -dimensional torus (the n -fold power of the circle S^1) and P^2 is a suitable target space that will be described in more detail later; moreover, the cyclic group \mathbb{Z}_3 naturally acts on both T^n and P^2 , and the map f_* preserves these symmetries (it is *equivariant*). This first step uses *homomorphism complexes* (a well-known construction in topological combinatorics that goes back to the work of Lovász [24], see Section 2.3). Second, we show that equivariant continuous maps $T^n \rightarrow P^2$, when considered up to a natural equivalence relation of symmetry-preserving continuous deformation (*equivariant homotopy*), are in bijection with affine maps $\mathbb{Z}_3^n \rightarrow \mathbb{Z}_3$. This second step uses *equivariant obstruction theory*.³

We remark that, with some additional work, our method could be extended to prove NP-hardness of $\text{PCSP}(\mathbf{LO}_k, \mathbf{LO}_{2k-2})$ (but as remarked above, this already follows from known hardness results for approximate graph colouring).

2 Preliminaries

We use the notation $[n]$ for the n -element set $\{1, \dots, n\}$, and identify tuples $a \in A^n$ with functions $a: [n] \rightarrow A$, and we use the notation a_i for the i th entry of a tuple. We denote by 1_X the identity function on a set X .

2.1 Promise CSPs

We start by recalling some fundamental notions from the theory of promise constraint satisfaction problems, following the presentation of [3] and [22].

³ By contrast, the topological argument in [23] required understanding maps from T^n to the circle S^1 that preserve natural \mathbb{Z}_2 -symmetries on both spaces, again up to equivariant homotopy; such maps can be classified by more elementary arguments using the *fundamental group* because S^1 is 1-dimensional.

A *relational structure* is a tuple $\mathbf{A} = (A; R_1^{\mathbf{A}}, \dots, R_k^{\mathbf{A}})$, where A is a set, and $R_i^{\mathbf{A}} \subseteq A^{\text{ar}(R_i)}$ is a relation of arity $\text{ar}(R_i)$. The *signature* of \mathbf{A} is the tuple $(\text{ar}(R_1), \dots, \text{ar}(R_k))$. For two relational structures $\mathbf{A} = (A; R_1^{\mathbf{A}}, \dots, R_k^{\mathbf{A}})$ and $\mathbf{B} = (B; R_1^{\mathbf{B}}, \dots, R_k^{\mathbf{B}})$ with the same signature, a *homomorphism* from \mathbf{A} to \mathbf{B} , denoted $h: \mathbf{A} \rightarrow \mathbf{B}$, is a function $h: A \rightarrow B$ that preserves all relations, i.e., such that $h(a) \in R_i^{\mathbf{B}}$ for each $i \in \{1, \dots, k\}$ and $a \in R_i^{\mathbf{A}}$ where $h(a)$ denotes the componentwise application of h on the elements of a . To express the existence of such a homomorphism, we will also use the notation $\mathbf{A} \rightarrow \mathbf{B}$. The set of all homomorphisms from \mathbf{A} to \mathbf{B} is denoted by $\text{hom}(\mathbf{A}, \mathbf{B})$.

Our focus is on structures with a single ternary relation R , i.e., pairs $(A; R^{\mathbf{A}})$ with $R^{\mathbf{A}} \subseteq A^3$. Moreover, most structures in this paper have a *symmetric* relation, i.e., the relation $R^{\mathbf{A}}$ is invariant under permuting coordinates. Such structures can be also viewed as 3-uniform hypergraphs, keeping in mind that edges of the form (a, a, b) are allowed.

► **Definition 2.1** (Promise CSP). *Fix two relational structures such that $\mathbf{A} \rightarrow \mathbf{B}$. The promise CSP with template \mathbf{A}, \mathbf{B} , denoted by $\text{PCSP}(\mathbf{A}, \mathbf{B})$, is a computational problem that has two versions:*

- *In the search version of the problem, we are given a relational structure \mathbf{X} with the same signature as \mathbf{A} and \mathbf{B} , we are promised that $\mathbf{X} \rightarrow \mathbf{A}$, and we are tasked with finding a homomorphism $h: \mathbf{X} \rightarrow \mathbf{B}$.*
- *In the decision version of the problem, we are given a relational structure \mathbf{X} , and we must answer Yes if $\mathbf{X} \rightarrow \mathbf{A}$, and No if $\mathbf{X} \not\rightarrow \mathbf{B}$. (These cases are mutually exclusive since $\mathbf{A} \rightarrow \mathbf{B}$ and homomorphisms compose.)*

The decision version reduces to the search version; thus for proving the hardness of both versions of problems, it is sufficient to prove the hardness of the decision version of the problem, and in order to prove tractability of both versions, it is enough to provide an efficient algorithm for the search version.

To complete this section, we define the relational structure \mathbf{LO}_k , $k \in \mathbb{N}$, that appears in our main result. The domain of \mathbf{LO}_k is $\{1, \dots, k\}$, and \mathbf{LO}_k has one ternary relation, containing precisely those triples (a, b, c) which contain a *unique maximum*. In other words, $(a, b, c) \in R^{\mathbf{LO}_k}$ if and only if $a = b < c$, $a = c < b$, $b = c < a$, or all three elements a, b, c are distinct. For example, $(1, 1, 2)$ or $(1, 2, 3)$ are triples of the relation of \mathbf{LO}_3 , but not $(2, 2, 1)$.

2.2 Polymorphisms and a hardness condition

Our proof of Theorem 1.1 uses a hardness criterion (Theorem 2.6 below) obtained as part of a general algebraic theory of PCSPs developed in [3], which we will briefly review.

► **Definition 2.2.** *Given a structure \mathbf{A} , we define its n -fold power to be the structure \mathbf{A}^n with the domain A^n and*

$$R_i^{\mathbf{A}^n} = \{(a_1, \dots, a_{\text{ar}(R_i)}) \mid (a_1(i), \dots, a_{\text{ar}(R_i)}(i)) \in R_i^{\mathbf{A}} \text{ for all } i \in [n]\}$$

for each i .

An n -ary polymorphism from a structure \mathbf{A} to a structure \mathbf{B} is a homomorphism from \mathbf{A}^n to \mathbf{B} . We denote the set of all polymorphisms from \mathbf{A} to \mathbf{B} by $\text{pol}(\mathbf{A}, \mathbf{B})$, and the set of all n -ary polymorphisms by $\text{pol}^{(n)}(\mathbf{A}, \mathbf{B})$.⁴

⁴ Untraditionally, we use lowercase notation for polymorphisms to highlight that we are not considering any topology on them contrary to the homomorphism complexes introduced below.

Concretely, in the special case of structures with a ternary relation, a polymorphism is a mapping $f: A^n \rightarrow B$ such that, for all triples $(u_1, v_1, w_1), \dots, (u_n, v_n, w_n) \in R^{\mathbf{A}}$, we have

$$(f(u_1, \dots, u_n), f(v_1, \dots, v_n), f(w_1, \dots, w_n)) \in R^{\mathbf{B}}.$$

Polymorphisms are enough to describe the complexity of a PCSP up to certain log-space reductions. Loosely speaking, the more complex the polymorphisms are, the easier the problem is. We will use a hardness criterion that essentially states that the problem is hard if the polymorphisms have no interesting structure. To define what do we mean by interesting structure, we have to define the notions of *minor*, *minion* and *minion homomorphism*.

► **Definition 2.3.** Fix two sets A and B , and let $f: A^n \rightarrow B$, $\pi: [n] \rightarrow [m]$ be functions. The π -minor of f is the function $g: A^m \rightarrow B$ defined by $g(x) = f(x \circ \pi)$, i.e., such that

$$g(x_1, \dots, x_m) = f(x_{\pi(1)}, \dots, x_{\pi(n)})$$

for all $x_1, \dots, x_m \in A$. We denote the π -minor of f by f^π .

Abstracting from the fact that the polymorphism of any template are closed under taking minors leads to the following notion of (abstract) *minions*:⁵

► **Definition 2.4.** An (abstract) minion \mathcal{M} is a collection of sets $\mathcal{M}^{(n)}$, where $n > 0$ is an integer, and mappings $\pi^{\mathcal{M}}: \mathcal{M}^{(n)} \rightarrow \mathcal{M}^{(m)}$ where $\pi: [n] \rightarrow [m]$ such that $\pi^{\mathcal{M}} \circ \sigma^{\mathcal{M}} = (\pi \circ \sigma)^{\mathcal{M}}$ for each π and σ , and $(1_{[n]})^{\mathcal{M}} = 1_{\mathcal{M}^{(n)}}$.⁶

The polymorphisms of a template \mathbf{A}, \mathbf{B} form a minion \mathcal{M} defined by $\mathcal{M}^{(n)} = \text{pol}^{(n)}(\mathbf{A}, \mathbf{B})$, and $\pi^{\mathcal{M}}(f) = f^\pi$. With a slight abuse of notation, we will use the symbol $\text{pol}(\mathbf{A}, \mathbf{B})$ for this minion. Conversely, if \mathcal{M} is an abstract minion, we will call $\pi^{\mathcal{M}}(f)$ the π -minor of f , and write f^π instead of $\pi^{\mathcal{M}}(f)$.

An important example is the minion of projections denoted by \mathcal{P} . Abstractly, it can be defined by $\mathcal{P}^{(n)} = [n]$ and $\pi^{\mathcal{P}} = \pi$. Equivalently, and perhaps more concretely, \mathcal{P} can also be described as follows: Given a finite set A with at least two elements and integers $i \leq n$, the i -th n -ary *projection* on A is the function $p_i: A^n \rightarrow A$ defined by $p_i(x_1, \dots, x_n) = x_i$. The set of coordinate projections is closed under minors as described above and forms a minion isomorphic to \mathcal{P} . In particular, \mathcal{P} is also isomorphic to the polymorphism minion $\text{pol}(\mathbf{LO}_2, \mathbf{LO}_2)$.

► **Definition 2.5.** A minion homomorphism from a minion \mathcal{M} to a minion \mathcal{N} is a collection of mappings $\xi_n: \mathcal{M}^{(n)} \rightarrow \mathcal{N}^{(n)}$ that preserve taking minors, i.e., such that for each $\pi: [n] \rightarrow [m]$, $\xi_m \circ \pi^{\mathcal{M}} = \pi^{\mathcal{N}} \circ \xi_n$. We denote such a homomorphism simply by $\xi: \mathcal{M} \rightarrow \mathcal{N}$, and write $\xi(f)$ instead of $\xi_n(f)$ when the index is clear from the context.⁷

Using the minion \mathcal{P} , we can now formulate the following hardness criterion (which follows from [3, Theorem 3.1 and Example 2.17] and can also be derived from [3, Corollary 5.2]; see also Section 5.1 of that paper for more details).

► **Theorem 2.6** ([3, corollary of Theorem 3.1]). For every promise template \mathbf{A}, \mathbf{B} such that there is a minion homomorphism $\xi: \text{pol}(\mathbf{A}, \mathbf{B}) \rightarrow \mathcal{P}$, $\text{PCSP}(\mathbf{A}, \mathbf{B})$ is NP-complete.

⁵ Abstract minions as defined here are a generalization of so-called *function minions* defined in [3]; the relation between a function minion and an abstract minion is analogous to the distinction between a permutation group and a group.

⁶ In the language of category theory, a minion is defined as a functor from the category of finite sets to the category of sets, which satisfies a non-triviality condition: $\mathcal{M}(X) = \emptyset$ if and only $X = \emptyset$. The definition given abuses the fact that the sets $[n]$ form a skeleton of the category of finite sets.

⁷ A minion homomorphism is a natural transformation between the two functors.

2.3 Homomorphism complexes

We will need a number of notions from topological combinatorics, which we will review briefly now. We refer the reader to [25] for a detailed and accessible introduction (see also [23], in particular for further background on homomorphism complexes).

A *(finite, abstract) simplicial complex* K is finite system of sets that is downward closed under inclusion, i.e., $F \subseteq G \in K$ implies $F \in K$. The (finite) set $V = \bigcup K$ is called the set of *vertices* of K , and the sets in K are called *simplices* or *faces* of the simplicial complex. A *simplicial map* $f: K \rightarrow L$ between simplicial complexes is a map between the vertex sets that preserves simplices, i.e., $f(F) \in L$ for all $F \in K$.

An important way of constructing simplicial complexes is the following: Let P be a partially ordered set (poset). A *chain* in P is a subset $\{p_0, \dots, p_k\} \subseteq P$ such that $p_0 < p_1 < \dots < p_k$. The set of all chains in P is a simplicial complex, called the *order complex* of P . Note that an order-preserving map between posets naturally induces a simplicial map between the corresponding order complexes.

With every simplicial complex K , one can associate a topological space $|K|$, called the *underlying space* or *geometric realization* of K , as follows: Identify the vertex set of K with a set of points *in general position* in a sufficiently high-dimensional Euclidean space (here, general position means that the points in $F \cup G$ are affinely independent for all $F, G \in K$). Then, in particular, the convex hull $\text{conv}(F)$ is a geometric simplex for every $F \in K$, and the geometric realization can be defined as the union $|K| = \bigcup_{F \in K} \text{conv}(F)$ of these geometric simplices (see, e.g., [25, Lemma 1.6.2]). We also say that the simplicial complex K is a triangulation of the space $|K|$. Every simplicial map $f: K \rightarrow L$ between abstract simplicial complexes induces a continuous map $|f|: |K| \rightarrow |L|$ between their geometric realizations. In what follows, we will often blur the distinction between a simplicial complex and its geometric realization (especially when considering properties that do not depend on a particular triangulation).

Following [25, Section 5.9], we define homomorphism complexes as order complexes of the poset of *multihomomorphisms* from one structure to another.⁸

► **Definition 2.7.** *Suppose \mathbf{A}, \mathbf{B} are relational structures. A multihomomorphism from \mathbf{A} to \mathbf{B} is a function $f: A \rightarrow 2^B \setminus \{\emptyset\}$ such that, for each relational symbol R and all tuples $(u_1, \dots, u_k) \in R^{\mathbf{A}}$, we have that*

$$f(u_1) \times \dots \times f(u_k) \subseteq R^{\mathbf{B}}.$$

We denote the set of all such multihomomorphisms by $\text{mhom}(\mathbf{A}, \mathbf{B})$.

Multihomomorphisms are partially ordered by component-wise comparison, i.e., $f \leq g$ if $f(u) \subseteq g(u)$ for all $u \in A$. They can also be composed in a natural way, i.e., if $f \in \text{mhom}(\mathbf{A}, \mathbf{B})$ and $g \in \text{mhom}(\mathbf{B}, \mathbf{C})$, then $(g \circ f)(a) = \bigcup_{b \in f(a)} g(b)$ is a multihomomorphism from \mathbf{A} to \mathbf{C} .

► **Definition 2.8.** *Let \mathbf{A} and \mathbf{B} be two structures of the same signature. The homomorphism complex $\text{Hom}(\mathbf{A}, \mathbf{B})$ is the order complex of the poset of multihomomorphisms from \mathbf{A} to \mathbf{B} , i.e., the vertices of this simplicial complex are multihomomorphisms from \mathbf{A} to \mathbf{B} , and faces correspond to chains $f_1 < f_2 < \dots < f_k$ of such multihomomorphisms.*

⁸ There are several alternative definitions of homomorphism complexes that lead to topologically equivalent spaces; e.g., the definition given here is the *barycentric subdivision* of the version of the homomorphism complex defined in [23, Definition 3.3].

By the discussion above, every homomorphism $f: \mathbf{A} \rightarrow \mathbf{B}$ induces a simplicial map $f_*: \text{Hom}(\mathbf{C}, \mathbf{A}) \rightarrow \text{Hom}(\mathbf{C}, \mathbf{B})$ between homomorphism complexes, and hence a continuous map between the corresponding spaces (defined on vertices by mapping a multihomomorphism m to the composition $f \circ m$, and then extended linearly).

In the case of graphs, the homomorphism complex $\text{Hom}(K_2, G)$ is commonly used⁹ to study graph colourings, including in [23]. In the present paper, we work instead with the homomorphism complex $\text{Hom}(\mathbf{R}_3, \mathbf{A})$ where \mathbf{R}_3 is the structure with 3 elements and all *rainbow tuples*, i.e., tuples (a, b, c) such that a, b , and c are pairwise distinct; this structure is a hypergraph analogue of the graph K_2 .

Note that a homomorphism $h: \mathbf{R}_3 \rightarrow \mathbf{A}$ can be identified with a triple $(h(1), h(2), h(3)) \in R^{\mathbf{A}}$; conversely, every triple $(a, b, c) \in R^{\mathbf{A}}$ also corresponds to a homomorphism as long as $R^{\mathbf{A}}$ is symmetric. Similarly, a multihomomorphism m can be identified with a triple $(m(1), m(2), m(3))$ of subsets of A such that $m(1) \times m(2) \times m(3) \subseteq R^{\mathbf{A}}$.

2.4 Group actions

Throughout this paper, we will work with actions of the cyclic group \mathbb{Z}_3 on various objects (relational structures, simplicial complexes, topological spaces, groups, etc.) by structure-preserving maps (homomorphisms, simplicial maps, continuous maps, etc.). Thinking of \mathbb{Z}_3 as the multiplicative group with three elements $1, \omega, \omega^2$ (with the understanding that $\omega^i \cdot \omega^j = \omega^{i+j \pmod{3}}$ and $\omega^0 = 1$), such an action is described by describing the action of the generator ω . Thus, specifying the action of \mathbb{Z}_3 on a structure \mathbf{A} amounts to specifying a homomorphism $\omega: \mathbf{A} \rightarrow \mathbf{A}$ such that $\omega^3 = 1_{\mathbf{A}}$ (hence, ω is necessarily an isomorphism; note that we are abusing notation here, writing ω both for the generator of the group and the isomorphism by which it acts). Analogously, an action of \mathbb{Z}_3 on a simplicial complex (or a topological space) is described by specifying a simplicial isomorphism (respectively, a homeomorphism) ω of order 3 from the complex (or space) to itself. We will mostly work with actions that are *free*, which in our special case of \mathbb{Z}_3 -actions simply means that ω has no fixed points.

In particular, consider the action of \mathbb{Z}_3 that acts on \mathbf{R}_3 by cyclically permuting elements. This action induces an action on multihomomorphisms $h: \mathbf{R}_3 \rightarrow \mathbf{A}$ by pre-composition, and this action extends naturally to an action of \mathbb{Z}_3 on $\text{Hom}(\mathbf{R}_3, \mathbf{A})$.¹⁰

It is not hard to show that the action on $\text{Hom}(\mathbf{R}_3, \mathbf{A})$ is free as long as \mathbf{A} has no constant tuples: If a multihomomorphism m is a fixed point of a non-trivial element of \mathbb{Z}_3 , then $m(1) = m(2) = m(3)$, and since $m(1) \neq \emptyset$ and $m(1) \times m(2) \times m(3) \subseteq R^{\mathbf{A}}$ then $R^{\mathbf{A}}$ contains a constant tuple (a, a, a) for any $a \in m(1)$. Consequently, we may observe that the action does not fix any face of the complex.

For every homomorphism $f: \mathbf{A} \rightarrow \mathbf{B}$, the induced simplicial map $f_*: \text{Hom}(\mathbf{R}_3, \mathbf{A}) \rightarrow \text{Hom}(\mathbf{R}_3, \mathbf{B})$ (defined on vertices by mapping multihomomorphism m to the composition $f \circ m$) is *equivariant*; as remarked above, we will often identify f_* with the corresponding continuous map between the underlying spaces.

⁹ Some papers use a different complex, the so-called *box complex*, that leads to a homotopically equivalent (see below) space.

¹⁰ This is analogous to the action of \mathbb{Z}_2 on graph homomorphism complexes $\text{Hom}(K_2, G)$ used, for example, in [23].

2.5 Homotopy

Two continuous maps $f, g: X \rightarrow Y$ between topological spaces are called *homotopic*, denoted $f \sim g$, if there is a continuous map $h: X \times [0, 1] \rightarrow Y$ such that $h(x, 0) = f(x)$ and $h(x, 1) = g(x)$; the map h is called a *homotopy* from f to g . Note that a homotopy can also be thought of as a family of maps $h(\cdot, t): X \rightarrow Y$ that varies continuously with $t \in [0, 1]$. In what follows, X and Y will often be given as simplicial complexes, but we emphasize that we will generally not assume that the maps (or homotopies) between them are simplicial maps. Two spaces X and Y are said to be *homotopy equivalent* if there are continuous maps $f: X \rightarrow Y$ and $g: Y \rightarrow X$ such that $fg \sim 1_Y$ and $gf \sim 1_X$.

These notions naturally generalize to the setting of spaces with group actions. If \mathbb{Z}_3 acts on two spaces X and Y then a continuous map $f: X \rightarrow Y$ is (\mathbb{Z}_3) -*equivariant* if f preserves the action, i.e., $f \circ \omega = \omega \circ f$. Two equivariant maps $f, g: X \rightarrow Y$ are said to be *equivariantly homotopic*, denoted by $f \sim_{\mathbb{Z}_3} g$, if there exists an *equivariant homotopy* between them, i.e., a homotopy $h: X \times [0, 1] \rightarrow Y$ from f to g such that all maps $h(\cdot, t): X \rightarrow Y$ are equivariant. We denote by $[X, Y]_{\mathbb{Z}_3}$ the set of all equivariant homotopy classes of (equivariant) maps from X to Y , i.e.,

$$[X, Y]_{\mathbb{Z}_3} = \{[f] \mid f: X \rightarrow Y \text{ is equivariant}\}$$

where $[f]$ denotes the set of all equivariant maps g such that $f \sim_{\mathbb{Z}_3} g$.

3 Overview of the proof

We give a brief overview of the proof and the core techniques used. The result is proved by a combination of topological, combinatorial, and algebraic methods. In particular, the hardness is provided by analysing the polymorphisms of the template together with a hardness criterion from [3], see Theorem 2.6. In short, our goal is to provide a minion homomorphism from the polymorphism minion $\text{pol}(\mathbf{LO}_3, \mathbf{LO}_4)$ to the minion of projections \mathcal{P} (which is incidentally isomorphic to the polymorphism minion of 3SAT). The core of our contribution is in providing deep-enough understanding of polymorphisms of our template so that the minion homomorphism follows. The proof has two parts: topological and combinatorial.

3.1 Topology

The first part builds on the topological method introduced by Krokhin, Opršal, Wrochna, and Živný [21, 29, 23]. The core idea is that, similarly to approximate graph colouring, there are unreasonably many polymorphisms between the linear-ordering hypergraphs, but most of them are very similar. This means that each polymorphism contains a lot of noise but relatively little information. We use topology to remove this noise, and uncover a signal. This is done by considering the polymorphisms “up to homotopy” – essentially claiming that the homotopy class of a polymorphism carries the information and everything else is noise.

In order to formalise this idea, we consider for each hypergraph \mathbf{A} the topological space $\text{Hom}(\mathbf{R}_3, \mathbf{A})$. Consequently, we get that each the homomorphism $f: \mathbf{A} \rightarrow \mathbf{B}$ induces a continuous map $f_*: \text{Hom}(\mathbf{R}_3, \mathbf{A}) \rightarrow \text{Hom}(\mathbf{R}_3, \mathbf{B})$. Consequently, we identify two homomorphisms f, g if f_* and g_* are homotopic to each other. The same is then extended to polymorphisms, although this requires to overcome a few subtle technical issues. The key observation for this extension is that the power of a homomorphism complex is homotopically equivalent to a homomorphism complex of the corresponding power (see, e.g., [20]).

This general idea requires a refinement to avoid trivial collapses, i.e., we have to avoid the case when f_* is homotopic to a constant map which is always a continuous map between topological spaces. In our case, this is avoided by keeping track of the action of \mathbb{Z}_3 on the spaces $\text{Hom}(\mathbf{R}_3, \mathbf{A})$ and $\text{Hom}(\mathbf{R}_3, \mathbf{B})$ described in the preliminaries. Consequently, we consider maps only up to \mathbb{Z}_3 -equivariant homotopy (note that the map f_* induced by a homomorphism is always equivariant). Further in this exposition, we will silently assume that the action is always present, and all notions are equivariant – the formal proof below is presented with the action in mind.

At this point we sketched how to construct a map that assigns to a polymorphism $f: \mathbf{A}^n \rightarrow \mathbf{B}$, an equivariant continuous map $f_*: \text{Hom}(\mathbf{R}_3, \mathbf{A}^n) \rightarrow \text{Hom}(\mathbf{R}_3, \mathbf{B})$. This map does not necessarily preserve minors, nevertheless, it preserves minors *up to homotopy*, i.e., for each $\pi: [n] \rightarrow [m]$, we have that $(f^\pi)_*$ and $(f_*)^\pi$ are equivariantly homotopic (this is since $\text{Hom}(\mathbf{R}_3, \mathbf{A})^n$ and $\text{Hom}(\mathbf{R}_3, \mathbf{A}^n)$ are only homotopically equivalent and not homeomorphic). This allows us to define a minion homomorphism between the polymorphism minion $\text{pol}(\mathbf{A}, \mathbf{B})$ and the minion of “homotopy classes of continuous maps” from powers of $\text{Hom}(\mathbf{R}_3, \mathbf{A})$ to $\text{Hom}(\mathbf{R}_3, \mathbf{B})$.

► **Definition 3.1.** *Let X and Y be two topological spaces with an action of G . The minion of homotopy classes of equivariant polymorphisms from X to Y is the minion $\text{hpol}(X, Y)$ defined by*

$$\text{hpol}^{(n)}(X, Y) = [X^n, Y]_G$$

and $[f]^\pi = [f^\pi]$.

Note that minors are well-defined in this minion since if f and g are equivariantly homotopic, then so are f^π and g^π for all maps π . Hence, we have a minion homomorphism

$$\zeta: \text{pol}(\mathbf{A}, \mathbf{B}) \rightarrow \text{hpol}(\text{Hom}(\mathbf{R}_3, \mathbf{A}), \text{Hom}(\mathbf{R}_3, \mathbf{B})).$$

This part of the proof follows [23], namely this minion homomorphism can be constructed by following the proof of [23, Lemma 3.22] while substituting \mathbf{R}_3 for \mathbf{K}_2 , and \mathbb{Z}_3 for \mathbb{Z}_2 . We give a general categorical proof in the full version of this paper [14, Appendix C].

In order to describe the minion $\text{hpol}(\text{Hom}(\mathbf{R}_3, \mathbf{A}), \text{Hom}(\mathbf{R}_3, \mathbf{B}))$, we need to classify all homotopy classes of maps between the corresponding topological spaces. The problem of classifying maps between two spaces up to homotopy is well-studied in algebraic topology, although it can be immensely difficult, e.g., maps between spheres of dimensions m and n (i.e., $[S^n, S^m]$) has been classified for many pairs m, n , but the classification for infinitely many remaining cases is still open, and it is considered to be a central open problem in algebraic topology. We take advantage of the topological methods developed to solve these problems. Moreover, we may simplify the matters considerably by replacing the spaces $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_3)$ and $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)$ with spaces that allow equivariant maps to and from, resp., these spaces, and are better behaved from the topological perspective. This is due to the fact that if there are equivariant maps $X' \rightarrow X$ and $Y \rightarrow Y'$, then there is a minion homomorphism

$$\eta: \text{hpol}(X, Y) \rightarrow \text{hpol}(X', Y').$$

This minion homomorphism is defined in the same way as a minion homomorphism from $\text{pol}(\mathbf{A}, \mathbf{B})$ to $\text{pol}(\mathbf{A}', \mathbf{B}')$ if \mathbf{A}', \mathbf{B}' is a *homomorphic relaxation* of \mathbf{A}, \mathbf{B} , i.e., if $\mathbf{A}' \rightarrow \mathbf{A}$ and $\mathbf{B} \rightarrow \mathbf{B}'$ [3, Lemma 4.8(1)]. To substantiate our choice of X' and Y' , let us start with describing some topological properties of the spaces $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_3)$ and $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)$.

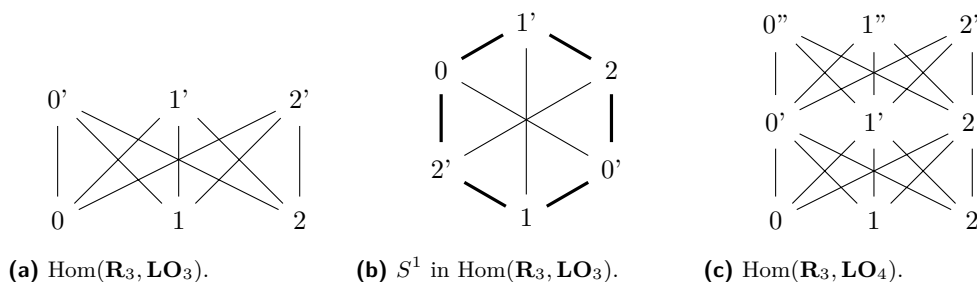


Figure 1 Some representations of spaces $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_3)$ and $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)$ up to \mathbb{Z}_3 homotopy equivalence.

It may be observed that the space $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_3)$ is homotopically equivalent to the simplicial complex depicted in Fig. 1a where the \mathbb{Z}_3 acts on each row cyclically. We choose X' so that $X' \rightarrow \text{Hom}(\mathbf{R}_3, \mathbf{LO}_3)$, and its powers are topologically simple but non-trivial. A natural choice is S^1 which can be obtained from the simplicial complex by removing all horizontal edges. The action of \mathbb{Z}_3 on the circle can be then equivalently described as a rotation by $2\pi/3$. Consequently, the powers of this space are n -dimensional tori T^n with component-wise (diagonal) action of \mathbb{Z}_3 ; by definition T^n is the n -th power of S^1 .

The space $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)$ is a bit more complicated, in particular it is not homotopically equivalent to a 1-dimensional space. Up to equivalence, it is the order complex of the partial order depicted in Fig. 1c where \mathbb{Z}_3 acts on rows cyclically.¹¹ It may be observed that $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)$ is simply connected, i.e., that $\pi_1(\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)) = 0$, and that $\pi_2(\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4))$ is a non-trivial group. Moreover, the action of \mathbb{Z}_3 on $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)$ induces a non-trivial action of \mathbb{Z}_3 on $\pi_2(\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4))$. The precise group and action is described in the full version of this article [14, Appendix A], nevertheless it is irrelevant for us at this point. The space that we use to replace $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)$, and denote by P^2 , shares these two properties with $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)$, and moreover $\pi_n(P^2) = 0$ for all $n > 2$. Spaces which have only one non-trivial homotopy group (and are sufficiently “nice”) are called *Eilenberg-MacLane spaces*, and denoted by $K(G, n)$ where $\pi_n(K(G, n)) = G$ is the only non-trivial homotopy group. These spaces are well-defined up to homotopy equivalence. They are also closely connected with cohomology: One of the core statements of obstruction theory provides a bijection $[X, K(G, n)] \simeq H^n(X; G)$ for each Abelian group G and $n \geq 1$. Consequently, it is much easier to classify maps into an Eilenberg-MacLane space up to homotopy. The space P^2 is in fact an Eilenberg-MacLane space $K(G, 2)$ where G is a suitable group with a free action of \mathbb{Z}_3 , it is chosen in such a way that it allows an equivariant homomorphism $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4) \rightarrow P^2$ while allowing for much easier classification of maps into it.

Next, we prove that the minion $\text{hpol}(S^1, P^2)$ is isomorphic to the minion \mathcal{Z}_3 of affine maps modulo 3, i.e., maps $\mathbb{Z}_3^n \rightarrow \mathbb{Z}_3$ of the form $(x_1, \dots, x_n) \mapsto \sum_{i=1}^n \alpha_i x_i$ where $\alpha_1, \dots, \alpha_n \in \mathbb{Z}_3$ are fixed constants such that $\sum_{i=1}^n \alpha_i = 1 \pmod{3}$. We construct this minion homomorphism by classifying equivariant continuous maps from T^n with the diagonal action of \mathbb{Z}_3 to P^2 . Since we are interested in equivariant maps and equivariant homotopy, we use a version of equivariant cohomology, called *Bredon cohomology*, introduced in [7]. For our purpose, this equivariant cohomology is defined analogously to regular cohomology except the coefficients

¹¹In the proof we will not need such a precise description of the space, and we will only provide an equivariant map $\text{Hom}(\mathbf{R}_3, \mathbf{LO}_4) \rightarrow L_4$ where L_4 is the space represented by the poset in Fig. 1c.

have a \mathbb{Z}_3 -action, and this action together with the action of \mathbb{Z}_3 on the space is taken into account in all computations. The space P^2 has the property that for every \mathbb{Z}_3 -space X such that there is an equivariant map $X \rightarrow P^2$, there is a bijection $[X, P^2]_{\mathbb{Z}_3} \simeq H_{\mathbb{Z}_3}^2(X; G)$ where G is the group with a \mathbb{Z}_3 action described above. Again, this is a consequence of the equivariant obstruction theory. We then compute that

$$H_{\mathbb{Z}_3}^2(T^n; G) \simeq \mathbb{Z}_3^{n-1},$$

and hence observe that there are 3^{n-1} elements in $\text{hpol}^{(n)}(S^1, P^2)$. This means that $\text{hpol}(S^1, P^2)$ and \mathcal{X}_3 have the same number of elements of each arity. To obtain the required minion isomorphism, we provide a minion homomorphism

$$\mathcal{X}_3 \rightarrow [S^1, P^2]_{\mathbb{Z}_3},$$

and show that it is injective. More precisely, this homomorphism is given by assigning to each affine map $f: \mathbb{Z}_3^n \rightarrow \mathbb{Z}_3$ (or a tuple of its coefficients), a continuous map $\mu(f): T^n \rightarrow P^2$, and showing that if $f \neq g$ then $\mu(f)$ and $\mu(g)$ are not equivariantly homotopic, and that $\mu(f^\pi)$ and $\mu(f)^\pi$ are equivariantly homotopic for all π . Since both minions have the same number of elements of each arity (and this number is finite), μ is bijective, and hence a minion isomorphism. All these computations are presented in detail in the full version of this paper [14, Appendix B].

The above isomorphism together with the composition of ζ , η , and ξ provides the following lemma.

► **Lemma 3.2.** *There is a minion homomorphism $\chi: \text{pol}(\mathbf{LO}_3, \mathbf{LO}_4) \rightarrow \mathcal{X}_3$ where \mathcal{X}_3 denotes the minion of affine maps over \mathbb{Z}_3 .*

This minion homomorphism is not enough to prove NP-hardness. Although we could conclude from it, for example, that $\text{PCSP}(\mathbf{LO}_3, \mathbf{LO}_4)$ is not solved by any level of Sherali-Adams hierarchy (this is a direct consequence of [11, Theorems 3.3 and 5.2]). To provide hardness, we need to further analyse the image of χ which is done using combinatorial arguments.

3.2 Combinatorics

In the second part, which is a combinatorial argument presented in Section 4, we show that the image of χ avoids all the affine maps except of projections. This is done by analysing binary polymorphisms from \mathbf{LO}_3 to \mathbf{LO}_4 .

We use the notion of *reconfiguration* of homomorphisms to achieve this. Loosely speaking, a homomorphism f is reconfigurable to a homomorphism g if there is a path of homomorphism starting with f and ending with g such that neighbouring homomorphisms differ in at most one value. (For graphs and hypergraphs without tuples with repeated entries this can be taken as a definition, but with repeated entries there are two sensible notions of reconfigurations that do not necessarily align.) The connection between reconfigurability and topology was described by Wrochna [28], and we use these ideas to connect reconfigurability with our minion homomorphism ξ .

We show that any binary polymorphism $f: \mathbf{LO}_3^2 \rightarrow \mathbf{LO}_4$ is reconfigurable to an essentially unary polymorphism, i.e., that there is an increasing function $h: \mathbf{LO}_3 \rightarrow \mathbf{LO}_4$ such that f is reconfigurable to the map $(x, y) \mapsto h(x)$ or to the map $(x, y) \mapsto h(y)$. Further, we show that if f and g are reconfigurable to each other, then $\chi(f) = \chi(g)$. Together with the above,

this means the image of $\chi_2: \text{hpol}^{(2)}(S^1, P^2) \rightarrow \mathcal{L}_3^{(2)}$ omits an element. More precisely, we have the following lemma where \mathcal{P}_3 denotes the minion of projections on a three element set (which is a subminion of \mathcal{L}_3).

► **Lemma 3.3.** *For each binary polymorphism $f \in \text{pol}^{(2)}(\mathbf{LO}_3, \mathbf{LO}_4)$, $\chi(f) \in \mathcal{P}_3^{(2)}$.*

This lemma is then enough to show that the image of χ omits all affine maps except projections.

► **Corollary 3.4.** *χ is a minion homomorphism $\text{pol}(\mathbf{LO}_3, \mathbf{LO}_4) \rightarrow \mathcal{P}_3$.*

Proof. We show that if a subminion $\mathcal{M} \subseteq \mathcal{L}_3$ contains any non-projection then it contains the map $g: (x, y) \mapsto 2x + 2y$. Let $f \in \mathcal{M}^{(n)}$ depends on at least 2 coordinates, and let $f(x_1, \dots, x_n) = \alpha_1 x_1 + \dots + \alpha_n x_n$. First assume that $\alpha_i = 2$ for some i . Then the binary minor given by $\pi: [n] \rightarrow [2]$ defined by $\pi(i) = 1$ and $\pi(j) = 2$ if $j \neq i$ is g since its first coordinate is 2 and the second is $1 - 2 = 2 \pmod{3}$. Otherwise, we have that $\alpha_i \in \{0, 1\}$ for all i . In particular, there are $i \neq j$ such that $\alpha_i = \alpha_k = 1$ since f depends on at least 2 coordinates. Consequently, the minor defined by $\pi': [n] \rightarrow [2]$ where $\pi'(i) = \pi'(j) = 1$ and $\pi'(k) = 2$ for $k \notin \{i, j\}$ is again g by a similar argument.

Finally, the image of ξ is a subminion of \mathcal{L}_3 , and since it omits g and every subminion of \mathcal{L}_3 contains \mathcal{P}_3 , it is equal to \mathcal{P}_3 which yields the desired. ◀

As mentioned before, the above corollary combined with Theorem 2.6 provides the main result of this paper, the NP-completeness of $\text{PCSP}(\mathbf{LO}_3, \mathbf{LO}_4)$ (Theorem 1.1).

4 Combinatorics of reconfigurations

The goal of this section is a careful combinatorial analysis of the binary polymorphisms. In particular, we will describe how the minion homomorphism $\xi: \text{pol}(\mathbf{LO}_3, \mathbf{LO}_4) \rightarrow \mathcal{P}$ acts on binary polymorphisms. This is the key to the argument that the image of ξ is the projection minion and the whole of $\text{pol}(\mathbb{Z}_3)$.

We say that two polymorphisms $f, g \in \text{pol}^{(n)}(\mathbf{LO}_3, \mathbf{LO}_4)$ are *reconfigurable* one to the other if a path between f and g exists within the homomorphism complex $\text{Hom}(\mathbf{LO}_3^n, \mathbf{LO}_4)$. (Note that every polymorphism is a homomorphism $\mathbf{LO}_3^n \rightarrow \mathbf{LO}_4$, and hence a vertex of the homomorphism complex.)

We will use the following combinatorial criterion that ensures that two polymorphisms are reconfigurable to each other. The proof is subtly dependent on some properties of the structure \mathbf{LO}_4 .

► **Lemma 4.1.** *Let \mathbf{A} be a symmetric relational structure. If $f, g: \mathbf{A} \rightarrow \mathbf{LO}_4$ are two homomorphisms such that f and g differ in exactly one value, i.e., there is $d \in A$ such that for all $a \in A \setminus \{d\}$ we have $f(a) = g(a)$, then f and g are reconfigurable.*

Proof. We first claim that under the above assumption, the multifunction $m: A \rightarrow 2^{[4]}$ given by $m(a) = \{f(a), g(a)\}$ is a multihomomorphism. Assume that $(a, b, c) \in R^{\mathbf{A}}$. Observe that for any $x \in A \setminus \{d\}$ we have $f(x) = g(x)$ and hence $m(x) = \{f(x)\} = \{g(x)\}$. We now have cases depending on how many times d appears in $\{a, b, c\}$.

d does not appear. In this case $m(a) \times m(b) \times m(c) = \{(f(a), f(b), f(c))\} \subseteq R^{\mathbf{LO}_4}$.

d appears once. Suppose $d = a, d \neq b, d \neq c$; then $m(a) \times m(b) \times m(c) = \{f(a), g(a)\} \times \{f(b)\} \times \{f(c)\} = \{(f(a), f(b), f(c)), (g(a), g(b), g(c))\} \subseteq R^{\mathbf{LO}_4}$, as $f(b) = g(b), f(c) = g(c)$.

34:14 Hardness of Linearly Ordered 4-Colouring of 3-Colourable 3-Uniform Hypergraphs

d appears twice. Suppose $d = a = b, d \neq c$; then as $(f(a), f(b), f(c)) = (f(d), f(d), f(c)) \in R^{\mathbf{LO}_4}$ and likewise $(g(d), g(d), g(c)) \in R^{\mathbf{LO}_4}$, we have $f(d) < f(c)$ and $g(d) < g(c) = f(c)$. Consequently, $m(a) \times m(b) \times m(c) = \{f(d), g(d)\}^2 \times \{f(c)\} \subseteq R^{\mathbf{LO}_4}$, since every tuple has a unique maximum, namely $f(c)$.

d appears thrice. This case (i.e., $d = a = b = c$) is impossible, as $\mathbf{A} \rightarrow \mathbf{LO}_4$, and thus \mathbf{A} has no constant tuples.

Thus m is a multihomomorphism in all cases.

We can now define a path $p: [0, 1] \rightarrow \text{Hom}(\mathbf{LO}_3, \mathbf{LO}_4)$ by $p(0) = f, p(1/2) = m, p(1) = g$, and extending linearly. \blacktriangleleft

We note, without a proof, if f and g are reconfigurable, then there is a sequence $f = f_0, \dots, f_k = g$ such that f_i and f_{i+1} differ in exactly one point. A polymorphism $f \in \text{pol}^{(2)}(\mathbf{LO}_3, \mathbf{LO}_4)$ has, as its domain, the set $[3]^2$, and thus it can naturally be represented as a matrix:

$$\begin{array}{ccc} f(1, 1) & f(1, 2) & f(1, 3) \\ f(2, 1) & f(2, 2) & f(2, 3) \\ f(3, 1) & f(3, 2) & f(3, 3) \end{array}.$$

When we speak of “rows” or “columns” of f this is what is meant.

We show the following lemma from which we will be able to derive that each binary polymorphism is reconfigurable to an essentially unary one. (Recall that a function $f: A^n \rightarrow B$ is essentially unary if it depends on at most one input coordinate.) The lemma is an analogue of the *Trash Colour Lemma* for polymorphisms from K_d to K_{2d-2} .

► **Lemma 4.2.** *For each $f \in \text{pol}^{(2)}(\mathbf{LO}_3, \mathbf{LO}_4)$ there exists an increasing function $h \in \text{pol}^{(1)}(\mathbf{LO}_3, \mathbf{LO}_4)$, a coordinate $i \in \{1, 2\}$, and a colour $t \in [4]$ (called trash colour) such that*

$$f(x_1, x_2) \in \{h(x_i), t\}$$

for all $x_1, x_2 \in [3]$.

Proof. Throughout we will implicitly use the fact that if $a < b$ and $c < d$ then $f(a, c) < f(b, d)$, as $((a, c), (a, c), (b, d)) \in R^{\mathbf{LO}_3}$.

First, we claim that every colour $c \in [4]$ appears inside only one row or only one column of f , i.e., that either there is $a \in [3]$ such that $f(x, y) = c$ implies $x = a$, or there is $b \in [3]$ such that $f(x, y) = c$ implies $y = b$. For contradiction, assume that this is not the case, i.e., there are x, y and $x', y' \in [3]$ such that $f(x, y) = f(x', y') = c$, $x \neq x'$, and $y \neq y'$. The claim is proved by case analysis as follows. First, observe that either $x < x'$ and $y > y'$, or $x > x'$ and $y < y'$, since otherwise (x, y) and (x', y') are comparable, and hence $f(x, y) \neq f(x', y')$. Since the two cases are symmetric, we may assume without loss of generality that $x < x'$ and $y > y'$. Furthermore, since $((x, y), (x', y'), (x, y')) \in R^{\mathbf{LO}_3}$, and $f(x, y) = f(x', y') = c$, we have $f(x, y') > c$. Similarly, as $x' > x, y > y'$ we have that $f(x', y) > f(x, y') > c$. This means that $c \in \{1, 2\}$. We consider each case separately.

Case $c = 1$. We claim that $x = y' = 1$ since if $x > 1$, then $f(1, y') < f(x, y) = 1$, and similarly if $y' > 1$. This implies that $f(1, 1) > 1$ since $((1, 1), (x, x'), (y, y')) = ((1, 1), (1, y), (x', 1)) \in R^{\mathbf{LO}_3}$ and $f(x, y) = f(x', y') = 1$. As $1 < f(1, 1) < f(2, 2) < f(3, 3) \leq 4$, we have that $f(1, 1) = 2, f(2, 2) = 3$, and $f(3, 3) = 4$. We now have three cases.

- $y = 3$. We argue that $f(1, 2)$ has no possible value. First, the value 1 is not possible since $((1, 2), (x, y), (x', y')) = ((1, 2), (1, 3), (x', 1)) \in R^{\mathbf{LO}_3^2}$, $f(x, y) = 1$, and $f(x', y') = 1$. $f(1, 2) = 2$ is not possible since $((1, 2), (1, 1), (x', y')) = ((1, 1), (1, 2), (x', 1)) \in R^{\mathbf{LO}_3^2}$, and $f(x', y') = 1, f(1, 1) = 2$. $f(1, 2) = 3$ is not possible since $((1, 2), (2, 2), (x, y)) = ((1, 2), (2, 2), (1, 3)) \in R^{\mathbf{LO}_3^2}$, and $f(x, y) = 1, f(2, 2) = 3$. Finally, $f(1, 2) < f(3, 3) = 4$, so $f(1, 2) \neq 4$.
- $x' = 3$. Here the contradiction follows analogously to the previous case.
- $x' = y = 2$. We consider the pair of values $f(1, 3)$ and $f(3, 1)$. First, we have $f(1, 3) > f(1, 2) = f(x, y) = 1$ and $f(3, 1) > f(2, 1) = f(x', y') = 1$. As $((1, 3), (1, 1), (x', y')) = ((1, 3), (1, 1), (2, 1)) \in R^{\mathbf{LO}_3^2}$ and $f(1, 1) = 2, f(x', y') = 2$ we have that $f(1, 3) \neq 2$; symmetrically $f(3, 1) \neq 2$. We also have $f(1, 3) \neq 3$ since $((1, 3), (x, y), (2, 2)) = ((1, 3), (1, 2), (2, 2)) \in R^{\mathbf{LO}_3^2}$ and $f(1, 2) = 1, f(2, 2) = 3$; symmetrically $f(3, 1) \neq 2$. Thus $f(1, 3) = f(3, 1) = 4$. However, then $(f(1, 2), f(1, 3), f(3, 1)) = (1, 4, 4) \notin R^{\mathbf{LO}_4}$, which is not possible, as $((1, 2), (1, 3), (3, 1)) \in R^{\mathbf{LO}_3^2}$, which yields our contradiction.

Case $c = 2$. As $f(x', y) > f(x, y') > c = 2$, we have that $f(x, y') = 3$ and $f(x', y) = 4$. Since $f(x, y') = 3$ then either $x > 1$ or $y' > 1$, otherwise $f(3, 3) > f(2, 2) > f(1, 1) = 3$ yields a contradiction. By symmetry it is enough to discuss the case $y' = 2$ and $y = 3$. Finally, we have $f(x, 1) < f(x', 2) = 2$, hence $f(x, 1) = 1$ which is in contradiction with

$$(1, 2, 2) = (f(x, 1), f(x', 2), f(x, 3)) \in R^{\mathbf{LO}_4}.$$

Thus we get a contradiction in all cases, and hence each colour appears in only one row or only one column.

We say that a colour $c \in [4]$ is of *column* type if $f(x, y) = c$ implies $x = a_c$ for some fixed $a_c \in [3]$, and is of *row* type if $f(x, y) = c$ implies $y = b_c$ for some $b_c \in [3]$. Note that a colour can be both row and column type, in which case we may choose either. We claim that there are at least three colours that share a type – otherwise there are two colours of row type and two colours of column type which would leave an element of \mathbf{LO}_3^2 uncoloured. A similar observation also yields that there has to be three colours of the same type that cover all rows or all columns, i.e., such that the constants a_c or b_c (depending on the type) are pairwise distinct. Let us assume they are of the column type; the other case is symmetric. Further, we may assume that the fourth colour is of the row type, since if two colours share a column, then one of the colours appears only once, and can be therefore considered to be of row type.

We define $h(a)$ to be the colour c of column type with $a_c = a$, then we have $f(x, y) \in \{h(x), t\}$ where t is the colour of the row type. Finally, we argue that h is increasing. This is since there are $y < y'$ with $y \neq b_t$ and $y' \neq b_t$, and consequently

$$h(1) = f(1, y) < f(2, y') = h(2) = f(2, y) < f(3, y') = h(3).$$

This concludes the proof of the lemma. ◀

► **Lemma 4.3.** *Every binary polymorphism $f \in \text{pol}^{(2)}(\mathbf{LO}_3, \mathbf{LO}_4)$ is reconfigurable to an essentially unary polymorphism.*

Proof. The proof relies on Lemma 4.2. We prove our result by induction on the number of appearances of the trash colour. The result is clear if the trash colour never appears; so assume it appears at least once. Thus suppose without loss of generality that $f(x, y) \in \{h(x), t\}$ for some increasing $h \in \text{pol}^{(1)}(\mathbf{LO}_3, \mathbf{LO}_4)$, and that in particular $f(x_0, y_0) = t$. Furthermore,

suppose that among all such pairs, (x_0, y_0) is the one that maximises x_0 . We claim that $f'(x, y)$, which is equal to $f(x, y)$ everywhere except that $f'(x_0, y_0) = h(x_0)$ is also a polymorphism, which gives us our inductive step.

Consider any $((x, y), (x', y'), (x'', y'')) \in R^{\mathbf{LO}_3^2}$; if $(x_0, y_0) \notin \{(x, y), (x', y'), (x'', y'')\}$, then $(f'(x, y), f'(x', y'), f'(x'', y'')) = (f(x, y), f(x', y'), f(x'', y'')) \in R^{\mathbf{LO}_4}$, so assume without loss of generality that $(x'', y'') = (x_0, y_0)$. We now have two cases, depending on where the unique maximum of $(f(x, y), f(x', y'), f(x_0, y_0)) \in R^{\mathbf{LO}_4}$ falls.

- **$f(x, y)$ is the unique maximum.** In this case, $f(x, y) > f(x_0, y_0) = t$ and $f(x, y) > f(x', y')$. We must show that $f'(x_0, y_0) = h(x_0) \neq f(x, y)$. Since we know that $f(x, y) \neq t$ and thus $f(x, y) = h(x)$, and furthermore that h is increasing, this is the same as showing that $x \neq x_0$. Suppose for contradiction that $x = x_0$; thus $x' > x$. If $f(x', y) = h(x') > h(x)$, then $f(x, y)$ would not be the unique maximum, so $f(x', y) = t$. This contradicts the choice of (x_0, y_0) , as $x' > x_0$.
- **$f(x', y')$ is the unique maximum.** This case is identical to the previous case.
- **$f(x_0, y_0)$ is the unique maximum.** It follows that $f(x, y) < t$ and $f(x', y') < t$, hence $f(x, y) = h(x)$ and $f(x', y') = h(x')$. Thus since $(x, x', x_0) \in R^{\mathbf{LO}_3}$ and h is increasing, it follows that $(f'(x, y), f'(x', y'), f'(x_0, y_0)) = (h(x), h(x'), h(x_0)) \in R^{\mathbf{LO}_4}$.

Thus we see that this f' is indeed a polymorphism, and contains one fewer trash colour. Thus our conclusion follows. ◀

In Figure 2, we can see the reconfiguration graph of $\text{pol}^{(2)}(\mathbf{LO}_3, \mathbf{LO}_4)$. This shows how one can reconfigure all polymorphisms to essentially unary ones. In the diagram, we show a polymorphism in its matrix representation.

It can be also observed that unary polymorphisms that depend on the same coordinate are reconfigurable to each other. Moreover, since every connected component of $\text{Hom}(\mathbf{LO}_3^2, \mathbf{LO}_4)$ contains a homomorphism, and hence a unary one, we can derive from these observation that $\text{Hom}(\mathbf{LO}_3^2, \mathbf{LO}_4)$ has at most two connected components. In the full version [14, Appendix B], we also prove that it has at least two components using topological methods.

Finally, we conclude with the statement that we actually use in the proof, which follows from well-known properties of homomorphism complexes.

► **Lemma 4.4.** *Let \mathbf{A} , \mathbf{B} , and \mathbf{C} be three structures, G a group acting on \mathbf{A} , and assume that $f, g \in \text{hom}(\mathbf{B}, \mathbf{C})$ are reconfigurable. Then the induced maps $f_*, g_* : \text{Hom}(\mathbf{A}, \mathbf{B}) \rightarrow \text{Hom}(\mathbf{A}, \mathbf{C})$ are G -homotopic.*

Proof. First, observe that the composition of multihomomorphisms as a map $\text{mhom}(\mathbf{A}, \mathbf{B}) \rightarrow \text{mhom}(\mathbf{B}, \mathbf{C}) \rightarrow \text{mhom}(\mathbf{A}, \mathbf{C})$ is monotone. This means that the composition extends linearly to a continuous map

$$c : \text{Hom}(\mathbf{B}, \mathbf{C}) \times \text{Hom}(\mathbf{A}, \mathbf{B}) \rightarrow \text{Hom}(\mathbf{A}, \mathbf{C})$$

(see also [20, Section 18.4.3]). Since the composition is associative, we obtain that the map c is equivariant (under an action of any automorphism of \mathbf{A} on the second coordinate).

Finally, we have that $f_*(x) = c(f, x)$ by the definition of f_* , and analogously, $g_*(x) = c(g, x)$. Consequently, if $h : [0, 1] \rightarrow \text{Hom}(\mathbf{B}, \mathbf{C})$ is an arc connecting f and g , i.e., such that $h(0) = f$ and $h(1) = g$, then the map $H : [0, 1] \times \text{Hom}(\mathbf{A}, \mathbf{B}) \rightarrow \text{Hom}(\mathbf{A}, \mathbf{C})$ defined by

$$H(t, x) = c(h(t), x)$$

is a homotopy between f_* and g_* . This H is also equivariant since c is equivariant. ◀

The following corollary then follows directly from the above and Lemma 4.3.

► **Corollary 4.5.** *For every binary polymorphism $f \in \text{pol}^{(2)}(\mathbf{LO}_3, \mathbf{LO}_4)$, the induced map $f_*: \text{Hom}(\mathbf{R}_3, \mathbf{LO}_3)^2 \rightarrow \text{Hom}(\mathbf{R}_3, \mathbf{LO}_4)$ is equivariantly homotopic either to the map $(x, y) \mapsto i_*(x)$, or to the map $(x, y) \mapsto i_*(y)$ where $i: \mathbf{LO}_3 \rightarrow \mathbf{LO}_4$ is the inclusion.*

References

- 1 Per Austrin, Venkatesan Guruswami, and Johan Håstad. $(2+\epsilon)$ -Sat is NP-hard. *SIAM Journal on Computing*, 46(5):1554–1573, 2017. doi:10.1137/15M1006507.
- 2 Libor Barto, Diego Battistelli, and Kevin M. Berg. Symmetric promise constraint satisfaction problems: Beyond the Boolean case. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16–19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPICs*, pages 10:1–10:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.STACS.2021.10.
- 3 Libor Barto, Jakub Bulín, Andrei Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. *J. ACM*, 68(4):28:1–66, 2021. doi:10.1145/3457606.
- 4 Joshua Brakensiek and Venkatesan Guruswami. New hardness results for graph and hypergraph colorings. In Ran Raz, editor, *Conference on Computational Complexity (CCC 2016)*, volume 50 of *LIPICs*, pages 14:1–14:27, Dagstuhl, Germany, 2016. Schloss Dagstuhl–LZI. doi:10.4230/LIPICs.CCC.2016.14.
- 5 Joshua Brakensiek and Venkatesan Guruswami. Promise constraint satisfaction: Algebraic structure and a symmetric Boolean dichotomy. *SIAM Journal on Computing*, 50(6):1663–1700, 2021. doi:10.1137/19M128212X.
- 6 Mark Braverman, Subhash Khot, Noam Lifshitz, and Dor Minzer. An invariance principle for the multi-slice, with applications. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science—FOCS 2021*, pages 228–236. IEEE Computer Soc., Los Alamitos, CA, 2022. doi:10.1109/FOCS52979.2021.00030.
- 7 Glen E. Bredon. *Equivariant Cohomology Theories*, volume 34 of *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 1967. doi:10.1007/BFb0082690.
- 8 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, Berkeley, CA, USA, 2017. IEEE. doi:10.1109/FOCS.2017.37.
- 9 Jakub Bulín, Andrei Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. In *Proc. of the 51st Annual ACM-SIGACT Symposium on the Theory of Computing, STOC 2019*, pages 602–613, New York, USA, 2019. ACM. doi:10.1145/3313276.3316300.
- 10 Lorenzo Ciardo, Marcin Kozik, Andrei A. Krokhin, Tamio-Vesa Nakajima, and Stanislav Živný. On the complexity of the approximate hypergraph homomorphism problem, 2023. doi:10.48550/arXiv.2302.03456.
- 11 Víctor Dalmau and Jakub Opršal. Local consistency as a reduction between constraint satisfaction problems, 2023. doi:10.48550/arXiv.2301.05084.
- 12 Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. *SIAM J. Comput.*, 39(3):843–873, 2009. doi:10.1137/07068062X.
- 13 Irit Dinur, Oded Regev, and Clifford Smyth. The hardness of 3-uniform hypergraph coloring. *Combinatorica*, 25(5):519–535, 2005. doi:10.1007/s00493-005-0032-4.
- 14 Marek Filakovský, Tamio-Vesa Nakajima, Jakub Opršal, Gianluca Tasinato, and Uli Wagner. Hardness of linearly ordered 4-colouring of 3-colourable 3-uniform hypergraphs, 2023. doi:10.48550/arXiv.2312.12981.
- 15 Venkatesan Guruswami and Sai Sandeep. d -to-1 hardness of coloring 3-colorable graphs with $O(1)$ colors. In *47th International Colloquium on Automata, Languages, and Programming*, volume 168 of *LIPICs. Leibniz Int. Proc. Inform.*, pages Art. No. 62, 12. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2020. doi:10.4230/LIPICs.ICALP.2020.62.

- 16 Magnús M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Inform. Process. Lett.*, 45(1):19–23, 1993. doi:10.1016/0020-0190(93)90246-6.
- 17 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, Mar 20–22, 1972, Yorktown Heights, New York*, pages 85–103, Boston, MA, 1972. Springer US. doi:10.1007/978-1-4684-2001-2_9.
- 18 Ken-ichi Kawarabayashi and Mikkel Thorup. Coloring 3-colorable graphs with less than $n^{1/5}$ colors. *J. ACM*, 64(1):4:1–4:23, 2017. doi:10.1145/3001582.
- 19 Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000. doi:10.1007/s004930070013.
- 20 Dmitry Kozlov. *Combinatorial algebraic topology*, volume 21 of *Algorithms and Computation in Mathematics*. Springer, 2008. doi:10.1007/978-3-540-71962-5.
- 21 Andrei Krokhin and Jakub Opršal. The complexity of 3-colouring H -colourable graphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS'19)*, pages 1227–1239, Baltimore, MD, USA, 2019. IEEE. doi:10.1109/FOCS.2019.00076.
- 22 Andrei Krokhin and Jakub Opršal. An invitation to the promise constraint satisfaction problem. *ACM SIGLOG News*, 9(3):30–59, 2022. doi:10.1145/3559736.3559740.
- 23 Andrei A. Krokhin, Jakub Opršal, Marcin Wrochna, and Stanislav Živný. Topology and adjunction in promise constraint satisfaction. *SIAM Journal on Computing*, 52(1):38–79, 2023. doi:10.1137/20M1378223.
- 24 László Lovász. Kneser’s conjecture, chromatic number, and homotopy. *J. Combin. Theory Ser. A*, 25(3):319–324, 1978. doi:10.1016/0097-3165(78)90022-5.
- 25 Jiří Matoušek. *Using the Borsuk-Ulam Theorem. Lectures on Topological Methods in Combinatorics and Geometry*. Springer-Verlag Berlin Heidelberg, first edition, 2003. doi:10.1007/978-3-540-76649-0.
- 26 Tamio-Vesa Nakajima and Stanislav Živný. Linearly ordered colourings of hypergraphs. *ACM Trans. Comput. Theory*, 14(3–4), February 2023. doi:10.1145/3570909.
- 27 Tamio-Vesa Nakajima and Stanislav Živný. Boolean symmetric vs. functional PCSP dichotomy. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2023. doi:10.1109/LICS56636.2023.10175746.
- 28 Marcin Wrochna. Homomorphism reconfiguration via homotopy. *SIAM J. Discret. Math.*, 34(1):328–350, 2020. doi:10.1137/17M1122578.
- 29 Marcin Wrochna and Stanislav Živný. Improved hardness for H -colourings of G -colourable graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’20)*, pages 1426–1435, Salt Lake City, UT, USA, 2020. SIAM. doi:10.1137/1.9781611975994.86.
- 30 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020. doi:10.1145/3402029.
- 31 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory Comput.*, 3:103–128, 2007. doi:10.4086/toc.2007.v003a006.

A Graph of reconfigurations of binary polymorphisms

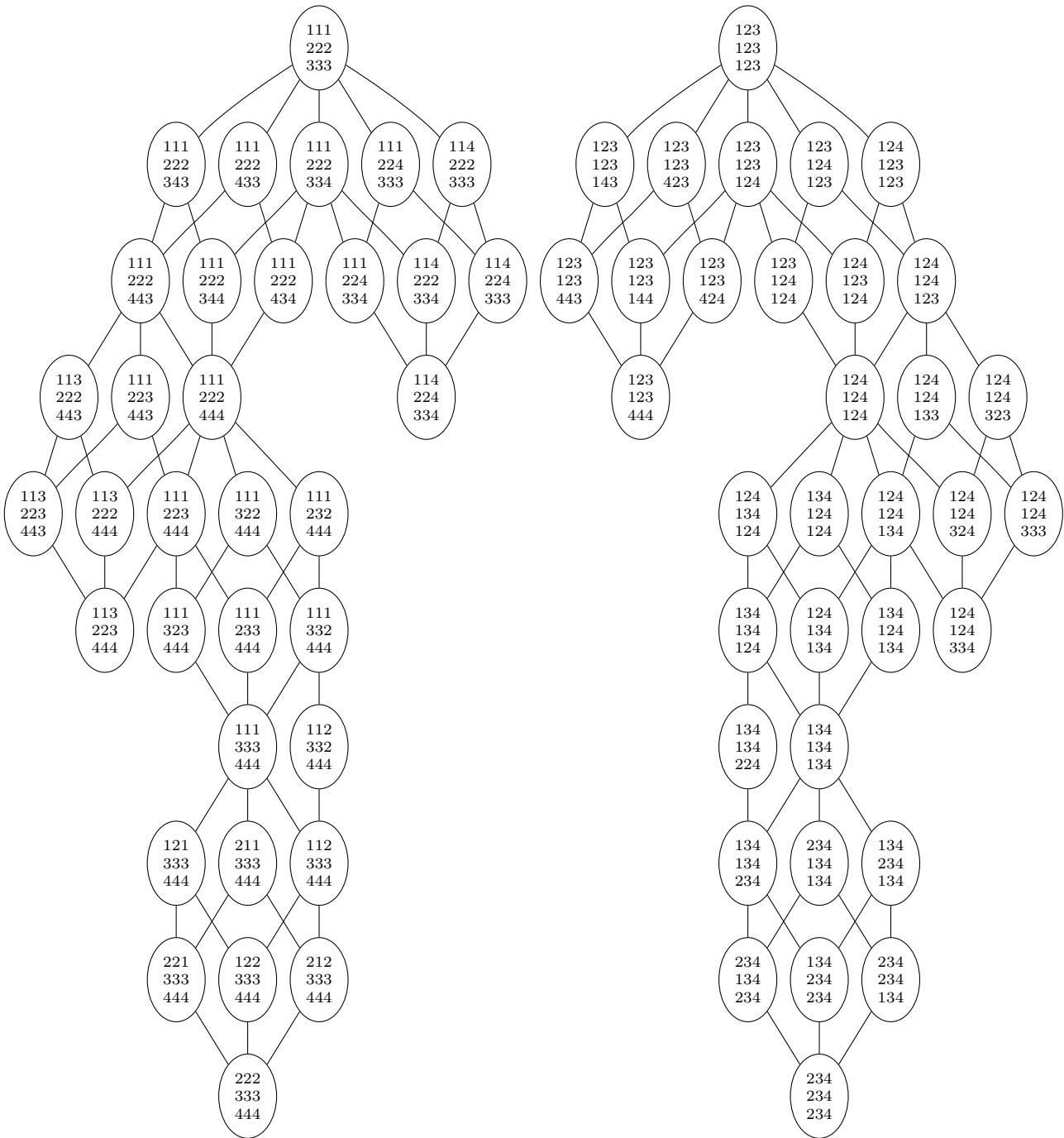





Figure 2 Graph of reconfigurations of $\text{pol}^{(2)}(\mathbf{LO}_3, \mathbf{LO}_4)$.

The 2-Attractor Problem Is NP-Complete

Janosch Fuchs  

RWTH Aachen University, Germany

Philip Whittington¹  

ETH Zürich, Switzerland

Abstract

A k -attractor is a combinatorial object unifying dictionary-based compression. It allows to compare the repetitiveness measures of different dictionary compressors such as Lempel-Ziv 77, the Burrows-Wheeler transform, straight line programs and macro schemes. For a string $T \in \Sigma^n$, the k -attractor is defined as a set of positions $\Gamma \subseteq [1, n]$, such that every distinct substring of length at most k is covered by at least one of the selected positions. Thus, if a substring occurs multiple times in T , one position suffices to cover it. A 1-attractor is easily computed in linear time, while Kempa and Prezza [STOC 2018] have shown that for $k \geq 3$, it is NP-complete to compute the smallest k -attractor by a reduction from k -set cover.

The main result of this paper answers the open question for the complexity of the 2-attractor problem, showing that the problem remains NP-complete. Kempa and Prezza's proof for $k \geq 3$ also reduces the 2-attractor problem to the 2-set cover problem, which is equivalent to edge cover, but that does not fully capture the complexity of the 2-attractor problem. For this reason, we extend edge cover by a color function on the edges, yielding the colorful edge cover problem. Any edge cover must then satisfy the additional constraint that each color is represented. This extension raises the complexity such that colorful edge cover becomes NP-complete while also more precisely modeling the 2-attractor problem. We obtain a reduction showing k -attractor to be NP-complete and APX-hard for any $k \geq 2$.

2012 ACM Subject Classification Theory of computation \rightarrow Problems, reductions and completeness; Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Data compression

Keywords and phrases String attractors, dictionary compression, computational complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.35

1 Introduction

Compressing text without losing information is usually achieved by exploiting structural properties of the text. For example, dictionary-based compression works by removing redundancy resulting from repeatedly occurring substrings. Thus, any measurement capturing the repetitiveness of strings is directly related to the performance of dictionary compression techniques.

In fact, Kempa and Prezza show in [16] that the solutions of famous compression algorithms, like Lempel-Ziv 77, the Burrows-Wheeler transform or straight-line programs, are approximations of a certain measurement for repetitiveness, the so called *string attractor*, which was introduced in [23]. These results are extended by Kempa and Saha [17] to include the LZ-End compression algorithm proposed by Kreft and Navarro [19], and Kempa and Kociumaka [14] apply string attractors to resolve the Burrows-Wheeler transform conjecture.

Further, Kempa and Prezza [16] build a universal data structure based on string attractors supporting random-access on any dictionary compression scheme. In [22], Navarro and Prezza improve data access when a string attractor is known, showing that the compression-based

¹ Parts of this work were produced as part of the author's Master's thesis at RWTH Aachen University.



string attractors suffice to support fast indexed queries, that is, searching for all occurrences of a given pattern in a text. Thus, string attractors are not only the basis of dictionary compression, they also allow for a universal indexing data structure that works on top of every dictionary compression scheme. Christiansen et al. continue this work in [8] by constructing an efficient indexing algorithm that is based on the underlying string attractor, but does not need to explicitly compute it, achieving optimal time results for locating and counting indices.

For a string of length n , an attractor is a set of positions $\Gamma \subseteq [1, n]$ covering all distinct substrings, that is, every distinct substring has an occurrence crossing at least one of the selected positions. If only distinct substrings up to a certain length k need to be covered, we speak of the k -attractor. As an example, for the string

$$T = \underline{a}bb\underline{c}ab\underline{c}c\underline{a}c,$$

position 2 covers the substrings b , ab and bb , and the set of markings $\Gamma = \{2, 7, 9\}$ forms a 2-attractor for T . It is not a valid 3-attractor and therefore not an attractor because the substrings bca and cab are not covered. Adding either 4 or 5 to Γ results in an attractor, and there is no smaller attractor for T .

In [16] Kempa and Prezza also show that computing the smallest k -attractor is NP-complete for any $k \geq 3$, by giving a reduction from k -set cover, and extend this proof to non-constant k , especially for $k = n$. On the other hand, the problem is trivially solvable in polynomial time for $k = 1$ by a greedy algorithm. The complexity for $k = 2$ was raised as an open problem.

Variants of the problem have been introduced, such as the sharp k -attractor in [15], which only considers distinct substrings of length exactly k , and the circular attractor in [20], which also requires to cover circular substrings. Interestingly, the sharp k -attractor problem can also be reduced from and to k -set cover and is therefore NP-complete for $k \geq 3$. However, the sharp 2-attractor problem reduces to 2-set cover, which is equivalent to edge cover and therefore solvable in polynomial time. Notably, the sharp variant does not exhibit the same gap as the k -attractor problem.

Mantaci et. al [20] take a combinatorial approach and study the attractor sizes of infinite families of words, such as Sturmian words, Thue-Morse words and de Bruijn words. This lead to Schaeffer and Shallit [25] raising the definition of the string attractor problem to infinite words by considering automatic sequences and computing attractors of every finite prefix. Further work on attractors of infinite words, especially those generated by morphisms, has been done by Restivo, Romana, and Sciortino [24], Gheeraert, Romana, Stipulanti [12], and Dvořáková [11].

Akagi, Funakoshi, Inenaga [1] analyze the sensitivity of an attractor, i.e., how much can editing a single position change the result. Bannai et. al [3] formulate the attractor and other dictionary compressors as instances of the maximum satisfiability problem and present computational studies showing that an attractor can be computed in reasonable time with this approach.

A more recent development in the field of data compression is the *relative substring complexity* measure δ [18], which counts the number of different substrings of length l and scales it by l . It is efficient to compute and also smaller than the size of the optimal string attractor by up to a logarithmic factor, but allows to use the results on string attractors with that overhead.

Our result is obtained by a technique in which we make a problem *colorful* by adding colors to its edges (or vertices) and requiring all colors to appear in a solution. This idea or similar versions of it are spread out throughout the literature and thus also known by other names such as *labeled*, *rainbow*, *tropical*, or *color-spanning*. It is mostly used in relation to paths in the graph [4, 10, 2] and matching problems [21, 7, 9, 5].

Our Contributions

The main result of this paper answers the open question for the complexity of the 2-attractor problem, showing that the problem remains NP-complete and thus closing the last remaining gap in the complexity analysis of the k -attractor problem.

We introduce a more general version of the k -attractor, where the input is a set of strings and each distinct substring only needs to be covered by an attractor position in at least one of the input strings. We call this a k -set attractor and show that it can be simulated by a single k -attractor, showing the equivalence of the two problems. The ability to construct multiple strings as input makes the proof of the main result more convenient.

Kempa and Prezza [16] give reductions for k -attractor from k -set cover and to $k(k+1)/2$ -set cover, placing 2-attractor between 2-set cover and 3-set cover. A k -set cover reduction is also used in [15] to show that sharp 2-attractor is solvable in polynomial time. Essentially, both the k -attractor and the less restrictive sharp k -attractor problem are reduced from k -set cover, indicating that some of the complexity of the k -attractor is lost in the process. For this reason, we further investigate the relation between 2-attractor and 2-set cover problem, which is equivalent to the edge cover problem.

Thus, we extend the edge cover problem with a color function on the edges, yielding the colorful edge cover problem. Any edge cover must then satisfy the additional constraint that each color is represented. The complexity of edge cover combined with the additional complexity from the colorfulness condition raises the complexity of colorful edge cover to NP-complete, although the problems each on their own are solvable in polynomial time. The hardness is shown by a reduction from a variable-bounded SAT variant. The colors are used to model a truth assignment of the variables and the edge cover condition verifies that this assignment is satisfying. The colorful edge cover problem captures the constraints of the 2-attractor problem more tightly. We later use this result to extend the structure of the reduction to the k -attractor problem. With the reduction we not only show the NP-hardness of the 2-attractor problem, we also obtain an APX-hardness result.

Our paper is organized as follows, we introduce in Section 2 the set of strings attractor problem and discuss its relation to the already introduced variations, showing that it is as hard as the classical k -string attractor problem. In Section 3 we define the colorful edge cover problem and show that it is NP-complete. The main result, the NP-completeness of the 2-attractor problem, is presented in Section 4. Afterwards, in Section 5, we discuss the implicit APX-hardness that results from our reduction.

2 Attractors of Strings, Circular Strings, and Sets of Strings

We start with the definition of the k -attractor and explain already introduced variations before we define the k -set attractor. Afterwards, we discuss how to solve the corresponding problems in polynomial time, if there exists an algorithm that solves one of the problems in polynomial time. Thus, we show that the problems are equivalent in their complexity.

► **Definition 1** (*k*-attractor [16]). A set $\Gamma \subseteq [1, n]$ is a *k*-attractor of a string $T \in \Sigma^n$ if every substring $T[i \dots j]$ such that $i \leq j < i + k$ has an occurrence $T[i' \dots j']$ with $j'' \in [i' \dots j']$ for some $j'' \in \Gamma$.

A solution Γ is called a *string attractor* or simply *attractor* if $k = n$. The corresponding optimization and decision problems are the minimum-*k*-attractor problem and the *k*-attractor problem.

For the circular *k*-string attractor problem, the input string is circular, resulting in additional substrings starting at the last letters of the input string and continuing at the beginning. Thus, there are more substrings that need to be covered compared to the *k*-string attractor problem. However, these additional substrings can make the solution smaller.

It is convenient for the proof of Theorem 9 to allow *k*-attractors over multiple strings. This does not change the problem much, in fact *k*-set attractors can be easily simulated by unique delimiter symbols. We formally define a set attractor over m strings of possibly different lengths n_1, \dots, n_m as a set of tuples, with the first entry of the tuple denoting the string and the second entry denoting the position.

► **Definition 2** (*k*-set attractor). A set $\Gamma \subseteq \bigcup_{x=1}^m \bigcup_{y=1}^{n_x} \{(x, y)\}$ is a *k*-set attractor of a set of m strings $\mathbf{T} = \{T_1, \dots, T_m\}$ with $T_x \in \Sigma^{n_x}$ if every substring $T_x[i \dots j]$ such that $i \leq j < i + k$ has an occurrence $T_{x'}[i' \dots j']$ with $j'' \in [i' \dots j']$ for some $(x', j'') \in \Gamma$.

A string is a circular string cut once, applying more cuts gives a set of strings. Therefore, if we can describe the behavior of attractors under cuts, we can show those three to be the same. The equivalence of circular *k*-attractors and *k*-attractors is already shown in [20]. We extend the idea of adding a delimiter, which must be part of any solution, to remove the impact of circularity and thereby acting as a cut, i.e., substrings that stretch over the delimiter are already covered. The remaining substrings are then the same as the substrings of two separate strings.

► **Lemma 3.** An algorithm solving the *k*-attractor problem can solve the *k*-set-attractor problem with linear overhead in the input size, and vice versa.

Proof. Given a set of strings $\mathbf{T} = \{T_1, \dots, T_m\}$ over an alphabet Σ , create $m - 1$ new symbols $\#_1, \dots, \#_{m-1} \notin \Sigma$ and use them to stitch the strings together as $T = T_1\#_1T_2\#_2 \dots \#_{m-1}T_m$. Then, T has a *k*-attractor of size $p + m - 1$ if and only if \mathbf{T} has a *k*-set attractor of size p , because any attractor for T has to mark each position of the unique delimiter symbols. The remaining markings induce a *k*-set attractor for \mathbf{T} , and a *k*-set attractor combined with those markings yields a *k*-attractor for T .

For the other direction, given a string T just use the singleton $\{T\}$ as input for *k*-set attractor problem. ◀

Table 1 shows the combined results of [20] and our proof. Given an input string T , T^* or \mathbf{T} and solution size p for the *k*-attractor, circular *k*-attractor or *k*-set attractor problem and a function solving one of the problems, potentially a different one, the corresponding entry describes how to modify the input to decide the problem corresponding to the input with the given function. Transforming a circular string into a set of strings or vice versa is done using strings as an intermediate step. A variant of the problem on a set of circular strings is also equivalent by transforming each circular string into a string with the given operations, obtaining a set of strings which can be further transformed as desired.

■ **Table 1** Equivalence of different attractor problems.

input	$\langle T, p \rangle$	$\langle T^*, p \rangle$	$\langle \mathbf{T}, p \rangle$
string	$\langle T, p \rangle$	$\langle T^*T^*T^*, p \rangle$	$\langle T_1\#_1T_2 \dots \#_{m-1}T_m, p+m-1 \rangle$
circular	$\langle T\#, p+1 \rangle$	$\langle T^*, p \rangle$	$\langle T_1\#_1T_2 \dots \#_{m-1}T_m\#_m, p+m \rangle$
set	$\langle \{T\}, p \rangle$	$\langle \{T^*T^*T^*\}, p \rangle$	$\langle \mathbf{T}, p \rangle$

3 The Colorful Edge Cover Problem

The key idea behind colorfulness is to extend a problem in P, i.e. edge cover, by a color function on the set of solution elements to raise the complexity to NP-complete. Any solution to the initial problem must then satisfy the additional condition that each color is represented. The colors can be used to model guessing a certificate for an NP-complete problem, and the structure of the initial problem is used to verify that certificate.

We define an edge coloring on a set of colors \mathcal{C} as a surjective function $\text{col} : E \rightarrow \mathcal{C}$. It is required that col is surjective to avoid trivially unsolvable instances. Note that this is different from the edge coloring problem where the colors are subject to the constraint that no two edges of the same color are adjacent. The set of edges E is allowed to contain self-loops, i.e., an edge of the form $\{v, v\}$. Normally, self-loops are not of interest for the edge cover problem, because they only cover one vertex, making every other adjacent edge more desirable. However, the additional colorfulness constraint can make the self-loops necessary as part of an optimal solution.

► **Definition 4** (Colorful Edge Cover). *For an undirected edge-colored graph $G = (V, E, \text{col})$, with $\text{col} : E \rightarrow \mathcal{C}$, a subset $E' \subseteq E$ is called a colorful edge cover of G , if for each vertex $v \in V$ there is an edge $\{v, w\} \in E'$, and for each color $c \in \mathcal{C}$ there is an edge $e \in E'$ with $\text{col}(e) = c$.*

By minimum colorful edge cover, we denote the optimization problem of finding a smallest colorful edge cover. The set $\{(G, p) : G \text{ has a colorful edge cover of size } p\}$ defines the corresponding decision problem, the colorful edge cover problem. Before we show its hardness, we prove that any algorithm that solves the minimum colorful edge cover problem on simple graphs, also solves the problem on graphs with self-loops. We achieve this by constructing a gadget of constant size replacing all self-loops.

► **Lemma 5.** *An algorithm solving the colorful edge cover problem on simple graphs also solves the colorful edge cover problem on graphs with self-loops.*

Proof. Self-loops can be simulated by a gadget consisting of two new vertices and a new color. Introduce a new color b and two new vertices x, y that are connected by an edge of color b . For any vertex v with a self-loop of color a , instead connect v and x with color a . Any valid solution contains the unique edge of color b such that x and y are always covered. Choosing the edge $\{v, x\}$ then only covers v and color a , which is the same behaviour as for the self-loop. The resulting graph does not contain self-loops and has a colorful edge cover on $\mathcal{C} + 1$ colors of size $p + 1$ if and only if the original graph has a colorful edge cover on \mathcal{C} colors of size p . ◀

In the following, we introduce a special, balanced version of the satisfiability problem that enables us to show the NP-hardness of the colorful edge cover problem.

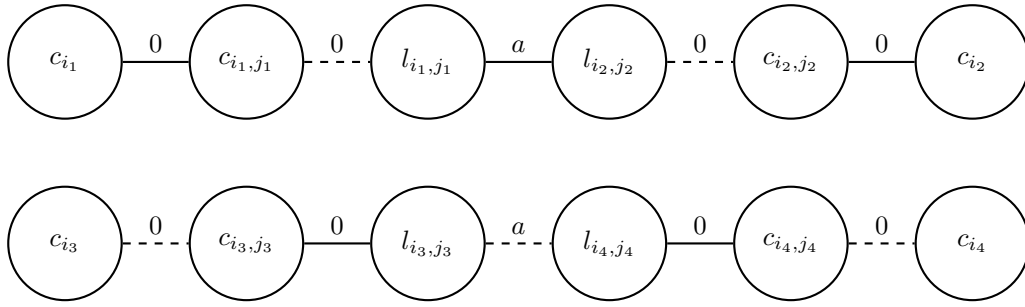


Figure 1 Gadgets for a variable x_a in the colorful edge cover reduction.

► **Definition 6** ((3, B2)-SAT). A Boolean formula $\phi = \bigwedge_{i=1}^m c_i = \bigwedge_{i=1}^m \bigvee_{j=1}^{L(c_i)} l_{i,j}$ over the variables $\mathbf{X}_n = \{x_1, \dots, x_n\}$ is a (3, B2)-SAT instance if each clause consists of exactly three literals and each literal occurs exactly two times, thus each variable occurs exactly four times. We denote the satisfiability problem for (3, B2)-SAT instances by (3, B2)-SAT.

By a result from Berman, Karpinski and Scott [6], (3, B2)-SAT is NP-complete.

► **Theorem 7.** The colorful edge cover problem is NP-complete.

Proof. The problem is in NP, as a subset $E' \subseteq E$ forming a colorful edge cover can be encoded and verified in polynomial space and time. We show its hardness by a reduction from (3, B2)-SAT.

Given a (3, B2)-SAT formula ϕ , for each clause $c_i = \bigvee_{j=1}^3 l_j$, construct a clause vertex c_i , and for each $j \in [1, 3]$ construct an intermediate vertex $c_{i,j}$ connected to c_i , and a literal vertex $l_{i,j}$ connected to $c_{i,j}$. All those edges are assigned the color 0. Further, for each variable x_a , connect the two vertices $l_{i_1, j_1}, l_{i_2, j_2}$ corresponding to its positive literal with an edge of color a , and the two vertices $l_{i_3, j_3}, l_{i_4, j_4}$ corresponding to its negative literals with another edge of color a . Figure 1 shows the subgraph for a variable x_a . Note that the clause vertices of the form c_i have two more neighbors $c_{i,j'}, c_{i,j''}$ not depicted here as they also belong to the gadgets of their other literals. The dashed lines form a solution representing a false assignment to the variable x_a , whereas the solid lines represent setting the variable to true. We claim that the constructed graph has a colorful edge cover of size $n + |\phi| = 5n$ if and only if ϕ is satisfiable.

Assume a colorful edge cover of size $n + |\phi| = 5n$ exists. These costs are always a lower bound for the colorful edge cover, as there has to be one chosen edge for each color other than 0, and one edge for each intermediate vertex $c_{i,j}$, as they are pairwise non-adjacent and all their incident edges have color 0.

Another way to see this is to consider the vertices unique to each variable x_a , which are four pairs of the form $c_{i,j}, l_{i,j}$. Because only the edges $\{l_{i_1, j_1}, l_{i_2, j_2}\}$ and $\{l_{i_3, j_3}, l_{i_4, j_4}\}$ have the color a , it is not possible to cover all the intermediate vertices $c_{i,j}$ and the color with less than five edges. The dashed and solid lines in Figure 1 each refer to one way to cover all unique elements with five edges, while also covering either c_{i_1} and c_{i_2} or c_{i_3} and c_{i_4} . Without loss of generality, we assume that the edges adjacent to the included edge of color a are not included, so our solution follows this form. Then, a clause vertex c_i is only covered by an edge $\{c_i, c_{i,j}\}$ if the edge $\{c_{i,j}, l_{i,j}\}$ is not included. In turn, this means that the edge of color a incident to $l_{i,j}$ is included, indicating that variable X_a is assigned a truth value that satisfies c_i .

Given a satisfying assignment of \mathbf{X}_n for ϕ , construct a colorful edge cover for G as follows. For each variable x_a , consider the two gadgets $c_{i_1}c_{i_1,j_1}l_{i_1,j_1}l_{i_2,j_2}c_{i_2}c_{i_2}$ and $c_{i_3}c_{i_3,j_3}l_{i_3,j_3}l_{i_4,j_4}c_{i_4}c_{i_4}$. If x_a is positive, choose the edges

$$\{c_{i_1}, c_{i_1,j_1}\}, \{l_{i_1,j_1}, l_{i_2,j_2}\}, \{c_{i_2,j_2}, c_{i_2}\}, \{c_{i_3,j_3}, l_{i_3,j_3}\}, \{l_{i_4,j_4}c_{i_4,j_4}\}.$$

Note that $\{l_{i_1,j_1}, l_{i_2,j_2}\}$ has color a . If x_a is negative, choose the edges

$$\{c_{i_1,j_1}, l_{i_1,j_1}\}, \{l_{i_2,j_2}, c_{i_2,j_2}\}, \{c_{i_3}, c_{i_3,j_3}\}, \{l_{i_3,j_3}, l_{i_4,j_4}\}, \{c_{i_4,j_4}c_{i_4}\}.$$

Note that $\{l_{i_3,j_3}, l_{i_4,j_4}\}$ has color a . In both cases, we infer a cost of $5n$. It is clear for all vertices except the clause vertices c_i that they are covered. Because ϕ is satisfied by the given assignment, any clause c_i is satisfied by some literal l_j , so by our choice the edge $\{c_i, c_{i,j}\}$ is included, covering c_i . Further, the color 0 is covered $4n$ times, and each other color is covered exactly once. ◀

4 NP-Completeness of the 2-Attractor Problem

We now give a formal definition of a graph interpretation of strings and their substrings of length 2 which we call the *2-substring graph*. Each position in a string corresponds to an edge in this graph. This interpretation was used to show that computing sharp 2-attractors can be done in time $\mathcal{O}(n\sqrt{n})$ [15] by solving the edge cover problem on this graph. However, we additionally use symbols to label the edges, yielding instances of the colorful edge cover problem instead.

▶ **Definition 8.** *Given a set of strings $\mathbf{T} = \{T_1, \dots, T_m\}$ with $T_i \in \Sigma^{n_i}$, we define the 2-substring graph of \mathbf{T} by $G(\mathbf{T}) = (V, E = E_1 \cup E_2 \cup E_3, \sigma)$ with $V = \{xy \in \Sigma^2 \mid xy \text{ is a substring of any } T_i \in \mathbf{T}\}$,*

$$E_1 = \{(xy, yz) \in (\Sigma^2)^2 \mid xyz \text{ is a substring of any } T_i \in \mathbf{T}\}$$

$$E_2 = \{(xy, xy) \in (\Sigma^2)^2 \mid xy \text{ is the prefix of any } T_i \in \mathbf{T}\}$$

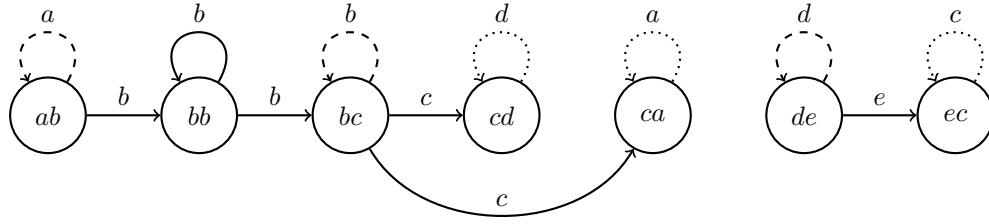
$$E_3 = \{(yz, yz) \in (\Sigma^2)^2 \mid yz \text{ is the suffix of any } T_i \in \mathbf{T}\}$$

and $\sigma : E \rightarrow \Sigma$ is a labeling function on the edges defined by

$$\sigma(e) = \begin{cases} y, & \text{if } e = (xy, yz) \in E_1, \\ x, & \text{if } e = (xy, xy) \in E_2, \\ z, & \text{if } e = (yz, yz) \in E_3. \end{cases}$$

Figure 2 shows the 2-substring graph of a set of strings $\{abbbcd, bca, dec\}$, where the solid lines are from E_1 , the dashed lines are the prefix self-loops from E_2 and the dotted lines are the suffix self-loops from E_3 . Note that the 2-substring graph does not capture words of length 1, as they do not have a substring of length 2 and therefore no associated vertex. However, these are not relevant to our problem. In a preprocessing step, we can remove the word of length 1 if its symbol also occurs in a longer string, or we have to add it to the solution if this is the only occurrence of the symbol.

The 2-attractor problem as a special case of the k -attractor problem is in NP [16]. To show NP-hardness, we essentially consider the reduction for colorful edge cover and find an assignment of symbols to the edges such that the resulting graph is a 2-substring graph representing a set of strings. The key idea is that this 2-substring graph with the edge labels



■ **Figure 2** The 2-substring graph for $\{abbbcd, bca, dec\}$.

interpreted as a set of colors has a colorful edge cover of a fixed size p if and only if the underlying set of strings has a k -attractor of size p . Towards this step, we assign a 2-substring to each vertex created in the proof of Theorem 7 and exchange colors with symbols. It is vital that each vertex has a unique label so that each vertex uniquely corresponds to a 2-substring. For that reason, we need to introduce more symbols and pay attention that they are all covered. The 2-substring graph representation of the constructed gadgets is shown in Figure 3. Comparing this to Figure 1 shows the similarities in the proofs.

Note that we do not give a reduction from the colorful edge cover problem to the 2-attractor problem, but use the structure of the colorful edge cover to give a reduction directly from $(3, B2)$ -SAT to the 2-attractor problem on sets. Combined with Lemma 3, this shows the NP-completeness of the 2-attractor problem.

► **Theorem 9.** *The 2-attractor problem is NP-complete.*

Proof. We again start with a $(3, B2)$ -SAT formula ϕ . We create a set of strings based on an alphabet $\Sigma = \mathbf{C} \cup \mathbf{L} \cup \bar{\mathbf{L}} \cup \mathbf{X} = \{C_1, \dots, C_m\} \cup \{L_1, L_2, L_3\} \cup \{\bar{L}_1, \bar{L}_2, \bar{L}_3\} \cup \{X_1, \dots, X_n\}$. Note that the symbols for the literals $\mathbf{L}, \bar{\mathbf{L}}$ only indicate the position of the literal in its clause and whether it is negated, but not the clause itself.

We now construct a set of strings \mathbf{T} . For every variable X_a that appears positively in clauses c_{i_1} and c_{i_2} at literals l_{i_1, j_1} and l_{i_2, j_2} respectively with $i_1 \leq i_2$ (and $j_1 < j_2$ if $i_1 = i_2$), we add a string $C_{i_1} C_{i_1} L_{j_1} X_a L_{j_2} C_{i_2} C_{i_2}$. For every variable X_a that appears negatively in clauses c_{i_3} and c_{i_4} at literals l_{i_3, j_3} and l_{i_4, j_4} respectively with $i_3 \leq i_4$ (and $j_3 < j_4$ if $i_3 = i_4$), we add a string $C_{i_3} C_{i_3} \bar{L}_{j_3} X_a \bar{L}_{j_4} C_{i_4} C_{i_4}$. We also add six auxiliary strings $L_1 L_1, \bar{L}_1 \bar{L}_1, L_2 L_2, \bar{L}_2 \bar{L}_2, L_3 L_3, \bar{L}_3 \bar{L}_3$ to make sure all symbols L_j and \bar{L}_j are covered.

This construction is shown in Figure 3 as subgraphs of the 2-substring graph. Note that the clause vertices of the form $C_i C_i$ have two more adjacent edges that are not shown here, and the auxiliary strings are not shown. The dashed lines form a solution representing a false assignment to the variable x_a , whereas the solid lines represent setting the variable to true. Note that there is always an optimal solution that does not use the self-loops at vertices of the form $C_i C_i$ which are represented by dotted lines.

The set of 2-substrings that need to be covered consists of $C_i C_i, C_i L_1, C_i L_2$ and $C_i L_3$ for each clause C_i , as well as $L_{j_1} X_a, X_a L_{j_2}, \bar{L}_{j_3} X_a, X_a \bar{L}_{j_4}$ for each variable X_a . Note that each of these 2-substrings except for $C_i C_i$ appears only once in \mathbf{T} , so they need to be covered at that occurrence. The auxiliary strings each add one unique 2-substring $L_j L_j$ or $\bar{L}_j \bar{L}_j$ for $j \in \{1, 2, 3\}$ that also need to be covered in the respective string. Of course, all 1-substrings, i.e. Σ , also need to be covered.

We claim that \mathbf{T} has a 2-set attractor of size $n + |\phi| + 6 = 5n + 6$ if and only if ϕ is satisfiable. The proof follows the same arguments as the proof of Theorem 7 as the 2-substring graph $G(\mathbf{T})$ has a colorful edge cover if and only if \mathbf{T} has a 2-set attractor.

Assume that a 2-set attractor of size $5n + 6$ for \mathbf{T} exists. We first show that the n symbols \mathbf{X} corresponding to the variables, which all appear twice, are covered exactly once each. The substrings $L_j L_j$ and $\bar{L}_j \bar{L}_j$ for $j \in \{1, 2, 3\}$ are unique to the six auxiliary strings, so they always induce a cost of six and ensure that all L_i, \bar{L}_i are covered independent of the remaining strings. It is also necessary to expend $4n$ markings to cover the substrings of the form $C_i L_j$, $L_j C_i$, $C_i \bar{L}_j$ and $\bar{L}_j C_i$, as each of these are pairwise non-overlapping with each other. The remaining n markings are then needed to cover all n symbols \mathbf{X} , so each is covered once.

We can therefore deduce a truth assignment by considering where each symbol X_a corresponding to a variable is covered. We show that this assignment satisfies ϕ . Consider any substring $C_i C_i$. Without loss of generality it is covered at the position adjacent to some L_j or \bar{L}_j , not at the position corresponding to the start or the end of a string. Therefore, the substrings and markings are of the form $C_i \underline{C_i} L_j$, $L_j \underline{C_i} C_i$ or $C_i \underline{C_i} \bar{L}_j$, $\bar{L}_j \underline{C_i} C_i$. This also covers the substrings of the form $C_i L_j$, $L_j C_i$, $C_i \bar{L}_j$ and $\bar{L}_j C_i$. By our counting argument, each of these substrings is covered only once, so L_j respectively \bar{L}_j is not marked. To then cover $L_j X_a$, $X_a L_j$ or $\bar{L}_j X_a$, X_a must be marked, which means the variable x_a is assigned a truth value such that it satisfies c_i . This holds for all c_i , so ϕ is satisfied.

Assume ϕ is satisfiable by some truth assignment of the variables \mathbf{X} . If x_a is set to true, mark its two strings by

$$C_{i_1} \underline{C_{i_1}} L_{j_1} X_a L_{j_2} \underline{C_{i_2}} C_{i_2}, C_{i_3} C_{i_3} \bar{L}_{j_3} X_a \bar{L}_{j_4} \underline{C_{i_4}} C_{i_4}$$

otherwise mark

$$C_{i_1} C_{i_1} L_{j_1} X_a L_{j_2} \underline{C_{i_2}} C_{i_2}, C_{i_3} C_{i_3} \bar{L}_{j_3} X_a \bar{L}_{j_4} \underline{C_{i_4}} C_{i_4}.$$

Both ways of marking the strings cover all eight substrings unique to x_a . The solid edges in Figure 3 show the included positions in the 2-substring graph for a positively assigned variable, and the dashed lines refer to a negative assignment. Each substring $C_i C_i$ is covered in the substring corresponding to the variable satisfying it. Further, each symbol of Σ is also covered. Each C_i is covered just as $C_i C_i$ is covered, each L_j is covered due to the six auxiliary strings, and X_a is covered by definition of our chosen markings.

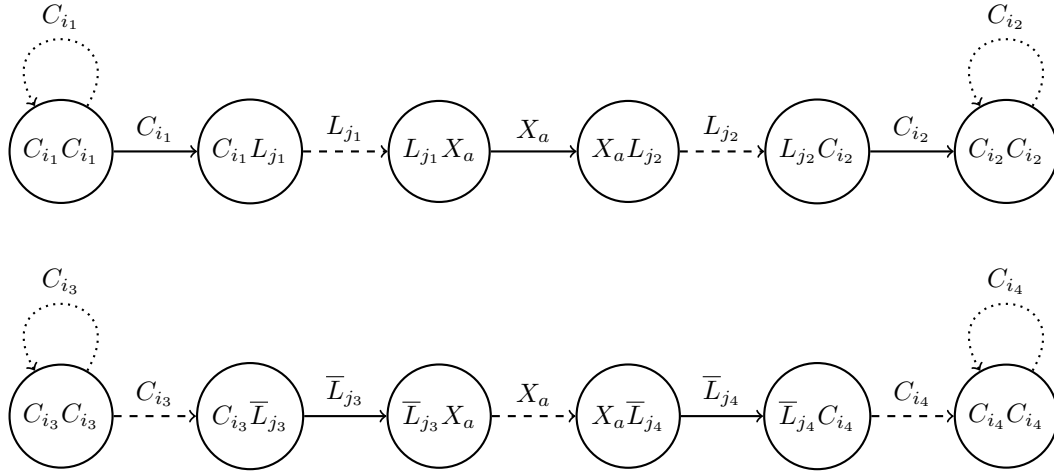
In total, given a formula with m clauses and n variables, we compute for a set of $2n + 6$ strings whether it has a 2-set attractor of size $5n + 6$. Each string has a size of 7 except for the auxiliary strings of size 2, and all strings in total have length $14n + 12$. The reduction can be computed naively in time $\mathcal{O}(n^2)$. By Lemma 3, the set of strings can be condensed into a single string of linear size in polynomial time. ◀

This reduction can be used for all $k \geq 2$ and for the general attractor problem, as there is a delimiter symbol or unique substring at least every 3 symbols. Any valid k -attractor then needs to put a marking every 3 symbols, such that any possibly uncovered substring has length at most 2. This enforces that there is a 2-attractor of any fixed size p if and only if there is a k -attractor of size p for any $k \geq 3$.

5 APX-Hardness

Our reduction to prove Theorem 9 also suffices to show the APX-hardness of the k -attractor problem for $k \geq 2$, which was shown for $k \geq 3$ by Kempa and Prezza [16]. The same paper also shows containment in APX for constant k . We also slightly improve the explicit lower bound to which k -attractor cannot be approximated. To this end, we analyse the behavior of the reduction's output string for unsatisfiable formulas.

35:10 The 2-Attractor Problem Is NP-Complete



■ **Figure 3** Gadgets for a variable x_a in the 2-attractor reduction in the 2-substring graph.

► **Lemma 10.** For a MAX-(3, B2)-SAT formula ϕ with m clauses and an optimal assignment satisfying $m - u$ clauses, $u \geq 0$, the optimal attractor for $T(\phi)$ has size between $\frac{21}{4}m + 11 + \lceil u/2 \rceil$ and $\frac{21}{4}m + 11 + u$.

Proof. A MAX-(3, B2)-SAT formula ϕ with m clauses contains $n = (3/4)m$ variables, thus the resulting set of strings is of size $2m + 6$ including auxiliary strings, inducing $2m + 5$ delimiter symbols to combine those strings into a single string T . Each variable induces a cost of at least 5 to cover its unique substrings, also each auxiliary string and delimiter induces a cost of 1. In total, we obtain a lower bound of $5n + 6 + 2n + 5 = \frac{21}{4}m + 11$ that can be matched if and only if ϕ is satisfiable with the markings given in the proof of Theorem 9.

If ϕ is not satisfiable, this marking still yields the most efficient way to cover the unique substrings. Then, all substrings are covered except for $C_i C_i$ (and C_i , which will always be covered implicitly) for clauses c_i that are not covered by the assignment. This gives u many 2-substrings that are not covered, which can be covered one by one using u additional positions, yielding the upper bound.

Consider a variable x_a such that in the optimal assignment two of the four clauses it appears in are not satisfied. Then, without loss of generality x_a is set to true and appears negatively in clauses c_{i_1}, c_{i_2} which are not satisfied. The corresponding string is then $C_{i_1}C_{i_1}\bar{L}_{j_1}X_a\bar{L}_{j_2}C_{i_2}C_{i_2}$ but remarking it to $C_{i_1}C_{i_1}\bar{L}_{j_1}X_a\bar{L}_{j_2}C_{i_2}C_{i_2}$ covers both $C_{i_1}C_{i_1}$ and $C_{i_2}C_{i_2}$ with just one additional position. If this can be done for all clauses, only $u/2$ additional markings are needed. Each variable still has to induce a cost of at least 5 and now induces a cost of at most 6, so this is optimal, otherwise the assignment we started with was not optimal. ◀

In [6], Berman, Karpinski and Scott show the APX-hardness of MAX-(3, B2)-SAT by a reduction from the MAX-E3-Lin-2 problem on linear equations, which was studied by Håstad [13]. We extend the given reduction to the k -attractor problem.

► **Theorem 11.** For every $k \geq 2$ and $0 < \varepsilon < 1$, it is NP-hard to approximate the k -attractor problem to within an approximation ratio smaller than $(10669 - \varepsilon)/10668$.

Proof. Berman, Karpinski and Scott [6] show that it is NP-hard to distinguish MAX-(3, B2)-SAT instances with $1016n$ clauses of which at least $(1016 - \varepsilon)n$ are satisfiable, and instances with $1016n$ clauses of which at most $(1015 + \varepsilon)n$ are satisfiable.

Let ϕ be a $(3, B2)$ -SAT formula with $1016n$ clauses, and $T(\phi)$ the string resulting from applying the reduction in Theorem 9 to this formula. The formula ϕ has $1016n \cdot 3/4 = 762n$ variables and thus $T(\phi)$ consists of $2 \cdot 762n + 6$ substrings with $2 \cdot 762n + 5$ delimiter symbols between them. If ϕ is satisfiable, $T(\phi)$ has an optimal k -attractor of size $(5 \cdot 762n + 6) + (2 \cdot 762n + 5) = 5334n + 11$ by Lemma 10. Further, if we can satisfy at least $(1016 - \epsilon)n$ clauses in ϕ , we need at most

$$(5334 + \epsilon)n + 11 = (10668 + 2\epsilon)n/2 + 11$$

many positions to find a 2-attractor for $T(\phi)$. Otherwise, we can satisfy at most $(1015 + \epsilon)n$ clauses and thus need at least $5334n + \frac{1-\epsilon}{2}n + 11 = (10669 - \epsilon)n/2 + 11 > (10669 - 2\epsilon)n/2 + 11$ many positions to find a 2-attractor for $T(\phi)$.

If we could approximate k -attractor better than $(10669 - \epsilon)/10668$, we could plug in ϕ and see if we get a result within

$$\frac{10669 - \epsilon}{10668} \left(\frac{(10668 + 2\epsilon)}{2}n + 11 \right) = \left(\frac{10669}{2} + \frac{10669\epsilon}{10668} - \frac{\epsilon}{2} - \frac{2\epsilon\epsilon}{10668} \right) n + \frac{10669 \cdot 11}{10668}$$

and accept if and only if this is true. For large enough n and ϵ , it holds that

$$\begin{aligned} & \left(\frac{10669}{2} + \frac{10669\epsilon}{10668} - \frac{\epsilon}{2} - \frac{2\epsilon\epsilon}{10668} \right) n + \frac{10669 \cdot 11}{10668} < \frac{10669 - 2\epsilon}{2}n + 11 \\ \iff & \frac{21337\epsilon}{10668} - \frac{2\epsilon\epsilon}{10668} + \frac{11}{10668n} < \frac{\epsilon}{2} \end{aligned}$$

and thus we are able to distinguish whether $(1016 - \epsilon)n$ clauses or $(1015 + \epsilon)n$ clauses in ϕ are satisfiable, which is NP-hard. Note that ϵ can converge to 0 as ϵ converges to 0, thus the proof works for all $\epsilon > 0$. ◀

A similar construction can be used to show that the colorful edge cover problem is APX-hard and cannot be approximated by a factor smaller than $(7621 - \epsilon)/7620$ for $0 < \epsilon < 1$.

6 Conclusion

In this paper, we have answered the open problem of the complexity of the 2-attractor problem and discussed the implicit APX-hardness that results from our reduction. Additionally, we introduced a more general variation of the k -attractor problem, the k -set attractor problem, to make our reduction more convenient. Moreover, motivated by the previous reductions between the k -set cover problem and the k -attractor problem, we introduced the colorful edge cover problem. Although it is not contained in the reduction chain for the main result, it shows where the sharp 2-attractor problem and the 2-attractor problem differ in their complexity.

In general, adding colorfulness to a problem solvable in polynomial time, like matching, network flow or spanning tree can help to find a different perspective on problems of unresolved complexity that are close to the mentioned problems, but not properly modeled by their unmodified variants. In the best case, it helps to understand where the borderline of complexity is and which combination of constraints make a problem hard. Colorful problems may also be of interest in the context of parameterized complexity or approximability.

References

- 1 Tooru Akagi, Mitsuru Funakoshi, and Shunsuke Inenaga. Sensitivity of string compressors and repetitiveness measures. *Inf. Comput.*, 291:104999, 2021. doi:10.1016/j.ic.2022.104999.
- 2 Saieed Akbari, Vahid Liaghat, and Afshin Nikzad. Colorful paths in vertex coloring of graphs. *Electr. J. Comb.*, 18, January 2011. doi:10.37236/504.
- 3 Hideo Bannai, Keisuke Goto, Masakazu Ishihata, Shunsuke Kanda, Dominik Köppl, and Takaaki Nishimoto. Computing np-hard repetitiveness measures via MAX-SAT. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPICs*, pages 12:1–12:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ESA.2022.12.
- 4 Matthias Bentert, Leon Kellerhals, and Rolf Niedermeier. Fair short paths in vertex-colored graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37:12346–12354, June 2023. doi:10.1609/aaai.v37i10.26455.
- 5 Sergey Bereg, Feifei Ma, Wencheng Wang, Jian Zhang, and Binhai Zhu. On some matching problems under the color-spanning model. *Theoretical Computer Science*, 786:26–31, 2019. *Frontiers of Algorithmics*. doi:10.1016/j.tcs.2018.08.008.
- 6 Piotr Berman, Marek Karpinski, and Alex D. Scott. Approximation hardness of short symmetric instances of max-3sat. *Electron. Colloquium Comput. Complex.*, TR03, 2003. URL: <https://api.semanticscholar.org/CorpusID:40181844>.
- 7 Christina Büsing and Martin Comis. Budgeted colored matching problems. *Electronic Notes in Discrete Mathematics*, 64:245–254, 2018. 8th International Network Optimization Conference - INOC 2017. doi:10.1016/j.endm.2018.01.026.
- 8 Anders Roy Christiansen, Mikko Berggren Ettiienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Algorithms*, 17(1), December 2021. doi:10.1145/3426473.
- 9 J. Cohen, Y. Manoussakis, H.P. Phong, and Zs. Tuza. Tropical matchings in vertex-colored graphs. *Electronic Notes in Discrete Mathematics*, 62:219–224, 2017. LAGOS'17 – IX Latin and American Algorithms, Graphs and Optimization. doi:10.1016/j.endm.2017.10.038.
- 10 Riccardo Dondi and Mohammad Mehdi Hosseinzadeh. Finding colorful paths in temporal graphs. In Rosa Maria Benito, Chantal Cherifi, Hocine Cherifi, Esteban Moro, Luis M. Rocha, and Marta Sales-Pardo, editors, *Complex Networks & Their Applications X*, pages 553–565, Cham, 2022. Springer International Publishing.
- 11 Lubomíra Dvořáková. String attractors of episturmian sequences. *CoRR*, abs/2211.01660v2, 2022. doi:10.48550/arXiv.2211.01660.
- 12 France Gheeraert, Giuseppe Romana, and Manon Stipulanti. String attractors of fixed points of k-bonacci-like morphisms. *CoRR*, abs/2302.13647, 2023. doi:10.48550/arXiv.2302.13647.
- 13 Johan Håstad. Some optimal inapproximability results. *Electron. Colloquium Comput. Complex.*, TR97, 2001. URL: <https://api.semanticscholar.org/CorpusID:5120748>.
- 14 Dominik Kempa and Tomasz Kociumaka. Resolution of the burrows-wheeler transform conjecture. *Commun. ACM*, 65(6):91–98, 2022. doi:10.1145/3531445.
- 15 Dominik Kempa, Alberto Policriti, Nicola Prezza, and Eva Rotenberg. String attractors: Verification and optimization. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPICs*, pages 52:1–52:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.52.
- 16 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

- 17 Dominik Kempa and Barna Saha. An upper bound and linear-space queries on the lz-end parsing. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 2847–2866. SIAM, 2022. doi:10.1137/1.9781611977073.111.
- 18 Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive measure of repetitiveness. In Yoshiharu Kohayakawa and Flávio Keidi Miyazawa, editors, *LATIN 2020: Theoretical Informatics - 14th Latin American Symposium, São Paulo, Brazil, January 5-8, 2021, Proceedings*, volume 12118 of *Lecture Notes in Computer Science*, pages 207–219. Springer, 2020. doi:10.1007/978-3-030-61792-9_17.
- 19 Sebastian Kreft and Gonzalo Navarro. Lz77-like compression with fast random access. In *2010 Data Compression Conference*, pages 239–248, 2010. doi:10.1109/DCC.2010.29.
- 20 Sabrina Mantaci, Antonio Restivo, Giuseppe Romana, Giovanna Rosone, and Marinella Sciortino. A combinatorial view on string attractors. *Theor. Comput. Sci.*, 850:236–248, 2021. doi:10.1016/j.tcs.2020.11.006.
- 21 Jérôme Monnot. The labeled perfect matching in bipartite graphs. *Inf. Process. Lett.*, 96(3):81–88, 2005. doi:10.1016/J.IPL.2005.06.009.
- 22 Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theoretical Computer Science*, 762:41–50, 2019. doi:10.1016/j.tcs.2018.09.007.
- 23 Nicola Prezza. String attractors. *CoRR*, abs/1709.05314, 2017. arXiv:1709.05314.
- 24 Antonio Restivo, Giuseppe Romana, and Marinella Sciortino. String attractors and infinite words. *CoRR*, abs/2206.00376, 2022. doi:10.48550/arXiv.2206.00376.
- 25 Jeffrey O. Shallit and Luke Schaeffer. String attractors for automatic sequences. *CoRR*, abs/2012.06840, 2020. arXiv:2012.06840.

Directed Regular and Context-Free Languages

Moses Ganardi  

Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Irmak Sağlam  

Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Georg Zetsche  

Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

We study the problem of deciding whether a given language is directed. A language L is *directed* if every pair of words in L have a common (scattered) superword in L . Deciding directedness is a fundamental problem in connection with ideal decompositions of downward closed sets. Another motivation is that deciding whether two *directed* context-free languages have the same downward closures can be decided in polynomial time, whereas for general context-free languages, this problem is known to be coNEXP-complete.

We show that the directedness problem for regular languages, given as NFAs, belongs to AC^1 , and thus polynomial time. Moreover, it is NL-complete for fixed alphabet sizes. Furthermore, we show that for context-free languages, the directedness problem is PSPACE-complete.

2012 ACM Subject Classification Theory of computation \rightarrow Problems, reductions and completeness; Theory of computation \rightarrow Concurrency; Theory of computation \rightarrow Formal languages and automata theory

Keywords and phrases Subword, ideal, language, regular, context-free, equivalence, downward closure, compression

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.36

Related Version *Extended Version*: <https://arxiv.org/abs/2307.13396> [27]

Funding Funded by the European Union (ERC, FINABIS, 101077902). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.



1 Introduction

We study the problem of deciding whether a given language is directed. A language L is called (*upward*) *directed* if for every $u, v \in L$, there exists a $w \in L$ with $u \preceq w$ and $v \preceq w$. Here, \preceq denotes the (non-contiguous) *subword* relation: We have $u \preceq v$ if there are decompositions $u = u_1 \cdots u_n$ and $v = v_0 u_1 v_1 \cdots u_n v_n$ for some $u_1, \dots, u_n \in \Sigma^*$ and $v_0, v_1, \dots, v_n \in \Sigma^*$.

Downward closures and ideals. The *downward closure* of a language $L \subseteq \Sigma^*$ is the set $L \downarrow = \{u \in \Sigma^* \mid \exists v \in L: u \preceq v\}$. Over the last ca. 15 years, downward closures have been used in several approaches to verifying concurrent systems. This has two reasons: First, $L \downarrow$ is a regular language for every set $L \subseteq \Sigma^*$ [32] and an NFA can often be computed effectively [6, 21, 23, 30, 31, 47, 49, 50]. Second, many verification tasks are *downward closure invariant* w.r.t. subsystems: This means, a (potentially infinite-state) subsystem (e.g. a recursive program represented by a context-free language) can be replaced with another with the same downward closure, without affecting the verified property. This has been applied to parameterized systems with non-atomic reads and writes [46], concurrent programs with dynamic thread creation [5, 11, 13], asynchronous programs [10, 41], and thread pools [14].



© Moses Ganardi, Irmak Sağlam, and Georg Zetsche;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 36; pp. 36:1–36:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



In addition to finite automata, there is a second representation of downward closed languages: Every non-empty downward closed set can be written as a finite union of ideals. An *ideal* (in the terminology of well-quasi orderings) is a non-empty downward closed set that is directed. Moreover, ideals have a simple representation themselves: They are precisely the products of languages of the forms $\{a, \varepsilon\}$ and Δ^* , where a is a letter and Δ is an alphabet. Clearly, a language L is directed if and only if $L\downarrow$ is itself an ideal.

Ideal decompositions of downward closed sets have recently been the center of significant attention: They have been instrumental in computing downward closures [9, 28, 49] and deciding separability by piecewise testable languages [29, 51]. Over other orderings, ideals play a crucial role in forward analysis of well-structured transition systems (WSTS) [16, 25, 26], infinitely branching WSTS [18], well-behaved transition systems [17] for clarifying reachability problems in vector addition systems [36–38], and for deciding regular separability [24].

Given the importance of ideals, it is a fundamental problem to decide whether a given language is directed, in other words, whether the ideal decomposition of its downward closure consists of a single ideal. It is a basic task for computing ideal decompositions, but also an algorithmic lens on the structure of ideals.

Efficient comparison. Aside from being a fundamental property, checking directedness is also useful for deciding equivalence. It is well-known that equivalence is PSPACE-complete for NFAs and undecidable for context-free languages. However, in some situations, it suffices to decide *downward closure equivalence*: Due to the aforementioned downward closure invariance in concurrent programs, if $L_1, L_2 \subseteq \Sigma^*$ describe the behaviors of sequential programs inside of a concurrent program, and we have $L_1\downarrow = L_2\downarrow$, then L_1 can be replaced with L_2 without affecting safety, boundedness, and termination properties in concurrent [11] and asynchronous programs [41]. Downward closure equivalence is known to be coNP-complete for NFAs [8, Thm. 12&13] and coNEXP-complete for context-free languages [50]. This is better than PSPACE and undecidable, but our results imply that if L_1 and L_2 are directed, then deciding $L_1\downarrow = L_2\downarrow$ is *in polynomial time*, both for NFAs and for context-free languages! Thus, directedness drastically reduces the complexity of downward closure equivalence.

Constraint satisfaction problems. Directedness has recently also been studied in the context of constraint satisfaction problems (CSPs) for infinite structures. If we view finite words in the usual way as finite relational structures (as in first-order logic), then a set of words is directed if and only if it has the *joint embedding property (JEP)*. More generally, a class \mathcal{C} of finite structures has the JEP if for any two structures in \mathcal{C} , there is a third in which they both embed. The JEP is important for CSPs because if \mathcal{C} is definable by a universal first-order formula and has the JEP, then it is the *age* of some (potentially infinite) structure, which then has a constraint satisfaction problem in NP [19, p. 1].

Motivated by this, it was recently shown that the JEP is undecidable for universal formulas by Braunfeld [20] (and even for universal Horn formulas by Bodirsky, Rydval, and Schrottenloher [19]). In the special case of finite words, the JEP (and thus directedness) was shown to be decidable in polynomial time by Atminas and Lozin [7] for regular languages of the form $\{w \in \Sigma^* \mid w_1, \dots, w_n \not\prec w\}$ for given $w_1, \dots, w_n \in \Sigma^*$. However, to our knowledge, for general regular languages (or even context-free languages), the complexity is not known.

Contribution. Our first main result is that for NFAs, directedness is decidable in AC^1 , a circuit complexity class within polynomial time, defined by Boolean circuits of polynomial size, logarithmic depth, and unbounded fan-in. If we fix the alphabet size, directedness becomes NL-complete. Our second main result is that for context-free languages, directedness is PSPACE-complete, and hardness already holds for input alphabets of size two.

The proof techniques for the main results also yield algorithms for downward closure equivalence. Given L_1 and L_2 , we show that deciding $L_1\downarrow = L_2\downarrow$ is in AC^1 if L_1 and L_2 are directed and given by NFAs. As above, we obtain NL-completeness for fixed alphabets. If L_1 and L_2 are context-free languages, then deciding $L_1\downarrow = L_2\downarrow$ becomes P-complete.

Finally, we mention that counting the number of ideals in the ideal decomposition of $L\downarrow$ is #P-complete if L is given as an NFA. Here, hardness follows from #P-hardness of counting words in NFAs of a given length, and #P-membership is a consequence of our methods.

Key ingredients. The upper bounds as well as the lower bounds in our results rely on new techniques. With only slight extensions of existing techniques, one would obtain an NP upper bound for regular languages and an NEXP upper bound for context-free languages. This is because given a regular or context-free language, one can construct an acyclic graph where every path corresponds to an ideal of its downward closure. If the input is an NFA, this graph is polynomial-sized, and for CFGs, it is exponential-sized. One could then guess a path and verify that the entire language is included in this candidate ideal.

To obtain our upper bounds, we introduce a *weighting technique*, where each ideal is assigned a weight in the natural numbers. The weighting function has the property that if there is an ideal that contains the entire language, it must be one with maximal weight. Using either (i) matrix powering over the semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ for NFAs or (ii) dynamic programming for context-free grammars, this allows us to compute in NL/ AC^1 /polynomial time a unique candidate ideal (which is compressed in the context-free case), which is then verified in NL resp. in PSPACE.

For the PSPACE lower bound for context-free languages, we first observe that directedness is equivalent to deciding whether $L \subseteq I$, where L is a context-free language, and I is an SLP-compressed ideal. The problem is thus a slight generalization of the *compressed subword problem*, where we are given two SLP-compressed words u and v and are asked whether $u \preceq v$. This problem is known to be in PSPACE and PP-hard [39, Theorem 13], but its exact complexity is a long-standing open problem [40].

To exploit the increased generality of our problem, we proceed as follows. Our key insight is that for a given SLP-compressed word $w \in \Sigma^*$, one can construct an SLP-compressed infinite *complement ideal* I_w , meaning that $I_w \cap \Sigma^{|w|} = \Sigma^{|w|} \setminus \{w\}$. To this end, we apply a construction from definability of languages in the subword ordering [12, Lemma 3.1]. We use the complement ideal for PSPACE-hardness follows. We reduce from the PSPACE-complete problem of deciding, given two equal-length SLP-compressed words $v, w \in \{a, b\}^*$, whether their convolution $v \otimes w$ belongs to a fixed regular language [40, p. 269]. We reduce this to the problem of deciding $w \in L$, where w is SLP-compressed and L is a context-free of words of length $|w|$. Then $L \subseteq I_w$ if and only if $w \notin L$, hence $L \cup I_w$ is directed if and only if $w \notin L$.

2 Main results

Our first main result is that for a given NFA, one can decide directedness of its language in polynomial time, and even in $AC^1 \subseteq NC$.

► **Theorem 2.1.** *Given an NFA, one can decide in AC^1 whether its language is directed.*

Recall that AC^1 is the class of all languages that are accepted by a family of unbounded fan-in Boolean circuits of polynomial size and logarithmic depth, see [48] for more details. In particular, the directedness problem for regular languages can be efficiently parallelized.

The same techniques show that fixing the input alphabet leads to NL-completeness:

► **Theorem 2.2.** *For every fixed k , given an NFA over k letters, it is NL-complete to decide whether its language is directed.*

As mentioned before, slight extensions of known techniques would yield an NP upper bound for directedness of regular languages: Given an NFA \mathcal{A} , it is not difficult to construct an acyclic graph whose paths correspond to ideals for which $L(\mathcal{A}) = I_1 \cup \dots \cup I_n$. One can then guess such a path with ideal I_i and verify $L(\mathcal{A}) \subseteq I_i$ in NL. Clearly, $L(\mathcal{A})$ is directed iff such an I_i exists. The key challenge is to compute in AC^1 a single ideal I_i for which we check $L(\mathcal{A}) \subseteq I_i$.

Note that Theorems 2.1 and 2.2 also apply to one-counter languages. Given a one-counter language L , one can compute in logspace an NFA \mathcal{A} with $L(\mathcal{A}) = L\downarrow$ [6, Theorem 7]¹. Then $L(\mathcal{A})$ is directed iff L is directed, and we can just decide directedness for \mathcal{A} .

Our second main result is that for context-free languages (given, e.g. by a grammar), directedness is PSPACE-complete:

► **Theorem 2.3.** *Given a context-free grammar, it is PSPACE-complete to decide whether its language is directed. Moreover, PSPACE-hardness holds already for binary input alphabets.*

Our methods also provide more efficient algorithms for downward closure equivalence in the case of directed input languages. The *downward closure equivalence (DCE)* problem is to decide, for given languages L_1 and L_2 , whether $L_1\downarrow = L_2\downarrow$. While DCE is coNP -complete for general regular languages given as NFAs [8, Thm. 12&13], we show that for directed input languages, the complexity drops to AC^1 , and NL for fixed alphabets.

► **Theorem 2.4.** *For directed languages given as NFAs, DCE belongs to AC^1 , for fixed alphabets even to NL.*

For context-free languages, DCE is known to be coNEXP -complete [50]. For directed input languages, our methods yield a drastic drop in complexity down to polynomial time:

► **Theorem 2.5.** *For directed context-free languages, DCE is P-complete.*

Since our directedness algorithms decide whether the unique decomposition of $L\downarrow$ into maximal ideals consists of a single ideal, it is natural to ask about the complexity of counting all ideals of this decomposition. It follows easily using methods developed here that this problem is in $\#\text{P}$. Moreover, the well-known $\#\text{P}$ -hardness of $\#\text{NFA}$ (i.e. counting words of a given length in an NFA) [4] provides a $\#\text{P}$ lower bound.

► **Theorem 2.6.** *Given an NFA \mathcal{A} , it is $\#\text{P}$ -complete to count the number of ideals in the decomposition of $L(\mathcal{A})\downarrow$ into maximal ideals.*

The proof is given in [27, App. E] $\#\text{P}$ is the class of functions f computable by some non-deterministic polynomial-time Turing machine (TM), in the sense that for a given input x , the computed value $f(x)$ is the number of accepting runs. $\#\text{P}$ -complete problems are very hard, as evidenced by Toda's well-known result that $\text{P}^{\#\text{P}}$ (i.e. polynomial-time algorithms with access to $\#\text{P}$ oracles) includes the entire polynomial time hierarchy [45]. This represents an interesting contrast between the complexity of directedness and counting all ideals.

¹ Theorem 7 in [6] only states a polynomial time computation, but it is clear that it can be performed in (deterministic) logspace .

3 Preliminaries

Ideals. We will use the notation $[m_1, m_2] := \{i \in \mathbb{Z} \mid m_1 \leq i \leq m_2\}$. Consider the context-free languages $K_1 = \{wcw \mid w \in \{ab\}^*\}$ and $K_2 = \{wcw \mid w \in \{a\}^* \cup \{b\}^*\}$. Note that K_1 is directed, whereas K_2 is not: The words aca and bcb in K_2 have no common superword in K_2 . However, $K_1 \cup K_2$ is directed. Let Σ be a finite alphabet. A set $D \subseteq \Sigma^*$ is *downward closed* if $D \downarrow = D$. A subset $I \subseteq \Sigma^*$ is an *ideal* if it is non-empty, downward closed, and (upward) directed. Thus clearly, a non-empty $L \subseteq \Sigma^*$ is directed if and only if $L \downarrow$ is an ideal (note that taking the downward closure does not affect directedness). It is known that every ideal can be written as products of so called *atoms*: We identify two types of atoms over Σ :

Single atoms: $a^{(?)}$ where $a \in \Sigma$,

Alphabet atoms: $\Delta^{(\otimes)}$ where $\emptyset \neq \Delta \subseteq \Sigma$.

Formally, each atom α is a formal symbol that describes an ideal $\text{Idl}(\alpha)$. For a single atom $a^{(?)}$, we define it as $\text{Idl}(a^{(?)}) = \{a, \varepsilon\}$, whereas for an alphabet atom $\Delta^{(\otimes)}$, $\text{Idl}(\Delta^{(\otimes)}) = \Delta^*$. By $\text{atoms}(\Sigma)$, we denote the set of atoms over Σ . Note that $|\text{atoms}(\Sigma)| = 2^{|\Sigma|} - 1 + |\Sigma|$.

An *ideal representation* is a finite (possibly empty) sequence $r = \alpha_1 \cdots \alpha_n$ of atoms α_i . Its language is the concatenation $\text{Idl}(\alpha_1 \cdots \alpha_n) = \text{Idl}(\alpha_1) \cdots \text{Idl}(\alpha_n)$ where the empty concatenation is interpreted as $\{\varepsilon\}$. It is a classical fact that every downward closed set can be decomposed into a finite union of ideals. For example, observe that $K_1 \downarrow = (K_1 \cup K_2) \downarrow = \{a, b\}^* \{c, \varepsilon\} \{a, b\}^* = \text{Idl}(\{a, b\}^{(\otimes)} c^{(?)})$ and $K_2 \downarrow = \{a\}^* \{c, \varepsilon\} \{a\}^* \cup \{b\}^* \{c, \varepsilon\} \{b\}^* = \text{Idl}(\{a\}^{(\otimes)} c^{(?)}) \cup \text{Idl}(\{b\}^{(\otimes)} c^{(?)})$. This decomposition result was first shown by Jullien [34] and the equivalent fact that every downward closed set can be expressed as a *simple regular expression* was shown independently by Abdulla, Bouajjani, and Jonsson [1] (see [33] for a general treatment). If R is a set of ideal representations, we set $\text{Idl}(R) := \bigcup_{r \in R} \text{Idl}(r)$.

Reduced ideal representations. Note that one ideal can have multiple different representations. For instance, the representations $a^{(\otimes)} \cdot a^{(?)}$, $b^{(?)}$, $b^{(\otimes)} \cdot a^{(?)}$ and $a^{(\otimes)} \cdot b^{(\otimes)} \cdot a^{(?)}$ represent the same ideal, namely all words that start with a (possibly empty) sequence of a 's, followed by a (possibly empty) sequence of b 's, and possibly end with an a . This is because in the first representation, all the words $a^{(?)}$ and $b^{(?)}$ generate are produced by their neighboring alphabet atoms. Two representations are called *equivalent* if they represent the same ideal.

To achieve unique ideal representations, one can use *reduced representations*, which we define next. Two atoms α and β are *absorptive* if $\text{Idl}(\alpha\beta) = \text{Idl}(\alpha)$ or $\text{Idl}(\alpha\beta) = \text{Idl}(\beta)$. In the first case we say α *absorbs* β and in the second case, β *absorbs* α . Note that two single atoms are always non-absorptive since $\text{Idl}(a^{(?)}) \cdot b^{(?)} \supsetneq \text{Idl}(a^{(?)})$. An atom α is said to *contain* an atom β if $\text{Idl}(\alpha) \supseteq \text{Idl}(\beta)$. α is said to *strictly contain* β if also $\text{Idl}(\alpha) \neq \text{Idl}(\beta)$. An ideal representation $\alpha_1 \cdots \alpha_n$ is said to be *reduced* if for all $i \in [1, n-1]$, α_i and α_{i+1} are non-absorptive. The following is obvious (and well-known [2, Lemma 5.4]), because we can just repeatedly merge neighboring absorptive atom pairs:

► **Lemma 3.1.** *For every ideal representation $\alpha_1 \cdots \alpha_n$, there exists a reduced ideal representation $\beta_1 \cdots \beta_m$ such that $\text{Idl}(\alpha_1 \cdots \alpha_n) = \text{Idl}(\beta_1 \cdots \beta_m)$ and $m \leq n$.*

Representing downward closed sets. We will use two classical facts about ideals. First, every downward closed set $D \subseteq \Sigma^*$ can be written as a finite union of ideals. Moreover, ideals are “prime” in the sense that if an ideal is included in a union $D_1 \cup D_2$ of downward closed sets, it is already included in one of them:

► **Lemma 3.2** ([26, 28, 35]). *For every downward closed set $D \subseteq \Sigma^*$, there exist $n \in \mathbb{N}$ and ideals $I_1, \dots, I_n \subseteq \Sigma^*$ with $D = I_1 \cup \dots \cup I_n$. Moreover, if I is an ideal with $I \subseteq D_1 \cup D_2$ for downward closed $D_1, D_2 \subseteq \Sigma^*$, then $I \subseteq D_1$ or $I \subseteq D_2$.*

The representation $D = I_1 \cup \dots \cup I_n$ is also called an *ideal decomposition* of D . Observe that the second statement implies that this decomposition is unique (up to the order of ideals) if we require the ideals I_1, \dots, I_n to be pairwise incomparable. This is sometimes called the unique *decomposition into maximal ideals*.

Non-deterministic finite automata. We start by formally introducing NFAs. A *non-deterministic finite automaton (NFA)* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of *states*, $q_0 \in Q$ is the unique *initial state*, $F \subseteq Q$ is the set of *final states*, Σ is a *finite alphabet* and $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is the set of *transitions*. A transition $(p, a, q) \in \delta$ is usually displayed as $p \xrightarrow{a} q$, and we write $p_0 \xrightarrow{w} p_n$ if there exists a sequence of transitions $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n$ such that $w = a_1 \dots a_n$. The language accepted by an NFA \mathcal{A} is the set of all words $w \in \Sigma^*$ such that $q_0 \xrightarrow{w} q$ for some $q \in F$, and is denoted by $L(\mathcal{A})$.

4 Solution on Regular Languages

In this section, we prove Theorem 2.1 and Theorem 2.2. Let us quickly observe the NL lower bound of the directedness problem: We reduce from the emptiness problem for NFAs. Given an NFA \mathcal{A} , we may assume that there is exactly one final state that is different from the initial state, and that all edges are labeled with a . We construct an NFA \mathcal{A}' with $L(\mathcal{A}') = L(\mathcal{A}) \cup \{b\}$. Then clearly, $L(\mathcal{A}) \neq \emptyset$ if and only if $L(\mathcal{A}')$ contains some word in $\{a\}^+$. The latter is true if and only if $L(\mathcal{A}')$ is not directed.

Thus, the interesting part of Theorems 2.1 and 2.2 are the upper bounds. To explain the main steps, we need some terminology. We say that a function $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ is *computable in NL* if there exists a non-deterministic **logspace** TM with a write-only output tape such that (i) on every input word $x \in \{0, 1\}^*$ there exists an accepting computation, and (ii) every accepting computation on x produces the same output $f(x)$ on the output tape. We say that f is *computable in AC^1* if the language $\{x01^i \mid \text{the } i\text{-th bit of } f(x) \text{ is } 1\}$ belongs to AC^1 . The main difficulty in proving Theorem 2.1 is the following:

► **Lemma 4.1.** *Given a non-empty NFA \mathcal{A} , one can compute in AC^1 an ideal I such that (i) $I \subseteq L(\mathcal{A})\downarrow$ and (ii) $L(\mathcal{A})$ is directed if and only if $L(\mathcal{A})\downarrow \subseteq I$. Moreover, for every fixed alphabet size, this computation can be carried out in NL.*

Since the inclusion $L(\mathcal{A})\downarrow \subseteq I$ for a given NFA \mathcal{A} and ideal I can be decided in NL [50], Lemma 4.1 immediately implies the upper bounds: Just compute I and check $L(\mathcal{A})\downarrow \subseteq I$.

Let us briefly outline the proof of Lemma 4.1. Given an NFA \mathcal{A} for a regular language L , we first construct an NFA \mathcal{R}^{id1} accepting representations ideals in an ideal decomposition of $L\downarrow$. This is then transformed into an NFA \mathcal{R}^{red} that accepts *reduced* ideal representations. Reducedness of the ideal representations enables us to efficiently compute a maximal ideal in $L(\mathcal{R}^{\text{red}})$, by solving a maximum weight path problem. This step can be carried out in AC^1 for arbitrary alphabets and in NL for fixed alphabets. Figure 1 depicts the mentioned transitionary automata and serves as a running example throughout the section.

4.1 Computing the ideal automaton \mathcal{R}^{id1}

Our first step towards Lemma 4.1 is to transform the input NFA into one that is partially ordered. Here, an NFA is *partially ordered* if the state set Q is equipped with a partial order (Q, \leq) such that for every transition from p to q , we have $p \leq q$. In particular, the automaton does not contain any cycles except for self-loops. The following is a standard fact:

► **Lemma 4.2.** *Given any NFA \mathcal{A} , one can compute in NL a partially ordered NFA \mathcal{R} such that $L(\mathcal{R}) = L(\mathcal{A})\downarrow$.*

Essentially, one collapses each strongly connected component (SCC) C of \mathcal{A} into a new state q_C of \mathcal{R} and adds a self-loop to q_C for each letter that appears in C . Here, we require non-determinism in our logspace computation, because we need to determine whether a given letter appears in a strongly connected component. See [27, App. B] for details.

Next, we want to construct an NFA \mathcal{R}^{id1} over a finite alphabet of *atoms* of Σ , that will accept (as its *words*) the ideal representations given by the accepted paths of \mathcal{R} .

► **Lemma 4.3.** *Given any partially ordered NFA \mathcal{R} on the finite alphabet Σ , one can compute in NL an acyclic NFA \mathcal{R}^{id1} over some polynomial-sized alphabet $\Gamma \subseteq \text{atoms}(\Sigma)$ such that $\text{Id1}(L(\mathcal{R}^{\text{id1}})) = L(\mathcal{R})\downarrow$.*

Since the only cycles \mathcal{R} contains are self loops, we can write $L(\mathcal{R})$ as the finite union,

$$L(\mathcal{R}) = \bigcup_{i \in [1, r]} a_{1,i} \Delta_{1,i}^* a_{2,i} \Delta_{2,i}^* \cdots a_{k_i,i} \Delta_{k_i,i}^* \quad \text{with } a_{n,i} \in \Sigma \cup \{\varepsilon\} \text{ and } \Delta_{n,i} \subseteq \Sigma \text{ for } n \in [1, k_i]$$

Since $L(\mathcal{R})$ is downward closed, it is equivalent to the following ideal decomposition of $L(\mathcal{A})\downarrow$:

$$L(\mathcal{R}) = \bigcup_{i \in [1, r]} \{\varepsilon, a_{1,i}\} \Delta_{1,i}^* \{\varepsilon, a_{2,i}\} \Delta_{2,i}^* \cdots \{\varepsilon, a_{k_i,i}\} \Delta_{k_i,i}^* \quad (1)$$

Proof sketch. To construct \mathcal{R}^{id1} from \mathcal{R} , for each state q in \mathcal{R} , we add two copies q and q' to \mathcal{R}^{id1} . We keep the initial state the same, and make the final states of \mathcal{R}^{id1} the copies of final states of \mathcal{R} . Each state q with self-loops is turned into a transition reading an alphabet atom $q \xrightarrow{\Delta^{\odot}} q'$ where Δ contains all letters read on self-loops on q . Furthermore, each transition $p \xrightarrow{a} q$ in \mathcal{R} where $p \neq q$ is turned into a transition reading a single atom $p' \xrightarrow{a^{\odot}} q$.

It is easily verified $L(\mathcal{R}^{\text{id1}})$ is the ideal decomposition of $L(\mathcal{R})$ given in equation (1). ◀

4.2 Weighting functions for ideals

Lemma 4.3 tells us that for our given NFA \mathcal{A} , we can construct an NFA \mathcal{R}^{id1} over $\text{atoms}(\Sigma)$ that is acyclic and whose paths correspond to the ideals of $L(\mathcal{A})\downarrow$. Observe that $L(\mathcal{A})$ is directed iff there exists a path π in \mathcal{R}^{id1} such that $L(\mathcal{A})$ is included in the ideal of π : Clearly, if there is such a path, then $L(\mathcal{A})\downarrow$ must equal this ideal and is thus directed. Conversely, if $L(\mathcal{A})$ is directed and $L(\mathcal{A})\downarrow = I_1 \cup \cdots \cup I_n$, where I_1, \dots, I_n are the ideals of the paths in \mathcal{R}^{id1} , then directedness implies that $L(\mathcal{A})\downarrow$ is an ideal. By Lemma 3.2, we must have that $L(\mathcal{A})\downarrow$ coincides with some I_i for $i \in [1, n]$, which lies on some path π . Therefore, to show Lemma 4.1, it remains to pick a path π such that if there is a greatest ideal among the paths in \mathcal{R}^{id1} , then it must lie on π . This is the main challenge in our decision procedures.

Our key insight is that this can be accomplished by a *weighting function*. Roughly speaking, we construct a function μ from the set of ideal representations to \mathbb{N} such that μ is *strictly monotone*, meaning (i) if $\text{Id1}(\alpha_1 \cdots \alpha_n) \subseteq \text{Id1}(\beta_1 \cdots \beta_m)$, then $\mu(\alpha_1 \cdots \alpha_n) \leq \mu(\beta_1 \cdots \beta_m)$ and (ii) if in addition $\text{Id1}(\alpha_1 \cdots \alpha_n) \neq \text{Id1}(\beta_1 \cdots \beta_m)$, then $\mu(\alpha_1 \cdots \alpha_n) < \mu(\beta_1 \cdots \beta_m)$. Moreover, the function will be *additive*, meaning $\mu(\alpha_1 \cdots \alpha_n) = \mu(\alpha_1) + \cdots + \mu(\alpha_n)$. Given such a function, we can find the aforementioned path π by picking one with maximal weight.

The weight function. Strictly speaking, an additive strictly monotone function as above is impossible: It would imply $n \leq \mu(a^{(?)}\cdots a^{(?)}) < \mu(\{a\}^{(*)})$ for every n (here, the product $a^{(?)}\cdots a^{(?)}$ has n factors). Therefore, we will only satisfy strict monotonicity on ideal representations of some maximal length k . Given $k \in \mathbb{N}$, the (k) -weight of an ideal representation $\alpha_1 \cdots \alpha_n$ is defined as $\mu_k(\alpha_1 \cdots \alpha_n) = \sum_{i=1}^n \mu_k(\alpha_i)$ where for each atom α :

$$\mu_k(\alpha) = \begin{cases} 1, & \text{if } \alpha \text{ is a single atom} \\ (k+1)^{|\Delta|}, & \text{if } \alpha = \Delta^{(*)} \text{ for some } \Delta \subseteq \Sigma \text{ where } \Delta \neq \emptyset. \end{cases} \quad (2)$$

The function μ_k is clearly additive. However, it is not strictly monotone: For instance, the products $a^{(*)} \cdot a^{(*)}$ and $a^{(*)} \cdot b^{(*)}$ receive the same weight, but the former represents a strict subset of the latter. In the remainder of this subsection, we will show that μ_k is strictly monotone on reduced ideal representations.

Exponential weights are needed. Before we continue with our algorithm for directedness, let us quickly remark that μ_k cannot be chosen much smaller. If the alphabet Σ is not fixed, then μ_k can have exponential values, because $|\Delta|$ appears in the exponent. In fact, dealing with exponentially large numbers is the reason our upper bound in the general NFA case is AC^1 rather than NL . This raises the question of whether there is a weighting function that is strictly monotone on reduced ideal representations of length k with polynomial values. This is not the case. To see this, consider the exponential-length chain C_ℓ of ideals over the alphabet $\Sigma = \{a_0, a_1, \dots, a_\ell\}$ constructed as follows. For $i \in [1, \ell]$, let $\Sigma_i = \{a_0, \dots, a_i\}$. Set $C_0 := (a_0^{(?)})$. For each $i \in [1, \ell]$, assuming $C_{i-1} = (I_1, \dots, I_t)$, define C_i as

$$C_i := (I_1, \dots, I_t, (\Sigma_{i-1})^{(*)} \cdot a_i^{(?)}) \cdot (I_1, \dots, I_t, (\Sigma_{i-1})^{(*)} \cdot a_i^{(?)}) \cdot (I_1, \dots, I_t).$$

Clearly, the number of ideals in each chain C_ℓ is exponential in ℓ . Moreover, the maximal length k of ideals in C_ℓ is polynomial in ℓ . Since for each i , a_i does not appear in C_{i-1} , we know that (a) the ideal representations in C_ℓ are reduced and (b) the chain C_ℓ is strict. Therefore, any weight function that is strictly monotone on ideal representations of length k maps each ideal in C_ℓ to a distinct value, requiring exponentially high values.

Strict monotonicity. We now prove our strict monotonicity property for μ_k :

► **Proposition 4.4.** *Let $\alpha_1 \cdots \alpha_n$ and $\beta_1 \cdots \beta_m$ be reduced ideal representations of ideals I and J , respectively, with $m, n \leq k$. If $I \subseteq J$ then $\mu_k(\alpha_1 \cdots \alpha_n) \leq \mu_k(\beta_1 \cdots \beta_m)$. Moreover, if $I \subsetneq J$, then $\mu_k(\alpha_1 \cdots \alpha_n) < \mu_k(\beta_1 \cdots \beta_m)$.*

To prove Proposition 4.4, we use Lemma 4.5, which roughly states that inclusion of ideals behaves similarly to the subword ordering: Inclusion is witnessed by some embedding map.

► **Lemma 4.5.** *Let $\alpha_1 \cdots \alpha_n$ and $\beta_1 \cdots \beta_m$ be representations of ideals I and J on Σ , respectively. If $I \subseteq J$, then there exists a function $f: [1, n] \rightarrow [1, m]$ such that*

1. $f(i) \leq f(i+1)$ for all $i \in [1, n-1]$,
2. α_i is contained in $\beta_{f(i)}$ for all $i \in [1, n]$,
3. if β_j is a single atom, then $|f^{-1}(j)| \leq 1$

Proof sketch. For each atom α , we generate a unique word w_α . If $\alpha = a^{(?)}$, then $w_\alpha = a$. If $\alpha = \Delta^{(*)}$, then we fix an order on Σ and for each $\Delta \subseteq \Sigma$ let \mathbf{w}_Δ be a word that contains each letter in Δ once, in the increasing order and set $w_\alpha = \mathbf{w}_\Delta^{m+1}$. We define f so that it sends each i to the j for which $\beta_1 \cdots \beta_j$ is the shortest prefix for which $w_{\alpha_1} \cdots w_{\alpha_i} \in \text{Idl}(\beta_1 \cdots \beta_j)$. Then f satisfies the premises of Item 1-3. Details can be found in [27, App. B]. ◀

Proof of Proposition 4.4. For the given ideal representations $\alpha_1 \cdots \alpha_n$ and $\beta_1 \cdots \beta_m$, let $f: [1, n] \rightarrow [1, m]$ be the embedding function introduced in Lemma 4.5.

▷ **Claim.** For all $j \in [1, m]$, $\mu_k(\beta_j) \geq \sum_{i \in f^{-1}(j)} \mu_k(\alpha_i)$

Proof of claim. If β_j is a single atom, then by Item 3, there exists at most one α_i embedded in β_j and by Item 2, β_j contains α_i ; thus $\alpha_i = \beta_j$ is a single atom. In this case, $\mu_k(\beta_j) = \mu_k(\alpha_i) = 1$. Otherwise, $\beta_j = \Delta^{\otimes}$. By Item 1, the elements of $f^{-1}(j)$ have to be consecutive numbers, i.e. $f^{-1}(j) = [i_1, i_2]$. Then f embeds $\alpha_{i_1}, \dots, \alpha_{i_2}$ in β_j . Since $\alpha_1 \cdots \alpha_n$ is reduced, each pair α_i, α_{i+1} is non-absorptive. Since they are all contained in β_j , either $|f^{-1}(j)| = 1$, or for all $i \in [i_1, i_2]$, $|\alpha_i| < |\beta_j|$ where $|\alpha_i| = 0$ if α_i is a single atom; otherwise it is the size of the alphabet of α_i . In the case $|f^{-1}(j)| = 1$, since the only atom in $f^{-1}(j)$ is contained in β_j , the claim trivially holds. In the latter case, the inequality (3)

$$\sum_{i \in f^{-1}(j)} \mu_k(\alpha_i) \leq n \cdot (k+1)^{|\Delta|^{-1}} < (k+1)^{|\Delta|} = \mu_k(\beta_j) \quad (3)$$

follows from the fact that f can embed at most n many atoms into β_j and that $n \leq k$. ◁

$\mu_k(\beta_1 \cdots \beta_m) \geq \mu_k(\alpha_1 \cdots \alpha_n)$ follows from the claim due to the weight of an ideal representation being defined additively. This concludes the first part of the proof.

Assume $I \subsetneq J$. Then there exists an α_i strictly contained in $\beta_{f(i)}$, or f is not surjective. In the latter case, equation (4) follows from our previous argument.

$$\mu_k(\alpha_1 \cdots \alpha_n) < \mu_k(\beta_1 \cdots \beta_m) \quad (4)$$

In the former case, $\beta_{f(i)}$ is an alphabet atom, otherwise it cannot strictly contain α_i . If α_i is a single atom, (4) follows from $\mu_k(\alpha_i) = 1$. If it is an alphabet atom, (4) follows from (3). ◀

4.3 Reducing ideals

We will apply the weighting function with k being an upper bound on the path length in \mathcal{R}^{id1} (e.g. the number of states). We have seen that the weighting function is strictly monotone on *reduced* ideal representations. Therefore, if all paths in \mathcal{R}^{id1} had reduced ideals, we could prove Lemma 4.1 by picking the path with the largest weight. This is because, if I_1, \dots, I_n are the ideals on paths of \mathcal{R}^{id1} and I_m has maximal μ_k among them, then Proposition 4.4 implies that $L(\mathcal{A})$ is directed if and only if $L(\mathcal{A}) \subseteq I_m$: Here, the “if” is trivial. Conversely, if $L(\mathcal{A})$ is directed, then $L(\mathcal{A}) \downarrow = I_1 \cup \dots \cup I_n$ is an ideal and hence $I_1 \cup \dots \cup I_n = I_i$ for some i by Lemma 3.2. But then we must have $I_i = I_m$, because $I_m \subseteq I_i$ by the choice of I_i , and if I_i were a strict superset of I_m , $\mu_k(I_m)$ would not be maximal. Thus, $L(\mathcal{A}) \subseteq I_m$.

Thus, our next task is to transform \mathcal{R}^{id1} so as to make all ideal representations reduced:

► **Lemma 4.6.** *Given any partially ordered NFA \mathcal{R} on the finite alphabet Σ , one can compute in NL an acyclic NFA \mathcal{R}^{red} over some polynomial-sized alphabet $\Gamma \subseteq \text{atoms}(\Sigma)$ such that $\text{Id1}(L(\mathcal{R}^{\text{red}})) = L(\mathcal{R}) \downarrow$ and the ideal representations \mathcal{R}^{red} accepts are reduced.*

Reducing an individual ideal representation is easy: repeatedly merge consecutive atoms, as briefly sketched in Lemma 3.1. Reducing *all* ideal representations accepted by an NFA at the same time is not obvious: For example, we cannot just merge two transitions $p \xrightarrow{\{a\}^{\otimes}} q \xrightarrow{a^{\otimes}} r$, since each of them might be needed for other paths. We achieve this using transducers.

Transducers. A transducer is a tuple $\mathcal{T} = \langle Q, \Gamma^i, \Gamma^o, t^0, F, E \rangle$ where Q is a finite set of states, Γ^i and Γ^o are finite (input and output) alphabets, $t^0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $E \subseteq Q \times (\Gamma^i \cup \{\varepsilon\}) \times (\Gamma^o \cup \{\varepsilon\}) \times Q$ is the transition relation. Each transition, reads a letter (or ε) from the input alphabet, writes a letter (or ε) from the output alphabet and moves to a new state. A sequence $r = (q_1, a_1, b_1, q_2)(q_2, a_2, b_2, q_3) \cdots (q_m, a_m, b_m, q_{m+1})$ is called a run of \mathcal{T} if each (q_i, a_i, b_i, q_{i+1}) is in E for $i \in [1, m]$ with $t^0 = q_1$ and $q_{m+1} \in F$. For such a run, let the projection of the transitions to the input (similarly, output) alphabet be denoted by $\text{inp}(r)$ (similarly, $\text{out}(r)$). That is, for the r defined above $\text{inp}(r)$ is the subword of $a_1 a_2 \dots a_m$ and $\text{out}(r)$ is the subword of $b_1 b_2 \dots b_m$. Then, the set of $(\text{inp}(r), \text{out}(r))$ over runs r of \mathcal{T} is called the language of \mathcal{T} , is denoted by $L(\mathcal{T})$, and defines a rational relation on $\Gamma^i \times \Gamma^o$. For a set $Y \subseteq (\Gamma^i)^*$, $\mathcal{T}(Y)$ denotes the set of words \mathcal{T} outputs upon a run from Y , i.e. $\mathcal{T}(Y) = \{b \mid a \in Y \text{ and } (a, b) \in L(\mathcal{T})\}$.

Composition of two transducers is again a transducer [15, 44].

Left- and right-reduced representations. We will later define two transducers $\mathcal{T}_{\mathcal{L}}$ and $\mathcal{T}_{\mathcal{R}}$ the composition of which will take an ideal representation $\alpha_1 \cdots \alpha_n$ produced by \mathcal{R}^{id1} and return an equivalent reduced ideal representation $\beta_1 \cdots \beta_m$ with $m \leq n$. In particular, $\mathcal{T}_{\mathcal{L}}$ will turn $\alpha_1 \cdots \alpha_n$ into an equivalent ideal representation that is *left-reduced*, and $\mathcal{T}_{\mathcal{R}}$ will turn it into an equivalent ideal representation that is *right-reduced*. *Left- and right-reducedness* are defined as follows. An ideal representation $\alpha_1 \cdots \alpha_n$ is called *left-reduced* if for all $i \in [1, n-1]$, α_i does not absorb α_{i+1} . Similarly, it is called *right-reduced* if α_{i+1} does not absorb α_i . Clearly, if a representation is both left- and right-reduced, then it is reduced.

Building the transducers. Intuitively, the transducer $\mathcal{T}_{\mathcal{L}}$ scans an ideal representation and after reading its first alphabet atom Δ^{\otimes} , it outputs Δ^{\otimes} , but then skips (i.e. reads without producing output) all atoms that are absorbed by Δ^{\otimes} . Formally, we have $\mathcal{T}_{\mathcal{L}} = \langle Q_{\mathcal{L}}, \Gamma_{\mathcal{L}}^i, \Gamma_{\mathcal{L}}^o, t_{\mathcal{L}}^0, F_{\mathcal{L}}, E_{\mathcal{L}} \rangle$. $Q_{\mathcal{L}}$ contains a state corresponding to each alphabet atom in $\Gamma \subseteq \text{atoms}(\Sigma)$ (as given in Lemma 4.3), with a new initial state $t_{\mathcal{L}}^0$ and a state t_1 for all of the single atoms. We set $\Gamma_{\mathcal{L}}^i = \Gamma_{\mathcal{L}}^o = \Gamma$. Let Φ be a mapping from Γ to $Q_{\mathcal{L}}$, which sends each alphabet atom to its corresponding state, and each single atom to t_1 . $F_{\mathcal{L}} = Q_{\mathcal{L}} \setminus \{t_{\mathcal{L}}^0\}$ and $E_{\mathcal{L}}$ is defined as follows;

- For all $\alpha \in \Gamma$, $(t_{\mathcal{L}}^0, \alpha, \alpha, \Phi(\alpha))$ and $(t_1, \alpha, \alpha, \Phi(\alpha))$ are in $E_{\mathcal{L}}$,
- For each $t \in Q_{\mathcal{L}} \setminus \{t_{\mathcal{L}}^0, t_1\}$ and each atom $\alpha \in \Gamma$, if t (as an alphabet atom) does not absorb α (as an atom), then $(t, \alpha, \alpha, \Phi(\alpha))$ is in $E_{\mathcal{L}}$.
- For each $t \in Q_{\mathcal{L}} \setminus \{t_{\mathcal{L}}^0, t_1\}$ and each atom $\alpha \in \Gamma$, if t absorbs α , then $(t, \alpha, \varepsilon, t)$ is in $E_{\mathcal{L}}$.

It is easy to see that for an ideal representation $\alpha_1 \cdots \alpha_n$, $\mathcal{T}_{\mathcal{L}}(\alpha_1 \cdots \alpha_n)$ is left-reduced and represents the same ideal. Clearly the size of $Q_{\mathcal{L}}$, as well as the sizes of the alphabets is polynomial. Furthermore, for a word w and all $(\text{inp}(w), \text{out}(w))$, $|\text{out}(w)| \leq |\text{inp}(w)|$.

Reversing the edges and flipping the initial and final states of $\mathcal{T}_{\mathcal{L}}$ we obtain the reverse transducer $\mathcal{T}_{\mathcal{R}}$. It can be inductively shown that for any ideal representation $\alpha_1 \cdots \alpha_n$, $\mathcal{T}_{\mathcal{R}}(\alpha_1 \cdots \alpha_n)$ is right-reduced. To show Lemma 4.6, we will apply the composition $\mathcal{T}_{\mathcal{L}} \circ \mathcal{T}_{\mathcal{R}}$ to $L(\mathcal{R}^{\text{id1}})$. Here, we need to show that applying $\mathcal{T}_{\mathcal{L}}$ after $\mathcal{T}_{\mathcal{R}}$ does not spoil right-reducedness:

► **Lemma 4.7.** *If $\alpha_1 \cdots \alpha_n$ is right-reduced, then $\mathcal{T}_{\mathcal{L}}(\alpha_1 \cdots \alpha_n)$ is also right-reduced.*

Proof. Since $\alpha_1 \cdots \alpha_n$ is right-reduced, for $i \in [1, n]$, α_{i+1} does not absorb α_i . By construction, $\mathcal{T}_{\mathcal{L}}(\alpha_1 \cdots \alpha_n)$ is a subword of $\alpha_1 \cdots \alpha_n$, say $\alpha_{i_1} \cdots \alpha_{i_k}$. We show that for all $j \in [1, k]$, $\alpha_{i_{j+1}}$ does not absorb α_{i_j} . Let $i_j = k$ and $i_{j+1} = k'$. If $k' = k+1$, then the claim follows from the right-reducedness of $\alpha_1 \cdots \alpha_n$. Otherwise, α_k absorbs all atoms between itself and $\alpha_{k'}$. In particular it absorbs $\alpha_{k'-1}$. Since $\alpha_{k'}$ does not absorb $\alpha_{k'-1}$, it cannot absorb α_k . ◀

Thus, by applying $\mathcal{T}_{\mathcal{L}} \circ \mathcal{T}_{\mathcal{R}}$ to $L(\mathcal{R}^{\text{id1}})$, we obtain an NFA that reads ideal representations of the same set of ideals (i.e. $L(\mathcal{A})\downarrow$) and every ideal representation is reduced. The resulting NFA \mathcal{R}^{red} can be computed in NL. For details of the construction, see [27, Corollary B.5].

4.4 Deciding directedness

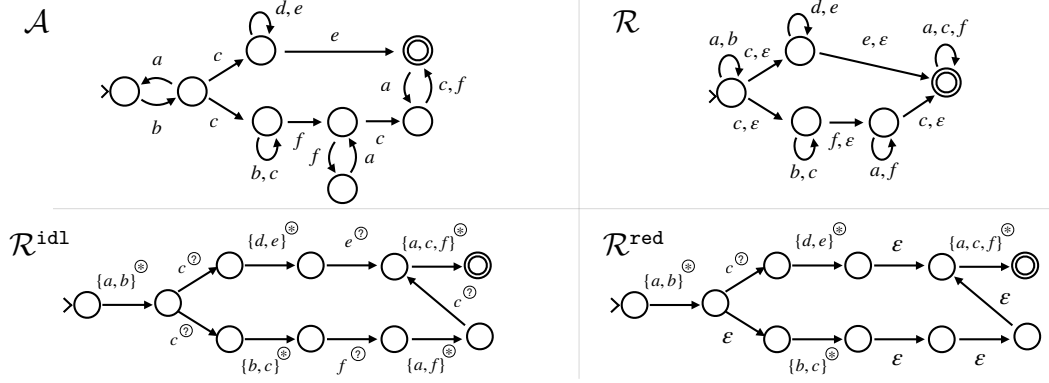
We now present our algorithms to decide directedness for a given NFA. We first complete the proof of Lemma 4.1. For this, in light of Lemma 4.6 and Proposition 4.4, it remains to compute a path of \mathcal{R}^{red} of maximal weight.

Computing the maximal weight. It is well known that the maximum weight path problem can be reduced to matrix multiplication over the max-plus semiring [3, Lemma 5.11]. This yields AC^1 (resp. NL) algorithms for binary (unary) encoded weights [22]. For completeness sake we provide a proof of this fact. Let $\mathcal{R}^{\text{red}} = (Q^{\text{red}}, \Gamma, \delta^{\text{red}}, q_0^{\text{red}}, F^{\text{red}})$ be the NFA from Lemma 4.6. We may assume \mathcal{R}^{red} has a unique final state q_f^{red} , is acyclic except for an ε self-loop in q_f^{red} , and between each pair of states, there is at most one edge. The goal is to compute, for each state q , the maximal weight of any path starting in q .

Let $m = |Q^{\text{red}}|$. Since \mathcal{R}^{red} is acyclic apart from the self-loop on q_f^{red} , any ideal representation accepted by \mathcal{R}^{red} has length $\leq m$. Observe that due to the ε -loop on q_f^{red} , every ideal $\alpha_1 \cdots \alpha_n \in L(\mathcal{R}^{\text{red}})$ is read on some path of length *exactly* m . We want to find an accepted path with the maximum summation of m -weights of each transition (recall that $\mu_m(\varepsilon) = 0$). To do so, we fix an order $\{q_1, \dots, q_m\}$ on the states of Q^{red} such that $q_1 := q_0^{\text{red}}$ and $q_m := q_f^{\text{red}}$ and construct a $m \times m$ -matrix \mathcal{M} , the elements of which takes values from the *max-plus semiring* $(\mathbb{N} \cup \{-\infty\}, +, \max, 0, -\infty)$. For each $i, j \in [1, m]$, we set $\mathcal{M}(i, j)$ to (i) $\mu_m(x)$, if there is an edge $(q_i, x, q_j) \in \delta^{\text{red}}$ with $x \in \text{atoms}(\Sigma) \cup \{\varepsilon\}$, (ii) $-\infty$, otherwise. We can now apply the standard fact from weighted automata that for every $n \geq 0$, in the matrix power \mathcal{M}^n , the entry (i, j) is the maximum weight of all paths of length exactly n from q_i to q_j [3]. Therefore, the largest weight among all paths from q_s to q_m is the entry (s, m) in the matrix power \mathcal{M}^m . A single matrix product can be computed in AC^0 since binary addition and the maximum of multiple numbers can be computed in AC^0 . Moreover, for n given in unary, a matrix power \mathcal{M}^n can be computed in AC^1 by repeated squaring: One writes $n = \sum_{i=0}^{\ell} b_i 2^i$ with $b_0, \dots, b_{\ell} \in \{0, 1\}$ and computes $\mathcal{M}'_0 = \mathcal{M}^{b_{\ell}}$, $\mathcal{M}'_i = (\mathcal{M}'_{i-1})^2 \cdot \mathcal{M}^{b_{\ell-i}}$, yielding $\mathcal{M}^n = \mathcal{M}'_{\ell}$. Thus, by applying the (constant depth) AC^0 circuit for matrix multiplication $\ell \in O(\log n)$ times, we obtain a circuit of logarithmic depth. In particular, we can compute \mathcal{M}^m , and hence the maximal path weights $\mathcal{M}^m(s, m)$, in AC^1 .

In case the alphabet is fixed, we compute the maximal weights in NL. Observe that in this case, all weights $\mu_m(\alpha)$ for atoms $\alpha \in \text{atoms}(\Sigma)$ have $\mu_m(\alpha) \leq (m+1)^{|\Sigma|}$, which is polynomial as $|\Sigma|$ is constant. In particular, all the maximal path weights from q_s to q_m , are bounded by $m \cdot (m+1)^{|\Sigma|}$ and can be stored in logarithmic space. Thus we can proceed as follows. Given $s \in [1, m]$, for every $\ell = m \cdot (m+1)^{|\Sigma|}, \dots, 0$, we decide in NL whether there exists a path of weight ℓ from q_s to q_m . If so, then ℓ is the maximal weight. Since $\text{NL} = \text{coNL}$, we can also determine the non-existence of such a path and continue with $\ell - 1$.

Computing a maximal-weight ideal representation. We have now computed, for each s , the maximal weight M_s of any path from q_s to the final state q_m . For Lemma 4.1, we now need to compute in NL a path from q_1 to q_m of maximal weight. Here, it is important that this computation only depends on the input (even though our NL computation is non-deterministic). Starting from q_1 , we successively compute the next transition in our path. If q_i is the current state, then we compute the next state q_j as follows. We compute



■ **Figure 1** Initial automaton \mathcal{A} on alphabet $\{a, b, c, d, e, f\}$ and the corresponding \mathcal{R} , \mathcal{R}^{id1} and \mathcal{R}^{red} are depicted. The initial states of the automata are marked with a half arrow sign and the final states are encircled. Since \mathcal{R}^{red} has 10 states all ideal representations in $L(\mathcal{R}^{\text{red}})$ contain ≤ 10 atoms. Therefore, m is set to 10 and we calculate the maximal 10-weight ideal, which is $I = \text{Id1}(\{a, b\}^{\otimes} \cdot c^{\otimes} \cdot \{d, e\}^{\otimes} \cdot \{a, c, f\}^{\otimes})$ with the 10-weight $11^3 + 2 \cdot 11^2 + 1$. $L(\mathcal{A}) \not\subseteq I$ witnessed by $cb \in L(\mathcal{A}) \setminus I$; proving that $L(\mathcal{A})$ is not directed.

M as the maximal M_ℓ , where q_ℓ ranges over all states reachable in one step from q_i . Then, we pick the smallest j with $M_j = M$. This way, we successively output a path of maximal weight, such that the path only depends on the input. This completes Lemma 4.1.

Deciding directedness. The upper bounds in Theorems 2.1 and 2.2 follow by applying Lemma 4.1 to obtain an ideal $\alpha_1 \cdots \alpha_n$, either in AC^1 if Σ is part of the input, or in NL for fixed Σ . Finally, checking whether $L(\mathcal{A}) \subseteq \text{Id1}(\alpha_1 \cdots \alpha_n)$ can be done in NL [50].

5 Solution on Context-free Languages

We now prove Theorem 2.3. As in Section 4, the upper bound uses the weighting function to compute a candidate ideal in $L(G) \downarrow$. However, the ideal representation may be exponentially long and will thus be compressed by a straight-line program. For the lower bound, the key idea is to employ a construction from [12] to compute a compressed ideal that contains all words of some length N (given in binary), except a particular word specified as an SLP.

5.1 Deciding directedness of context-free languages

We begin with the PSPACE upper bound, which requires some terminology. A *context-free grammar* (CFG) is a tuple $G = \langle N, \Sigma, P, S \rangle$ where N is the finite set of *nonterminals*, Σ is the finite set of *terminals*, or the finite *alphabet*, $S \in N$ is the *start nonterminal* and $P \subseteq N \times (N \cup \Sigma)^*$ is the finite set of *productions*. We use the arrow notation to denote productions. $A \rightarrow w$ denotes $(A, w) \in P$. We write $w \rightarrow^* w'$ for some $w, w' \in (N \cup \Sigma)^*$ to express that w' can be produced by w through a finite sequence of productions.

For $w \in (N \cup \Sigma)^*$, we denote by $L(w) = \{w' \in \Sigma^* \mid w \rightarrow^* w'\}$ all sequences of terminals w can produce and call it the *language of w* . We define the language of a grammar to be the language of its start nonterminal. That is, $L(G) := L(S)$. WLOG we assume that all nonterminals are reachable from S . A CFG is said to be in *Chomsky Normal Form* (CNF), if

all its productions are of the form $A \rightarrow BC$, $A \rightarrow a$ or $S \rightarrow \varepsilon$, where $A, B, C, S \in N$, $a \in \Sigma$ and $B, C \neq S$. It is well known that one can bring a given grammar into CNF in polynomial time. A CFG G is called *acyclic*, if non of its nonterminals produce itself.

A *straight line program (SLP)* is a CFG that produces a single word. Formally, an SLP is a CFG $G = \langle N, \Sigma, P, S \rangle$ where (i) for each $A \in N$, there is exactly one production $A \rightarrow w$ in P , (ii) G is acyclic. We denote the unique word $a_1 \cdots a_n$ an SLP \mathbb{A} produces by $\text{val}(\mathbb{A})$. If the letters of \mathbb{A} belong to $\text{atoms}(\Sigma)$, $\text{val}(\mathbb{A})$ is an ideal representation over Σ . Thus \mathbb{A} is a *compressed ideal representation for $I = \text{Idl}(\text{val}(\mathbb{A}))$* , or shortly *compressed ideal I* .

Our algorithm is analogous to the one for NFAs. First, an analogue of Lemma 4.1:

► **Lemma 5.1.** *There is a polynomial time algorithm that given a non-empty CFG G , computes a compressed ideal $I \subseteq L(G)\downarrow$ such that $L(G)\downarrow$ is directed if and only if $L(G)\downarrow \subseteq I$.*

And given Lemma 5.1, it remains to decide whether $L(G) \subseteq I$:

► **Lemma 5.2.** *Given any CFG G and a compressed ideal I , one can decide in PSPACE whether $L(G) \subseteq I$.*

A grammar of ideals. The remainder of this subsection is devoted to Lemmas 5.1 and 5.2. Analogously to Lemma 4.3, we first transform G into an acyclic grammar G^{id1} that produces ideal representations of an ideal decomposition of $L(G)\downarrow$.

► **Lemma 5.3.** *Given any CFG G in CNF over Σ , one can compute in polynomial time an acyclic CFG G^{id1} over a polynomial-sized alphabet $\Gamma \subseteq \text{atoms}(\Sigma)$ with $\text{Idl}(L(G^{\text{id1}})) = L(G)\downarrow$.*

The procedure is similar to Courcelle's construction [23] (see [27, App. C] for details).

Reducing ideals. The next step is analogous to Lemma 4.6: We want to transform G^{id1} so as to only produce reduced ideal representations. Luckily, we can directly apply the transducers $\mathcal{T}_{\mathcal{L}}$ and $\mathcal{T}_{\mathcal{R}}$ constructed for Lemma 4.6: Since for a given CFL K and a transducer \mathcal{T} , one can compute in polynomial time a grammar for $\mathcal{T}(K)$ [44], we obtain the following:

► **Lemma 5.4.** *Given any CFG G in CNF over alphabet Σ , one can compute in polynomial time an acyclic CFG G^{red} in CNF over some polynomial-sized alphabet $\Gamma \subseteq \text{atoms}(\Sigma)$ such that (i) $\text{Idl}(L(G^{\text{red}})) = L(G)\downarrow$ and (ii) all ideal representations in $L(G^{\text{red}})$ are reduced.*

Similar to Lemma 4.6, we apply $\mathcal{T}_{\mathcal{L}} \circ \mathcal{T}_{\mathcal{R}}$ to $L(G^{\text{id1}})$ (see [27, Lem. C.1]) and convert to CNF.

Calculating the maximum weight ideal. Similar to Section 4, the next step is to compute for each nonterminal A of G^{red} the maximal weight of any ideal representation produced by A . Let $G^{\text{red}} = \langle N^{\text{red}}, \Gamma, P^{\text{red}}, S^{\text{red}} \rangle$ denote the grammar from Lemma 5.4. With the same argument as in Section 4, an ideal of maximal weight will be as desired in Lemma 5.1. We use the weighting function μ_m , where $m = 3 \cdot 2^{2^{|N^{\text{red}}|}}$ is an upper bound on the length of words in G^{red} . A notable difference to Section 4 is that here m is exponential.

For each nonterminal A of G^{red} , we denote by $\mu_m(A)$ the maximal possible weight of any ideal representation generated by A . To calculate $\mu_m(A)$ for each A , we employ a simple dynamic programming approach. We maintain a table T that contains for each nonterminal A a number $T(A) \in \mathbb{N}$, which is the maximal weight of a derivable ideal representation observed so far. We initialize $T(A) = -\infty$ for every A . Then, we set $T(A)$ to the maximal value of $\mu_m(a)$, where a ranges over all $a \in \text{atoms}(\Sigma)$ for which $A \rightarrow a$ is a production. Finally, we perform the following update step. For each nonterminal A , if there is a production $A \rightarrow BC$

such that currently $T(A)$ is smaller than $T(B) + T(C)$, then we update $T(A) := T(B) + T(C)$. It can be shown by induction that after i update steps, $T(A)$ contains the correct value $\mu_m(A)$ for each nonterminal A that has a depth $\leq i$ derivation tree that attains $\mu_m(A)$. When we apply the update step $|N^{\text{red}}|$ times, we arrive at $T(A) = \mu_m(A)$ for every nonterminal A .

Computing the candidate ideal. Given the numbers $\mu_m(A)$, it is easy to prove Lemma 5.1. For each nonterminal A , there must exist a “max-weight” production $A \rightarrow BC$, resp. $A \rightarrow a$, such that $\mu_m(A) = \mu_m(B) + \mu_m(C)$, resp. $\mu_m(A) = \mu_m(a)$. We build a new grammar \mathbb{S} by selecting for each nonterminal A of G^{red} this max-weight production. Then \mathbb{S} contains at most one production for each nonterminal and is thus an SLP. Moreover, it clearly generates an ideal representation $\text{val}(\mathbb{S})$ of maximal weight. We only need to argue that $L(G)$ is directed iff $L(G) \subseteq \text{Idl}(\text{val}(\mathbb{S}))$. As before, the “if” direction is obvious, because $L(G) \subseteq \text{Idl}(\text{val}(\mathbb{S}))$ implies that $L(G)$ is an ideal. Conversely, suppose $L(G)$ is directed and let $L(G)\downarrow = I_1 \cup \dots \cup I_n$ be the ideal decomposition given by $L(G^{\text{red}})$ with $\text{Idl}(\text{val}(\mathbb{S})) = I_i$. Since $L(G)$ is directed, $L(G)\downarrow$ is an ideal and thus $I_1 \cup \dots \cup I_n = I_j$ for some j by Lemma 3.2. In particular, we have $I_i \subseteq I_j$. Moreover, if the inclusion were strict, I_i would not have maximal weight. Hence, $I_i = I_j$ and thus $L(G) \subseteq I_i = \text{Idl}(\text{val}(\mathbb{S}))$ as required.

Deciding directedness. With Lemma 5.1 in hand, it remains to prove Lemma 5.2. Suppose we are given a grammar G and an SLP \mathbb{S} for $I = \text{Idl}(\text{val}(\mathbb{S}))$, and we want to check $L(G) \subseteq \text{val}(\mathbb{S})$. Since this is equivalent to $L(G)\downarrow \subseteq \text{val}(\mathbb{S})$, we first construct G^{red} given in Lemma 5.4. Recall that $L(G^{\text{red}})$ generates representations of ideals of $L(G)\downarrow$. The algorithm guesses an ideal representation in $L(G^{\text{red}})$ whose ideal *does not embed in* I .

We guess an ideal representation generated by G^{red} , atom by atom, via its leftmost derivation. This word can be exponentially long, but we only store one (polynomial-length) path in the derivation tree, leading to the terminal atom that we are currently guessing (see [27, App.D] for an example). While guessing the representation, we simultaneously maintain a (binary encoded) pointer into $\text{val}(\mathbb{S})$. Suppose $\alpha_1 \dots \alpha_{j-1}$ is guessed so far. While α_j is being guessed, the pointer holds the length of the shortest prefix of $\text{val}(\mathbb{S})$, $\alpha_1 \dots \alpha_{j-1}$ embeds in. Let $\text{val}(\mathbb{S})[i]$ denote the i^{th} index of $\text{val}(\mathbb{S})$. If there is an atom $\text{val}(\mathbb{S})[i']$ with $i' \geq i$ (if $\text{val}(\mathbb{S})[i]$ is an alphabet atom) or $i' > i$ (if $\text{val}(\mathbb{S})[i]$ is a single atom) that α_j embeds in, we update the pointer to the smallest such i' . If there is no such atom, the guessed ideal does not embed in I . On the other hand, if $j - 1$ is the last atom guessed, then the guessed ideal embeds in I . Details are in [27]. This establishes Lemma 5.2 and thus Theorem 2.3.

5.2 PSPACE Lower Bound

Let us now come to the lower bound in Theorem 2.3. It remains to show:

► **Lemma 5.5.** *Given a CFG G over $\{0, 1\}$, directedness of $L(G)$ is PSPACE-hard.*

To this end, we reduce from compressed membership in automatic relations. Given two words $u = a_1 \dots a_n, v = b_1 \dots b_n \in \{0, 1\}^*$, their *convolution* is defined as $u \otimes v = (a_1, b_1) \dots (a_n, b_n) \in (\{0, 1\} \times \{0, 1\})^*$. The following was shown in [39, Corollary 8]:

► **Lemma 5.6.** *There exists a regular language $R \subseteq (\{0, 1\} \times \{0, 1\})^*$ such that for given two SLPs \mathbb{A} and \mathbb{B} with $|\text{val}(\mathbb{A})| = |\text{val}(\mathbb{B})|$, deciding $\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B}) \in R$ is PSPACE-hard.*

From Lemma 5.6, we deduce the following:

► **Lemma 5.7.** *Given an SLP \mathbb{B} and a CFG G such that all words in $L(G)$ have length exactly $|\text{val}(\mathbb{B})|$, both over the alphabet $\Sigma = \{0, 1\}$, deciding $\text{val}(\mathbb{B}) \in L(G)$ is PSPACE-hard.*

Proof. We reduce from the PSPACE-complete problem in Lemma 5.6. Let R be the regular language from Lemma 5.6, and let \mathbb{A} and \mathbb{B} be SLPs with $n = |\text{val}(\mathbb{A})| = |\text{val}(\mathbb{B})|$. Observe that $\text{val}(\mathbb{A}) \otimes \text{val}(\mathbb{B}) \in R$ if and only if $\text{val}(\mathbb{B})$ belongs to the language $K = \{w \in \{0, 1\}^n \mid \text{val}(\mathbb{A}) \otimes w \in R\}$. We can construct a context-free grammar G for K in polynomial-time by viewing an automaton for R as a transducer and applying it to the SLP \mathbb{A} . ◀

Now in order to reduce the compressed membership problem $\text{val}(\mathbb{B}) \in L(G)$ in Lemma 5.7 to an inclusion $L(G) \subseteq I$, the key trick is to construct an ideal I that acts like a complement of $\{\text{val}(\mathbb{B})\}$. We expect that this will be of independent interest.

► **Lemma 5.8.** *Given an SLP \mathbb{B} over Σ , one can construct in polynomial time an SLP \mathbb{I} over $\text{atoms}(\Sigma)$ so that $\text{Idl}(\text{val}(\mathbb{B}))$ is infinite and $\text{Idl}(\text{val}(\mathbb{I})) \cap \Sigma^{|\text{val}(\mathbb{B})|} = \Sigma^{|\text{val}(\mathbb{B})|} \setminus \{\text{val}(\mathbb{B})\}$.*

The proof uses a construction from [12]. The authors of the latter were interested in defining languages in the existential fragment of first-order logic over the structure the set Σ^* , ordered by \preceq . In one step [12, Lemma 3.1], given a word $u \in \Sigma^*$, they construct a word $\bar{w} \in \Sigma^*$ such that $\{\bar{w}\} \downarrow \cap \Sigma^{|w|} = \Sigma^{|w|} \setminus \{w\}$. To this end, they write $w = a_1 \cdots a_n$ and define u_i to be a word that contains every letter from Σ , except for a_i . Then, they argue that $\bar{w} = u_1 a_1 \cdots u_{n-1} a_{n-1} u_n$ is as desired. Here, we cannot use \bar{w} directly, because we want I to be infinite. However, we can use a similar construction.

Proof of Lemma 5.8. Suppose $\text{val}(\mathbb{B}) = b_1 \cdots b_n$ and define $I = \text{Idl}(w)$ with

$$w = (\Sigma \setminus \{b_1\})^{\otimes} \cdot b_1^{\otimes} \cdot (\Sigma \setminus \{b_2\})^{\otimes} \cdot b_2^{\otimes} \cdots (\Sigma \setminus \{b_{n-1}\})^{\otimes} \cdot b_{n-1}^{\otimes} \cdot (\Sigma \setminus \{b_n\})^{\otimes}.$$

We first show $I \cap \Sigma^n = \Sigma^n \setminus \{\text{val}(\mathbb{B})\}$. Clearly, $b_1 \cdots b_n \notin I$: Let j_i be length of the shortest prefix of w whose ideal contains $b_1 \cdots b_i$. Clearly $j_1 = 2$, since the first atom $(\Sigma \setminus \{b_1\})^{\otimes}$ does not embed b_1 . Inductively we get $j_2 = 4, \dots, j_{n-1} = 2n - 2$, which leaves only the last atom $(\Sigma \setminus \{b_n\})^{\otimes}$ to embed b_n , but this is not possible. We now prove $\Sigma^n \setminus \{b_1 \cdots b_n\} \subseteq I$. Let $c_1 \cdots c_n \neq b_1 \cdots b_n$ and choose d minimally with $c_d \neq b_d$. Let h_i to be the length of the of shortest prefix of w whose ideal contains $c_1 \cdots c_i$. Then $h_{d-1} = j_{d-1} = 2d - 2$. Since c_d differs from b_d , it embeds in the $(2d - 1)$ -th atom $(\Sigma \setminus \{b_d\})^{\otimes}$, i.e. $h_d = 2d - 1$. The remaining $(n - d)$ -length suffix $c_{d+1} \cdots c_n$ embeds in the $2(n - i)$ length suffix $(\Sigma \setminus \{b_d\})^{\otimes} \cdot b_d^{\otimes} \cdots (\Sigma \setminus \{b_{n-1}\})^{\otimes} \cdot b_{n-1}^{\otimes}$: Indeed, since $(\Sigma \setminus \{b_k\})^{\otimes} \cdot b_k^{\otimes}$ embeds every letter, the aforementioned suffix even embeds every word from Σ^{n-d} .

It remains to be shown that we can compute an SLP for w . Note that w is *almost* a homomorphic image of $\text{val}(\mathbb{B})$. Given SLP \mathbb{B} , we obtain an SLP \mathbb{I}' for w by replacing each production $A \rightarrow b$ with $A \rightarrow (\Sigma \setminus \{b\})^{\otimes} b^{\otimes}$. Then, $\text{val}(\mathbb{I}') = (\Sigma \setminus \{b_1\})^{\otimes} \cdot b_1^{\otimes} \cdots (\Sigma \setminus \{b_n\})^{\otimes} \cdot b_n^{\otimes}$. To get w exactly, we construct \mathbb{I} so that $\text{val}(\mathbb{I})$ is $\text{val}(\mathbb{I}')$ without its last letter. It is easy to see that this can be done in polynomial time (it follows, e.g. from [43, Theorem 7.1]). ◀

We are now ready to prove Lemma 5.5. Given a grammar G and an SLP \mathbb{B} as in Lemma 5.7, we use Lemma 5.8 to construct an SLP \mathbb{I} with $\text{Idl}(\mathbb{I}) \cap \Sigma^{|\text{val}(\mathbb{B})|} = \Sigma^{|\text{val}(\mathbb{B})|} \setminus \{\text{val}(\mathbb{B})\}$. Observe that now $\text{val}(\mathbb{B}) \notin L(G)$ if and only if $L(G) \subseteq \text{Idl}(\mathbb{I})$. Moreover, observe that $L(G)$ is finite and $\text{Idl}(\mathbb{I})$ is infinite. Therefore, the following lemma implies that $\text{val}(\mathbb{B}) \notin L(G)$ if and only if the context-free language $L(G) \cup \text{Idl}(\text{val}(\mathbb{I}))$ is directed, yielding PSPACE-hardness.

► **Lemma 5.9.** *For finite $L \subseteq \Sigma^*$ and an infinite ideal I , we have $L \subseteq I$ iff $L \cup I$ is directed.*

Proof. Clearly, if $L \subseteq I$, then $L \cup I = I$ is an ideal and thus directed. Conversely, suppose $L \cup I$ is directed. Consider an ideal decomposition $L \downarrow = I_1 \cup \dots \cup I_n$. Here, all I_i are finite since L is finite. Since $L \cup I$ is directed, the downward closure $(L \cup I) \downarrow$ must coincide with one of the ideals in the ideal decomposition $(L \cup I) \downarrow = I_1 \cup \dots \cup I_n \cup I$. Since I is the only infinite ideal, this is only possible with $(L \cup I) \downarrow = I$. In particular, $L \subseteq I$. ◀

6 Downward closure comparison

Regular languages. We now show how to obtain Theorem 2.4 as a byproduct of our results. For the upper bounds, we use Lemma 4.1 to compute, in AC^1 resp. NL , a candidate ideal I_i for each input language L_i . Since L_1 and L_2 are directed, we must have $L_i \downarrow = I_i$ and we can decide in deterministic logspace whether $I_1 = I_2$ [50]. This yields an NL upper bound for fixed alphabets and an AC^1 upper bound for arbitrary alphabets.

For the NL lower bound, we reduce from emptiness of NFAs: Given an NFA \mathcal{A} , we may assume that all transitions are labeled with the empty word ε . We take an NFA \mathcal{A}' that just accepts $\{\varepsilon\}$. Then $L(\mathcal{A}) \neq \emptyset$ if and only if $L(\mathcal{A}') \downarrow = L(\mathcal{A}) \downarrow$, proving NL -hardness.

Context-free languages. We now show Theorem 2.5. We first use Lemma 5.1 to compute an SLP \mathbb{A}_i for a candidate ideal for each L_i . By directedness, $L_i \downarrow = \text{Idl}(\text{val}(\mathbb{A}_i))$. To decide $\text{Idl}(\text{val}(\mathbb{A}_1)) = \text{Idl}(\text{val}(\mathbb{A}_2))$, we use the fact that two reduced ideal representations yield the same ideal if and only if they are syntactically identical [33, Theorem 6.1.12]. To check $\text{val}(\mathbb{A}_1) = \text{val}(\mathbb{A}_2)$ we may apply the well-known result of Plandowski [42] that equality of SLPs can be decided in polynomial time (see also [40]).

For the P lower bound, we reduce from emptiness of CFL: Given a CFG G , we may assume that the only word G can produce is the empty word (otherwise, just replace all occurrences of terminal letters with the empty word). We also take a grammar G' with $L(G') = \{\varepsilon\}$. Then clearly, $L(G) \neq \emptyset$ if and only if $L(G) \downarrow = L(G') \downarrow$, yielding P -hardness.

7 Conclusion

We have initiated the investigation of the directedness problem and determined the exact complexity for context-free languages and for NFAs over fixed alphabets. Over variable alphabets, we show an AC^1 upper bound for NFAs. Despite serious efforts, we leave the exact complexity open. Note that the complexity of directedness is the same for DFAs and NFAs [27, App.F]. Also, the complexity of the maximum weight path problem is not known [22].

The developed techniques could be of independent interest. The idea to analyze ideals by their weights might apply to other procedures for reachability involving ideals [16–18, 25, 26, 36–38]. Furthermore, our PSPACE lower bound can be viewed as progress towards resolving the complexity of the *compressed subword problem*: Our lower bound applies in particular to deciding $L \subseteq I$ for context-free L and a compressed ideal I . Compressed subword, on the other hand, is equivalent to deciding $I \subseteq J$ for compressed ideals I, J . As mentioned before, it is a long-standing open problem to close the gap between the PP lower bound and the PSPACE upper bound [39] (see [40] for a survey) for compressed subword.

The surprisingly low complexity of downward closure equivalence (DCE) for directed CFL calls for an investigation of further applications of directed CFL. As previously stated, safety properties of concurrent programs only depend on the downward closure of the participating threads [5, 11, 41]. It is conceivable that deciding safety [5, 13] or other notoriously difficult problems such as refinement [10] are more tractable for directed threads as well.

References

- 1 Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318. Springer, 1998. doi:10.1007/BFb0028754.
- 2 Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods Syst. Des.*, 25(1):39–65, 2004. doi:10.1023/B:FORM.0000033962.51898.1a.
- 3 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- 4 Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993. doi:10.1016/0304-3975(93)90252-0.
- 5 Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Log. Methods Comput. Sci.*, 7(4), 2011. doi:10.2168/LMCS-7(4:4)2011.
- 6 Mohamed Faouzi Atig, Dmitry Chistikov, Piotr Hofman, K. Narayan Kumar, Prakash Saivasan, and Georg Zetsche. The complexity of regular abstractions of one-counter languages. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 207–216. ACM, 2016. doi:10.1145/2933575.2934561.
- 7 Aistis Atminas and Vadim V. Lozin. Deciding atomicity of subword-closed languages. In Volker Diekert and Mikhail V. Volkov, editors, *Developments in Language Theory - 26th International Conference, DLT 2022, Tampa, FL, USA, May 9-13, 2022, Proceedings*, volume 13257 of *Lecture Notes in Computer Science*, pages 69–77. Springer, 2022. doi:10.1007/978-3-031-05578-2_5.
- 8 Georg Bachmeier, Michael Luttenberger, and Maximilian Schlund. Finite automata for the sub- and superword closure of CFLs: Descriptive and computational complexity. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lecture Notes in Computer Science*, pages 473–485. Springer, 2015. doi:10.1007/978-3-319-15579-1_37.
- 9 David Barozzini, Lorenzo Clemente, Thomas Colcombet, and Pawel Parys. Cost automata, safe schemes, and downward closures. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPICs*, pages 109:1–109:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.109.
- 10 Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. Checking refinement of asynchronous programs against context-free specifications. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 110:1–110:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ICALP.2023.110.
- 11 Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. Context-bounded analysis of concurrent programs (invited talk). In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 3:1–3:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ICALP.2023.3.
- 12 Pascal Baumann, Moses Ganardi, Ramanathan S. Thinniyam, and Georg Zetsche. Existential definability over the subword ordering. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPICs*, pages 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.STACS.2022.7.

- 13 Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. The complexity of bounded context switching with dynamic thread creation. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPICs*, pages 111:1–111:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.111.
- 14 Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. Context-bounded verification of thread pools. *Proc. ACM Program. Lang.*, 6(POPL):1–28, 2022. doi:10.1145/3498678.
- 15 Jean Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.
- 16 Michael Blondin, Alain Finkel, and Jean Goubault-Larrecq. Forward analysis for WSTS, part III: Karp-Miller trees. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPICs*, pages 16:1–16:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.FSTTCS.2017.16.
- 17 Michael Blondin, Alain Finkel, and Pierre McKenzie. Well behaved transition systems. *Log. Methods Comput. Sci.*, 13(3), 2017. doi:10.23638/LMCS-13(3:24)2017.
- 18 Michael Blondin, Alain Finkel, and Pierre McKenzie. Handling infinitely branching well-structured transition systems. *Inf. Comput.*, 258:28–49, 2018. doi:10.1016/j.ic.2017.11.001.
- 19 Manuel Bodirsky, Jakub Rydval, and André Schrottenloher. Universal Horn sentences and the joint embedding property. *Discret. Math. Theor. Comput. Sci.*, 23(2), 2021. doi:10.46298/dmtcs.7435.
- 20 Samuel Braunfeld. The undecidability of joint embedding and joint homomorphism for hereditary graph classes. *Discret. Math. Theor. Comput. Sci.*, 21(2), 2019. doi:10.23638/DMTCS-21-2-9.
- 21 Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 96–105. ACM, 2016. doi:10.1145/2933575.2934527.
- 22 Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Inf. Control.*, 64(1-3):2–21, 1985. doi:10.1016/S0019-9958(85)80041-3.
- 23 Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, January 1991.
- 24 Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular separability of well-structured transition systems. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPICs*, pages 35:1–35:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.CONCUR.2018.35.
- 25 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part II: complete WSTS. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikolettseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2009. doi:10.1007/978-3-642-02930-1_16.
- 26 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: completions. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPICs*, pages 433–444. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. doi:10.4230/LIPICs.STACS.2009.1844.

- 27 Moses Ganardi, Irmak Sağlam, and Georg Zetsche. Directed regular and context-free languages, 2024. [arXiv:2401.07106](https://arxiv.org/abs/2401.07106).
- 28 Jean Goubault-Larrecq, Simon Halfon, Prateek Karandikar, K. Narayan Kumar, and Philippe Schnoebelen. The ideal approach to computing closed subsets in well-quasi-ordering. *CoRR*, abs/1904.10703, 2019. [arXiv:1904.10703](https://arxiv.org/abs/1904.10703).
- 29 Jean Goubault-Larrecq and Sylvain Schmitz. Deciding piecewise testable separability for regular tree languages. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 97:1–97:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.97.
- 30 Peter Habermehl, Roland Meyer, and Harro Winkelmann. The downward-closure of Petri net languages. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 466–477. Springer, 2010. doi:10.1007/978-3-642-14162-1_39.
- 31 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163. ACM, 2016. doi:10.1145/2837614.2837627.
- 32 Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969. doi:10.1016/S0021-9800(69)80111-0.
- 33 Simon Halfon. *On Effective Representations of Well Quasi-Orderings*. PhD thesis, Université Paris Saclay, 2018. URL: <https://theses.hal.science/tel-01945232>.
- 34 P. Jullien. Sue un théorème d’extension dans la théorie des mots. *C. R. Acad. Sci. Paris, Ser. A*, 266:851–854, 1968.
- 35 Mustapha Kabil and Maurice Pouzet. Une extension d’un théorème de p. jullien sur les âges de mots. *RAIRO Theor. Informatics Appl.*, 26:449–482, 1992. doi:10.1051/ita/1992260504491.
- 36 Ranko Lazic and Sylvain Schmitz. The ideal view on Rackoff’s coverability technique. *Inf. Comput.*, 277:104582, 2021. doi:10.1016/j.ic.2020.104582.
- 37 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.
- 38 Jérôme Leroux and Sylvain Schmitz. Ideal decompositions for vector addition systems (invited talk). In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPICs*, pages 1:1–1:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.STACS.2016.1.
- 39 Markus Lohrey. Leaf languages and string compression. *Inf. Comput.*, 209(6):951–965, 2011. doi:10.1016/j.ic.2011.01.009.
- 40 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complex. Cryptol.*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.
- 41 Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. General decidability results for asynchronous shared-memory programs: Higher-order and beyond. *Log. Methods Comput. Sci.*, 18(4), 2022. doi:10.46298/lmcs-18(4:2)2022.
- 42 Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In Jan van Leeuwen, editor, *Algorithms - ESA ’94, Second Annual European Symposium, Utrecht, The Netherlands, September 26-28, 1994, Proceedings*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470. Springer, 1994. doi:10.1007/BFb0049431.
- 43 Saul Schleimer. Polynomial-time word problems. *Commentarii mathematici helvetici*, 83(4):741–765, 2008.

- 44 Jeffrey O. Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, 2008. URL: http://www.cambridge.org/gb/knowledge/isbn/item1173872/?site_locale=en_GB.
- 45 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991. doi:10.1137/0220053.
- 46 Salvatore La Torre, Anca Muscholl, and Igor Walukiewicz. Safety of parametrized asynchronous shared-memory systems is almost always decidable. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1-4, 2015*, volume 42 of *LIPICs*, pages 72–84. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.CONCUR.2015.72.
- 47 Jan van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978.
- 48 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. doi:10.1007/978-3-662-03927-4.
- 49 Georg Zetsche. An approach to computing downward closures. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2015. doi:10.1007/978-3-662-47666-6_35.
- 50 Georg Zetsche. The complexity of downward closure comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 123:1–123:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.123.
- 51 Georg Zetsche. Separability by piecewise testable languages and downward closures beyond subwords. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 929–938. ACM, 2018. doi:10.1145/3209108.3209201.

Online Simple Knapsack with Bounded Predictions

Matthias Gehnen  

RWTH Aachen University, Germany

Henri Lotze¹  

RWTH Aachen University, Germany

Peter Rossmanith  

RWTH Aachen University, Germany

Abstract

In the ONLINE SIMPLE KNAPSACK problem, an algorithm has to pack a knapsack of unit size as full as possible with items that arrive sequentially. The algorithm has no prior knowledge of the length or nature of the instance. Its performance is then measured against the best possible packing of all items of the same instance, over all possible instances.

In the classical model for online computation, it is well known that there exists no constant bound for the ratio between the size of an optimal packing and the size of an online algorithm's packing. A recent variation of the classical online model is that of *predictions*. In this model, an algorithm is given knowledge about the instance in advance, which is in reality distorted by some factor δ that is commonly unknown to the algorithm. The algorithm only learns about the actual nature of the elements of an input once they are revealed and an irrevocable and immediate decision has to be made. In this work, we study a slight variation of this model in which the error term, and thus the range of sizes that an announced item may actually lay in, is given to the algorithm in advance. It thus knows the range of sizes from which the actual size of each item is selected from.

We find that the analysis of the ONLINE SIMPLE KNAPSACK problem under this model is surprisingly involved. For values of $0 < \delta \leq \frac{1}{7}$, we prove a tight competitive ratio of 2. From there on, we are able to prove that there are at least three alternating functions that describe the competitive ratio. We provide partially tight bounds for the whole range of $0 < \delta < 1$, showing in particular that the function of the competitive ratio depending on δ is not continuous.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online problem, Simple Knapsack, Predictions, Machine-Learned Advice

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.37

1 Introduction

The ONLINE (SIMPLE) KNAPSACK problem has received a lot of attention since it has been found to be non-competitive in the classical model for online computation by Marchetti-Spaccamela and Versellis [27].

In this model, an online algorithm receives a finite sequence of requests, one item after another. It has no knowledge of the nature of future items or the length of the sequence, and is tasked to either pack a given item into a knapsack of unit size or to reject it. Every decision is irrevocable. The objective of the algorithm is to maximize the sum of sizes of packed items in the knapsack, without this sum exceeding the knapsack size of 1. Its performance is measured against the best possible packing of the same sequence of items. The worst ratio between this optimal packing and the packing of an algorithm over all possible sequences is then called the *competitive ratio* of the algorithm, as first defined by Sleator and Tarjan [29]. For a more thorough introduction to competitive analysis, we refer to the books by Borodin and El-Yaniv [13] and by Komm [24].

¹ Corresponding author



The classical counterexample to show that even the ONLINE SIMPLE KNAPSACK problem is not competitive, is arguably pathological: An adversary chooses between two instances with an identical prefix of an arbitrarily small first item of size $\varepsilon > 0$. The first instance only contains this small first item, while a second instance contains an item of size 1 as the second item. Depending on whether a given algorithm deterministically discards or packs this first item, the first or the second instance is given to the algorithm, resulting in an unbounded competitive ratio.

Thus, many variations of the problem itself or indeed the classical online model itself have been studied, each aiming to disallow these pathological instances. We will only give an incomplete list of all studied variants. The variations of the problem itself include allowing an online algorithm to pack a slightly larger knapsack than the offline counterpart, called *resource augmentation* by Iwama and Zhang [22] and allowing an algorithm to intermediately store items in a *buffer* of a certain size, as introduced by Han et al. [18]. Thielen, Tiedemann and Westphal [31] studied a model in which the capacity of the knapsack increases step-wise over a given number of discrete time periods. Iwama and Taketomi [21] introduced a variation in which packed items could be removed (also called *preempted*) from the knapsack, not to be packed again. Building on this result, Han and Makino [19] additionally allowed for a limited number of *cuts*, i.e. splitting of items into two sub-items. Zhou, Chakrabarty and Lukose [33] analyzed the online knapsack problem under the assumption that the size of each item is much smaller than the knapsack capacity and the ratio between the value and the weight of an item is bounded within a given range $[L, U]$. Further variations include allowing for randomization and allowing for an oracle to communicate information about the instance via so-called *advice bits*. These variations were studied by Böckenhauer et al. [12]. The recently introduced model by Böckenhauer et al. [11] of *reservation costs* allows to pay a fee for a presented item in order to delay a decision on it for an arbitrary amount of time.

When acting in an online setting, the assumption that the future is completely unknown is often unrealistic. When buying boxes for moving to a different house, one usually has a pretty good idea of what one owns and can then buy a number of boxes on this estimate. When planning to drive a group of friends home, one can already pre-plan a route with the rough knowledge on where these persons actually live.

A common theme is thus that the knowledge of the world and the future is actually not unknown, on the contrary: it is often known but the details are uncertain. If we want to model this kind of behavior, we could thus assume that an instance is already given in advance: all elements are revealed beforehand. When the elements arrive, i.e. we have to actually pack the moving boxes or ask our friends where they live, we learn the *actual* nature of the element. For the sake of a simple model, let us assume that the actual number of elements is correctly given. We also assume that our estimates do not deviate beyond a certain bound, i.e., that the maximum uncertainty is bounded and this bound is known in advance.

2 Problem Definitions and Notation

We will abuse notation by using x_i for both the label of an item and for the size (or gain) of the same item.

► **Definition 1** (The ONLINE SIMPLE KNAPSACK WITH BOUNDED PREDICTIONS Problem). *Given a constant $\delta \in [0, 1]$, called distortion. Given a set of items $I = (x_1, \dots, x_n)$ as a request sequence of items that arrive sequentially. Let $P = (x_1^-, \dots, x_n^-)$ be a sequence of items called the prediction such that $x_i \in [x_i^-, \min(x_i^+, 1)]$, where $x_i^+ := \frac{1+\delta}{1-\delta}x_i^-$. Let K be an*

initially empty set called the knapsack. An algorithm is given P and δ before the first item of I is revealed. At each step $i \in \{1, \dots, n\}$, the item x_i of the request sequence is given. An algorithm then has one of the following options:

- **Pack** If $\sum_{x_k \in K} x_k + x_i \leq 1$, set $K := K \cup x_i$.
- **Reject** Do nothing.

The ONLINE SIMPLE KNAPSACK WITH BOUNDED PREDICTIONS problem (OSKP) is then for an algorithm **ALG** to maximize the sum of items in K , i.e. $\sum_{x_k \in K} x_k$.

We will write $P_{[i,n]}$ to denote the infix of P from index i to n , including both endpoints. When referring to an item x^- , we will sometimes speak of the *minimum (possible) size* of an item, as well as speaking of the *maximum (possible) size* of an item when referring to an item x^+ . For every instance I with predictions P , we define $b^- := \max P$ as the *announced largest item* of the instance. This is of course not necessarily the item of largest *actual* size. If b^- is not unique, we refer to the first of such items regarding the ordering of the prediction sequence.

Note that restricting the maximum *actual* size of items to 1 is necessary to ensure competitiveness, as otherwise the classical counterexample by Marchetti-Spaccamela and Vercellis [27] can be reconstructed by letting $P = (\varepsilon, 1)$ for some $\varepsilon > 0$ and setting the second item to a value larger than one if an algorithm rejects the first item.

Strictly speaking, any competitive ratio given in this work is a function of a fixed value δ . To simplify notation, we will write c for $c(\delta)$, as it is always either clear from the context or not of relevance which concrete δ the competitive ratio refers to.

3 Related Work and Our Contributions

It is important to note the difference between this model and the one of *machine learned advice*, which has recently been called *untrusted predictions* or just *predictions*. The latter has been introduced by Lykouris and Vassilvitskii [26] and works as follows. An algorithm is given a prediction on the input, just as is in our model. However, no bound is given on the error, which is rather treated as a variable and algorithms are designed with the aim of them meeting three characteristics. The aim in algorithm design in this model is that they output an optimal solution when the given prediction is correct (*consistency*), that they perform as well as a regular online algorithm on the problem when the predictions become arbitrarily bad (*robustness*) and that they degrade with increasing unreliability of the prediction (*smoothness*).

The very important difference between this model and ours, the first one, is that the algorithm is given knowledge about the maximum error that it can expect, namely δ , as part of its input. Classically, this error term is unknown and an algorithm is to perform as well as possible. We believe this model to be equally reasonable as that without bounds on the predictions, as it is not unnatural for a person to be sure that their prediction will not be arbitrarily far off from the truth.

The classical prediction model has seen a big influx of results in the past few years, with the model being applied to several different online problems, such as scheduling [25, 14, 7], metric algorithms [2, 3], matching problems [16, 23], spanning tree problems [17, 10] and many more. Our modified model is, however, not our original modification. Azar et al. [5] recently discussed scheduling problems based on this model, where they developed algorithms that can learn about the maximum distortion and make decisions based upon this value. In a

more recent work, they extended their analysis to the scheduling on multiple machines [6], in which they also analyzed the case in which an algorithm is oblivious to the actual distortion of the instance.

In order to avoid confusion, as Azar et al. still speak of *problems with predictions* in their works, we name this specific model *bounded predictions* in the context of this work. We believe that using this slightly modified version of the original prediction model can yield an even more fine-grained way of worst-case analysis, when one can assume that an oracle can only be wrong up to a known, bounded degree.

While we assume that an adversary is able to control both the predicted instance and the actual distortion of the items, there is a related model of *smoothed analysis*, in which an adversary can fix an instance, which is then subject to some random (commonly Gaussian) distortion, or *noise*. This model of an adversary not having complete control over its prepared instance was first made popular when showing that the simplex algorithm runs in expected polynomial time when its input is subjected to such random noise [30]. Since then, there was a large influx of results in this area for a wide range of problems, such as multi-level feedback algorithms [8], analyzing the k -means method [4] for clustering, and especially the 0/1 knapsack problem [9]. The model of smoothed analysis thus gives evidence that the worst-case running time or worst-case approximation ratios often seem to suffer from very specific and limited adversarial inputs which break down if even only a very slight perturbation of the instance is given. This model of an announced instance being perturbed has been studied in slight variations already. The *robust knapsack problem* by Monaci, Pferschy and Serafini [28] is very similar in that it also allow for an uncertain input with a multiplicative factor, but the authors look at *offline* algorithms that see the complete permuted instance at once and are compared to the performance of a non-perturbed instance. Im et al. [20] recently looked at the general knapsack problem, which they study under a model predicting the frequency of items of each size. Angelopoulos, Kamali and Shadkami [1] look at the online bin packing problem with predictions on the frequency of item sizes in the instance. Boyar, Favrholdt and Larsen [15] very recently studied the online simple knapsack problem with predictions, but working with predictions on the *average* size of the items an optimal solution would pack. Xu and Zhang [32] recently studied the simple knapsack problem in a learning-augmented setting, where they design algorithms that are able to learn and use the error of prediction.

We study the behavior of the OSKP problem with a distortion of $0 < \delta < 1$. The difference between the predicted size and the actual size of an item is determined by a relative error.

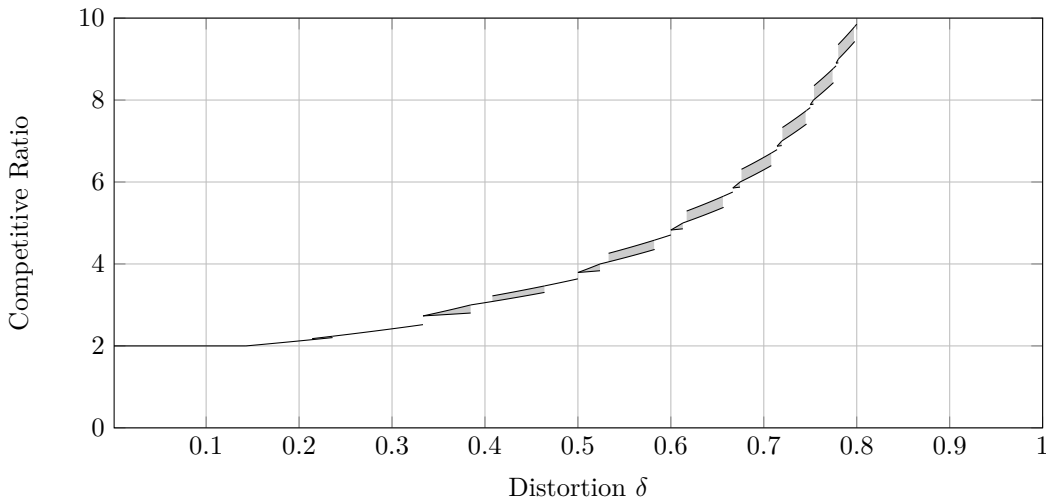
While we are not able to give tight bounds on the competitive ratio for all values of δ between 0 and 1, we are able to carve out the following, partial picture, which is visualized in Figure 1, with the bounds that we prove also given in Table 1. For reasons of readability, we will refrain from stating these terms found in the table in the following text. They will of course be stated in their respective theorems later.

Up to $\delta \leq \frac{1}{7}$, we give a tight bound of 2 on the competitive ratio. From there on, three bounds on the competitive ratio repeat periodically, given a fixed $k \in \mathbf{N}$: Between values of δ between $\frac{k^2+k-1}{k^2+3k+3}$ and δ_k (as defined in Table 1), we are able to prove a tight bound of $\sqrt{(2+k)\frac{1+\delta}{1-\delta}}$. From δ_k to $-1 - k + \sqrt{4k+k^2}$, we are able to prove a non-matching lower and upper bound. For the penultimate segment of $-1 - k + \sqrt{4k+k^2} \leq \delta \leq \frac{1}{k+2}$, we give a matching upper and lower bound. The final segment, which connects to the first segment for the next higher value of k , we are again only able to prove a non-matching pair of bounds. Noticeably, the function of the competitive ratio depending on δ is not continuous between segments two and three.

■ **Table 1** Competitive ratios of the OSKP problem, given a fixed $k \in \mathbb{N}$ and distortion δ .

Distortion	Competitive ratio
$0 < \delta \leq \frac{1}{7}$	2
$\frac{k^2+k-1}{k^2+3k+3} < \delta \leq \delta_k$	$\sqrt{(2+k)\frac{1+\delta}{1-\delta}}$
$\delta_k \leq \delta < -1-k+\sqrt{4k+k^2}$	$\geq \sqrt{(2+k)\frac{1+\delta}{1-\delta}} \leq \frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$
$-1-k+\sqrt{4k+k^2} \leq \delta \leq \frac{k}{k+2}$	$\frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$
$\frac{k}{k+2} < \delta < \frac{k^2+k-1}{k^2+3k+3}$	$\geq \frac{1+k}{2} - \frac{\sqrt{(-1+\delta)(-5-k(2+k)+\delta(-1+k)(3+k))}}{-2+2\delta} \leq \frac{1+\delta+\sqrt{5+2\delta-3\delta^2}}{2-2\delta}$

where $\delta_k := \frac{12k+8}{32^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}}} + \frac{1}{3}2^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}} + 5/3$



■ **Figure 1** Competitive ratio of the OSKP problem, depending on the distortion δ . The gray areas signify a gap between the best known lower and upper bounds.

To the best of our knowledge, our work is the first one systematically analyzing a knapsack problem in the setting of predictions, where the distortion (or error) is linked to the size of the presented items. We show that the additional knowledge of a bound on the distortion can significantly help an online algorithm to choose an appropriate strategy, as e.g. the algorithms used for the repeating segments two and three differ in their nature. Thus, a much more fine-grained analysis is possible. Noticeably, even for an uncertain prediction where the actual value of an item may deviate by a factor of three from its prediction, one can guarantee a competitive ratio of less than four. Yet, even for arbitrarily small distortions, no competitive ratio better than two can be achieved.

The remainder of this work is structured as follows: We analyze the model of relative errors by first proving a number of helpful structural lemmata in Section 4, followed by upper bound proofs in Section 5 and lower bounds in Section 6. We conclude with a short discussion of open problems and possible future work in Section 7.

4 Structural Observations

When trying to design algorithms that are supposed to achieve some given competitive ratio $c \geq 1$, there are many instances for which a solution can be found more or less trivially. Since these solutions have to be handled explicitly by almost every of our algorithms, we handle them such that we can refer to them in our algorithm designs without risking redundancy. We first start with four very simple cases, that nevertheless need to be explicitly handled by our algorithms.

The first lemma deals with instances of very small total size.

► **Lemma 2.** *Any instance I of the OSKP problem with predictions P , such that $\sum_{x^- \in P} x^- \leq \frac{1-\delta}{1+\delta}$, can be solved optimally by packing all items greedily.*

Proof. Since $x^+ \leq \frac{1+\delta}{1-\delta}x^-$, it holds that $\sum_{x \in I} x \leq \sum_{x^- \in P} \frac{1+\delta}{1-\delta}x^- = \frac{1+\delta}{1-\delta} \sum_{x^- \in P} x^- \leq \frac{1+\delta}{1-\delta} \frac{1-\delta}{1+\delta} = 1$. Thus, all items of the instance are guaranteed to fit into the knapsack. ◀

The second lemma explicitly deals with instances that contain a subset of items that is both guaranteed to fit together and of sufficient size.

► **Lemma 3.** *Any instance I of the OSKP problem with predictions P such that there is some $S \subseteq P$ such that $\sum_{x^- \in S} x^- \in [\frac{1}{c}, \frac{1-\delta}{1+\delta}]$ can be solved with a competitive ratio of at most c by packing exactly S .*

Proof. By definition, the items of S are both guaranteed to fit together in the knapsack and guaranteed to be of total size at least $\frac{1}{c}$. ◀

The third lemma ensures that we will only need to deal with instances in which no single item yields the wanted competitive ratio.

► **Lemma 4.** *Any instance I of the OSKP problem with predictions P such that $x^- \in P$ with $x^- \geq \frac{1}{c}$ can be solved with a competitive ratio of at most c by packing exactly x .*

The fourth lemma is slightly more interesting. While the previous statement upper bounded the size of the announced largest item b^- by $\frac{1}{c}$, we are also able to lower bound the size of this item as follows.

► **Lemma 5.** *Any instance I of the OSKP problem with predictions P such that $b^- \leq (1 - \frac{1}{c})/\frac{1+\delta}{1-\delta}$ admits a c -competitive, greedy algorithm.*

Proof. Since b^- is by definition the announced largest item of the instance, we know that the size of the largest actual item is upper bounded by $\frac{1+\delta}{1-\delta}b^-$. Thus, the largest item that a greedy algorithm can *not* fit into its knapsack is of size at most $\frac{1+\delta}{1-\delta}b^- \leq \frac{1+\delta}{1-\delta}(1 - \frac{1}{c})/\frac{1+\delta}{1-\delta} = 1 - \frac{1}{c}$. If such an item does not fit, then the knapsack is already packed to size at least $\frac{1}{c}$. ◀

Thus, we can assume that $b^- \in [(1 - \frac{1}{c})/\frac{1+\delta}{1-\delta}, \frac{1}{c}]$, which allows us to partition the predicted items into two categories, which we will call *small* and *large*. We call an item small if its announced size together with b^- does not guarantee a packing of sufficient size, assuming they are presented as small as possible. Formally, an item $x^- \in P$ is small iff $x^- + b^- < \frac{1}{c} \Leftrightarrow x^- < \frac{1}{c} - b^-$. On the other hand, we call an item large if, together with b^- , it *may* be the case that the two items do not fit together in the knapsack (or if the sum of their announced sizes already exceeds 1.) Formally, an item $x^- \in P$ is large iff $\frac{1+\delta}{1-\delta}x^- + \frac{1+\delta}{1-\delta}b^- > 1 \Leftrightarrow x^- > \frac{1-\delta}{1+\delta} - b^-$.

Note that the existence of any item that falls in neither category already implies a trivial solution of size at least $\frac{1}{c}$ by Lemma 3, with the exception of b^- itself. However, we can see that b^- exceeds the minimum size of a large item, as

$$b^- > \frac{1-\delta}{1+\delta} - b^- \Leftrightarrow 2\left(1 - \frac{1}{c}\right) / \frac{1+\delta}{1-\delta} > \frac{1-\delta}{1+\delta}$$

holds true for all δ iff $c \geq 2$ – even when substituting b^- by its minimum possible value – which will be the case for all bounds that we prove in this paper. Thus, the size of b^- is always large enough to call it a large item as well.

This partition has some very useful implications, with the very central one being that an algorithm can essentially ignore small items of the instance. To show this, we first prove that the total sum of small items in P is limited or the instance has a trivial solution.

► **Lemma 6.** *Let I be an instance of the OSKP problem with predictions P and $S := \{x^- \in P \mid x^- < \frac{1}{c} - b^-\}$ for $c \geq \frac{\delta^2+3}{2-2\delta}$. If P does not admit a solution due to Lemmata 2, 3, 4 or 5 and if $\sum_{x^- \in S} x^- \geq \frac{1}{c} - b^-$ holds, then there exists an algorithm that is c -competitive.*

Proof. The algorithm works as follows: It first selects an arbitrary subset of small items $S \subseteq P$ such that $\sum_{x^- \in S} x^- \in [\frac{1}{c} - b^-, 2(\frac{1}{c} - b^-)]$. Such a subset always exists and can be found greedily since each individual small item is announced smaller than $\frac{1}{c} - b^-$.

The algorithm packs b^- when it is revealed and small items from S greedily, but stopping to pack further small items as soon as their total size is at least $\frac{1}{c} - b^-$. Together with b^- , the gain is obviously at least $\frac{1}{c}$, so the only thing left to show is that such a subset is always guaranteed to fit in the knapsack and does not exceed its capacity.

In the worst case, the algorithm packs a small item of size $\frac{1}{c} - b^- - \varepsilon_1$ and is then presented an item of size $\frac{1+\delta}{1-\delta}(\frac{1}{c} - b^- - \varepsilon_2)$ for some $\varepsilon_1, \varepsilon_2 > 0$. Afterwards, b^- is presented of maximum possible size $\frac{1+\delta}{1-\delta}b^-$. Yet even in this case, all items fit into the knapsack, as

$$\begin{aligned} & \frac{1}{c} - b^- - \varepsilon_1 + \frac{1+\delta}{1-\delta}\left(\frac{1}{c} - b^- - \varepsilon_2\right) + \frac{1+\delta}{1-\delta}b^- \\ & < \frac{1}{c} - b^- + \frac{1+\delta}{1-\delta}\left(\frac{1}{c} - b^-\right) + \frac{1+\delta}{1-\delta}b^- \\ & = \frac{1}{c} - b^- \frac{1+\delta}{1-\delta} \\ & = \left(1 + \frac{1+\delta}{1-\delta}\right) \frac{1}{c} - b^- \\ & < \left(1 + \frac{1+\delta}{1-\delta}\right) \frac{1}{c} - \left(1 - \frac{1}{c}\right) / \frac{1+\delta}{1-\delta} \end{aligned}$$

Solving $\left(1 + \frac{1+\delta}{1-\delta}\right) \frac{1}{c} - \left(1 - \frac{1}{c}\right) / \frac{1+\delta}{1-\delta} \leq 1$ for c yields that these items fit if $c \geq \frac{\delta^2+3}{2-2\delta}$, which is lower than every upper bound proven in this work, for all δ . ◀

Thus, we can assume that the *sum* of all announced small items is smaller than $\frac{1}{c} - b^-$, or else a trivial solution exists. This in turn lets us show a nice relation between small and large items, which is that the larger the sum of small items is, the smaller any large item may be announced or there is again a trivial solution.

► **Lemma 7.** *Let I be an instance of the OSKP problem with predictions P and $S := \{x^- \in P \mid x^- < \frac{1}{c} - b^-\}$ with $c \geq 1$. If P does not admit a solution due to Lemmata 2, 3, 4 or 6 and $b^- \geq \frac{1}{c} - \sum_{x^- \in S} x^-$, then there exists an algorithm that is c -competitive.*

37:8 Online Simple Knapsack with Bounded Predictions

Proof. The algorithm packs b^- together with the items from S . Since $\sum_{x^- \in S} x^- < \frac{1}{c} - b^-$ due to Lemma 6, the total size is at least $b^- + \sum_{x^- \in S} x^- \geq \frac{1}{c} - \sum_{x^- \in S} x^- + \sum_{x^- \in S} x^- = \frac{1}{c}$. ◀

This lemma has the pleasant consequence that we will not have to worry about small items in most of our algorithm design beyond checking whether they are part of a trivial solution. Since in the upcoming analysis, we often bound the packed items of our algorithm against the largest item of the instance, it makes no difference to us whether there are no small items in the instance and thus a very large b^- , or if b^- is made smaller for the sake of additional small items in the instance.

The next lemma allows us to lower bound the announced size of large items depending on the size of b^- .

► **Lemma 8.** *Let I be an instance of the OSKP problem with predictions P and $x^- \in P$ be a large item such that $x^- \neq b^-$. If $x + \frac{1+\delta}{1-\delta}b^- \leq 1$, packing x together with b guarantees a gain of $\frac{1}{c}$ with $c \geq 1$.*

Proof. A large item has actual size at least $\frac{1-\delta}{1+\delta} - b^-$, thus $x + \frac{1+\delta}{1-\delta}b^- > \frac{1-\delta}{1+\delta} - b^- + b^- = \frac{1-\delta}{1+\delta} > \frac{1}{c}$. ◀

This means that whenever an algorithm is guaranteed to be able to pack any large item together with b^- , its gain is sufficient.

Using these observations, we define with Algorithm 1 a small subroutine that will be called by our other algorithms in order to rule out trivial solutions to an instance.

■ **Algorithm 1** Subroutine *filter_trivial*.

```

1: if  $\sum_{x^- \in P} x^+ \leq 1$  then                                ▷ Lemma 2
2:   Pack  $b$ . END
3: if  $b^- \geq \frac{1}{c}$  then                                       ▷ Lemma 4
4:   Pack  $b$ . END
5: if  $b^- \leq \frac{1}{c} / \frac{1+\delta}{1-\delta}$  then                             ▷ Lemma 5
6:   Greedily pack items. END
7: if  $\exists S \in 2^P : \sum_{x^- \in S} x^- \in [\frac{1}{c}, \frac{1-\delta}{1+\delta}]$  then     ▷ Lemma 3
8:   Pack all items of  $S$ . END
9:  $T := \{x^- \in P \mid x^- < \frac{1}{c} - b^-\}$ 
10: if  $\sum_{x^- \in T} x^- \geq \frac{1}{c} - b^-$  then                          ▷ Lemma 6
11:   Pack a subset  $R \in T$  with  $\sum_{x^- \in R} x^- \in [\frac{1}{c} - b^-, 2(\frac{1}{c} - b^-)]$  and  $b$ . END
12: if  $b^- \geq \frac{1}{c} - \sum_{x^- \in T} x^-$  then                          ▷ Lemma 7
13:   Pack all items of  $T$  and  $b$ . END

```

5 Upper Bounds

The analysis of upper bound algorithms remains complicated, even with the filters of Section 4. While we are quite confident that the lower bounds of Section 6 should not be improvable, finding matching upper bounds is an aim that we can only fulfill for parts of the whole range of values for δ .

We start with a simple 2-competitive algorithm for all values of δ up to $\frac{1}{7}$.

► **Theorem 9.** *Given a fixed δ with $0 < \delta \leq \frac{1}{7}$. Algorithm 2 solves the OSKP problem with a competitive ratio of at most $c = 2$.*

■ **Algorithm 2** 2-competitive Algorithm for $0 < \delta \leq \frac{1}{7}$.

-
- 1: *filter-trivial*()
 - 2: Reveal items up to and including the first *large* item x_1 .
 - 3: **if** $x_1 \leq 1 - \frac{1+\delta}{1-\delta}b^-$ and $x_1 \neq b$ **then**
 - 4: **Pack** x_1 with b . **END**
 - 5: **else**
 - 6: Greedily **pack** large items, including x_1 .
-

Proof. If the algorithm ends after the call of *filter_trivial*, the algorithm is at most 2-competitive. Assuming the condition in line 3 is met, the algorithm is at least 2-competitive by Lemma 8. Thus, we only have to prove that the algorithm is not worse than 2-competitive if it ends in line 6.

Let us first assume that $x_1 \neq b$. Then x_1 is packed and $x_1 > 1 - \frac{1+\delta}{1-\delta}b^-$. If any other large item x_i can be packed by the algorithm, the knapsack will be filled up to at least $\frac{1}{2}$, since

$$x_1 + x_i > 1 - \frac{1+\delta}{1-\delta}b^- + \frac{1-\delta}{1+\delta}b^- > 1 - \frac{1+\delta}{1-\delta}\frac{1}{2} + \frac{1-\delta}{1+\delta} - \frac{1}{2} \geq \frac{1}{2},$$

where the last inequality holds for $\delta \leq 3 - 2\sqrt{2} \sim 0.171$.

If no second large item fits into the algorithm's knapsack, an optimal solution cannot contain more than one large item either. This item of the optimal solution is bounded by $\frac{1+\delta}{1-\delta}b^- < \frac{1+\delta}{1-\delta}\frac{1}{2}$. The remaining items of an optimal solution then consist of all small items that the algorithm has ignored. We can bound the total announced size of these small items, by $\frac{1}{2} - b^-$, using Lemma 6 and in consequence their actual size by $\frac{1+\delta}{1-\delta}(\frac{1}{2} - b^-)$. Putting it all together, in the worst case Algorithm 2 packs exactly one large item of size slightly larger than $1 - \frac{1+\delta}{1-\delta}b^-$. The optimal solution packs one large item of size slightly smaller than $\frac{1+\delta}{1-\delta}\frac{1}{2}$ and the maximum sum of small items. This results in a competitive ratio of

$$\frac{\frac{1+\delta}{1-\delta}b^- + \frac{1+\delta}{1-\delta}(\frac{1}{2} - b^-)}{1 - \frac{1+\delta}{1-\delta}b^-} = \frac{\frac{1+\delta}{1-\delta}\frac{1}{2}}{1 - \frac{1+\delta}{1-\delta}b^-} < \frac{\frac{1+\delta}{1-\delta}\frac{1}{c}}{1 - \frac{1+\delta}{1-\delta}\frac{1}{c}} \leq 2,$$

for $\delta \leq \frac{1}{7}$.

The only case left to handle is that $x_1 = b$. If $b \geq 1 - \frac{1+\delta}{1-\delta}b^-$, we can use the same argumentation as before. If however $b < 1 - \frac{1+\delta}{1-\delta}b^-$, then, since b is by definition the announced largest item, *all* large items are announced smaller or equal than $1 - \frac{1+\delta}{1-\delta}b^-$ and thus of actual size at most $\frac{1+\delta}{1-\delta}(1 - \frac{1+\delta}{1-\delta}b^-) < \frac{1+\delta}{1-\delta}(1 - \frac{1+\delta}{1-\delta}\frac{1}{2}/\frac{1+\delta}{1-\delta}) = \frac{1+\delta}{1-\delta}\frac{1}{2}$. As each large item is of size at least $\frac{1-\delta}{1+\delta}b^-$, packing a second large item is still sufficient, since $2(\frac{1-\delta}{1+\delta}b^-) > 2\frac{1-\delta}{1+\delta} - 1 \geq 1/2$ for $\delta \leq \frac{1}{7}$. Since no large item can exceed an actual size of $\frac{1+\delta}{1-\delta}b^-$ and since $x_1 < 1 - \frac{1+\delta}{1-\delta}b^-$, we are guaranteed that a second large item fits into the knapsack of the algorithm. The only case left is that there is no second large item. Then the algorithm and the optimal solution both pack the same large item $x_1 = b$, while the optimal solution can still add the maximum number of small items. The resulting competitive ratio is then $\frac{b^- + \frac{1+\delta}{1-\delta}(\frac{1}{2} - b^-)}{b^-} < 2$ for all δ in the given range. ◀

The second repeating upper bound matches the lower bound of Theorem 14 for almost the complete range of δ . For brevity, let $\delta_k := \frac{12k+8}{32^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}}} + \frac{1}{3}2^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}} + 5/3$.

■ **Algorithm 3** $\sqrt{(2+k)\frac{1+\delta}{1-\delta}}$ -competitive Algorithm for a $k \in \mathbf{N}$ and $\delta < \delta_k$.

```

1: filter_trivial()
2: Reveal items up to and including the first large item  $x_1$ .
3: if  $x_1 \geq \frac{1}{k+2}$  or  $x_1 = b$  then
4:   Greedily pack large items, including  $x_1$ . END
5: if  $x_1 \leq 1 - \frac{1+\delta}{1-\delta}b^-$  then
6:   Pack  $x_1$  with  $b$ . END
7: Let  $B := \{x^- \in P \mid x^- > \frac{1-\delta}{1+\delta} - b^-\}$ 
8: if  $\exists x_j^- \in B : x_j^- \leq (1-x_1)/\frac{1+\delta}{1-\delta}$  then
9:   Pack  $x_1$  and  $x_j$ . END
10: for Reveal next  $x_i \in P_{[i,n-1]}$  do
11:   if  $\exists S \subseteq P_{[i,n-1]} : x_1 + \sum_{x^- \in S} x^- \geq \frac{1}{c_k} \wedge x_1 + \sum_{x^- \in S} x^+ \leq 1$  then
12:     Pack  $x_1$  and  $S$ . END
13: Pack  $x_n$ .

```

► **Theorem 10.** *Given a fixed $k \in \mathbf{N}$ and a fixed δ with $\delta < \delta_k$. Algorithm 3 solves the OSKP problem with a competitive ratio of at most $c_k = \sqrt{(2+k)\frac{1+\delta}{1-\delta}}$.*

Proof. If the algorithm ends after the call of *filter_trivial*, the algorithm is at most c_k -competitive. The algorithm first reveals and discards small items and reveals the first large item x_1 . If $x_1 \geq \frac{1}{k+2}$, the algorithm packs x_1 and from there on greedily any large item that still fits into the knapsack. If any second large item fits into the knapsack of the algorithm, its gain is at least $\frac{1}{k+2} + \frac{1-\delta}{1+\delta} - b^- > \frac{1}{c_k}$. Thus, an optimal solution consists of at most one large item of maximum size or a sum of large and small items adding up to at most the same size, due to Lemma 7. The competitive ratio is then $\frac{1+\delta}{1-\delta} \frac{1}{c_k} / \frac{1}{k+2} = c_k$.

Assuming $x_1 = b$, we know from Lemma 8 that any large item that is guaranteed to fit with b yields a ratio of at most c_k . Again, the algorithm packs x_1 and large items greedily. If any second large item fits into the knapsack of the algorithm, its gain is at least $b + (1-b)/\frac{1+\delta}{1-\delta} > \frac{1}{c_k}$ for the complete range of δ . Thus, an optimal solution consists of at most one large item of maximum size. The competitive ratio is then at worst $\frac{1+\delta}{1-\delta} b/b < c_k$ for the complete range of α .

Finally, if $x_1 \leq 1 - \frac{1+\delta}{1-\delta}b^-$, the algorithm packs x_1 together with b and has sufficient gain by Lemma 8.

Thus, we assume that $1 - \frac{1+\delta}{1-\delta}b^- < x_1 < \frac{1}{k+2}$. The algorithm next checks whether any other large item is of announced size such that it is both guaranteed to fit together with x_1 and guaranteed to fit into the knapsack together with x_1 , i.e. if any $x_j^- \in B$ exists with $x_j^- \in [\frac{1}{c_k} - x_1, \frac{1-x_1}{\frac{1+\delta}{1-\delta}}]$. Note that $\frac{1}{c_k} - x_1 < \frac{1-\delta}{1+\delta} - b^-$, i.e. there exists no large item on the lower end of this bound if $\delta < \delta_k$. Thus, each large item apart from x_1 is of actual size larger than $(1-x_1)/\frac{1+\delta}{1-\delta}$.

If all of these bounds hold and thus do not admit a solution, the algorithm discards x_1 and continues to reveal large items. It only packs anything before the last large item if either the revealed item itself is of size $\frac{1}{c_k}$ or if it admits a packing that is guaranteed to fit and of size at least $\frac{1}{c_k}$. If neither is the case, the algorithm packs the last large item of size larger than $(1-x_1)/\frac{1+\delta}{1-\delta}$.

In the worst case, the optimal solution then consists of $k + 1$ large items of size slightly smaller than $\frac{1}{c_k}$ each. The competitive ratio is then bounded by

$$\frac{k+1}{\sqrt{(2+k)\frac{1+\delta}{1-\delta}}} / \frac{1 - \frac{1}{k+2}}{\frac{1+\delta}{1-\delta}} = \sqrt{(2+k)\frac{1+\delta}{1-\delta}}.$$

Note that the first item is too large to be packed on top of the $k + 1$ items as

$$\frac{k+1}{\sqrt{(2+k)\frac{1+d}{1-d}}} + \left(1 - \frac{1+d}{1-d} \frac{1}{\sqrt{(2+k)\frac{1+d}{1-d}}}\right) > 1$$

for $\delta < \frac{k}{k+2}$. ◀

The next segment's proof is the most involved one. It tightly matches the lower bound of Theorem 15 for $-1 - k + \sqrt{4k + k^2} \leq \delta \leq \frac{k}{k+2}$, for any $k \in \mathbb{N}$.

■ **Algorithm 4** $\frac{(1+\delta)(\sqrt{-1+\delta^2} + \sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$ -competitive Algorithm for a $k \in \mathbb{N}$ and $\delta \leq \frac{k}{k+2}$.

```

1: filter_trivial()
2: Reveal items up to and including the first large item  $x_1$ .
3: Let  $\ell_k := \frac{8}{9+8\delta-\delta^2-\sqrt{-1+\delta^2}\sqrt{-17-16\delta+\delta^2-16k}+8k}$ 
4: if  $x_1 \geq \ell_k$  or  $x_1 = b$  then
5:   Greedily pack large items, including  $x_1$ . END
6: if  $x_1 \leq 1 - \frac{1+\delta}{1-\delta}b^-$  then
7:   Pack  $x_1$  with  $b$ . END
8: Let  $B := \{x^- \in P \mid x^- > \frac{1-\delta}{1+\delta} - b^-\}$ 
9: if  $\exists x_j^-, x_l^- \in B : x_1 + x_j^+ + x_l^+ \leq 1$  then
10:  Pack  $x_1, x_j$  and  $x_l$ . END
11: else if  $\exists x_L^- \in B : x_1 + x_L^- \geq \frac{1}{c_k} \wedge x_1 + x_L^+ \leq 1$  then
12:  Pack  $x_1$  and  $x_L$ . END
13: for Reveal next  $x_i \in P_{[i,n-1]}$  do
14:  if  $\exists S \subseteq P_{[i,n-1]} : x_1 + \sum_{x^- \in S} x^- \geq \frac{1}{c_k} \wedge x_1 + \sum_{x^- \in S} x^+ \leq 1$  then
15:   Pack  $x_1$  and  $S$ . END
16: Pack  $x_n$ .

```

► **Theorem 11.** Given a fixed $k \in \mathbb{N}$ and a fixed δ with $\delta \leq k/(k+2)$. Algorithm 4 solves the OSKP problem with a competitive ratio of at most $c_k = \frac{(1+\delta)(\sqrt{-1+\delta^2} + \sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$.

Proof. If the algorithm ends after the call of *filter_trivial*, the algorithm is at most c_k -competitive. Let $\ell_k = \frac{8}{9+8\delta-\delta^2-\sqrt{-1+\delta^2}\sqrt{-17-16\delta+\delta^2-16k}+8k}$, which is the solution to the equation $\frac{1+\delta}{1-\delta} \frac{1}{c_k} / \ell_k = c_k$.

The algorithm first reveals and discards small items and reveals the first large item x_1 . If $x_1 \geq \ell_k$, the algorithm packs x_1 and from there on greedily any large item that still fits into the knapsack. If any second large item fits into the knapsack of the algorithm, its gain is at least $\ell_k + \frac{1-\delta}{1+\delta} - b^- > \frac{1}{c_k}$. Thus, an optimal solution consists of at most one large item of maximum size. The competitive ratio is then $\frac{1+\delta}{1-\delta} \frac{1}{c_k} / \ell_k = c_k$.

Assuming $x_1 = b$, we know from Lemma 8 that any large item that is guaranteed to fit with b yields a ratio of at most c_k . Again, the algorithm packs x_1 and large items greedily. If any second large item fits into the knapsack of the algorithm, its gain is at least

37:12 Online Simple Knapsack with Bounded Predictions

$b + (1 - b)/\frac{1+\delta}{1-\delta} > \frac{1}{c_k}$ for the complete range of δ . Thus, an optimal solution consists of at most one large item of maximum size. The competitive ratio is then at worst $\frac{1+\delta}{1-\delta}b/b < c_k$ for the complete range of α .

Finally, if $x_1 \leq 1 - \frac{1+\delta}{1-\delta}b^-$, the algorithm packs x_1 together with b and has sufficient gain by Lemma 8.

Thus, we assume that $1 - \frac{1+\delta}{1-\delta}b^- < x_1 < \ell_k$. If another two large items are guaranteed to fit together with x_1 , the gain of the algorithm is again sufficient: $1 - \frac{1+\delta}{1-\delta}b^- + 2(\frac{1-\delta}{1+\delta} - b^-) > 1 - \frac{1+\delta}{1-\delta}\frac{1}{c_k} + 2(\frac{1-\delta}{1+\delta} - \frac{1}{c_k}) > \frac{1}{c_k}$. Thus, we may assume that there is at most one other large item x_j that is guaranteed to fit into the knapsack together with x_1 . Its announced size is then in between $\frac{1-\delta}{1+\delta} - b^-$ and $\frac{1}{c_k} - x_1 < \frac{1}{c_k} - (1 - \frac{1+\delta}{1-\delta}b^-) < \frac{1}{c_k} - (1 - \frac{1+\delta}{1-\delta}\frac{1}{c_k}) = (1 + \frac{1+\delta}{1-\delta})\frac{1}{c_k} - 1$. Obviously, it has to hold that $x_1 + x_j < \frac{1}{c_k}$, otherwise the two items together are sufficient. Since the remaining large items $x_L \in I$ must not be announced with a size that guarantees that they fit into the knapsack together with x_1 and guarantee a packing of at least $\frac{1}{c_k}$, they have to be of announced size $x_L^- > \frac{1-x_1}{1-\delta}$. Since $x_1 < \ell_k < \frac{1}{k+2}$, we can lower bound the size of these large items by $x_L^- > (1 - x_1)/\frac{1+\delta}{1-\delta} > (1 - \frac{1}{k+2})/\frac{1+\delta}{1-\delta} \geq \frac{1}{k+2}$, for all $\delta \leq \frac{1}{k+2}$. Since $\frac{1}{k+2} + \frac{1-\delta}{1+\delta} - b^- > \frac{1}{c_k}$, the item x_j that was guaranteed to fit together with x_1 must not be guaranteed to fit together with any other large item as well. This lower bounds its size by $x_j^- > (1 - \frac{1}{c_k})/(\frac{1+\delta}{1-\delta}) > x_1/\frac{1+\delta}{1-\delta}$.

If all of these bounds hold and thus do not admit a solution, the algorithm discards x_1 and continues to reveal large items. It only packs anything before the last large item if either the revealed item itself is of size $\frac{1}{c_k}$ or if it admits a packing that is guaranteed to fit and of size at least $\frac{1}{c_k}$. If neither is the case, the algorithm packs the last large item of size larger than $(1 - x_1)/\frac{1+\delta}{1-\delta}$.

Recall that $x_L^- > (1 - x_1)/\frac{1+\delta}{1-\delta}$, which means that $(k+1)x_L^- + x_1 > 1$ for $\delta \leq \frac{1}{k+2}$. Since also $x_1 + x_j > \frac{1}{c_k} > x_L^-$, the largest packing of a knapsack consists of k items x_L or size slightly smaller than $\frac{1}{c_k}$, together with both items x_1 and x_j . The competitive ratio is then bounded by

$$(x_1 + x_j + k\frac{1}{c_k})/\frac{1-x_1}{\frac{1+\delta}{1-\delta}}.$$

We are now interested in maximizing this ratio by choosing appropriate values for x_1 and x_j . The gain of the algorithm is minimized if the value of x_1 is maximized. Since $x_j^- > x_1/\frac{1+\delta}{1-\delta}$ and $x_1 + x_j < \frac{1}{c_k}$ both have to hold, the ratio is maximized to a value of c_k when choosing $x_1 = \frac{2\sqrt{-1+\delta^2}}{\sqrt{-1+\delta^2} + \sqrt{-17-16\delta+\delta^2-16k}}$ and thus $x_j = \frac{1+\delta}{1-\delta}x_1$. ◀

For the remainder of unmatched lower bounds, we re-use Algorithm 2 that was used to prove the upper bound of 2 for $0 < \delta \leq \frac{1}{7}$. The only difference is the competitive ratio that is aimed for, which is $\frac{1+\delta+\sqrt{5+2\delta-3\delta^2}}{2-2\delta}$ for $\frac{1}{7} \leq \delta$. The proof is almost identical to that of Theorem 9.

► **Theorem 12.** *Given a fixed $k \in \mathbb{N}$ and a fixed δ with $\frac{1}{7} \leq \delta$. Algorithm 2 solves the OSKP problem with a competitive ratio of at most $c = \frac{1+\delta+\sqrt{5+2\delta-3\delta^2}}{2-2\delta}$.*

Proof in Appendix. ◀

6 Lower Bounds

We are able to provide lower bounds for all values of δ . We can observe that different bounds dominate one another in an alternating fashion, which is a consequence of the scaling factor $\frac{1+\delta}{1-\delta}$ crossing certain thresholds.

Intuitively, the lower bounds – after the initial one of 2 – each work once sufficient distance between the lowest possible size and the largest possible size of an announced item crosses certain thresholds. They are then valid for higher values of δ as well, but are quickly dominated by yet other lower bounds, that again use certain thresholds between the range of possible sizes of an item. These bounds turn out to be rather tricky to find but easy to verify with the assistance of computer algebra systems. We found them by fixing concrete values of δ and carefully building a decision tree regarding an announced instance of variable values. We were then able to generalize them to surrounding values of δ using computer algebra systems.

We start with a simple lower bound of 2 for all values of $0 < \delta < 1$. This lower bound works quite similar to the bound by Marchetti-Spaccamela and Vercellis [27] used to show non-competitiveness. The main difference is that the predictions have to be almost correct for very small δ , thus presenting two items that have a large size difference allow an algorithm to gravitate towards the larger one.

► **Theorem 13.** *Given a fixed δ with $0 < \delta < 1$. There exists no algorithm solving the OSKP problem with a competitive ratio better than 2.*

Proof. Let $\varepsilon > 0$ such that $\varepsilon < \frac{\delta}{4}$. The algorithm is given the following prediction:

$$P = \left(\frac{1}{2} + \varepsilon, \frac{1}{2} - \varepsilon, \frac{1}{2} \right)$$

We do a full case distinction on the possible behaviors of an algorithm. The first item is revealed to be of size $x_1 = \frac{1}{2} + \varepsilon$.

Case 1: An algorithm packs x_1 with $x_1 = \frac{1}{2} + \varepsilon$. The next two items are presented of size $x_2 = x_3 = \frac{1}{2}$, fitting together with one another but not with x_1 . The competitive ratio is then $\frac{1}{\frac{1}{2} + \varepsilon} = 2 \frac{1}{1 + 2\varepsilon}$.

Case 2: An algorithm rejects x_1 with $x_1 = \frac{1}{2} + \varepsilon$. The second item is then presented to be of size $x_2 = \frac{1}{2} - \varepsilon$, with the last item presented of size $x_3 = \frac{1}{2} + 2\varepsilon$, independent of the algorithm's decision on the item x_2 . The optimal solution consists of the first two items, while the biggest packing an algorithm can achieve is $\frac{1}{2} + 2\varepsilon$. The competitive ratio is then $\frac{1}{\frac{1}{2} + 2\varepsilon} = 2 \frac{1}{1 + 4\varepsilon}$. ◀

The next bound connects seamlessly to the previous bound of 2. As with all following bounds, the following theorem describes a set of bounds that only become valid once a certain value of δ is reached.

► **Theorem 14.** *For every $k \in \mathbb{N}$ and $\frac{k^2 + k - 1}{k^2 + 3k + 3} \leq \delta$, there exists no algorithm solving the OSKP problem with a competitive ratio better than $c_k = \sqrt{(2+k) \frac{1+\delta}{1-\delta}}$ -competitive.*

Proof. Let $\varepsilon > 0$. The algorithm is given the following prediction:

$$P = \left(\frac{1}{2+k}, \underbrace{\frac{1}{c_k}, \dots, \frac{1}{c_k}}_{k+1 \text{ many}}, \frac{1 - \frac{1}{2+k}}{\frac{1+\delta}{1-\delta}} \right)$$

37:14 Online Simple Knapsack with Bounded Predictions

We do a full case distinction on the possible behaviors of an algorithm. The first item is presented as $\frac{1}{2+k} + \varepsilon$.

Case 1: An algorithm packs $\frac{1}{2+k} + \varepsilon$. The next items are all presented of maximum possible size. Since $\frac{1+\delta}{1-\delta} \frac{1}{c_k} + \frac{1}{2+k} + \varepsilon > 1$ for each choice of k and all $\delta \geq \frac{k^2+k-1}{k^2+3k+3}$, this means that no two items fit together into the knapsack. The biggest item of an optimal packing is then $\frac{1}{c_k} \frac{1+\delta}{1-\delta}$. The competitive ratio is then $\frac{1+\delta}{1-\delta} \frac{1}{c_k} / (\frac{1}{2+k} + \varepsilon) \stackrel{\varepsilon \rightarrow 0}{=} c_k$.

Case 2: An algorithm rejects $\frac{1}{2+k} + \varepsilon$. The next item is presented of size $\frac{1}{c_k}$.

Case 2.1: An algorithm packs $\frac{1}{c_k}$. The next items are each presented of size $1 - \frac{1}{2+k} - \varepsilon < \frac{1+\delta}{1-\delta} \frac{1}{c_k}$, allowing an optimal packing to reach a size of 1. Since $\frac{1}{c_k} > \frac{1}{2+k}$ for all values of δ in the given range for the given k , the algorithm cannot pack either of these items, resulting in the target competitive ratio.

Case 2.2: An algorithm rejects $\frac{1}{c_k}$. The next items, up to and including the penultimate one, are presented to be of size $\frac{1}{c_k}$ as well. If an algorithm packs one of these items, Case 2.1 applies. We thus assume that all these items are rejected. The last item is then presented to be of size $\frac{1 - \frac{1}{2+k}}{\frac{1+\delta}{1-\delta}}$.

If the algorithm does not pack this item, it is not competitive. An optimal solution can pack all $k+1$ items of size $\frac{1}{c_k}$ if $\delta \geq \frac{k^2+k-1}{k^2+3k+3}$. The competitive ratio is then $\frac{k+1}{c_k} / \frac{1 - \frac{1}{2+k}}{\frac{1+\delta}{1-\delta}} = c_k$. ◀

We continue with a lower bound superseding the previous ones for (relative) higher values of δ .

► **Theorem 15.** *For every $k \in \mathbb{N}$ and $-1-k + \sqrt{4k+k^2} \leq \delta$, there exists no algorithm solving the OSKP problem with a competitive ratio better than $c_k = \frac{(1+\delta)(\sqrt{-1+\delta^2} + \sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$.*

Proof. Let $i_k = \frac{2\sqrt{-1+\delta^2}}{\sqrt{-1+\delta^2} + \sqrt{-17-16\delta+\delta^2-16k}}$, which is the solution to the equation $i_k + \frac{i_k}{\frac{1+\delta}{1-\delta}} = \frac{1}{c_k}$ and $\varepsilon > 0$. The algorithm is given the following prediction:

$$P = (i_k, \underbrace{\frac{i_k}{\frac{1+\delta}{1-\delta}} + \varepsilon, \frac{1-i_k}{\frac{1+\delta}{1-\delta}} + \varepsilon, \dots, \frac{1-i_k}{\frac{1+\delta}{1-\delta}} + \varepsilon}_{i \text{ many}})$$

Note that $\frac{1-i_k}{\frac{1+\delta}{1-\delta}} \leq \frac{1}{c_k}$ only if $\delta \geq -1-k + \sqrt{4k+k^2}$.

We do a full case distinction on the possible behaviors of an algorithm. The first item is presented as i_k .

Case 1: An algorithm packs i_k . The next item is revealed as $i_k / \frac{1+\delta}{1-\delta} + \varepsilon$.

Case 1.1: An algorithm packs $i_k / \frac{1+\delta}{1-\delta} + \varepsilon$. The next item is revealed as $1 - i_k$, thus fitting together with the first item, but not into the knapsack of the algorithm. All subsequent items are presented as $1 - i_k$ as well. The optimal packing is thus of size 1, while the algorithm only achieves a gain of $i_k + i_k / \frac{1+\delta}{1-\delta} + \varepsilon = \frac{1}{c_k} + \varepsilon$.

Case 1.2: An algorithm rejects $i_k/\frac{1+\delta}{1-\delta} + \varepsilon$. All subsequent items are each revealed to be of size $1 - i_k + \varepsilon$, thus not fitting into the knapsack of the algorithm. An optimal strategy packs the items $i_k/\frac{1+\delta}{1-\delta} + \varepsilon$ and $1 - i_k + \varepsilon$, while the algorithm only has the first item of size i_k in its knapsack, resulting in a ratio of $(\frac{i_k}{\frac{1+\delta}{1-\delta}} + 1 - i_k + 2\varepsilon)/(i_k) > c_k$.

Case 2: An algorithm rejects i_k . The second item is revealed of size $i_k + \varepsilon$.

Case 2.1: An algorithm packs $i_k + \varepsilon$. The remaining items are each revealed to be of size $1 - i_k$, thus not fitting into the knapsack of the algorithm. An optimal packing is the first item together with one of the last items, adding up to a gain of 1. The algorithm can only pack $i_k + \varepsilon$, resulting in a ratio of $\frac{1}{i_k} \geq c_i$.

Case 2.2: An algorithm rejects $i_k + \varepsilon$. The next item is revealed of size $\frac{1}{c_k}$.

Case 2.2.1: An algorithm packs $\frac{1}{c_k}$. The remaining items are revealed of size $1 - i_k$, not fitting into the knapsack of the algorithm but fully filling an optimal knapsack together with the first item, resulting exactly in a ratio of c_k .

Case 2.2.2: An algorithm rejects $\frac{1}{c_k}$. Up to and including the penultimate item, the next items are also revealed as $\frac{1}{c_k}$. If an algorithm packs any of them, the rest of the instance behaves as in Case 2.2.1.

Assuming this is not the case, the last item is presented as $(1 - i_k)/(\frac{1+\delta}{1-\delta}) + \varepsilon$, which is the only item the algorithm can pack. An optimal algorithm packs all items except the last one. These items are of sum at most one if $\delta \geq -1 - k + \sqrt{4k + k^2}$. The resulting competitive ratio is

$$\frac{2i_k + \varepsilon + (k - 1)\frac{1}{c_k}}{\frac{1-i_k}{\frac{1+\delta}{1-\delta}} + \varepsilon},$$

which converges to c_k for ε going to 0. ◀

Just like with the two previous bounds, the last lower bound dominates the other two for yet (relative) higher values of δ .

► **Theorem 16.** *For every $k \in \mathbf{N}_{>1}$ and $\frac{k-1}{k+1} < \delta < \frac{k}{k+2}$, there exists no algorithm solving the OSKP problem with a competitive ratio better than $c_k = \frac{1+k}{2} - \frac{\sqrt{(-1+\delta)(-5-k(2+k)+\delta(-1+k)(3+k))}}{-2+2\delta}$.*

Proof. Let $i_k = \frac{2(\delta-1)}{3\delta - \sqrt{-(\delta-1)(-4\delta(k-1)+k^2+2k+5)+\delta k-k-3}}$ be the solution to the equation $c_k = \frac{1}{i_k} - 1$ and let $j_k = \frac{i_k^2}{1-(k+2)i_k}$. The algorithm is given the following prediction:

$$P = (i_k + \varepsilon, \underbrace{j_k}_{i \text{ many}})$$

We do a full case distinction on the possible behaviors of an algorithm. The first item is presented as i_k .

Case 1: An algorithm packs $i_k + \varepsilon$. Each of the next items are presented of maximum possible size $\frac{1+\delta}{1-\delta}j_k = 1 - i_k$, which is the counterpart to the first item, except for the additional ε . Since $1 - i_k > \frac{1}{2}$, no two items fit into the knapsack in this case. The optimal solution thus packs an item of size $1 - i_k$. The competitive ratio is then $\frac{1-i_k}{i_k+\varepsilon} \stackrel{\varepsilon \rightarrow 0}{=} c_k$.

Case 2: An algorithm rejects $i_k + \varepsilon$. The next item is presented to be of size j_k .

Case 2.1: An algorithm packs j_k . If an algorithm packs such an item before the last item, each subsequent item is presented to be of almost maximum size $\frac{1+\delta}{1-\delta}j_k - \varepsilon$. For $\delta < \frac{k}{k+2}$ it holds that $j_k + \frac{1+\delta}{1-\delta}j_k - \varepsilon > 1$. Thus, the algorithm only packs an item of size j_k , while the optimal solution is 1, as such an item of almost maximum size perfectly fits together with the first item of the instance. The competitive ratio is then $\frac{1}{j_k} > c_k$, as $j_k < \frac{1}{c_k}$ for $\delta > \frac{1}{3}$.

Case 2.2: An algorithm rejects j_k . Up to and including the penultimate item, the next items are also revealed as j_k . If an algorithm packs any of them, the rest of the instance behaves as in Case 2.1.

Assuming this is not the case, the last item is presented as j_k , which is the only item the algorithm can pack. An optimal algorithm packs all items except the last one. These items are of sum at most 1 if $\delta > \frac{1}{3}$. The competitive ratio is then $\frac{i_k + \varepsilon + k j_k}{j_k} \stackrel{\varepsilon \rightarrow 0}{=} c_k$. ◀

7 Open Problems

The OSKP problem turns out to be very involved problem. While we were able to give tight bounds on the competitive ratio for some values of δ , the analysis is not complete. We are quite confident that the lower bounds that we determined should reflect the actual competitive ratio in the remaining open segments, but finding a proof to do so is beyond our capability.

A logical next step would be to look at the general ONLINE KNAPSACK problem in the setting of predictions. However, one would first have to determine which part of the input is predicted: The size, the weight or the density of the items. The model of predictions offers to be applied to – and already has been applied to – further problems beyond knapsack problems. To our knowledge, more discretized problems, such as graph problems like the MINIMUM VERTEX COVER problem have not been studied in the setting of predictions. Finding an appropriate, unifying model on what to predict in such problems would be interesting.

References

- 1 Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. Online bin packing with predictions. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 4574–4580. ijcai.org, 2022. doi:10.24963/ijcai.2022/635.
- 2 Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 2020. URL: <http://proceedings.mlr.press/v119/antoniadis20a.html>.
- 3 Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Mixing predictions for online metric algorithms. *CoRR*, abs/2304.01781, 2023. doi:10.48550/arXiv.2304.01781.

- 4 David Arthur, Bodo Manthey, and Heiko Röglin. k-means has polynomial smoothed complexity. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 405–414. IEEE Computer Society, 2009. doi: 10.1109/FOCS.2009.14.
- 5 Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1070–1080. ACM, 2021. doi:10.1145/3406325.3451023.
- 6 Yossi Azar, Eldad Peretz, and Noam Touitou. Distortion-oblivious algorithms for scheduling on multiple machines. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPICs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ISAAC.2022.16.
- 7 Eric Balkanski, Vasilis Gkatzelis, and Xizhi Tan. Strategyproof scheduling with predictions. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPICs*, pages 11:1–11:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/LIPICs.ITCS.2023.11.
- 8 Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 462–471. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238219.
- 9 René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 232–241. ACM, 2003. doi:10.1145/780542.780578.
- 10 Magnus Berg, Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Online minimum spanning trees with weight predictions. *CoRR*, abs/2302.12029, 2023. doi:10.48550/arXiv.2302.12029.
- 11 Hans-Joachim Böckenhauer, Elisabet Burjons, Juraj Hromkovic, Henri Lotze, and Peter Rossmanith. Online simple knapsack with reservation costs. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPICs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.STACS.2021.16.
- 12 Hans-Joachim Böckenhauer, Dennis Komm, Richard Královic, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014. doi:10.1016/j.tcs.2014.01.027.
- 13 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
- 14 Joan Boyar, Lene M. Favrholdt, Shahin Kamali, and Kim S. Larsen. Online interval scheduling with predictions. *CoRR*, abs/2302.13701, 2023. doi:10.48550/arXiv.2302.13701.
- 15 Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Online unit profit knapsack with untrusted predictions. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022, June 27-29, 2022, Tórshavn, Faroe Islands*, volume 227 of *LIPICs*, pages 20:1–20:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SWAT.2022.20.
- 16 Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 10393–10406, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/5616060fb8ae85d93f334e7267307664-Abstract.html>.

- 17 Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter. Learning-augmented query policies for minimum spanning tree with uncertainty. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPICs*, pages 49:1–49:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ESA.2022.49.
- 18 Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku. Online knapsack problems with a resource buffer. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPICs*, pages 28:1–28:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ISAAC.2019.28.
- 19 Xin Han and Kazuhisa Makino. Online removable knapsack with limited cuts. *Theor. Comput. Sci.*, 411(44-46):3956–3964, 2010. doi:10.1016/j.tcs.2010.08.009.
- 20 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Online knapsack with frequency predictions. In Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 2733–2743, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/161c5c5ad51fcc884157890511b3c8b0-Abstract.html>.
- 21 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 293–305. Springer, 2002. doi:10.1007/3-540-45465-9_26.
- 22 Kazuo Iwama and Guochuan Zhang. Online knapsack with resource augmentation. *Inf. Process. Lett.*, 110(22):1016–1020, 2010. doi:10.1016/j.ipl.2010.08.013.
- 23 Billy Jin and Will Ma. Online bipartite matching with advice: Tight robustness-consistency tradeoffs for the two-stage model. In *NeurIPS, 2022*. URL: http://papers.nips.cc/paper_files/paper/2022/hash/5d68a3f05ee2aae6a0fb2d94959082a0-Abstract-Conference.html.
- 24 Dennis Komm. *An Introduction to Online Computation - Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016. doi:10.1007/978-3-319-42749-2.
- 25 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877. SIAM, 2020. doi:10.1137/1.9781611975994.114.
- 26 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311. PMLR, 2018. URL: <http://proceedings.mlr.press/v80/lykouris18a.html>.
- 27 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995. doi:10.1007/BF01585758.
- 28 Michele Monaci, Ulrich Pferschy, and Paolo Serafini. Exact solution of the robust knapsack problem. *Comput. Oper. Res.*, 40(11):2625–2631, 2013. doi:10.1016/j.cor.2013.05.005.
- 29 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update rules. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 488–492. ACM, 1984. doi:10.1145/800057.808718.

- 30 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 296–305. ACM, 2001. doi:10.1145/380752.380813.
- 31 Clemens Thielen, Morten Tiedemann, and Stephan Westphal. The online knapsack problem with incremental capacity. *Math. Methods Oper. Res.*, 83(2):207–242, 2016. doi:10.1007/s00186-015-0526-9.
- 32 Chenyang Xu and Guochuan Zhang. Learning-augmented algorithms for online subset sum. *J. Glob. Optim.*, 87(2):989–1008, 2023. doi:10.1007/S10898-022-01156-W.
- 33 Yunhong Zhou, Deeparnab Chakrabarty, and Rajan M. Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In Christos H. Papadimitriou and Shuzhong Zhang, editors, *Internet and Network Economics, 4th International Workshop, WINE 2008, Shanghai, China, December 17-20, 2008. Proceedings*, volume 5385 of *Lecture Notes in Computer Science*, pages 566–576. Springer, 2008. doi:10.1007/978-3-540-92185-1_63.

A Additional Proofs

Proof of Theorem 12. If the algorithm ends after the call of *filter_trivial*, the algorithm is at most c -competitive. Assuming the condition in line 3 is met, the algorithm is at least c -competitive by Lemma 8. Thus, we only have to prove that the algorithm is not worse than c -competitive if it ends in line 6.

Let us first assume that $x_1 \neq b$. Then x_1 is packed and $x_1 > 1 - \frac{1+\delta}{1-\delta}b^-$. If any other large item x_i can be packed by the algorithm, the knapsack will be filled up to at least $\frac{1}{2}$, since

$$x_1 + x_i > 1 - \frac{1+\delta}{1-\delta}b^- + \frac{1-\delta}{1+\delta}b^- > 1 - \frac{1+\delta}{1-\delta}\frac{1}{c} + \frac{1-\delta}{1+\delta}\frac{1}{c} \geq \frac{1}{c},$$

where the last inequality holds for all values of δ between 0 and 1.

If no second large item fits into the algorithm's knapsack, an optimal solution cannot contain more than one large item either. This item of the optimal solution is bounded by $\frac{1-\delta}{1+\delta}b^- < \frac{1-\delta}{1+\delta}\frac{1}{c}$. The remaining items of an optimal solution then consist of all small items that the algorithm has ignored. We can bound the total announced size of these small items, by $\frac{1}{c} - b^-$, using Lemma 6 and in consequence their actual size by $\frac{1+\delta}{1-\delta}(\frac{1}{c} - b^-)$. Putting it all together, in the worst case Algorithm 2 packs exactly one large item of size slightly larger than $1 - \frac{1+\delta}{1-\delta}b^-$. The optimal solution packs one large item of size slightly smaller than $\frac{1-\delta}{1+\delta}\frac{1}{c}$ and the maximum sum of small items. This results in a competitive ratio of

$$\frac{\frac{1+\delta}{1-\delta}b^- + \frac{1+\delta}{1-\delta}(\frac{1}{c} - b^-)}{1 - \frac{1+\delta}{1-\delta}b^-} = \frac{\frac{1+\delta}{1-\delta}\frac{1}{c}}{1 - \frac{1+\delta}{1-\delta}b^-} < \frac{\frac{1+\delta}{1-\delta}\frac{1}{c}}{1 - \frac{1+\delta}{1-\delta}\frac{1}{c}} = c,$$

for $\delta \leq \frac{1}{7}$.

The only case left to handle is that $x_1 = b$. If $b \geq 1 - \frac{1+\delta}{1-\delta}b^-$, we can use the same argumentation as before. If however $b < 1 - \frac{1+\delta}{1-\delta}b^-$, then, since b is by definition the announced largest item, all large items are announced smaller than $1 - \frac{1+\delta}{1-\delta}b^-$ and thus of actual size at most $\frac{1+\delta}{1-\delta}(1 - \frac{1+\delta}{1-\delta}b^-) < \frac{1+\delta}{1-\delta}(1 - \frac{1+\delta}{1-\delta}\frac{1}{c}/\frac{1+\delta}{1-\delta}) = \frac{1+\delta}{1-\delta}\frac{1}{c}$. Since no large item can exceed an actual size of $\frac{1+\delta}{1-\delta}b^-$ and since $x_1 < 1 - \frac{1+\delta}{1-\delta}b^-$, we are guaranteed that a second large item fits into the knapsack of the algorithm. In the worst case, an optimal solution

37:20 Online Simple Knapsack with Bounded Predictions

packs b and an item of size $\frac{1+\delta}{1-\delta}b^-$, while the algorithm only packs b and a large item of minimal size. The competitive ratio is then

$$\left(\frac{1+\delta}{1-\delta}b^- + b^-\right) / \left(b^- + \frac{1-\delta}{1+\delta}b^-\right) < c,$$

where the last inequality holds for all values of δ between 0 and 1.

The only case left is that there is no second large item. Then the algorithm and the optimal solution both pack the same large item $x_1 = b$, while the optimal solution can still add the maximum number of small items. The resulting competitive ratio is then $\frac{b^- + \frac{1+\delta}{1-\delta}(\frac{1}{c}b^-)}{b^-} < c$ for all δ between 0 and 1. ◀

The AC^0 -Complexity of Visibly Pushdown Languages

Stefan Göller 

School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Nathan Grosshans   

Independent Scholar, Paris Region, France

Abstract

We study the question of which visibly pushdown languages (VPLs) are in the complexity class AC^0 and how to effectively decide this question. Our contribution is to introduce a particular subclass of one-turn VPLs, called *intermediate VPLs*, for which the raised question is entirely unclear: to the best of our knowledge our research community is unaware of containment or non-containment in AC^0 for *any* language in our newly introduced class. Our main result states that there is an algorithm that, given a visibly pushdown automaton, correctly outputs exactly one of the following: that its language L is in AC^0 , some $m \geq 2$ such that MOD_m (the words over $\{0, 1\}$ having a number of 1's divisible by m) is constant-depth reducible to L (implying that L is not in AC^0), or a finite disjoint union of intermediate VPLs that L is constant-depth equivalent to. In the latter of the three cases one can moreover effectively compute $k, l \in \mathbb{N}_{>0}$ with $k \neq l$ such that the concrete intermediate VPL $L(S \rightarrow \varepsilon \mid ac^{k-1}Sb_1 \mid ac^{l-1}Sb_2)$ is constant-depth reducible to the language L . Due to their particular nature we conjecture that either all intermediate VPLs are in AC^0 or all are not. As a corollary of our main result we obtain that in case the input language is a visibly counter language our algorithm can effectively determine if it is in AC^0 – hence our main result generalizes a result by Krebs et al. stating that it is decidable if a given visibly counter language is in AC^0 (when restricted to well-matched words).

For our proofs we revisit so-called Ext-algebras (introduced by Czarnetzki et al.), which are closely related to forest algebras (introduced by Bojańczyk and Walukiewicz), and use Green's relations.

2012 ACM Subject Classification Theory of computation \rightarrow Grammars and context-free languages; Theory of computation \rightarrow Circuit complexity

Keywords and phrases Visibly pushdown languages, Circuit Complexity, AC^0

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.38

Related Version *Full Version, Extended Version*: <https://doi.org/10.48550/arXiv.2302.13116>

Funding *Stefan Göller*: The author was supported by the Agence Nationale de la Recherche grant no. ANR-17-CE40-0010.

1 Introduction

This paper studies the circuit complexity of formal word languages. It is well-known that the regular word languages are characterized as the languages recognizable by finite monoids. When restricting the finite monoids to be aperiodic Schützenberger proved that one obtains precisely the star-free regular languages [22]. In terms of logic, these correspond to the languages definable in first-order logic $FO[<]$ by a result of McNaughton and Papert [23]. The more general class of regular languages expressible in $FO[arb]$, i.e. first-order logic with arbitrary numerical predicates, coincides with the regular languages in AC^0 [13, 16]. These can be characterized algebraically as the regular languages whose syntactic morphism is quasi-aperiodic [5]. The latter algebraic characterization also shows that it is decidable if a regular language is in AC^0 .



© Stefan Göller and Nathan Grosshans;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 38; pp. 38:1–38:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Generalizing regular languages, input-driven languages were introduced by Mehlhorn [21]. They are described by pushdown automata whose input alphabet is partitioned into letters that are either of type call, internal, or return. Rediscovered by Alur and Madhusudan in 2004 [2] under the name of *visibly pushdown languages (VPLs)*, it was shown that they enjoy many of the desirable effective closure properties of the regular languages. For instance, the visibly pushdown languages form an effective Boolean algebra. Algebraically, VPLs were characterized by Alur et al. [1] by congruences on well-matched words of finite index. Extending upon these, Czarnetzki et al. introduced so-called Ext-algebras [9]; these involve pairs of monoids (R, O) where O is a submonoid of R^R . Being tailored towards recognizing word languages, Ext-algebras are closely connected to forest algebras, introduced by Bojańczyk and Walukiewicz [7]: in [9] it is shown that a language of well-matched words is visibly pushdown if, and only if, its syntactic Ext-algebra is finite. While context-free languages are generally in $LOGCFL = SAC^1$, the visibly pushdown languages, as the regular languages, are known to be in NC^1 [10]. By a famous result of Barrington [4], there already exist regular languages that are NC^1 -hard.

Related work. Visibly pushdown languages (VPLs) were introduced [2] via deterministic visibly pushdown automata (DVPA for short). Inspired by forest algebras [7] the paper [9] introduces Ext-algebras. Unfortunately, the definition of Ext-algebra morphisms in [9] is incorrect in that it provably does not lead to freeness.

The regular languages that are in AC^0 were effectively characterized by Barrington et al. [5]. By an automata-theoretic approach, Krebs et al. [19] effectively characterized the visibly counter languages that are in AC^0 . These are particular VPLs that are essentially accepted by visibly pushdown automata that use only one stack symbol. In his PhD thesis [20] Ludwig already considers the question of which VPLs are in AC^0 . Yet, his conjectural characterization contains several serious flaws – a detailed discussion of these shortcomings can be found in Section 8 in [12].

Our contribution. We reintroduce Ext-algebras, fix the notion of Ext-algebra morphisms and define the languages they recognize. We also reintroduce the syntactic Ext-algebra of languages of well-matched words. We rigorously prove classical results like freeness and minimality of syntactic Ext-algebras with respect to recognition. We prove that a language of well-matched words is a VPL if, and only if, it is recognizable by a finite Ext-algebra. While these results essentially revisit the constructions of [9], we use Ext-algebras as a technical tool for studying the complexity of visibly pushdown languages.

Fix a visibly pushdown alphabet Σ , i.e. Σ is partitioned into Σ_{call} (call letters), Σ_{int} (internal letters), and Σ_{ret} (return letters). Denoting $\Delta(u)$ as the difference between the number of occurrences of call and return letters in $u \in \Sigma^*$, a word $w \in \Sigma^*$ is *well-matched* if $\Delta(w) = 0$ and $\Delta(u) \geq 0$ for all prefixes u of w . A *context* is a pair (u, v) such that uv is well-matched – contexts have a natural composition operation: $(u, v) \circ (u', v') = (uu', v'v)$.

A set of contexts \mathcal{R} is *length-synchronous* if $|u|/|v| = |u'|/|v'|$ for all $(u, v), (u', v') \in \mathcal{R}$ with $\Delta(u), \Delta(u') > 0$ and *weakly length-synchronous* if $u = u'$ implies $|v| = |v'|$ and $v = v'$ implies $|u| = |u'|$ for all $(u, v), (u', v') \in \mathcal{R}$ with $\Delta(u), \Delta(u') > 0$. Any language L of well-matched words induces a congruence \equiv_L on contexts: $(u, v) \equiv_L (u', v')$ if $xuwvy \in L \Leftrightarrow xu'wv'y \in L$ for all contexts (x, y) and all well-matched words w . We introduce the notion of quasi-counterfreeness: a language is *quasi-counterfree* if for all contexts $\sigma \in \Sigma^k \times \Sigma^l$ for arbitrary k and l at least one of the following holds: (1) there exists some $n \in \mathbb{N}$ such that $\sigma^n \equiv_L \sigma^{n+1}$ or (2) no context in $\Sigma^k \times \Sigma^l$ is \equiv_L -equivalent to $\sigma \circ \sigma$. Finally, we introduce

our central class of *intermediate VPLs*: a VPL is *intermediate* if it is quasi-counterfree and generated by a context-free grammar containing the production $S \rightarrow_G \varepsilon$, where S is the start nonterminal and whose other productions are of the form $T \rightarrow_G uT'v$ where uv is well-matched, $u \in (\Sigma_{\text{int}}^* \Sigma_{\text{call}} \Sigma_{\text{int}}^*)^+$ and $v \in (\Sigma_{\text{int}}^* \Sigma_{\text{ret}} \Sigma_{\text{int}}^*)^+$, such that the set of contexts $\{(u, v) \mid S \Rightarrow_G^* uSv\}$ is weakly length-synchronous but not length-synchronous. Note that intermediate VPLs are particular one-turn visibly pushdown languages, that is, visibly pushdown languages that are subsets of $(\Sigma \setminus \Sigma_{\text{ret}})^* (\Sigma \setminus \Sigma_{\text{call}})^*$. As an example, for all $k, l \geq 1$ with $k \neq l$, a concrete intermediate VPL, denoted by $\mathcal{L}_{k,l}$, is the one that is generated by the context-free grammar $S \rightarrow \varepsilon \mid ac^{k-1}Sb_1 \mid ac^{l-1}Sb_2$: here a is a call letter, c is an internal letter and b_1 and b_2 are return letters.

As far as we know the techniques known to our community do not directly suffice to show whether *at all* there is some intermediate VPL that is *provably in* AC^0 or *provably not in* AC^0 – analogous remarks apply to ACC^0 . Our main result states that there is an algorithm that, given a DVPA A correctly outputs exactly one of the following: $L(A) \in \text{AC}^0$, some $m \geq 2$ such that MOD_m is constant-depth reducible to L (thus witnessing that $L(A) \notin \text{AC}^0$), or a non-empty disjoint finite union of intermediate VPLs that $L(A)$ is constant-depth equivalent to. In the latter of the three cases one can moreover effectively compute $k, l \in \mathbb{N}_{>0}$ with $k \neq l$ such that the above-mentioned $\mathcal{L}_{k,l}$ is constant-depth reducible to $L(A)$. We conjecture that either all intermediate VPLs are in AC^0 or all are not: note that together with our main result this conjecture implies the existence of an algorithm that can effectively determine if a given visibly pushdown language is in AC^0 . As a corollary of our main result we obtain that in case the input language is a visibly counter language our algorithm can effectively determine if it is in AC^0 , hence our main result generalizes a result by Krebs et al. stating that it is decidable if a given visibly counter language is in AC^0 (when restricted to well-matched words).

For our main result we extensively study Ext-algebras, the syntactic morphisms of VPLs, and make use of Green's relations.

Organization. Our paper is organized as follows. We introduce notation and give an overview of our main result in Section 2. In Section 3 we first recall general algebraic concepts and then revisit Ext-algebras and their correspondence to visibly pushdown languages. Section 4 introduces central notions like length-synchronicity and weak length-synchronicity for Ext-algebra morphisms and visibly pushdown languages. The proof of our main result is content of Section 5. We conclude in Section 6. Full proofs can be found in [12].

2 Preliminaries

By \mathbb{N} we denote the non-negative integers and by $\mathbb{N}_{>0}$ the positive integers. For all $a \in \Gamma$ and all $w \in \Gamma^*$ we write $|w|_a$ to denote the number of occurrences of a in w . We define the languages $\text{EQUALITY} = \{w \in \{0,1\}^* \mid |w|_0 = |w|_1\}$ and $\text{MOD}_m = \{w \in \{0,1\}^* \mid m \text{ divides } |w|_1\}$ for each $m \geq 2$. A *visibly pushdown alphabet* is a finite alphabet $\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$, where the alphabets Σ_{call} , Σ_{int} , and Σ_{ret} are pairwise disjoint. The set of *well-matched words over a visibly pushdown alphabet* Σ , denoted by Σ^Δ , is the smallest set satisfying the following: $\varepsilon \in \Sigma^\Delta$ and $c \in \Sigma^\Delta$ for all $c \in \Sigma_{\text{int}}$, $awb \in \Sigma^\Delta$ for all $w \in \Sigma^\Delta$, $a \in \Sigma_{\text{call}}$ and $b \in \Sigma_{\text{ret}}$, and $uv \in \Sigma^\Delta$ for all $u, v \in \Sigma^\Delta \setminus \{\varepsilon\}$. A well-matched word $w \in \Sigma^\Delta$ is *one-turn* if $w \in (\Sigma \setminus \Sigma_{\text{ret}})^* (\Sigma \setminus \Sigma_{\text{call}})^*$. A language $L \subseteq \Sigma^\Delta$ is *one-turn* if it contains only one-turn words. Let Σ be a visibly pushdown alphabet. We define $\Delta: \Sigma^* \rightarrow \mathbb{Z}$ to be the height monoid morphism such that $\Delta(w) = |w|_{\Sigma_{\text{call}}} - |w|_{\Sigma_{\text{ret}}}$ for all $w \in \Sigma^*$.

A *context* is a pair $(u, v) \in \Sigma^* \times \Sigma^*$ such that $uv \in \Sigma^\Delta$. The *composition* of two contexts $(u, v), (x, y) \in \text{Con}(\Sigma)$ is defined as $(u, v) \circ (x, y) = (ux, yv)$. For $\sigma \in \text{Con}(\Sigma)$ by σ^k we denote the k -fold composition $\sigma \circ \dots \circ \sigma$. For any context $(u, v) \in \text{Con}(\Sigma)$ and well-matched word $w \in \Sigma^\Delta$ we define $(u, v)w = uwv$. An equivalence relation \equiv on $\text{Con}(\Sigma)$ is a *congruence* if for all $\chi, \chi', \sigma, \tau \in \text{Con}(\Sigma)$ we have that $\sigma \equiv \tau$ implies $\chi \circ \sigma \circ \chi' \equiv \chi \circ \tau \circ \chi'$. Given a congruence \equiv over $\text{Con}(\Sigma)$ we denote by $[\sigma]_{\equiv}$ the equivalence class of σ . Given a language of well-matched words $L \subseteq \Sigma^\Delta$ we write $\sigma \equiv_L \tau$ if for all $\chi \in \text{Con}(\Sigma)$ and all $w \in \Sigma^\Delta$ we have $(\chi \circ \sigma)w \in L$ if, and only if, $(\chi \circ \tau)w \in L$. Clearly, \equiv_L is a congruence.

A *context-free grammar* is a tuple $G = (V, \Sigma, P, S)$, where V is a finite set of *nonterminals*, Σ is a non-empty finite *alphabet*, $P \subseteq V \times (V \cup \Sigma)^*$ is a finite set of *productions*, and $S \in V$ is the *start nonterminal*. We write $T \rightarrow_G y$ whenever $(T, y) \in P$. The binary relation \Rightarrow_G over $(V \cup \Sigma)^*$ is defined as $u \Rightarrow_G v$ if there exists a production $T \rightarrow_G y$ and $x, z \in (V \cup \Sigma)^*$ such that $u = xTz$ and $v = xyz$. By $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$ we denote the *language* of G where \Rightarrow_G^* is the reflexive transitive closure of \Rightarrow_G .

In the following we introduce deterministic visibly pushdown automata, remarking that nondeterministic visibly pushdown automata are determinizable [2]. A *deterministic visibly pushdown automaton* (DVPA) is a tuple $A = (Q, \Sigma, \Gamma, \delta, q_0, F, \perp)$, where Q is a finite set of *states*, Σ is a visibly pushdown alphabet, the *input alphabet*, Γ is a finite alphabet, the *stack alphabet*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states*, $\perp \in \Gamma$ is the *bottom-of-stack symbol*, and $\delta: Q \times \Sigma \times \Gamma \rightarrow Q \times (\{\varepsilon\} \cup \Gamma \cup (\Gamma \setminus \{\perp\})\Gamma)$ is the *transition function* such that for all $q \in Q, a \in \Sigma, \alpha \in \Gamma$: if $a \in \Sigma_{\text{call}}$, then $\delta(q, a, \alpha) \in Q \times (\Gamma \setminus \{\perp\})\alpha$, if $a \in \Sigma_{\text{ret}}$, then $\delta(q, a, \alpha) \in Q \times \{\varepsilon\}$, and if $a \in \Sigma_{\text{int}}$, then $\delta(q, a, \alpha) \in Q \times \{\alpha\}$. We define the *extended transition function* $\hat{\delta}: Q \times \Sigma^* \times \Gamma^* \rightarrow Q \times \Gamma^*$ inductively as $\hat{\delta}(q, \varepsilon, \beta) = (q, \beta)$ for all $q \in Q$ and $\beta \in \Gamma^*$, $\hat{\delta}(q, w, \varepsilon) = (q, \varepsilon)$ for all $q \in Q$ and $w \in \Sigma^+$, and $\hat{\delta}(q, aw, \alpha\beta) = \hat{\delta}(p, w, \gamma\beta)$, where $\delta(q, a, \alpha) = (p, \gamma)$ for all $q \in Q, a \in \Sigma, w \in \Sigma^*, \alpha \in \Gamma$ and $\beta \in \Gamma^*$. The *language* accepted by A is the language $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w, \perp) \in F \times \{\perp\}\}$. We call such a language a *visibly pushdown language* (VPL). We remark that visibly pushdown languages are always subsets of Σ^Δ .

We refer to [14] for further details on formal language theory.

2.1 Complexity and logic

We assume familiarity with standard circuit complexity, we refer to [24, 18] for an introduction to the topic. Recall the following Boolean functions: the AND-function, the OR-function, the majority function (that outputs 1 if the majority of its inputs are 1s), and the mod_m function (that outputs 1 if the number of its inputs that are 1s is divisible by m) for all $m \geq 2$.

A circuit family $(C_n)_{n \in \mathbb{N}}$ *decides* a binary language $L \subseteq \{0, 1\}^*$ if C_n is a circuit with n inputs such that $L \cap \{0, 1\}^n = \{x_1 \dots x_n \in \{0, 1\}^n \mid C_n(x_1, \dots, x_n) = 1\}$ for all $n \in \mathbb{N}$. In this paper, we will consider circuits deciding languages over arbitrary finite alphabets: to do this, we just consider implicitly that any language over an arbitrary finite alphabet comes with a fixed binary encoding that encodes each letter as a block of bits of fixed size. By \leq_{cd} we mean *constant-depth truth table reducibility* (or just constant-depth reducibility) as introduced in [8]. Formally for two languages $K \subseteq \Gamma^*$ and $L \subseteq \Sigma^*$ for finite alphabets Σ, Γ , we write $K \leq_{\text{cd}} L$ in case there is a polynomial p , a constant $d \in \mathbb{N}$, and circuit family $(C_n)_{n \in \mathbb{N}}$ deciding L such that each circuit C_n satisfies the following: it is of depth at most d and size at most $p(n)$ and its non-input gates are either AND-labeled, OR-labeled, or so-called oracle gates, labeled by L , that are gates deciding $L \cap \Sigma^m$, where $m \leq p(n)$, such that there is no path from the output of an oracle gate to an input of another oracle gate.

We write $K =_{\text{cd}} L$ if $K \leq_{\text{cd}} L$ and $L \leq_{\text{cd}} K$; we also say that K and L are *constant-depth equivalent*. We say a language L is *hard* for a complexity class C (or just C -hard) if $L' \leq_{\text{cd}} L$ for all $L' \in C$. We say L is C -complete if L is C -hard and $L \in C$. The following complexity classes are relevant in this paper: AC^0 is the class of all languages decided by circuit families with NOT gates, AND, OR gates of unbounded fan-in, constant depth and polynomial size; ACC^0 is the class of all languages decided by circuit families with NOT gates, AND, OR and modular gates (for some fixed m) of unbounded fan-in, constant depth and polynomial size; TC^0 is the class of all languages decided by circuit families with NOT gates, AND, OR and majority gates of unbounded fan-in, constant depth and polynomial size; NC^1 is the class of all languages decided by circuit families with NOT gates, AND, OR gates of bounded fan-in, logarithmic depth and polynomial size.

We also consider the framework of first order logic over finite words. (See [17, 23] for a proper introduction to the topic.) A *numerical predicate of arity* $r \in \mathbb{N}_{>0}$ is a symbol of arity r associated to a subset of $\mathbb{N}_{>0}^r$. Given a class \mathcal{C} of numerical predicates and a finite alphabet Σ , we call $\text{FO}_{\Sigma}[\mathcal{C}]$ -formula a first order formula over finite words using the alphabet Σ and numerical predicates from the class \mathcal{C} . On occasions, we might also consider $\text{FO}_{\Sigma, \rightsquigarrow}[\mathcal{C}]$ -formulas that in comparison to the previous ones can use an additional binary predicate \rightsquigarrow and are interpreted on structures (w, M) with $w \in \Sigma^*$ and $M \subseteq [1, |w|]^2$, where everything is interpreted as for $\text{FO}_{\Sigma}[\mathcal{C}]$ -formulas on w excepted for \rightsquigarrow that is interpreted by M . Given a class \mathcal{C} of numerical predicates, by $\text{FO}[\mathcal{C}]$ we denote the class of all languages over any finite alphabet Σ defined by a $\text{FO}_{\Sigma}[\mathcal{C}]$ -sentence. A classical result at the interplay of circuit complexity and logic is that $\text{AC}^0 = \text{FO}[\text{arb}]$, where arb denotes the class of all numerical predicates (see [23, Theorem IX.2.1] or [17, Corollary 5.32]). The other numerical predicates that we will encounter in this paper are $<$, $+$ and MOD_m for all $m \in \mathbb{N}_{>0}$, where MOD_m tests if m divides the number of 1's (gathered together in the set $\text{MOD} = \{\text{MOD}_m \mid m > 0\}$).

2.2 Main result

The notion of length-synchronicity and weak length-synchronicity will be a central notion in our main result. In the following Σ always denotes a visibly pushdown alphabet.

► **Definition 1** ((Weak) Length-Synchronicity). *Let $\mathcal{R} \subseteq \text{Con}(\Sigma)$ be a set of contexts. We say \mathcal{R} is length-synchronous if $|u|/|v| = |u'|/|v'|$ for all $(u, v), (u', v') \in \mathcal{R}$ with $\Delta(u), \Delta(u') > 0$; we say \mathcal{R} is weakly length-synchronous if $u = u'$ implies $|v| = |v'|$ and $v = v'$ implies $|u| = |u'|$ for all $(u, v), (u', v') \in \mathcal{R}$ with $\Delta(u), \Delta(u') > 0$.*

Note that a set of contexts \mathcal{R} is weakly length-synchronous if \mathcal{R} is length-synchronous. Indeed, if, say $(u, v), (u', v') \in \mathcal{R}$, where $|v| \neq |v'|$ and $\Delta(u) > 0$, then $|u|, |v|, |v'| > 0$ and so the quotients $\frac{|u|}{|v|}$ and $\frac{|u|}{|v'|}$ are distinct, thus violating length-synchronicity of \mathcal{R} .

► **Definition 2** (Quasi-Counterfree). *A VPL $L \subseteq \Sigma^{\Delta}$ is quasi-counterfree if for all $\sigma = (u, v) \in \text{Con}(\Sigma)$ at least one of the following holds: (1) there exists some $n \in \mathbb{N}$ such that $\sigma^n \equiv_L \sigma^{n+1}$ or (2) for all $\tau \in \Sigma^{|u|} \times \Sigma^{|v|} \cap \text{Con}(\Sigma)$ we have $\tau \not\equiv_L \sigma \circ \sigma$.*

We will later show that quasi-counterfreeness of a VPL $L \subseteq \Sigma^{\Delta}$ is equivalent to the condition that there is no $k, l \in \mathbb{N}$ such that there is a subset of $\text{Con}(\Sigma) \cap \Sigma^k \times \Sigma^l$ that forms a non-trivial group when considering the associated equivalence classes with respect to \equiv_L .

► **Example 3.** Consider the visibly pushdown alphabet Σ , where $\Sigma_{\text{call}} = \{a\}$, $\Sigma_{\text{int}} = \{c\}$ and $\Sigma_{\text{ret}} = \{b_1, b_2\}$. For all $k, l \in \mathbb{N}_{>0}$ satisfying $k \neq l$, consider the language $\mathcal{L}_{k,l}$ generated by the context-free grammar $S \rightarrow ac^{k-1}Sb_1 \mid ac^{l-1}Sb_2 \mid \varepsilon$. We have that the set of contexts

38:6 The AC^0 -Complexity of Visibly Pushdown Languages

$\{(u, v) \in \text{Con}(\Sigma) \mid S \Rightarrow_G^* uSv\}$ is weakly length-synchronous since both the relation and its reverse is a partial function – however, it is not length-synchronous. It is also not hard to see that $\mathcal{L}_{k,l}$ is quasi-counterfree. Indeed, given $(u, v) \in \text{Con}(\Sigma)$, if u contains the letter b_1 or b_2 , or v contains the letter a or the letter c along with the letter b_1 or b_2 , we have that $(\chi \circ (u, v))w \notin \mathcal{L}_{k,l}$ for all $\chi \in \text{Con}(\Sigma)$ and all $w \in \Sigma^\Delta$, so $(u, v)^2 \equiv_{\mathcal{L}_{k,l}} (u, v)^3$. If u and v happen to contain the letter c and only this letter, we can argue similarly. In the cases remaining, u contains only letters from $\{a, c\}$ and v letters from $\{b_1, b_2\}$.

We say a context-free grammar $G = (V, \Sigma, P, S)$ is *vertically visibly pushdown* if the underlying alphabet Σ is a visibly pushdown alphabet, $S \rightarrow_G \varepsilon$, and all other productions of G are of the form $T \rightarrow_G uT'v$, where $uv \in \Sigma^\Delta$ is one-turn such that $u \in (\Sigma_{\text{int}}^* \Sigma_{\text{call}} \Sigma_{\text{int}}^*)^+$ and $v \in (\Sigma_{\text{int}}^* \Sigma_{\text{ret}} \Sigma_{\text{int}}^*)^+$. Note that each grammar generating $\mathcal{L}_{k,l}$ in Example 3 is vertically visibly pushdown. Note that languages generated by vertically visibly pushdown grammars are obviously one-turn VPLs.

► **Definition 4** (Intermediate VPL). *A VPL L is intermediate if it is quasi-counterfree and $L = L(G)$ for some vertically visibly pushdown grammar G for which $\mathcal{R}(G) = \{(u, v) \in \text{Con}(\Sigma) \mid S \Rightarrow_G^* uSv\}$ is weakly length-synchronous but not length-synchronous.*

We remark that every intermediate languages is in TC^0 . Thus the languages $\mathcal{L}_{k,l}$ from Example 3 are all intermediate VPLs. Loosely speaking, they are the simplest intermediate VPLs. We have the following conjecture.

► **Conjecture 5.** *There is no intermediate VPL that is in ACC^0 or TC^0 -hard.*

In fact, the authors are not even aware of any intermediate VPL that is provably not in AC^0 . An indication for the inadequacy of known techniques to prove it is that the *robustness* [18] of intermediate VPLs can be proven to be constant. Further techniques, based for instance on the *switching lemma* [15] or on the *polynomial method* [6] also do not seem to be applicable.

Our main result is the following theorem.

► **Theorem 6.** *There is an algorithm that, given a DVPA A , correctly outputs either*

- $L(A) \in AC^0$,
- $m \geq 2$ such that $\text{MOD}_m \leq_{\text{cd}} L(A)$ (hence implying $L(A) \notin AC^0$),
- vertically visibly pushdown grammars G_1, \dots, G_m each generating intermediate VPLs such that $L =_{\text{cd}} \biguplus_{i=1}^m L(G_i)$. In this case one can moreover effectively compute $k, l \in \mathbb{N}$ with $k \neq l$ such that $\mathcal{L}_{k,l} \leq_{\text{cd}} L(A)$.

Theorem 6 and the following conjecture imply the existence of an algorithm that decides if a given visibly pushdown language is in AC^0 .

► **Conjecture 7.** *Either all intermediate VPLs are in AC^0 or all are not.*

We refer the reader to [3] for the definition of visibly counter automata. Visibly counter automata (m -VCA) are essentially restricted visibly pushdown automata manipulating a counter which can moreover explicitly test if the current counter has a value in $[0, m - 1]$ or at least m . The following corollary of Theorem 6 implies the main result of [19] when restricted to well-matched words.

► **Corollary 8.** *There is an algorithm that, given an m -VCA A , correctly outputs either that $L(A)$ is in AC^0 or some $m \geq 2$ such that $\text{MOD}_m \leq_{\text{cd}} L(A)$ (hence implying $L(A) \notin AC^0$).*

3 Ext-Algebras

This section builds on [9], but identifies an inaccuracy in the definition of Ext-algebra morphisms to establish freeness.

Let $(M, \cdot, 1_M)$ be a monoid. For each $m \in M$, we shall respectively denote by left_m and right_m the *left-multiplication map* $x \mapsto m \cdot x$ and the *right-multiplication map* $x \mapsto x \cdot m$.

An Ext-algebra (R, O, \cdot, \circ) consists of a monoid $(R, \cdot, 1_R)$ and a monoid $(O, \circ, 1_O)$ that is a submonoid of (R^R, \circ) containing the maps left_r and right_r for each $r \in R$. An Ext-algebra morphism from Ext-algebra (R, O) to Ext-algebra (S, P) is a pair (φ, ψ) of monoid morphisms $\varphi: R \rightarrow S$ and $\psi: O \rightarrow P$ such that: for all $e \in O$ and $r \in R$ we have $\psi(e)(\varphi(r)) = \varphi(e(r))$ and for all $r \in R$ we have $\psi(\text{left}_r) = \text{left}_{\varphi(r)}$ and $\psi(\text{right}_r) = \text{right}_{\varphi(r)}$. When it is clear from the context, we shall write *morphism* to mean Ext-algebra morphism. We remark that in the above definition, φ is totally determined by ψ , because for each $r \in R$, we have $\varphi(r) = \varphi(\text{left}_r(1_R)) = \psi(\text{left}_r)(\varphi(1_R)) = \psi(\text{left}_r)(1_S)$.

For the rest of this section, let us fix some visibly pushdown alphabet Σ . For all $(u, v) \in \text{Con}(\Sigma)$, consider the function $\text{ext}_{u,v}: \Sigma^\Delta \rightarrow \Sigma^*$ such that $\text{ext}_{u,v}(x) = u xv$ for all $x \in \Sigma^\Delta$. It is not hard to prove that $\text{ext}_{u,v}$ is a function from Σ^Δ to Σ^Δ . Consider now the set $\mathcal{O}(\Sigma^\Delta)$ of all functions $\text{ext}_{u,v}$ for $(u, v) \in \text{Con}(\Sigma)$: it is a subset of $(\Sigma^\Delta)^{\Sigma^\Delta}$ closed under composition. Thus, $(\mathcal{O}(\Sigma^\Delta), \circ)$ is a submonoid of $((\Sigma^\Delta)^{\Sigma^\Delta}, \circ)$. As for all $w \in \Sigma^\Delta$ we have $\text{left}_w = \text{ext}_{w,\varepsilon}$ and $\text{right}_w = \text{ext}_{\varepsilon,w}$, the set $\mathcal{O}(\Sigma^\Delta)$ contains left_w and right_w for all $w \in \Sigma^\Delta$. Thus, $(\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta), \cdot, \circ)$ is an Ext-algebra.

The following important proposition establishes freeness of $(\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta))$.

► **Proposition 9.** *Let (R, O) be an Ext-algebra and consider two functions $\varphi: \Sigma_{\text{int}} \rightarrow R$ and $\psi: \{\text{ext}_{a,b} \mid a \in \Sigma_{\text{call}}, b \in \Sigma_{\text{ret}}\} \rightarrow O$. Then there exists a unique Ext-algebra morphism $(\bar{\varphi}, \bar{\psi})$ from $(\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta))$ to (R, O) satisfying $\bar{\varphi}(c) = \varphi(c)$ for each $c \in \Sigma_{\text{int}}$ and $\bar{\psi}(\text{ext}_{a,b}) = \psi(\text{ext}_{a,b})$ for each $a \in \Sigma_{\text{call}}, b \in \Sigma_{\text{ret}}$.*

We remark that the requirement that for all $r \in R$ we have $\psi(\text{left}_r) = \text{left}_{\varphi(r)}$ and $\psi(\text{right}_r) = \text{right}_{\varphi(r)}$ does not appear in the definition of Ext-algebra morphisms given in [9]. But this is actually problematic, because then Proposition 9 would not hold in general. A counter-example is given in [12].

A language $L \subseteq \Sigma^\Delta$ is *recognized* by an Ext-algebra (R, O) whenever there exists a morphism $(\varphi, \psi): (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) \rightarrow (R, O)$ such that $L = \varphi^{-1}(F)$ for some $F \subseteq R$. The *syntactic Ext-algebra congruence* of a language $L \subseteq \Sigma^\Delta$ is the congruence \sim_L on Σ^Δ defined by $u \sim_L v$ for $u, v \in \Sigma^\Delta$ whenever $xuy \in L \Leftrightarrow xvy \in L$ for all $(x, y) \in \text{Con}(\Sigma)$. By $[x]_{\sim_L}$ we denote the *congruence class* of $x \in \Sigma^\Delta$.

► **Example 10.** Consider the language $L = \mathcal{L}_{1,2} = L(S \rightarrow aSb_1 \mid acSb_2 \mid \varepsilon)$ from Example 3 over the visibly pushdown alphabet Γ , where $\Gamma_{\text{int}} = \{c\}$, $\Gamma_{\text{call}} = \{a\}$ and $\Gamma_{\text{ret}} = \{b_1, b_2\}$. Set $R_L = \{[acb_1]_{\sim_L}, [\varepsilon]_{\sim_L}, [c]_{\sim_L}, [cab_1]_{\sim_L}, [ab_1]_{\sim_L}\}$, and, given for all $(u, v) \in \text{Con}(\Sigma)$ the function $f_{u,v} \in (R_L)^{R_L}$ satisfying $f_{u,v}([x]_{\sim_L}) = [uxv]_{\sim_L}$ for all $x \in \Sigma^\Delta$, set

$$O_L = \{f_{acb_1, \varepsilon}, f_{\varepsilon, \varepsilon}, f_{c, \varepsilon}, f_{\varepsilon, c}, f_{ab_1, \varepsilon}, f_{\varepsilon, ab_1}, f_{cab_1, \varepsilon}, f_{a, b_2}, f_{ca, b_2}, f_{ca, ab_1 b_2}, f_{ca, b_1}, f_{a, ab_1 b_2}, f_{a, b_1}\}.$$

For instance, note that $[ab_1]_{\sim_L} = [acb_2]_{\sim_L}$, that $[acb_1]_{\sim_L}$ is the zero of R_L and that $f_{acb_1, \varepsilon}$ is the zero of O_L . Then (R_L, O_L) is an Ext-algebra recognizing L thanks to the morphism $(\varphi_L, \psi_L): (\Gamma^\Delta, \mathcal{O}(\Gamma^\Delta)) \rightarrow (R_L, O_L)$ satisfying $\varphi_L(c) = [c]_{\sim_L}$, $\psi_L(\text{ext}_{a, b_1}) = f_{a, b_1}$ and $\psi_L(\text{ext}_{a, b_2}) = f_{a, b_2}$. Note that $L = \varphi_L^{-1}(\{[\varepsilon]_{\sim_L}, [ab_1]_{\sim_L}\})$. Finally, note that for instance $\psi_L(\text{ext}_{ca, ab_1 b_2}) = f_{ca, ab_1 b_2} \neq f_{a, ab_1 b_2} = \psi_L(\text{ext}_{a, ab_1 b_2})$ since we have $\psi_L(\text{ext}_{a, b_2}) \circ \psi_L(\text{ext}_{ca, ab_1 b_2})([c]_{\sim_L}) = [acacab_1 b_2 b_2]_{\sim_L} = [ab_1]_{\sim_L}$ whereas $\psi_L(\text{ext}_{a, b_2}) \circ \psi_L(\text{ext}_{a, ab_1 b_2})([c]_{\sim_L}) = [aacab_1 b_2 b_2]_{\sim_L} = [acb_1]_{\sim_L}$.

The following lemma is proven in several steps. For this, classical notions like sub-Ext-algebra, division and quotients are introduced.

► **Lemma 11.** *Let $L \subseteq \Sigma^\Delta$. The pair $(R_L, O_L) = (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) / \sim_L$, where $R_L = \Sigma^\Delta / \sim_L$ and where $O_L = \{e' \in (R_L)^{R_L} \mid \exists \text{ext}_{u,v} \in \mathcal{O}(\Sigma^\Delta) \forall x \in \Sigma^\Delta : e'([x]_{\sim_L}) = [\text{ext}_{u,v}(x)]_{\sim_L}\}$ is an Ext-algebra. Moreover the pair (φ_L, ψ_L) of functions $\varphi_L: \Sigma^\Delta \rightarrow R_L$ and $\psi_L: \mathcal{O}(\Sigma^\Delta) \rightarrow O_L$ satisfying $\varphi_L(x) = [x]_{\sim_L}$ for all $x \in \Sigma^\Delta$ and $\psi(\text{ext}_{u,v})([x]_{\sim_L}) = [\text{ext}_{u,v}(x)]_{\sim_L}$ for all $\text{ext}_{u,v} \in \mathcal{O}(\Sigma^\Delta)$ and $x \in \Sigma^\Delta$ is an Ext-algebra morphism. We have $L = \varphi_L^{-1}(\varphi_L(L))$.*

We call (R_L, O_L) the *syntactic Ext-algebra* of L along with its unique associated morphism $(\varphi_L, \psi_L): (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) \rightarrow (R_L, O_L)$, the *syntactic Ext-algebra morphism* of L . (An example of each of these is already given in Example 10.)

We say that an Ext-algebra (R, O) is *finite* whenever R is finite (which is the case if and only if O is finite). The following theorem establishes the equivalence between visibly pushdown languages and languages recognizable by finite Ext-algebras. Its proof provides effective translations from DVPAs to Ext-algebras and vice versa.

► **Theorem 12.** *A language $L \subseteq \Sigma^\Delta$ is a VPL if, and only if, it is recognized by a finite Ext-algebra.*

4 (Weak) length-synchronicity, nesting depth, and quasi-aperiodicity

For the rest of this section let us fix a visibly pushdown alphabet Σ , a finite Ext-algebra (R, O) and consider a morphism $(\varphi, \psi): (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) \rightarrow (R, O)$.

In this section we introduce the notions of weak length-synchronicity and length-synchronicity for Ext-algebra morphisms and visibly pushdown languages. Before we do that, let us give some motivation how TC⁰-hardness can be proven if the syntactic morphism maps certain $\text{ext}_{u,v}, \text{ext}_{u',v}$ with $|u| \neq |u'|$ to particular idempotents. For these we require the following notion of reachability.

For $F \subseteq R$ we say that an element $r \in R$ is *F-reaching* if $e(r) \in F$ for some $e \in O$. We say $e \in O$ is *F-reaching* if $e(r)$ is *F-reaching* for some $r \in R$. Fix any VPL L , its syntactic Ext-algebra (R_L, O_L) along with its syntactic morphism (φ_L, ψ_L) . Assume there exists some idempotent $e \in O_L$ that is $\varphi_L(L)$ -reaching.

We claim that if $\psi_L(\text{ext}_{u,v}) = \psi_L(\text{ext}_{u',v}) = e$ and $\Delta(u), \Delta(u') > 0$ for some $\text{ext}_{u,v}, \text{ext}_{u',v} \in \mathcal{O}(\Sigma^\Delta)$ with $|u| \neq |u'|$, then L is TC⁰-hard. We remark that we must have $\Delta(u) = \Delta(u')$. Exploiting the fact that $|u| \neq |u'|$ we give a constant-depth reduction from the TC⁰-complete language EQUALITY to L . As $\psi_L(\text{ext}_{u,v})$ is $\varphi_L(L)$ -reaching, we can fix some $x, y, z \in \Sigma^*$ such that $xuyvz \in L$. Given a word $w \in \{0, 1\}^*$ of length $2n$ (binary words of odd length can directly be rejected) we map it to $xh(w)yv^{n \cdot (|u| + |u'|)}z$, where $h: \{0, 1\}^* \rightarrow \Sigma^*$ is the length-multiplying morphism (i.e. $\exists l \in \mathbb{N} : h(0), h(1) \in \Sigma^l$) satisfying $h(0) = u^{|u'|}$ and $h(1) = u^{|u|}$. We have $w \in \text{EQUALITY}$ if, and only if, $|w|_0 = |w|_1 = n$ if, and only if, $\Delta(h(w)) = n \cdot (|u| + |u'|) \cdot \Delta(u) = -n \cdot (|u| + |u'|) \cdot \Delta(v)$ if, and only if, $h(w)v^{n \cdot (|u| + |u'|)} \in \Sigma^\Delta$. Hence, since $\psi_L(\text{ext}_{u^s, v^s}) = \psi_L(\text{ext}_{(u')^t, v^t}) = e$ for all $s, t \geq 1$ it follows that $w \in \text{EQUALITY}$ if, and only, if $xh(w)yv^{n \cdot (|u| + |u'|)}z \in \Sigma^\Delta$ if, and only if, $xh(w)yv^{n \cdot (|u| + |u'|)}z \in L$.

Dually, one can show that L is TC⁰-hard in case $\psi_L(\text{ext}_{u,v}) = \psi_L(\text{ext}_{u,v'}) = e$ and $\Delta(u) > 0$ for some $\text{ext}_{u,v}, \text{ext}_{u,v'} \in \mathcal{O}(\Sigma^\Delta)$ with $|v| \neq |v'|$.

The following definition of weak length-synchronicity captures the situation when such idempotents do not exist – it adapts the notion of weak length-synchronicity of sets of contexts, given in Definition 1, to morphisms and VPLs, respectively.

The morphism $(\varphi, \psi): (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) \rightarrow (R, O)$ is F -weakly-length-synchronous (where $F \subseteq R$) if for all F -reaching idempotents $e \in O$ the set of contexts $\mathcal{R}_e := \{(u, v) \in \text{Con}(\Sigma) \mid \psi(\text{ext}_{u,v}) = e\}$ is weakly length-synchronous. We call a VPL $L \subseteq \Sigma^\Delta$ *weakly length-synchronous* if its syntactic morphism (φ_L, ψ_L) is $\varphi_L(L)$ -weakly-length-synchronous.

Instead of considering those pairs (u, v) such that $\text{ext}_{u,v}$ is being mapped to an F -reaching idempotent, the following characterization of weak length-synchronicity consider pairs (u, v) such that $\text{ext}_{u,v}$ is being mapped to an element that behaves neutrally with respect to right multiplication with an F -reaching element that is not necessarily idempotent.

► **Proposition 13.** *For all $F \subseteq R$ we have that (φ, ψ) is F -weakly-length-synchronous if, and only if, for all F -reaching $e \in O$ the set of contexts $\mathcal{U}_e := \{(u, v) \in \text{Con}(\Sigma) \mid e \circ \psi(\text{ext}_{u,v}) = e\}$ is weakly length-synchronous.*

One can prove that each $\text{ext}_{u,v}$ has a unique *stair factorization* $\text{ext}_{u,v} = \text{ext}_{x_1, y_1} \circ \text{ext}_{a_1, b_1} \circ \dots \circ \text{ext}_{x_{h-1}, y_{h-1}} \circ \text{ext}_{a_{h-1}, b_{h-1}} \circ \text{ext}_{x_h, y_h}$ satisfying $h \geq 1$, $x_i, y_i \in \Sigma^\Delta$ for all $i \in [1, h]$, $a_i \in \Sigma_{\text{call}}$, and $b_i \in \Sigma_{\text{ret}}$ for all $i \in [1, h-1]$. We refer to the x_i and y_i as *hills* of the stair factorization. From the following proposition it follows that weak length-synchronicity of a VPL L implies that for all $\varphi_L(L)$ -reaching $e \in O_L$ and all $(u, v) \in \mathcal{U}_e$, the stair factorization of $\text{ext}_{u,v}$ has small hills of constant size.

► **Proposition 14.** *There is a constant $n \in \mathbb{N}$ such that for all $F \subseteq R$, all F -reaching $e \in O$, and all $(u, v) \in \mathcal{U}_e$, if (φ, ψ) is F -weakly-length-synchronous, then the stair factorization $\text{ext}_{u,v} = \text{ext}_{x_1, y_1} \circ \text{ext}_{a_1, b_1} \circ \dots \circ \text{ext}_{x_{h-1}, y_{h-1}} \circ \text{ext}_{a_{h-1}, b_{h-1}} \circ \text{ext}_{x_h, y_h}$ satisfies $|x_i|, |y_i| \leq n$ for all $i \in [1, h]$.*

As above, the following definition adapts the notion of length-synchronicity of sets of contexts, given in Definition 1, to Ext-algebra morphisms and VPLs, respectively.

The morphism $(\varphi, \psi): (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) \rightarrow (R, O)$ is F -length-synchronous (where $F \subseteq R$) if for all F -reaching idempotents $e \in O$ the set of contexts $\mathcal{R}_e = \{(u, v) \in \text{Con}(\Sigma) \mid \psi(\text{ext}_{u,v}) = e\}$ is length-synchronous. We call a VPL $L \subseteq \Sigma^\Delta$ *length-synchronous* if its syntactic morphism (φ_L, ψ_L) is $\varphi_L(L)$ -length-synchronous.

Consider our running example $\mathcal{L}_{1,2} = L(S \rightarrow aSb_1 \mid acSb_2 \mid \varepsilon)$. Recall that the monoid $O_{\mathcal{L}_{1,2}}$ of the syntactic Ext-algebra $(R_{\mathcal{L}_{1,2}}, O_{\mathcal{L}_{1,2}})$ and syntactic morphism $(\varphi_{\mathcal{L}_{1,2}}, \psi_{\mathcal{L}_{1,2}})$ of $\mathcal{L}_{1,2}$, given in Example 10, has the idempotents $f_{\varepsilon, \varepsilon}$, $f_{acb_1, \varepsilon}$ and f_{a, b_1} . Also recall that $\varphi_{\mathcal{L}_{1,2}}(\mathcal{L}_{1,2}) = \{[\varepsilon]_{\sim_{\mathcal{L}_{1,2}}}, [ab_1]_{\sim_{\mathcal{L}_{1,2}}}\}$. Since $\psi_{\mathcal{L}_{1,2}}^{-1}(f_{\varepsilon, \varepsilon}) = \{\text{ext}_{\varepsilon, \varepsilon}\}$ and $f_{acb_1, \varepsilon}$ is a zero we have that $O_{\mathcal{L}_{1,2}}$'s only idempotent that is $\{[\varepsilon]_{\sim_{\mathcal{L}_{1,2}}}, [ab_1]_{\sim_{\mathcal{L}_{1,2}}}\}$ -reaching and whose pre-image under $\psi_{\mathcal{L}_{1,2}}$ contains at least one $\text{ext}_{u,v}$ with $\Delta(u) > 0$ is the idempotent f_{a, b_1} . However, both ext_{a, b_1} and ext_{ac, b_2} , where $\Delta(a) = \Delta(ac) = 1 > 0$, are sent to the idempotent $f_{a, b_1} = f_{a, b_2} \circ f_{c, \varepsilon}$. Since $|a|/|b_1| = 1 \neq 2 = |ac|/|b_2|$, we have that $\mathcal{L}_{1,2}$ is not length-synchronous. On the other hand, note that if any $\text{ext}_{u,v}$ and $\text{ext}_{u',v}$ (resp. $\text{ext}_{u,v}$ and $\text{ext}_{u',v'}$) are sent to f_{a, b_1} then $u = u'$ and thus $|u| = |u'|$ (resp. $v = v'$ and thus $|v| = |v'|$). Hence, $\mathcal{L}_{1,2}$ is weakly length-synchronous.

The two following propositions characterize length-synchronicity of Ext-algebra morphisms and of the set of contexts \mathcal{U}_e , which will be of particular importance when approximating the matching relation of a length-synchronous VPL in terms of $\text{FO}[+]$. This will be an important ingredient to proving that VPLs that both are length-synchronous and have a quasi-a-periodic syntactic morphism (a notion to be defined below) are in $\text{FO}[+]$ and thus in AC^0 .

► **Proposition 15.** *For all $F \subseteq R$ we have that (φ, ψ) is F -length-synchronous if, and only if, for all F -reaching $e \in O$ the set of contexts $\mathcal{U}_e = \{(u, v) \in \text{Con}(\Sigma) \mid e \circ \psi(\text{ext}_{u,v}) = e\}$ is length-synchronous.*

► **Proposition 16.** *Let $F \subseteq R$ and assume (φ, ψ) is F -weakly-length-synchronous. Then for all F -reaching $e \in O$ the following two statements are equivalent.*

1. $\mathcal{U}_e = \{(u, v) \in \text{Con}(\Sigma) \mid e \circ \psi(\text{ext}_{u,v}) = e\}$ is length-synchronous.
2. *There exist $\alpha \in \mathbb{Q}_{>0}$, $\beta \in \mathbb{N}$, $\gamma \in \mathbb{N}_{>0}$ such that for all $(u, v) \in \mathcal{U}_e$ with $\Delta(u) > 0$ we have:*
 - (a) $\frac{|u|}{|v|} = \alpha$,
 - (b) for all $u', v' \in \Sigma^+$ with u' prefix of u and v' suffix of v such that $\frac{|u'|}{|v'|} = \alpha$, we have that $-\Delta(v') - \beta \leq \Delta(u') \leq -\Delta(v') + \beta$,
 - (c) for all factors $u' \in \Sigma^*$ of u such that $|u'| = \gamma$, we have $\Delta(u') \geq 1$, and
 - (d) for all factors $v' \in \Sigma^*$ of v such that $|v'| = \gamma$, we have $\Delta(v') \leq -1$.

The nesting depth of visibly pushdown languages. Another central notion is the nesting depth of well-matched words, which is the Horton-Strahler number [11] of the underlying trees. The *nesting depth* of well-matched words is inductively defined as follows: $\text{nd}(\varepsilon) = 0$; $\text{nd}(c) = 0$ for all $c \in \Sigma_{\text{int}}$; $\text{nd}(uv) = \max\{\text{nd}(u), \text{nd}(v)\}$ for all $u \in \Sigma_{\text{call}}\Sigma^\Delta\Sigma_{\text{ret}} \cup \Sigma_{\text{int}}$ and $v \in \Sigma^\Delta \setminus \{\varepsilon\}$; $\text{nd}(awb) = \text{nd}(w) + 1$ if $w = uv$ for some $u, v \in \Sigma^\Delta$ with $\text{nd}(w) = \text{nd}(u) = \text{nd}(v)$ and $\text{nd}(w)$ otherwise, for all $a \in \Sigma_{\text{call}}$, $b \in \Sigma_{\text{ret}}$ and $w \in \Sigma^\Delta$.

An important property of weakly length-synchronous VPLs is that their words have bounded nesting depth. Assume any weakly length-synchronous VPL $L \subseteq \Sigma^\Delta$. Let n be the constant of Proposition 14. One can prove that if there exists $w \in L$ with $\text{nd}(w) > d = n + 1$, then there exists a factorization $w = uv = \text{ext}_{u,v}(\varepsilon)$ such that for stair factorization $\text{ext}_{u,v} = \text{ext}_{x_1, y_1} \circ \text{ext}_{a_1, b_1} \circ \dots \circ \text{ext}_{x_{h-1}, y_{h-1}} \circ \text{ext}_{a_{h-1}, b_{h-1}} \circ \text{ext}_{x_h, y_h}$ we must have $\max\{|x_i|, |y_i| : i \in [1, h]\} > n$, clearly contradicting Proposition 14. We obtain the following proposition.

► **Proposition 17.** *For each weakly length-synchronous VPL $L \subseteq \Sigma^\Delta$ there exists a constant $d \in \mathbb{N}$ such that $L \subseteq \{w \in \Sigma^\Delta \mid \text{nd}(w) \leq d\}$.*

Quasi-aperiodicity. Towards characterizing the circuit complexity of visibly pushdown languages the notion of quasi-aperiodicity has already been defined for visibly pushdown languages in [20]. Let $(\varphi, \psi): (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) \rightarrow (R, O)$ for a visibly pushdown alphabet Σ and a finite Ext-algebra (R, O) . We define $\mathcal{O}(\Sigma^\Delta)^{k,l} = \{\text{ext}_{u,v} \in \mathcal{O}(\Sigma^\Delta) : |u| = k, |v| = l\}$ for all $k, l \in \mathbb{N}$. We say (φ, ψ) is *quasi-aperiodic* if all semigroups contained in the set $\psi(\mathcal{O}(\Sigma^\Delta)^{k,l})$ are aperiodic for all $k, l \in \mathbb{N}$.

5 Proof of the main theorem

The following proposition implies that the syntactic Ext-algebra and the syntactic morphism of a given visibly pushdown language L is computable and that it is decidable if L is quasi-aperiodic, length-synchronous, and weakly length-synchronous, respectively.

► **Proposition 18.** *The following computability and decidability results hold:*

1. *Given a DVPA A , one can effectively compute the syntactic Ext-algebra of $L = L(A)$, its syntactic morphism (φ_L, ψ_L) and $\varphi_L(L)$.*
2. *Given a morphism $(\varphi, \psi): (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) \rightarrow (R, O)$ for a visibly pushdown alphabet Σ and a finite Ext-algebra (R, O) , all of the following are decidable for (φ, ψ) :*
 - (a) *Quasi-aperiodicity: in case (φ, ψ) is not quasi-aperiodic, one can effectively compute $k, l \in \mathbb{N}$ such that $\psi(\mathcal{O}(\Sigma^\Delta)^{k,l})$ is not aperiodic;*
 - (b) *F -length-synchronicity for a given $F \subseteq R$: in case (φ, ψ) is not F -length-synchronous, one can effectively compute a quadruple $(k, l, k', l') \in \mathbb{N}_{>0}^4$ such that there exist $uv, u'v' \in \Sigma^\Delta$ and some F -reaching idempotent $e \in O$ such that $\psi(\text{ext}_{u,v}) = \psi(\text{ext}_{u',v'}) = e$, $\Delta(u) > 0$, $\Delta(u') > 0$, $k = |u|, l = |v|, k' = |u'|, l' = |v'|$, and $\frac{k}{l} \neq \frac{k'}{l'}$;*
 - (c) *F -weakly-length-synchronicity for a given $F \subseteq R$.*

Proof outline for Theorem 6. Towards proving our main result (Theorem 6), given a DVPA A , where $L = L(A)$ is a VPL over a visibly pushdown alphabet Σ , we apply Proposition 18 and compute its syntactic Ext-algebra (R_L, O_L) along with its syntactic morphism (φ_L, ψ_L) and the subset $\varphi_L(L)$. Then the following effective case distinction implies Theorem 6.

1. If L is not weakly length-synchronous, then L is TC^0 -hard and hence not in AC^0 (Proposition 19 in Section 5.1). We output some $m > 1$ since $\text{MOD}_m \leq_{\text{cd}} \text{EQUALITY} \leq_{\text{cd}} L$.
2. If L is not quasi-aperiodic, then one can compute some $m \geq 2$ such that $\text{MOD}_m \leq_{\text{cd}} L$ (Proposition 20 in Section 5.1).
3. If L is length-synchronous and (φ_L, ψ_L) is quasi-aperiodic, then $L \in \text{AC}^0$ (Theorem 22 in Section 5.2).
4. If L that is weakly length-synchronous, not length-synchronous, and its syntactic morphism (φ_L, ψ_L) is quasi-aperiodic, one can compute vertically visibly pushdown grammars G_1, \dots, G_m generating intermediate VPLs such that $L =_{\text{cd}} \biguplus_{i=1}^m L(G_i)$ (Theorem 29 in Section 5.3). Already if L is weakly length-synchronous but not length-synchronous, one can compute $k, l \in \mathbb{N}_{>0}$ with $k \neq l$ such that $\mathcal{L}_{k,l} \leq_{\text{cd}} L$ (Proposition 30 in Section 5.3). \blacktriangleleft

5.1 Lower bounds

The following lower bound has already been given in Section 4.

► **Proposition 19.** *If L is not weakly length-synchronous, then L is TC^0 -hard.*

The following proposition has essentially already been shown in [20, Proposition 135], yet with some inaccuracies (we refer to Section 8 in [12]) that we fix. The idea is again a standard reduction from the word problem of non-trivial cyclic subgroups of $\psi_L(\mathcal{O}(\Sigma)^{k,l})$, in case the latter set contains a non-trivial group.

► **Proposition 20.** *If L is not quasi-aperiodic, then one can compute some $m \geq 2$ such that $\text{MOD}_m \leq_{\text{cd}} L$.*

As final lower bound result we prove a stronger lower bound, namely when the syntactic morphism not only is not quasi-aperiodic but the syntactic Ext-algebra is not solvable. We say the Ext-algebra (R, O) is *solvable* if all subsets of R or O that are groups (under the multiplication of R , resp. of O) are solvable. It is worth mentioning that one can prove that if $(\varphi, \psi): (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) \rightarrow (R, O)$ is quasi-aperiodic, then (R, O) is solvable.

Our proof that L is NC^1 -hard (and thus TC^0 -hard) when (R_L, O_L) is not solvable can essentially be reduced to the case for words [4], by showing that already $\psi_L(\mathcal{O}(\Sigma^\Delta)^{k,l})$ contains such a non-solvable group for some fixed $k, l \geq 0$.

► **Proposition 21.** *If (R_L, O_L) is not solvable, then L is NC^1 -hard and thus not in AC^0 .*

5.2 In AC^0 : Length-synchronous and quasi-aperiodic

In this section we concern ourselves with the following result.

► **Theorem 22.** *If L is length-synchronous and (φ_L, ψ_L) is quasi-aperiodic, then L is in $\text{FO}[+]$ and thus in AC^0 .*

For the rest of this section let us fix a VPL L , its syntactic Ext-algebra (R_L, O_L) , and its syntactic morphism $(\varphi_L, \psi_L): (\Sigma^\Delta, \mathcal{O}(\Sigma^\Delta)) \rightarrow (R_L, O_L)$. We first introduce suitably padded word languages mimicking the evaluation problem of the monoid R_L and the monoid O_L , respectively.

38:12 The AC⁰-Complexity of Visibly Pushdown Languages

For all $k \in \mathbb{N}$, we define $\mathcal{O}(\Sigma^\Delta)^{k,*} = \{\text{ext}_{u,v} \in \mathcal{O}(\Sigma^\Delta) : |u| = k\}$ and $\mathcal{O}(\Sigma^\Delta)^{*,k} = \{\text{ext}_{u,v} \in \mathcal{O}(\Sigma^\Delta) : |v| = k\}$. We also define $\mathcal{O}(\Sigma^\Delta)_\uparrow = \{\text{ext}_{u,v} \in \mathcal{O}(\Sigma^\Delta) \mid \Delta(u) > 0\}$ and finally for all $k \in \mathbb{N}$, we define $\mathcal{O}(\Sigma^\Delta)_\uparrow^{k,*} = \mathcal{O}(\Sigma^\Delta)^{k,*} \cap \mathcal{O}(\Sigma^\Delta)_\uparrow$ and $\mathcal{O}(\Sigma^\Delta)_\uparrow^{*,k} = \mathcal{O}(\Sigma^\Delta)^{*,k} \cap \mathcal{O}(\Sigma^\Delta)_\uparrow$. Consider the alphabets $\Gamma_{\varphi_L} = \varphi_L(\Sigma^\Delta \setminus \{\varepsilon\}) \cup \{\$\}$ and $\Gamma_{\psi_L} = \psi_L(\mathcal{O}(\Sigma^\Delta)_\uparrow) \cup \{\$\}$ for a letter $\$ \notin R_L \cup O_L$. We also define $\mathcal{V}_{\varphi_L} = \{\$^k s \mid k \in \mathbb{N}, s \in \varphi_L(\Sigma^{k+1})\}^*$ and $\mathcal{V}_{\psi_L} = \{\$^k f \mid k \in \mathbb{N}, f \in \psi_L(\mathcal{O}(\Sigma^\Delta)_\uparrow^{k+1,*})\}^*$. The following lemma holds irrespective of whether the syntactic morphism (φ_L, ψ_L) of L is quasi-aperiodic or not.

► **Lemma 23.** $\mathcal{V}_{\varphi_L}, \mathcal{V}_{\psi_L}$ are regular languages whose syntactic morphisms are quasi-aperiodic.

Define the φ_L -evaluation morphism $\text{eval}_{\varphi_L}: \Gamma_{\varphi_L}^* \rightarrow R_L$ by $\text{eval}_{\varphi_L}(s) = s$ for all $s \in \varphi_L(\Sigma^\Delta \setminus \{\varepsilon\})$ and $\text{eval}_{\varphi_L}(\$) = 1_{R_L}$. Similarly, let the morphism $\text{eval}_{\psi_L}: \Gamma_{\psi_L}^* \rightarrow O_L$ be defined as $\text{eval}_{\psi_L}(f) = f$ for all $f \in \psi_L(\mathcal{O}(\Sigma^\Delta)_\uparrow)$ and $\text{eval}_{\psi_L}(\$) = 1_{O_L}$. Finally, for all $r \in R_L$, we set $\mathcal{E}_{\varphi_L, r} = \mathcal{V}_{\varphi_L} \cap \text{eval}_{\varphi_L}^{-1}(r)$ and for all $e \in O_L$, $\mathcal{E}_{\psi_L, e} = \mathcal{V}_{\psi_L} \cap \text{eval}_{\psi_L}^{-1}(e)$.

The following proposition states that the respective evaluation languages $\mathcal{E}_{\varphi_L, r}$ and $\mathcal{E}_{\psi_L, e}$ are all quasi-aperiodic if the syntactic morphism (φ_L, ψ_L) of L is and L is additionally length-synchronous.

► **Proposition 24.** Let L be a VPL for which (φ_L, ψ_L) is quasi-aperiodic. Then $\mathcal{E}_{\varphi_L, r}$ is a regular language whose syntactic morphism is quasi-aperiodic for all $r \in R_L$. If L is length-synchronous, then $\mathcal{E}_{\psi_L, e}$ is a regular language whose syntactic morphism is quasi-aperiodic for all $e \in O_L$.

The following remark states that the length-synchronicity condition in the second point of Proposition 24 is important. In fact it shows that weak length-synchronicity is not sufficient.

► **Remark 25.** For the second point of Proposition 24 it is generally not sufficient to assume that L is weakly length-synchronous. Indeed, the VPL K generated by the grammar with rules $S \rightarrow aSb_1 \mid aTb_2 \mid \varepsilon$ and $T \rightarrow aTb_1 \mid aSb_2$ using S as start symbol is not length-synchronous (but weakly length-synchronous) and has a quasi-aperiodic syntactic morphism. However, for the syntactic Ext-algebra (R_K, O_K) and the syntactic morphism (φ_K, ψ_K) of K , one can prove that there exists $e \in O_K$ such that $\mathcal{E}_{\psi_K, e}$ is a regular language whose syntactic morphism is not quasi-aperiodic. Details can be found in [12].

Approximate matchings generalize the classical matching relation on well-matched words with respect to our VPL L in the sense that they are subsets of the matching relation but must equal the matching relation on all those words that are in L . Approximate matchings in the context of visibly pushdown languages were introduced by Ludwig [20].

For any word $w \in \Sigma^*$, we say that two positions $i, j \in [1, |w|]$ in w are *matched* whenever $i < j$, $w_i \in \Sigma_{\text{call}}$, $w_j \in \Sigma_{\text{ret}}$ and $w_{i+1} \cdots w_{j-1} \in \Sigma^\Delta$; we also say that i is *matched to* j in w . Given a word $w \in \Sigma^\Delta$, we denote by $M^\Delta(w)$ its *matching relation* (or *matching*), that is the relation $\{(i, j) \in [1, |w|]^2 \mid i \text{ is matched to } j \text{ in } w\}$. An *approximate matching relative to* $L \subseteq \Sigma^\Delta$ is a function $M: \Sigma^* \rightarrow \mathbb{N}_{>0}^2$ such that $M(w) = M^\Delta(w)$ for all $w \in L$ and $M(w) \subseteq M^\Delta(w)$ for all $w \in \Sigma^* \setminus L$.

The next lemma is an important tool for defining an approximate matching relation.

► **Lemma 26.** Assume that (φ_L, ψ_L) is weakly length-synchronous. Let $e \in O_L$ be $\varphi_L(L)$ -reaching and let $\mathcal{U}_e = \{(u, v) \in \text{Con}(\Sigma) \mid e \circ \psi_L(\text{ext}_{u,v}) = e\}$ be length-synchronous. Then there exists an $\text{FO}_{\Sigma}[+]$ -formula $\pi_e(x, x', y', y)$ such that for all $w \in \Sigma^+$ and $i, i', j', j \in [1, |w|]$, $i \leq i' < j' \leq j$ the following holds:

- if $w \models \pi_e(i, i', j', j)$, then $w_i \cdots w_{i'} w_{j'} \cdots w_j \in \Sigma^\Delta$ and
- if $w_i \cdots w_{i'} w_{j'} \cdots w_j \in \Sigma^\Delta$ and $(w_i \dots w_{i'}, w_{j'} \dots w_j) \in \mathcal{R}_e$, then $w \models \pi_e(i, i', j', j)$.

The proof of Lemma 26 takes several steps. The formula π_e expresses the characterization of length-synchronicity given by Proposition 16 via an $\text{FO}_{\Sigma}[+]$ -formula. To realize these we make use of aperiodicity of the following languages $L_{p,q}$. For each $p \in \mathbb{N}$ let $\Gamma_p = \{a_{-p}, \dots, a_{-1}, a_0, a_1, \dots, a_p\}$ and let $\Delta_p: \Gamma_p^* \rightarrow \mathbb{Z}$ be the morphism satisfying $\Delta_p(a_h) = h$ for all $a_h \in \Gamma_p$. Let $L_{p,q} = \{w \in \Gamma_p^* \mid \Delta_p(w) = 0 \wedge \forall i \in [1, |w|], -q \leq \Delta_p(w_1 \cdots w_i) \leq q\}$. One can prove that this language is recognized by a finite aperiodic monoid. This implies, by a theorem by McNaughton and Papert (see [23, Theorem VI.1.1]), that there exists an $\text{FO}_{\Gamma_{n+d}}[<]$ -sentence defining $L_{p,q}$. These ingredients are central for expressing the characterization of length-synchronicity given by Proposition 16 via an $\text{FO}_{\Sigma}[+]$ -formula.

With the help of the predicates π_e provided by Lemma 26 one can build an $\text{FO}_{\Sigma}[+]$ -definable approximate matching relative to any length-synchronous visibly pushdown language. The proof is by induction on the nesting depth.

► **Proposition 27.** *If $L \subseteq \Sigma^{\Delta}$ is length-synchronous, then there exists an $\text{FO}_{\Sigma}[+]$ -formula $\eta(x, y)$ such that $M: \Sigma^* \rightarrow \mathbb{N}_{>0}^2$ defined by $M(w) = \{(i, j) \in [1, |w|]^2 \mid w \models \eta(i, j)\}$ for all $w \in \Sigma^*$ is an approximate matching relative to L .*

The following proposition states that a VPL L is definable by some $\text{FO}_{\Sigma, \rightsquigarrow}[+]$ sentence in case L has bounded nesting depth, the evaluation languages $\mathcal{E}_{\varphi_L, r}$ and $\mathcal{E}_{\psi_L, e}$ are all quasi-aperiodic, and any approximate matching is present as built-in predicate.

► **Proposition 28.** *Assume a VPL L has bounded nesting depth and $\mathcal{E}_{\varphi_L, r}$ and $\mathcal{E}_{\psi_L, e}$ are regular languages whose syntactic morphisms are quasi-aperiodic for all $r \in R_L$ and for all $e \in O_L$. Then there exists an $\text{FO}_{\Sigma, \rightsquigarrow}[+]$ -sentence η such that for all approximate matchings M relative to L , we have $w \in L$ if, and only if, $(w, M(w)) \models \eta$ for all $w \in \Sigma^*$.*

Proof (Sketch). By hypothesis, there exists $d_L \in \mathbb{N}$ bounding the nesting depth of the words in L . By hypothesis also, for each $r \in R_L$, the language $\mathcal{E}_{\varphi_L, r}$ is regular and its syntactic morphism is quasi-aperiodic. This implies, by [23, Theorem VI.4.1], that for each $r \in R_L$, there exists an $\text{FO}_{\Gamma_{\varphi_L}}[<, \text{MOD}]$ -sentence $\nu_{\varphi_L, r}$ defining $\mathcal{E}_{\varphi_L, r}$. Finally, by hypothesis, for each $e \in O_L$, the language $\mathcal{E}_{\psi_L, e}$ is regular and its syntactic morphism is quasi-aperiodic. Analogously, for each $e \in O_L$, there exists an $\text{FO}_{\Gamma_{\psi_L}}[<, \text{MOD}]$ -sentence $\nu_{\psi_L, e}$ defining $\mathcal{E}_{\psi_L, e}$.

To build the $\text{FO}_{\Sigma, \rightsquigarrow}[+]$ -sentence η , we build $\text{FO}_{\Sigma, \rightsquigarrow}[+]$ -formulas $\eta_{d,r}^{\uparrow}(x, y)$ and $\eta_{d,r}(x, y)$ for all $d \in [0, d_L]$ and all $r \in R_L$. They will have the following properties for all $w \in \Sigma^{\Delta}$ and all $i, j \in [1, |w|]$, where M^{Δ} is the full matching relation: (1) if i is matched to j in w , then $(w, M^{\Delta}(w)) \models \eta_{d,r}^{\uparrow}(i, j)$ if, and only if, $\text{nd}(w_i \cdots w_j) \leq d$ and $\varphi_L(w_i \cdots w_j) = r$ and (2) if $w_i \cdots w_j \in \Sigma^{\Delta}$, then $(w, M^{\Delta}(w)) \models \eta_{d,r}(i, j)$ if, and only if, $\text{nd}(w_i \cdots w_j) \leq d$ and $\varphi_L(w_i \cdots w_j) = r$. It is not difficult to construct a formula $N_d(x, y)$ (that also accesses the full matching relation) such that for all $w \in \Sigma^*$ and all infixes $w_i \dots w_j \in \Sigma^{\Delta}$ we have $w \models N_d(i, j)$ if, and only if, $\text{nd}(w_i \dots w_j) \leq d$. Let the formula E be $\forall x(x \neq x)$ if $\varepsilon \in L$ and $\perp = \exists x(x \neq x)$ otherwise. Observe that $w \models \forall x(x \neq x)$ if, and only if, $w = \varepsilon$. Letting \rightsquigarrow being interpreted over any approximate matching relation, the formula η is then defined as the conjunction of $\forall z \exists t ((\Sigma_{\text{call}}(z) \rightarrow z \rightsquigarrow t) \wedge (\Sigma_{\text{ret}}(z) \rightarrow t \rightsquigarrow z))$ and $E \vee \exists x \exists y (\neg \exists x'(x' < x) \wedge \neg \exists y'(y < y') \wedge \bigvee_{r \in \varphi_L(L)} \eta_{d_L, r}(x, y))$.

Let us give some intuition on how to build $\eta_{d,r}^{\uparrow}(x, y)$ and $\eta_{d,r}(x, y)$ for all $d \in \mathbb{N}$ and $r \in R_L$. The construction is by induction on d . Let $r \in R_L$. We define $\eta_{0,r}^{\uparrow}(x, y) = \perp$. We define $\eta_{0,r}$ as $\eta_{0,r}(x, y) = \neg N_1(x, y) \wedge \tau_0(\nu_{\varphi_L, r})$, where the translation τ_0 is defined as follows: $\tau_0(z < z') = z < z'$, $\tau_0(s(z)) = \bigvee_{c \in \varphi_L^{-1}(s) \cap \Sigma_{\text{int}}} c(z)$ for all $s \in \varphi_L(\Sigma^{\Delta} \setminus \{\varepsilon\})$, $\tau_0(\text{MOD}_m(z)) = \exists t(z - x + 1 = t \cdot m)$ for all $m \in \mathbb{N}_{>0}$, $\tau_0(\$ (z)) = \perp$, $\tau_0(\rho_1(z_1) \wedge \rho_2(z_2)) = \tau_0(\rho_1(z_1)) \wedge \tau_0(\rho_2(z_2))$, $\tau_0(\neg \rho(z)) = \neg \tau_0(\rho(z))$, and $\tau_0(\exists z \rho(z, z)) = \exists z(x \leq z \leq y \wedge \tau_0(\rho(z, z)))$

Now let $d > 0$. Define the formula $A(x, y, z) = \exists x' \exists y' (x \leq x' \leq z < y' \leq y \wedge x' \leftrightarrow y')$ that expresses that for all $w \in \Sigma^\Delta$ and $i, j, k \in [1, |w|]$ satisfying $w_i \cdots w_j \in \Sigma^\Delta$, we have $(w, M^\Delta(w)) \models A(i, j, k)$ if, and only if, $i \leq k < j$ and $\Delta(w_i \cdots w_k) > 0$. Let us first define $\eta_{d,r}$ when assuming that we have already defined $\eta_{d,r}^\uparrow$. Note that in case $\text{nd}(u) \leq d$, then one can factorize u as $u = u_1 \cdots u_m$ such that $u_i \in \Sigma_{\text{int}}^+ \cup \Sigma_{\text{call}} \Sigma^\Delta \Sigma_{\text{ret}}$ and $\text{nd}(u_i) \leq d$ for all $i \in [1, m]$. Using this observation we define $\eta_{d,r}(x, y) = \neg N_{d+1}(x, y) \wedge \tau_1(\nu_{\varphi_L, r})$, where the translation τ_1 agrees with the above translation τ_0 (where, as expected, occurrences of τ_0 are replaced by τ_1) except for the following kinds of subformulas: $\tau_1(\$ (z)) = A(x, y, z)$ and $\tau_1(s(z)) = \neg A(x, y, z) \wedge \left(\bigvee_{c \in \varphi_L^{-1}(s) \cap \Sigma_{\text{int}}} c(z) \vee \exists t (x \leq t \leq y \wedge t \leftrightarrow z \wedge \eta_{d,s}^\uparrow(t, z)) \right)$.

Similarly, in the definition of $\eta_{d,r}^\uparrow$ we make use of our sentences $\nu_{\psi_L, e}$ for the evaluation languages $\mathcal{E}_{\psi_L, e}$ for all $e \in O_L$. For these however we make use of an auxiliary formula U such that for all $w \in \Sigma^\Delta$ and $i, i', k \in [1, |w|]$ we have $(w, M^\Delta(w)) \models U(i, i', k)$ if, and only if, $i \leq k \leq i'$ and k is matched with some position larger than i' , thus expressing that for some positions j, j' the position k is an ‘‘upward stair position’’ in the stair factorization of $\text{ext}_{u_i \cdots u_{i'}, u_j \cdots u_{j'}} = \text{ext}_{x_1, y_1} \circ \cdots \circ \text{ext}_{a_{h-1}, b_{h-1}} \circ \text{ext}_{x_h, y_h}$: more precisely k is one of the positions $\{i + |x_1 a_1| - 1, i + |x_1 a_1 x_2 a_2| - 1, \dots, i + |x_1 a_1 \dots x_{h-1} a_{h-1}| - 1\}$. These positions k will be precisely the ones where the predicate $\$$ does not hold in a suitable translation of $\nu_{\psi_L, e}$. ◀

Proof of Theorem 22. Proposition 24 implies that $\mathcal{E}_{\varphi_L, r}$ and $\mathcal{E}_{\psi_L, e}$ are regular languages whose respective syntactic morphisms are quasi-aperiodic for all $r \in R_L$ and all $e \in O_L$, respectively. Thus, the first two conditions of Proposition 28 are satisfied. Moreover, Proposition 27 provides a first-order definable approximate matching relation relative to L , being a predicate assumed by Proposition 28. Finally, Proposition 28 implies Theorem 22. ◀

5.3 The intermediate case

The following theorem effectively characterizes the remaining case, namely those VPLs that are weakly length-synchronous but not length-synchronous and whose syntactic morphism is quasi-aperiodic: such VPLs are shown to be constant-depth equivalent to a non-empty disjoint union of intermediate languages.

► **Theorem 29.** *If a VPL L is weakly length-synchronous, not length-synchronous, and its syntactic morphism (φ_L, ψ_L) is quasi-aperiodic, one can compute vertically visibly pushdown grammars G_1, \dots, G_m generating intermediate VPLs such that $L =_{\text{cd}} \biguplus_{i=1}^m L(G_i)$.*

Let $L \subseteq \Sigma^\Delta$ be a weakly length-synchronous VPL that is not length-synchronous, and whose syntactic morphism (φ_L, ψ_L) is quasi-aperiodic. By Proposition 18 one can compute its syntactic Ext-algebra (R_L, O_L) , (φ_L, ψ_L) and $\varphi_L(L)$ from (a given DVPA for) L . For all $\varphi_L(L)$ -reaching $e \in O_L$ and some fresh internal letter $\# \notin \Sigma$ let $M_e = \{u\#v \mid \Delta(u) > 0, (u, v) \in \mathcal{U}_e\}$, which can be shown to be a computable VPL.

The set $\mathcal{Z} = \{e \in O_L \mid e \text{ is } \varphi_L(L)\text{-reaching and } \mathcal{U}_e \text{ is not length-synchronous}\}$ can be proven to be computable. Observe that as L is not length-synchronous by assumption, we have $\mathcal{Z} \neq \emptyset$ (Proposition 15). Next make use of Lemma 14 stating that the hills in the stair factorization of any $\text{ext}_{u,v} \in \psi_L^{-1}(e)$ are constantly bounded for all $\varphi_L(L)$ -reaching $e \in O_L$. This gives rise to computable intermediate languages N_e such that $N_e =_{\text{cd}} M_e$ for all $e \in \mathcal{Z}$. Letting $L_f = \{u\#v \mid \psi(\text{ext}_{u,v}) = f\}$ for all $f \in O_L$, it is standard to show $L_f \leq_{\text{cd}} L$ for all $\varphi_L(L)$ -reaching $f \in O_L$. Finally, one proves $M_e \leq_{\text{cd}} \biguplus_{f \in O_L \text{ is } \varphi_L(L)\text{-reaching}} L_f$ for all $e \in \mathcal{Z}$ and $L \leq_{\text{cd}} \biguplus_{e \in \mathcal{Z}} M_e$ thus establishing $L =_{\text{cd}} \biguplus_{e \in \mathcal{Z}} N_e$. The proof of $L \leq_{\text{cd}} \biguplus_{e \in \mathcal{Z}} M_e$ is the technically most demanding and is an adaption of the proof of Proposition 28: alas, one

cannot assume presence of the evaluation $\text{FO}_{\Gamma_{\psi_L}}[<, \text{MOD}]$ -sentence $\nu_{\psi_L, e}$ for each $e \in O_L$ since $\mathcal{E}_{\psi_L, e}$ can possibly have a non-quasi-aperiodic syntactic morphism by Remark 25. Yet, one can realize evaluation via access to the oracle $\biguplus_{e \in \mathcal{Z}} M_e$.

The following proposition implies the computability of $k, l \in \mathbb{N}_{>0}$ such that $\mathcal{L}_{k,l} \leq_{\text{cd}} L$ already when a VPL L is weakly length-synchronous but not length-synchronous.

► **Proposition 30.** *If a VPL L is weakly length-synchronous but not length-synchronous, one can compute $k, l \in \mathbb{N}_{>0}$ with $k \neq l$ such that $\mathcal{L}_{k,l} \leq_{\text{cd}} L$.*

Proof. Let $L \subseteq \Sigma^\Delta$ be a weakly length-synchronous VPL that is not length-synchronous. According to Point 2 (b) of Proposition 18 one can compute a quadruple $(k_0, l_0, k'_0, l'_0) \in \mathbb{N}_{>0}^4$ for which there exist $\text{ext}_{u,v}, \text{ext}_{u',v'} \in \mathcal{O}(\Sigma^\Delta)$ such that $|u| = k_0, |v| = l_0, |u'| = k'_0, |v'| = l'_0, \psi_L(\text{ext}_{u,v}) = \psi_L(\text{ext}_{u',v'})$ is a $\varphi_L(L)$ -reaching idempotent, $\Delta(u), \Delta(u') > 0$, and $\frac{k_0}{l_0} = \frac{|u|}{|v|} \neq \frac{|u'|}{|v'|} = \frac{k'_0}{l'_0}$. We can compute such $\text{ext}_{u,v}$ and $\text{ext}_{u',v'}$ by just doing an exhaustive search. This enables us to assume without loss of generality that $\Delta(u) = \Delta(u')$: indeed, in case $\Delta(u) \neq \Delta(u')$, we can consider $\text{ext}_{u_1, v_1} = \text{ext}_{u\Delta(u'), v\Delta(u')}$ and $\text{ext}_{u_2, v_2} = \text{ext}_{(u')\Delta(u), (v')\Delta(u)}$.

Let us now define Green's relations on O_L . Let us consider two elements x, y of O_L . We write $x \leq_{\mathfrak{J}} y$ whenever there are elements e, f of O_L such that $x = e \circ y \circ f$. We write $x \mathfrak{J} y$ if $x \leq_{\mathfrak{J}} y$ and $y \leq_{\mathfrak{J}} x$. We write $x <_{\mathfrak{J}} y$ if $x \leq_{\mathfrak{J}} y$ and $x \not\mathfrak{J} y$. We write $x \leq_{\mathfrak{R}} y$ whenever there is an element e of O_L such that $x = y \circ e$. We write $x \mathfrak{R} y$ if $x \leq_{\mathfrak{R}} y$ and $y \leq_{\mathfrak{R}} x$. We write $x \leq_{\mathfrak{L}} y$ whenever there is an element e of O_L such that $x = e \circ y$. We write $x \mathfrak{L} y$ if $x \leq_{\mathfrak{L}} y$ and $y \leq_{\mathfrak{L}} x$. We write $x \mathfrak{H} y$ if $x \mathfrak{R} y$ and $x \mathfrak{L} y$.

Observe that because $\Delta(u) = \Delta(u')$, we have that $uv' \in \Sigma^\Delta$ and $u'v \in \Sigma^\Delta$, so that we can consider the elements $\text{ext}_{uuu, vv'v} = \text{ext}_{u,v} \circ \text{ext}_{u,v'} \circ \text{ext}_{u,v}$ and $\text{ext}_{uu'u, vvv} = \text{ext}_{u,v} \circ \text{ext}_{u',v'} \circ \text{ext}_{u,v}$ in $\mathcal{O}(\Sigma^\Delta)$. These elements satisfy $\psi_L(\text{ext}_{uuu, vv'v}) \leq_{\mathfrak{J}} \psi_L(\text{ext}_{u,v})$ and $\psi_L(\text{ext}_{uu'u, vvv}) \leq_{\mathfrak{J}} \psi_L(\text{ext}_{u,v})$. We claim that we actually have $\psi_L(\text{ext}_{uuu, vv'v}) <_{\mathfrak{J}} \psi_L(\text{ext}_{u,v})$ and $\psi_L(\text{ext}_{uu'u, vvv}) <_{\mathfrak{J}} \psi_L(\text{ext}_{u,v})$. Indeed, assume we had $\psi_L(\text{ext}_{uuu, vv'v}) \mathfrak{J} \psi_L(\text{ext}_{u,v})$. Set $x = \psi_L(\text{ext}_{u,v})$ and $y = \psi_L(\text{ext}_{uu'u, vvv})$. Since it would hold that $x \circ y \circ x \leq_{\mathfrak{R}} x$ and $x \circ y \circ x \mathfrak{J} x$, we would have $x \circ y \circ x \mathfrak{R} x$ and dually, since it would hold that $x \circ y \circ x \leq_{\mathfrak{L}} x$ and $x \circ y \circ x \mathfrak{J} x$, we would have $x \circ y \circ x \mathfrak{L} x$. Therefore, we would have $x \circ y \circ x \mathfrak{H} x$. As x is an idempotent, its \mathfrak{H} -class is a group, hence for $\omega \in \mathbb{N}_{>0}$ the idempotent power of O_L , we would have $(x \circ y \circ x)^\omega = x^\omega = x$ (as the only idempotent element in a group is the identity). This would finally entail that $\psi_L(\text{ext}_{(uu'u)^\omega, (vvv)^\omega}) = \psi_L(\text{ext}_{uuu, vv'v})$ is a $\varphi_L(L)$ -reaching idempotent and $\Delta((uu'u)^\omega) = \Delta((uuu)^\omega) > 0$ but $|(uu'u)^\omega| \neq |(uuu)^\omega|$, a contradiction to the fact that (φ_L, ψ_L) is $\varphi_L(L)$ -weakly-length-synchronous. Symmetrically, we can prove that if we had $\psi_L(\text{ext}_{uuu, vv'v}) \mathfrak{J} \psi_L(\text{ext}_{u,v})$, this would contradict the fact that (φ_L, ψ_L) is $\varphi_L(L)$ -weakly-length-synchronous.

Here we only treat the case when $|v| = |v'|$ and refer to [12] for the full proof of the other cases. We prove that there exist $k, l \in \mathbb{N}_{>0}, k \neq l$ such that $\mathcal{L}_{k,l} \leq_{\text{cd}} L_{\psi_L(\text{ext}_{u,v})}$, so that since $L_{\psi_L(\text{ext}_{u,v})} \leq_{\text{cd}} L$ (as already mentioned in Section 5.3 this is standard) and by transitivity of \leq_{cd} we have $\mathcal{L}_{k,l} \leq_{\text{cd}} L$. In that case, we necessarily have $|u| \neq |u'|$. Then, we can exploit the fact that matching u^3 with $vv'v$ or $uu'u$ with v^3 makes us fall down to a smaller \mathfrak{J} -class to reduce $\mathcal{L}_{3|u|, 2|u|+|u'|}$ to $L_{\psi_L(\text{ext}_{u,v})}$. The constant-depth reduction works as follows on input $w \in \Sigma^*$: (i) check if $w = xy$ with $x \in (ac^{3|u|-1} + ac^{2|u|+|u'|-1})^*$ and $y \in (b_1 + b_2)^*$, reject otherwise; (ii) build x' by sending $ac^{3|u|-1}$ to u^3 , $ac^{2|u|+|u'|-1}$ to $uu'u$ and y' by sending b_1 to v^3 and b_2 to $vv'v$; (iii) accept whenever $x' \# y' \in L_{\psi_L(\text{ext}_{u,v})}$. This forms a valid reduction. Indeed, take a word $w = xy$ with $x \in (ac^{3|u|-1} + ac^{2|u|+|u'|-1})^n$ for $n \in \mathbb{N}$ and $y \in (b_1 + b_2)^m$ for $m \in \mathbb{N}$ and consider $x' \# y'$ produced by the reduction with $x' \in (u^3 + uu'u)^n$ and $y' \in (v^3 + vv'v)^m$. If $w \in \mathcal{L}_{3|u|, 2|u|+|u'|}$, then it easily follows that

$x' \# y' \in L_{\psi_L(\text{ext}_{u,v})}$. Otherwise, if $w \notin \mathcal{L}_{3|u|, 2|u|+|u'|}$, then either $n \neq m$ and thus $x'y'$ is not well-matched because $\Delta(x') = n \cdot 3 \cdot \Delta(u)$ and $\Delta(y') = m \cdot 3 \cdot \Delta(v)$, or $n = m$ and thus $x'y'$ is well-matched, so $\text{ext}_{x',y'} = \text{ext}_{z'_1, t'_1} \circ \dots \circ \text{ext}_{z'_n, t'_n}$ with $z'_1, \dots, z'_n \in \{u^3, uu'u\}$ and $t'_1, \dots, t'_n \in \{v^3, vv'v\}$ such that there exists $i \in [1, n]$ satisfying $\text{ext}_{z'_i, t'_i} \in \{\text{ext}_{u^3, vv'v}, \text{ext}_{uu'u, v^3}\}$, thereby implying $\psi_L(\text{ext}_{x',y'}) \leq_{\mathfrak{J}} \psi_L(\text{ext}_{z'_i, t'_i}) <_{\mathfrak{J}} \psi_L(\text{ext}_{u,v})$. Hence, our algorithm outputs the pair $(k, l) = (3k_0, 2k_0 + k'_0)$. \blacktriangleleft

6 Conclusion

In this paper we have studied the question of which visibly pushdown languages lie in the complexity class AC^0 . We have introduced the notions of length-synchronicity, weak length-synchronicity and quasi-counterfreeness. We have introduced intermediate VPLs: these are quasi-counterfree VPLs generated by context-free grammars G involving the production $S \rightarrow_G \varepsilon$ for the start nonterminal S and whose further productions are all of the form $T \rightarrow_G uT'v$, where uv is well-matched, $u \in (\Sigma_{\text{int}}^* \Sigma_{\text{call}} \Sigma_{\text{int}}^*)^+$, $v \in (\Sigma_{\text{int}}^* \Sigma_{\text{ret}} \Sigma_{\text{int}}^*)^+$, and the set of contexts $\{(u, v) \in \text{Con}(\Sigma) \mid S \Rightarrow_G^* uSv\}$ is weakly length-synchronous but not length-synchronous. To the best of our knowledge it is unclear whether at all there is an intermediate VPL that is provably in AC^0 (even in ACC^0) or provably not in AC^0 . We conjecture that none of the intermediate VPLs are in ACC^0 nor TC^0 -hard. Our main result states that there is an algorithm that, given a visibly pushdown language L , outputs if L surely lies in AC^0 , surely does not lie in AC^0 (by providing some $m > 1$ such that MOD_m is constant-depth reducible to L), or outputs a disjoint finite union of intermediate VPLs that L is constant-depth equivalent to. In the latter of the three cases one can moreover compute distinct $k, l \in \mathbb{N}_{>0}$ such that already $\mathcal{L}_{k,l} = L(S \rightarrow \varepsilon \mid ac^{k-1}Sb_1 \mid ac^{l-1}Sb_2)$ is constant-depth reducible to L . We conjecture that due to the particular nature of intermediate VPLs, either all of them are in AC^0 or all are not: this conjecture together with our main result indeed implies that there is an algorithm that decides if a given visibly pushdown language is in AC^0 . As main tools we carefully revisited Ext-algebras, introduced by Czarnetzki et al. [9], being closely related to forest algebras, introduced by Bojańczyk and Walukiewicz [7]. For the reductions from $\mathcal{L}_{k,l}$ we made use of Green's relations.

Natural questions arise. Is there any *concrete* intermediate VPL that is provably in ACC^0 , provably not in AC^0 , or hard for TC^0 ? Another exciting question is whether one can effectively compute those visibly pushdown languages that lie in the complexity class TC^0 . Is there is a TC^0/NC^1 complexity dichotomy? For these questions new techniques seem to be necessary. In this context it is already interesting to mention there is an NC^1 -complete visibly pushdown language whose syntactic Ext-algebra is aperiodic. Another exciting question is to give an algebraic characterization of the visibly counter languages.

References

- 1 Rajeev Alur, Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Congruences for visibly pushdown languages. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 1102–1114. Springer, 2005. doi:10.1007/11523468_89.
- 2 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.

- 3 Vince Bárány, Christof Löding, and Olivier Serre. Regularity problems for visibly pushdown languages. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2006. doi:10.1007/11672142_34.
- 4 David A. Mix Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1 . *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. doi:10.1016/0022-0000(89)90037-8.
- 5 David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in nc^1 . *J. Comput. Syst. Sci.*, 44(3):478–499, 1992. doi:10.1016/0022-0000(92)90014-A.
- 6 Richard Beigel. The polynomial method in circuit complexity. In *Proceedings of the Eighth Annual Structure in Complexity Theory Conference, San Diego, CA, USA, May 18-21, 1993*, pages 82–95. IEEE Computer Society, 1993. doi:10.1109/SCT.1993.336538.
- 7 Mikolaj Bojanczyk and Igor Walukiewicz. Forest algebras. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 107–132. Amsterdam University Press, 2008.
- 8 Ashok K. Chandra, Larry J. Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM J. Comput.*, 13(2):423–439, 1984. doi:10.1137/0213028.
- 9 Silke Czarnetzki, Andreas Krebs, and Klaus-Jörn Lange. Visibly pushdown languages and free profinite algebras. *CoRR*, abs/1810.12731, 2018. arXiv:1810.12731.
- 10 Patrick W. Dymond. Input-driven languages are in $\log n$ depth. *Inf. Process. Lett.*, 26(5):247–250, 1988. doi:10.1016/0020-0190(88)90148-2.
- 11 Javier Esparza, Michael Luttenberger, and Maximilian Schlund. A brief history of strahler numbers. In Adrian-Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, volume 8370 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2014. doi:10.1007/978-3-319-04921-2_1.
- 12 Stefan Göller and Nathan Grosshans. The ac^0 -complexity of visibly pushdown languages. *CoRR*, abs/2302.13116, 2023. doi:10.48550/ARXIV.2302.13116.
- 13 Yuri Gurevich and Harry R. Lewis. A logic for constant-depth circuits. *Inf. Control.*, 61(1):65–74, 1984. doi:10.1016/S0019-9958(84)80062-5.
- 14 Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.
- 15 Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986. doi:10.1145/12130.12132.
- 16 Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987. doi:10.1137/0216051.
- 17 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.
- 18 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.
- 19 Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig. Visibly counter languages and constant depth circuits. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPICs*, pages 594–607. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.STACS.2015.594.
- 20 Michael Ludwig. *Tree-Structured Problems and Parallel Computation*. PhD thesis, University of Tübingen, Germany, 2019. URL: <https://publikationen.uni-tuebingen.de/xmlui/handle/10900/85960/>.

- 21 Kurt Mehlhorn. Pebbling Mountain Ranges and its Application of DCFL-Recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordwijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980. doi:10.1007/3-540-10003-2_89.
- 22 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control.*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.
- 23 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser Boston, 1994. doi:10.1007/978-1-4612-0289-9.
- 24 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. doi:10.1007/978-3-662-03927-4.

Quantum and Classical Communication Complexity of Permutation-Invariant Functions

Ziyi Guan ✉

EPFL, Lausanne, Switzerland

Yunqi Huang ✉

University of Technology Sydney, Australia

Penghui Yao ✉

State Key Laboratory for Novel Software Technology, Nanjing University, China
Hefei National Laboratory, China

Zekun Ye ✉

State Key Laboratory for Novel Software Technology, Nanjing University, China

Abstract

This paper gives a nearly tight characterization of the quantum communication complexity of the permutation-invariant Boolean functions. With such a characterization, we show that the quantum and randomized communication complexity of the permutation-invariant Boolean functions are quadratically equivalent (up to a logarithmic factor). Our results extend a recent line of research regarding query complexity [2, 16, 11] to communication complexity, showing symmetry prevents exponential quantum speedups.

Furthermore, we show the Log-rank Conjecture holds for any non-trivial total permutation-invariant Boolean function. Moreover, we establish a relationship between the quantum/classical communication complexity and the approximate rank of permutation-invariant Boolean functions. This implies the correctness of the Log-approximate-rank Conjecture for permutation-invariant Boolean functions in both randomized and quantum settings (up to a logarithmic factor).

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Communication complexity, Permutation-invariant functions, Log-rank Conjecture, Quantum advantages

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.39

Related Version *Full Version:* <https://arxiv.org/pdf/2401.00454.pdf> [26]

Funding *Ziyi Guan:* partially supported by the Ethereum Foundation.

Penghui Yao: supported by National Natural Science Foundation of China (Grant No. 62332009, 12347104, 61972191) and Innovation Program for Quantum Science and Technology (Grant No. 2021ZD0302901).

Zekun Ye: supported by National Natural Science Foundation of China (Grant No. 62332009, 12347104, 61972191) and Innovation Program for Quantum Science and Technology (Grant No. 2021ZD0302901).

1 Introduction

Exploring quantum advantages is a key problem in the realm of quantum computing. Numerous work focuses on analyzing and characterizing quantum advantages, such as [6, 14, 24, 20, 29, 44]. It has been known that quantum computing demonstrates a potential exponential speedup to solve certain problems than classical computers, such as Simon's problem [40] and integer factoring problem [39]. However, for some problems, the quantum



© Ziyi Guan, Yunqi Huang, Penghui Yao, and Zekun Ye;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 39; pp. 39:1–39:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



speedups can be at most polynomial, including the unstructured search problems [25] and collision finding problems [4]. In light of the aforementioned phenomenon, Aaronson [1] proposed such a problem: How much structure is needed for huge quantum speedups?

Regarding the above problem, there exist two major directions to explore the structure needed for quantum speedups in the query model, which is a complexity model commonly used to describe quantum advantages. On the one hand, Aaronson and Ambainis [2] conjectured the acceptance probability of a quantum query algorithm to compute a Boolean function can be approximated by a classical deterministic algorithm with only a polynomial increase in the number of queries, which is still one of most important conjecture in the field of Boolean analysis. On the other hand, Watrous conjectured that the quantum and randomized query complexities are also polynomially equivalent for any permutation-invariant function [2]. Along this direction, Aaronson and Ambainis [2] initiated the study on the quantum speedup of permutation-invariant functions with respect to query complexity. They demonstrated that a function invariant under full symmetry does not exhibit exponential quantum speedups, even if the function is partial, thereby resolving the Watrous conjecture. (Interested readers may refer to [2] for a more detailed introduction.) Furthermore, Chailloux [16] expanded upon their work by providing a tighter bound and removing a technical dependence of output symmetry. Recently, Ben-David, Childs, Gilyén, Kretschmer, Podder and Wang [11] further proved that hypergraph symmetries in the adjacency matrix model allow at most a polynomial separation between randomized and quantum query complexities. All the above results demonstrated that symmetries break exponential quantum speedups in the query model.

While the study of problem structure in the roles of quantum speedups has obtained considerable attention in the query model, it is a natural question to consider whether we can derive similar results in other computation models. The communication complexity model comes to attention as it is also extensively used to demonstrate quantum advantages. Furthermore, while the exponential gap between quantum and classical communication models has been shown in many works [35, 7, 21, 22, 33], there are also some problems in communication models that demonstrate at most polynomial quantum speedups, such as set disjointness problem [36] and (gap) Hamming distance problem [28, 38, 43, 18]. Therefore, it is a meaningful question to consider how much structure is needed for significant quantum speedups in the communication complexity model. More specifically, while symmetry breaks quantum exponential advantages in the query model, does there exist a similar conclusion in the communication complexity model? In this paper, we investigate a variant of the Watrous conjecture concerning the quantum and randomized communication complexities of permutation-invariant functions as follows. Briefly, a permutation-invariant Boolean function is a function that is invariant under permutations of its inputs (see Definition 8 for a formal definition).

► **Conjecture 1** (Communication complexity version of the Watrous Conjecture.). *For any permutation-invariant function $f : [m]^n \times [m]^n \rightarrow \{-1, 1, *\}$, $R(f) \leq Q^*(f)^{O(1)}$, where $R(f)$ and $Q^*(f)$ are the randomized and quantum communication complexities of f , respectively.*

Furthermore, we study the Log-rank Conjecture proposed by Lovasz and Saks [31], a long-standing open problem in communication complexity. Despite its slow progress on total functions [12, 32, 30], the conjecture has been shown for several subclasses of total permutation-invariant Boolean functions [15] and XOR-symmetric functions [45]. Lee and Shraibman further proposed the Log-approximate-rank Conjecture, stating that the randomized communication and the logarithm of the approximate rank of the input matrix are polynomially equivalent. Surprisingly, this conjecture was later proven false [19], even for its quantum counterpart [5, 41].

In this paper, we investigate both conjectures for permutation-invariant functions.

► **Conjecture 2** (Log-rank Conjecture for permutation-invariant functions.). *For any total permutation-invariant function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1\}$, $D(f) \leq (\log \text{rank}(f))^{O(1)}$, where $\text{rank}(f)$ is the rank of the input matrix of f .*

► **Conjecture 3** (Log-Approximate-Rank Conjecture for permutation-invariant functions.). *For any (total or partial) permutation-invariant function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1, *\}$, $R(f) \leq (\log \widetilde{\text{rank}}(f))^{O(1)}$, where $\widetilde{\text{rank}}(f)$ is the approximate rank of the input matrix of f (see Definition 11 for a formal definition).*

► **Conjecture 4** (Quantum Log-Approximate-Rank Conjecture for permutation-invariant functions.). *For any (total or partial) permutation-invariant function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1, *\}$, $Q(f) \leq (\log \widetilde{\text{rank}}(f))^{O(1)}$.*

1.1 Our Contribution

To study the communication complexity version of the Watrous conjecture, we start with permutation-invariant Boolean functions, which are essential to analyze general permutation-invariant functions. We show that for any permutation-invariant Boolean function, its classical communication complexity has at most a quasi-quadratic blowup comparing to its quantum communication complexity (Theorem 5). Thus, we cannot hope for exponential quantum speedups of permutation-invariant Boolean functions. Additionally, Theorem 5 gives a nearly tight bound on the quantum communication complexity. Furthermore, we show that every non-trivial permutation-invariant Boolean function satisfies the Log-rank Conjecture in Theorem 6. To resolve the (quantum) Log-Approximate-Rank Conjecture, we investigate the relationship between the quantum/classical communication complexities and the approximate rank of any permutation-invariant Boolean function in Theorem 7.

Consider a Boolean function f . Let $D(f), R(f), Q(f)$ and $Q^*(f)$ be the deterministic communication complexity, randomized communication complexity, quantum communication complexity without prior entanglement, and quantum communication complexity of f , respectively. Let $\text{rank}(f)$ and $\widetilde{\text{rank}}(f)$ be the rank and approximate rank of f . We summarize our results below¹.

► **Theorem 5.** *For any permutation-invariant function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1, *\}$ in Definition 8, the followings hold:*

$$\Omega(m(f)) \leq R(f) \leq \tilde{O}(m(f)^2) \leq \tilde{O}(Q^*(f)^2) \text{ and}$$

$$\Omega(m(f)) \leq Q^*(f) \leq Q(f) \leq \tilde{O}(m(f)),$$

where $m(f)$ is a measure defined in Definition 12. Hence, $R(f) \leq \tilde{O}(Q^*(f)^2)$ for any permutation-invariant function f .

The complexity measure $m(\cdot)$ is inspired by the work [23], where Gahzi et al. introduced a complexity measure to capture $R(f)$. It is worth noting that their complexity measure is equivalent up to a fourth power of $R(f)$, while our complexity measure $m(\cdot)$ is quadratically related to $R(f)$ and almost tightly characterizes the quantum communication complexity.

¹ In Theorems 5 and 7, $\tilde{O}(M(f)) = O(M(f) \log^2 n \log \log n)$ for any complexity measure M .

► **Theorem 6.** For any non-trivial total permutation-invariant function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1\}$ in Definition 8, we have

$$D(f) = O(\log^2 \text{rank}(f)).$$

► **Theorem 7.** For any permutation-invariant function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1, *\}$ in Definition 8, we have

$$R(f) = \tilde{O}(\log^2 \widetilde{\text{rank}}(f)) \quad \text{and} \quad Q(f) = \tilde{O}(\log \widetilde{\text{rank}}(f)).$$

1.2 Proof Techniques

In this section, we give a high-level technical overview of our main results.

1.2.1 Lower Bound

We outline our approaches to obtain the lower bound on the quantum communication complexity, rank and approximate rank of permutation-invariant functions below:

1. **Quantum communication complexity and approximate rank:** In Theorem 5, to prove $Q^*(f) = \Omega(m(f))$ for any permutation-invariant function f , we use the following two-step reduction (see Lemma 15 and Theorem 13): First, we reduce the lower bound of the quantum communication complexity of the Exact Set-Inclusion Problem (ESetInc, Definition 10) to Paturi's approximate degree of symmetric functions [34] by the pattern matrix method [37], a well-known method for lower bound analysis in quantum communication complexity. Second, we reduce the lower bound of any permutation-invariant function to the lower bound of ESetInc. In Theorem 7, we use a similar method to prove the lower bound of approximate rank: $\log \widetilde{\text{rank}}(f) = \Omega(m(f))$.
2. **Rank:** In Theorem 6, we reduce the lower bound of the rank of total permutation-invariant functions to the lower bound of the rank of some representative function instances, such as the set disjointness problem and the equality problem (see Lemma 24).

1.2.2 Upper Bound

We use the following methods to show the upper bounds on the communication complexity of permutation-invariant functions in the randomized, quantum, and deterministic models.

1. **Randomized and quantum models:** In Theorem 5, to prove $R(f) \leq \tilde{O}(m(f)^2)$ for any permutation-invariant function f , we first propose a randomized protocol to solve the Set-Inclusion problem (SetInc, Definition 10) using a well-suited sampling method according to the parameters of SetInc (see Lemma 20). Afterward, we use this protocol as a subroutine to solve any permutation-invariant function based on binary search (see Theorem 14). Furthermore, to prove $Q(f) \leq \tilde{O}(m(f))$, we use the quantum amplitude amplification technique [13, 27] to speed up the above randomized protocol to solve SetInc (see Lemma 21).
2. **Deterministic model:** In Theorem 6, to give an upper bound of deterministic communication complexity of total permutation-invariant functions, we propose a deterministic protocol as follows (see Lemma 25): Alice and Bob first share their Hamming weight of inputs, and decide who sends the input to the other party according to the definition of function and the Hamming weight of inputs. The party that has all the information of inputs will output the answer. Combining Lemmas 24 (described in Section 1.2.1) and 25, Theorem 6 can be proved.

1.3 Related Work

The need for structure in quantum speedups has been studied in the query model extensively. Beals, Buhrman, Cleve, Mosca and de Wolf [8] demonstrated that there exists at most a polynomial quantum speedup for total Boolean functions in the query model. Moreover, Aaronson and Ambainis [2] established that even partial symmetric functions do not allow super-polynomial quantum speedups. Chailloux [16] further improved this result to a broader class of symmetric functions. Ben-David, Childs, Gilyén, Kretschmer, Podder and Wang [11] later analyzed the quantum advantage for functions that are symmetric under different group actions systematically. Ben-David [10] established a quantum and classical polynomial equivalence for a certain set of functions satisfying a specific symmetric promise. Aaronson and Ben-David [3] proved that if domain D satisfies $D = \text{poly}(n)$, there are at most polynomial quantum speedups for computing an n -bit partial Boolean function.

In terms of communication complexity, there are a few results that imply the polynomial equivalence between quantum and classical communication complexity for several instances of permutation-invariant functions. Examples include AND-symmetric functions [36], Hamming distance problem [28, 17], XOR-symmetric functions [45]. While the above results characterized quantum advantage for a certain class of permutation-invariant Boolean functions, our work provides a systemic analysis of all permutation-invariant Boolean functions.

The study of the Log-rank Conjecture and the Log-Approximate-Rank Conjecture has a rich history. Here, we only survey the results about the Log-rank Conjecture and the Log-Approximate-Rank Conjecture about permutation-invariant Boolean functions. Buhuman and de Wolf [15] verified the correctness of the Log-rank Conjecture for AND-symmetric functions. Combining the results of Razborov [36], Sherstov [37] and Suruga [42], it is implied that the Log-Approximate-Rank Conjecture holds for AND-symmetric functions both in the randomized and quantum settings. Moreover, the result of Zhang and Shi [45] implies the Log-rank Conjecture and the (quantum) Log-Approximate-Rank Conjecture hold for XOR-symmetric functions.

In previous work, Ghazi, Kamath and Sudan [23] introduced a complexity measure, which is polynomially equivalent to the randomized communication complexity of permutation-invariant functions defined in Definition 8. This paper is inspired by their work.

1.4 Organization

The remaining part of the paper is organized as follows. In Section 2, we state some notations and definitions used in this paper. In Section 3, we study the quantum and classical communication complexities of permutation-invariant functions. In Section 4, we show the Log-rank Conjecture holds for non-trivial total permutation-invariant functions. In Section 5, we study the Log-approximate Conjecture of permutation-invariant functions both in quantum and classical setting. Finally, a conclusion is made in Section 6. The appendices contain a section on extended preliminaries and omitted proofs.

2 Preliminaries

We introduce the notations and definitions used in this paper.

A **multiset** is a set with possibly repeating elements. We use $\{\{\cdot\}\}$ to denote multiset and $\{\cdot\}$ to denote standard set. Let S be a multiset, $S \setminus \{a\}$ removes one occurrence of a from S if there is any.

2.1 Boolean Functions

A **partial function** is a function defined only on a subset of its domain \mathcal{X} . Formally, given a partial Boolean function $f: \mathcal{X} \rightarrow \{-1, 1, *\}$, $f(x)$ is **undefined** for $x \in \mathcal{X}$ if $f(x) = *$. A **total function** is a function that is defined on the entire domain. We say $f: \mathcal{X} \rightarrow \{-1, 1, *\}$ is a **subfunction** of $g: \mathcal{X} \rightarrow \{-1, 1, *\}$ if $f(x) = g(x)$ or $f(x) = *$ for any $x \in \mathcal{X}$.

A **Boolean predicate** is a partial function that has domain $\mathcal{X} = \{0, 1, \dots, n\}$ for any $n \in \mathbb{N}$.

An **incomplete Boolean matrix** is a matrix with entries in $\{-1, 1, *\}$, where undefined entries are filled with $*$.

A **submatrix** is a matrix that is obtained by extracting certain rows and/or columns from a given matrix.

A **half-integer** is a number of the form $n + 1/2$, where $n \in \mathbb{Z}$.

We introduce some Boolean operators as follows. For every $n \in \mathbb{N}$ and $x, y \in \{0, 1\}^n$:

- $\bar{x} := \{\bar{x}_0, \dots, \bar{x}_{n-1}\} = \{1 - x_0, \dots, 1 - x_{n-1}\}$;
- $x \wedge y := \{x_0 \wedge y_0, \dots, x_{n-1} \wedge y_{n-1}\}$; and
- $x \oplus y := \{x_0 \oplus y_0, \dots, x_{n-1} \oplus y_{n-1}\}$.

2.2 Communication Complexity Model

In the two-party communication model, Alice is given input x , and Bob is given input y . Then they aim to compute $f(x, y)$ for some function $f: \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1, *\}$ by communication protocols. The **deterministic communication complexity** $D(f)$ is defined as the cost of the deterministic protocol with the smallest communication cost, which computes f correctly on any input. The **randomized communication complexity** $R_\epsilon(f)$ is defined as the cost of the randomized protocol with the smallest communication cost, which has access to public randomness and computes f correctly on any input with probability at least $1 - \epsilon$. Similarly, the **quantum communication complexity** $Q(f)$ is defined as the cost of the quantum protocol with the smallest cost, which is not allowed to share prior entanglement, has access to public randomness and computes f correctly on any input with probability at least $1 - \epsilon$. If the quantum protocol is allowed with prior entanglement initially, then the corresponding quantum communication complexity is denoted $Q^*(f)$. If a protocol succeeds with probability at least $1 - \epsilon$ on any input for some constant $\epsilon < 1/2$, we say the protocol is with **bounded error**. If $\epsilon = 1/3$, we abbreviate $R_\epsilon(f), Q_\epsilon(f), Q_\epsilon^*(f)$ as $R(f), Q(f), Q^*(f)$.

2.3 Permutation-Invariant Functions

In the two-party communication model, the function value of a permutation-invariant function is invariant if we perform the same permutation to the inputs of Alice and Bob. Specifically, the formal definition is as follows.

► **Definition 8** (Permutation-invariant functions [23]). *A (total or partial) function $f: \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1, *\}$ is permutation-invariant if for all $x, y \in \{0, 1\}^n$, and every bijection $\pi: \{0, \dots, n-1\} \rightarrow \{0, \dots, n-1\}$, $f(x^\pi, y^\pi) = f(x, y)$, where x^π satisfies that $x^\pi_{(i)} = x_{\pi(i)}$ for any $i \in \{0, \dots, n-1\}$.*

Note that any permutation-invariant function f in Definition 8 depends only on $|x|, |y|$ and $|x \wedge y|$. Here $|\cdot|$ is the Hamming weight for the binary string, i.e., the number of 1's in the string. Thus, for any $a, b \in [n]$, there exists a function $f_{a,b}: \{\max\{0, a+b-n\}, \dots, \min\{a, b\}\} \rightarrow \{-1, 1\}$ such that

$$f_{a,b}(|x \wedge y|) = f(x, y), \tag{1}$$

for any $x, y \in \{0, 1\}^n$ satisfying $|x| = a, |y| = b$. If there exist $a, b \in [n]$ such that $f_{a,b}$ is not a constant function, we say f is **non-trivial**.

The following definition of jumps partitions the domain of $f_{a,b}$ into different intervals according to the transition of function values.

► **Definition 9** (Jump in $f_{a,b}$ [23]). (c, g) is a jump in $f_{a,b}$ if

1. $f_{a,b}(c-g) \neq f_{a,b}(c+g)$;
2. $f_{a,b}(c-g), f_{a,b}(c+g) \in \{-1, 1\}$;
3. $f_{a,b}(r)$ is undefined for $c-g < r < c+g$.

Moreover, we define $\mathcal{J}(f_{a,b})$ to be the set of all jumps in $f_{a,b}$:

$$\mathcal{J}(f_{a,b}) := \left\{ (c, g) : \begin{array}{l} f_{a,b}(c-g), f_{a,b}(c+g) \in \{0, 1\} \\ f_{a,b}(c-g) \neq f_{a,b}(c+g) \\ \forall i \in (c-g, c+g), f_{a,b}(i) = * \end{array} \right\}.$$

The following definition gives an important instance of permutation-invariant functions.

► **Definition 10** (Set-Inclusion Problem). We define the Set-Inclusion Problem $\text{SetInc}_{a,b,c,g}^n$ as the following partial function:

$$\text{SetInc}_{a,b,c,g}^n(x, y) := \begin{cases} -1 & \text{if } |x| = a, |y| = b \text{ and } |x \wedge y| \leq c-g, \\ 1 & \text{if } |x| = a, |y| = b \text{ and } |x \wedge y| \geq c+g, \\ * & \text{otherwise.} \end{cases}$$

Additionally, we define the Exact Set-Inclusion Problem $\text{ESetInc}_{a,b,c,g}^n$ as follows.

$$\text{ESetInc}_{a,b,c,g}^n(x, y) := \begin{cases} -1 & \text{if } |x| = a, |y| = b \text{ and } |x \wedge y| = c-g, \\ 1 & \text{if } |x| = a, |y| = b \text{ and } |x \wedge y| = c+g, \\ * & \text{otherwise.} \end{cases}$$

2.4 Rank and Approximate Rank

If F is a real matrix, let $\text{rank}(F)$ be the **rank** of F . Then we define the approximate rank for any incomplete matrix as follows.

► **Definition 11** (Approximate rank). For an incomplete matrix $F \in \{-1, 1, *\}^{m \times n}$ and $0 \leq \epsilon < 1$, we say a real matrix A approximates F with error ϵ if:

- (1) $|A_{i,j} - F_{i,j}| \leq \epsilon$ for any $i \in [m], j \in [n]$ such that $F_{i,j} \neq *$;
- (2) $|A_{i,j}| \leq 1$ for all $i \in [m], j \in [n]$.

Let \mathcal{F}_ϵ be the set of all the real matrices that approximate F with error ϵ . The approximate rank of F with error ϵ , denoted by $\widetilde{\text{rank}}_\epsilon(F)$, is the least rank among all real matrices in \mathcal{F}_ϵ . If $\epsilon = 2/3$, we abbreviate $\widetilde{\text{rank}}_\epsilon(F)$ as $\widetilde{\text{rank}}(F)$.

Let f be a Boolean function, $\text{rank}(f) := \text{rank}(M_f)$ and $\widetilde{\text{rank}}(f) := \widetilde{\text{rank}}(M_f)$, where M_f is the input matrix of f .

3 Polynomial Equivalence on Communication Complexity of Permutation-Invariant Functions

To show the polynomial equivalence between quantum and classical communication complexity of permutation-invariant functions as stated in Theorem 5, we prove the following two theorems (proved in Sections 3.1 and 3.2, respectively) for the quantum and randomized communication complexities of permutation-invariant functions using the measure in Definition 12.

► **Definition 12** (Measure $m(f)$). Fix $n \in \mathbb{Z}$. Let $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$ be a permutation-invariant function. We define the measure $m(f)$ of f as follows:

$$m(f) := \max_{\substack{a, b \in [n] \\ (c, g) \in \mathcal{J}(f_{a,b}) \\ n_1 := \min\{[a-c, c, b-c, n-a-b+c]\} \\ n_2 := \min(\{[a-c, c, b-c, n-a-b+c]\} \setminus \{n_1\})}} \frac{\sqrt{n_1 n_2}}{g} .$$

Note that this definition is motivated by Lemma 15.

The measure $m(\cdot)$ is inspired by the complexity measure introduced in [23], which was used to capture the randomized communication complexity of permutation-invariant functions.

► **Theorem 13** (Lower Bound). Fix $n \in \mathbb{Z}$. Let $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$ be a permutation-invariant function. We have

$$Q^*(f) = \Omega(m(f)) .$$

► **Theorem 14** (Upper Bound). Fix $n \in \mathbb{N}$. Given a permutation-invariant function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$ and the corresponding measure $m(f)$ defined in Definition 12, we have

- $R(f) = O(m(f)^2 \log^2 n \log \log n)$, and
- $Q(f) = O(m(f) \log^2 n \log \log n)$.

3.1 Quantum Communication Complexity Lower Bound

In this section, our goal is to obtain a lower bound on the quantum communication complexity for permutation-invariant functions (Theorem 13). Towards this end, we show that every permutation-invariant function f can be reduced to ESetInc (defined in Definition 10) and exhibit a lower bound for ESetInc (Lemma 15). Additionally, Lemma 15 implies if $|x| = a$, $|y| = b$, then the cost to distinguish $|x \wedge y| = c - g$ from $|x \wedge y| = c + g$ is related to the smallest and the second smallest number in $[a - c, c, b - c, n - a - b + c]$.

► **Lemma 15.** Fix $n, a, b \in \mathbb{Z}$. Consider c and g such that $c + g, c - g \in \mathbb{Z}$. Let $n_1 := \min\{[a - c, c, b - c, n - a - b + c]\}$ and $n_2 := \min(\{[a - c, c, b - c, n - a - b + c]\} \setminus \{n_1\})$. We have

$$Q^*(\text{ESetInc}_{a,b,c,g}^n) = \Omega\left(\frac{\sqrt{n_1 n_2}}{g}\right) .$$

Proof of Theorem 13. By the definitions of $f_{a,b}$ and jump of $f_{a,b}$, any quantum protocol computing f can also compute $\text{ESetInc}_{a,b,c,g}^n$ for any a, b and any jump $(c, g) \in \mathcal{J}(f_{a,b})$. Therefore, given a jump (c, g) for $f_{a,b}$, the cost of computing $\text{ESetInc}_{a,b,c,g}^n$ lower bounds the cost of computing f . By Lemma 15, we have $Q^*(f) \geq \frac{\sqrt{n_1 n_2}}{g}$ for any jump (c, g) in $f_{a,b}$, where n_1, n_2 are the smallest and the second smallest number in $\{[a - c, c, b - c, n - a - b + c]\}$. We conclude that $Q^*(f) = \Omega(m(f))$ as desired. ◀

Now we remain to show Lemma 15. We note that the following two lemmas imply Lemma 15 directly, where Lemma 16 reduces the instance such that the parameter only relies on n_1, n_2, g and Lemma 17 gives the final lower bound.

► **Lemma 16.** *Fix $n, a, b \in \mathbb{Z}$. Consider c and g such that $c + g, c - g \in \mathbb{Z}$. Let $n_1 := \min\{a - c, c, b - c, n - a - b + c\}$ and $n_2 := \min(\{[a - c, c, b - c, n - a - b + c]\} \setminus \{n_1\})$. We have*

$$Q^*(\text{ESetInc}_{a,b,c,g}^n) \geq Q^*(\text{ESetInc}_{n_1+n_2, n_1+n_2, n_1, g}^{n_1+3n_2}).$$

► **Lemma 17.** *For $n_1, n_2 \in \mathbb{N}$ such that $n_1 \leq n_2$, we have*

$$Q^*(\text{ESetInc}_{n_1+n_2, n_1+n_2, n_1, g}^{n_1+3n_2}) = \Omega\left(\frac{\sqrt{n_1 n_2}}{g}\right).$$

We use the following two results on ESetInc to show Lemmas 16 and 17 (See full version [26] for the detailed proofs). Specifically, Lemma 18 is a variant of Lemma 4.1 in [18] and shows some reduction methods to the instances of the Exact Set-Inclusion Problem. Lemma 19 is a generalization of Theorem 5 in [9] proved by pattern matrix method and shows the lower bound of a special instance of the Exact Set-Inclusion Problem.

► **Lemma 18.** *Fix $n, a, b \in \mathbb{Z}$. Consider c and g such that $c + g, c - g \in \mathbb{Z}$. The following relations hold.*

1. $Q^*(\text{ESetInc}_{a,b,c,g}^n) \leq Q^*(\text{ESetInc}_{a+\ell_1+\ell_3, b+\ell_2+\ell_3, c+\ell_3, g}^{n+\ell})$ for integers $\ell_1, \ell_2, \ell_3 \geq 0$ such that $\ell_1 + \ell_2 + \ell_3 \leq \ell$;
2. $Q^*(\text{ESetInc}_{a,b,c,g}^n) = Q^*(\text{ESetInc}_{a, n-b, a-c, g}^n) = Q^*(\text{ESetInc}_{n-a, b, b-c, g}^n)$;
3. $Q^*(\text{ESetInc}_{a,b,c,g}^n) \leq Q^*(\text{ESetInc}_{ka, kb, kc, kg}^{kn})$, where $k \geq 1$ is an integer.

► **Lemma 19.** *For every $k \in \mathbb{Z}$, if l is a half-integer and $l \leq k/2$, then $Q^*(\text{ESetInc}_{2k, k, l, 1/2}^{4k}) = \Omega(\sqrt{kl})$.*

Proof of Lemma 16. Using the second item of Lemma 18, we assume $n_1 = c$ without loss of generality. Furthermore, we assume $n_2 = a - c$. Let $n_3 := b - c, n_4 := n - a - b + c$. Then $n_3, n_4 \geq n_2 \geq n_1$ and $n = n_1 + n_2 + n_3 + n_4$. By Lemma 18, we have

$$\begin{aligned} Q^*(\text{ESetInc}_{n_1+n_2, n_1+n_2, n_1, g}^{n_1+3n_2}) &= Q^*(\text{ESetInc}_{n_1+n_2, n_1+n_2, n_1, g}^{n_1+n_2+n_2+n_2}) \\ &\leq Q^*(\text{ESetInc}_{n_1+n_2, n_1+n_3, n_1, g}^{n_1+n_2+n_3+n_4}) \\ &= Q^*(\text{ESetInc}_{a,b,c,g}^n). \end{aligned}$$

If $n_2 = b - c$ or $n - a - b + c$, the argument is similar. ◀

Proof of Lemma 17. Let $m_1 = \lfloor \frac{n_1}{2g} + \frac{1}{2} \rfloor - \frac{1}{2}$, i.e., m_1 is the largest half-integer no more than $\frac{n_1}{2g}$. Similarly, let $m_2 = \lfloor \frac{n_2}{2g} + \frac{1}{2} \rfloor - \frac{1}{2}$. By Lemma 18, we have

$$Q^*(\text{ESetInc}_{n_1+n_2, n_1+n_2, n_1, g}^{n_1+3n_2}) \geq Q^*(\text{ESetInc}_{m_1+m_2, m_1+m_2, m_1, 1/2}^{m_1+3m_2}).$$

Then we discuss the following three cases:

■ *Case 1:* $m_1 = m_2 = 1/2$. We have

$$Q^*(\text{ESetInc}_{m_1+m_2, m_1+m_2, m_1, 1/2}^{m_1+3m_2}) = \Omega(1) = \Omega(\sqrt{m_1 m_2}).$$

39:10 Communication Complexity of Permutation-Invariant Functions

- *Case 2:* $m_2 \geq 3/2$ and $m_1 = 1/2$. Let $m'_2 := \lfloor \frac{m_1+m_2}{2} \rfloor$, $l_1 := m_1 + m_2 - 2m'_2$, $l_2 := m_1 + m_2 - m'_2$, $l := m_1 + 3m_2 - 4m'_2$. Then,

$$l - (l_1 + l_2) = m_2 - m_1 - m'_2 \geq \frac{m_2 + m_1}{2} - m'_2 \geq 0.$$

By Lemmas 18 and 19, we have

$$\begin{aligned} Q^* \left(\text{ESetInc}_{m_1+m_2, m_1+m_2, m_1, 1/2}^{m_1+3m_2} \right) &= Q^* \left(\text{ESetInc}_{2m'_2+l_1, m'_2+l_2, m_1, 1/2}^{4m'_2+l} \right) \\ &\geq Q^* \left(\text{ESetInc}_{2m'_2, m'_2, m_1, 1/2}^{4m'_2} \right) \\ &= \Omega \left(\sqrt{m_1 m'_2} \right) \\ &= \Omega \left(\sqrt{m_1 m_2} \right). \end{aligned}$$

- *Case 3:* $m_1 \geq 3/2$. Let $m := \lfloor \frac{m_1}{6} + \frac{m_2}{2} \rfloor$, $k := \lfloor \frac{m_1}{3} + \frac{1}{2} \rfloor - \frac{1}{2}$, $l_3 := m_1 - k$, $l_1 := (m_1 + m_2 - 2m) - l_3$, $l_2 := (m_1 + m_2 - m) - l_3$, $l := m_1 + 3m_2 - 4m$. Since k is the largest half-integer smaller than $\frac{m_1}{3}$, we have $k \leq \frac{1}{2} \cdot \lfloor \frac{2m_1}{3} \rfloor$. Since $m_1 \leq m_2$, we have

$$k \leq \frac{1}{2} \cdot \left\lfloor \frac{2m_1}{3} \right\rfloor \leq \frac{1}{2} \cdot \left\lfloor \frac{m_1}{6} + \frac{m_2}{2} \right\rfloor \leq \frac{m}{2}, \quad (2)$$

and

$$l - (l_1 + l_2 + l_3) = m_2 - k - m \geq m_2 - \frac{m_1}{3} - \left(\frac{m_1}{6} + \frac{m_2}{2} \right) \geq 0. \quad (3)$$

Then we have

$$\begin{aligned} &Q^* \left(\text{ESetInc}_{m_1+m_2, m_1+m_2, m_1, 1/2}^{m_1+3m_2} \right) \\ &= Q^* \left(\text{ESetInc}_{2m+l_1+l_3, m+l_2+l_3, k+l_3, 1/2}^{4m+l} \right) \\ &\geq Q^* \left(\text{ESetInc}_{2m, m, k, 1/2}^{4m} \right) \quad (\text{by Lemma 18 and Equation (3)}) \\ &= \Omega \left(\sqrt{mk} \right) \quad (\text{by Lemma 19 and Equation (2)}) \\ &= \Omega \left(\sqrt{m_1 m_2} \right). \end{aligned}$$

We conclude that

$$Q^* \left(\text{ESetInc}_{n_1+n_2, n_1+n_2, n_1, g}^{n_1+3n_2} \right) \geq Q^* \left(\text{ESetInc}_{m_1+m_2, m_1+m_2, m_1, 1/2}^{m_1+3m_2} \right) = \Omega \left(\frac{\sqrt{n_1 n_2}}{g} \right). \quad \blacktriangleleft$$

3.2 Randomized and Quantum Communication Complexity Upper Bound

We show upper bounds on the randomized and quantum communication complexities for permutation invariant functions (Theorem 14). Similar to Section 3.1, we do so by giving upper bounds for a specific problem, SetInc (see Definition 10), and reducing permutation-invariant functions to SetInc .

The following two lemmas capture the randomized and quantum communication complexity for SetInc , respectively.

► **Lemma 20** (Classical Upper Bound). *Fix $n, a, b \in \mathbb{Z}$. Consider c, g such that $c + g, c - g \in \mathbb{Z}$. Let $n_1 := \min\{[a - c, c, b - c, n - a - b + c]\}$ and $n_2 := \min(\{[a - c, c, b - c, n - a - b + c]\} \setminus \{n_1\})$. For any input $x, y \in \{0, 1\}^n$ of $\text{SetInc}_{a,b,c,g}^n$, there exists a randomized communication protocol that computes $\text{SetInc}_{a,b,c,g}^n(x, y)$ using $O\left(\frac{n_1 n_2}{g^2} \log n \log \log n\right)$ bits of communication with success probability at least $1 - 1/(6 \log n)$.*

► **Lemma 21** (Quantum Upper Bound). *Fix $n, a, b \in \mathbb{Z}$. Consider c, g such that $c + g, c - g \in \mathbb{Z}$. Let $n_1 := \min\{[a - c, c, b - c, n - a - b + c]\}$ and $n_2 := \min(\{[a - c, c, b - c, n - a - b + c]\} \setminus \{n_1\})$. For any input $x, y \in \{0, 1\}^n$ of $\text{SetInc}_{a,b,c,g}^n$, there exists a quantum communication protocol without prior entanglement that computes $\text{SetInc}_{a,b,c,g}^n(x, y)$ using $O\left(\frac{\sqrt{n_1 n_2}}{g} \log n \log \log n\right)$ qubits of communication with success probability at least $1 - 1/(6 \log n)$.*

We note that Lemma 21 is a quantum speedup version of Lemma 20 by quantum amplitude amplification. The proof of Lemma 20 is given at the end of this section, and the proof of Lemma 21 can be seen in the full version [26].

Now we explain how to derive Theorem 14 from the lemmas above.

Proof of Theorem 14. We first present a randomized protocol to compute f based on binary search:

1. Alice and Bob exchange $a := |x|, b := |y|$.
2. Alice and Bob both derive $f_{a,b}$ such that $f_{a,b}(|x \wedge y|) = f(x, y)$.
3. Let $\mathcal{J}(f_{a,b}) = \{(c_i, g_i)\}_{i \in [k]}$ for some $k \leq n$ be the set of jumps of $f_{a,b}$ as in Definition 9.
4. Alice and Bob use binary search to determine $i \in \{0, 1, \dots, k\}$ such that $|x \wedge y| \in I_i$, where I_i is defined in Equation (4).

We first discuss the communication complexity of the above protocol. The first step takes $O(\log n)$ bits of communication. The fourth step costs $O(m(f)^2 \log^2 n \log \log n)$ bits of communication: For each $i \in [k]$, Alice and Bob can determine whether $|x \wedge y| \leq c_i - g_i$ or $|x \wedge y| \geq c_i + g_i$ by $O(m(f)^2 \log n \log \log n)$ with a success probability of at least $1 - 1/(6 \log n)$ by Lemma 20. Since binary search takes at most $\lceil \log(k + 1) \rceil = O(\log n)$ rounds, the total communication cost is $O(m(f)^2 \log^2 n \log \log n)$.

Now we argue for the correctness of the protocol. Notice that the set of jumps $\mathcal{J}(f_{a,b})$ invokes $k + 1$ intervals:

$$\{I_0 := [0, c_1 - g_1], I_1 := [c_1 + g_1, c_2 - g_2], \dots, I_{k-1} := [c_{k-1} + g_{k-1}, c_k - g_k], I_k := [c_k + g_k, n]\} . \quad (4)$$

In particular, the followings hold:

- For every $j \in [0, k]$ and $z_1, z_2 \in I_j$ such that $f_{a,b}(z_1) \neq *$ and $f_{a,b}(z_2) \neq *$, we have $f_{a,b}(z_1) = f_{a,b}(z_2)$.
- If $z \notin I_j$ for any $j \in [0, k]$, then $f_{a,b}(z) = *$.

Therefore, Alice and Bob start from $i = \lfloor (k + 1)/2 \rfloor$ to determine whether $|x \wedge y| \leq c_i - g_i$ or $|x \wedge y| \geq c_i + g_i$ with success probability of at least $1 - 1/(6 \log n)$. Depending on the result, they repeat the same process similar to binary search to find the interval that $|x \wedge y|$ falls in. After at most $\lceil \log(k + 1) \rceil = O(\log n)$ repetitions, there is only one remaining interval and they can easily determine $f_{a,b}(|x \wedge y|)$. For $n \geq 2$, the failure probability of the above protocol is at most

$$1 - \left(1 - \frac{1}{6 \log n}\right)^{\lceil \log(k+1) \rceil} \leq \frac{\lceil \log(k+1) \rceil}{6 \log n} \leq \frac{\lceil \log(n+1) \rceil}{3 \log n^2} \leq \frac{1}{3}.$$

39:12 Communication Complexity of Permutation-Invariant Functions

For the quantum case, Alice and Bob use the same protocol above, but we invoke Lemma 21 to analyze the communication complexity. ◀

Proof of Lemma 20. We rely on the following two claims to prove the lemma.

► **Fact 22** ([2, Lemma 30]). *Fix $0 < \epsilon < \beta < 1$ such that $\beta + \epsilon \leq 1$. For a set S , suppose there is a subset S' of S such that $\frac{|S'|}{|S|} \leq \beta - \epsilon$ or $\frac{|S'|}{|S|} \geq \beta + \epsilon$. Suppose we can sample from S uniformly and ask whether the sample is in S' . Then we can decide whether $\frac{|S'|}{|S|} \leq \beta - \epsilon$ or $\frac{|S'|}{|S|} \geq \beta + \epsilon$ by $O(\beta/\epsilon^2)$ samples, with success probability at least $2/3$.*

► **Fact 23.** *Suppose $x, y \in \{0, 1\}^n$ are the inputs of Alice and Bob such that $|x| \neq |y|$. Alice and Bob can sample an element from $S := \{i : x_i \neq y_i\}$ uniformly using $O(\log n)$ bits of communication.*

We refer interesting readers to the full version [26] for the proof of Fact 23. Now we prove the lemma by casing on the values of n_1 and n_2 .

- *Case 1: $n_1 = c$ and $n_2 = a - c$.* According to Definition 10, we have either $\frac{|x \wedge y|}{|x|} \leq \frac{c-g}{a}$ or $\frac{|x \wedge y|}{|x|} \geq \frac{c+g}{a}$. Alice and Bob estimate $\frac{|x \wedge y|}{|x|}$ as follows: Alice chooses an index i such that $x_i = 1$ uniformly at random. Then Alice sends i to Bob, and Bob checks whether $y_i = 1$. By Fact 22, setting $\beta := \frac{c}{a}$, $\epsilon := \frac{g}{a}$, Alice and Bob can decide whether $\frac{|x \wedge y|}{|x|} \leq \frac{c-g}{a}$ or $\frac{|x \wedge y|}{|x|} \geq \frac{c+g}{a}$ with bounded error using $O\left(\frac{ac}{g^2}\right) = O\left(\frac{n_1 n_2}{g^2}\right)$ samples. Since $|x| = a$, using $O\left(\frac{n_1 n_2}{g^2} \log \log n\right)$ samples, they can decide whether $|x \wedge y| \leq c - g$ or $|x \wedge y| \geq c + g$ with success probability at least $1 - 1/(6 \log n)$ by error reduction. Thus, the communication complexity is $O\left(\frac{n_1 n_2}{g^2} \log n \log \log n\right)$.
- *Case 2: $n_1 = a - c$ and $n_2 = c$, or $n_1 = a - c$ and $n_2 = c$, or $n_1 = c$ and $n_2 = b - c$.* A similar argument as in Case 1 applies.
- *Case 3: $n_1 = c$ and $n_2 = n - a - b + c$.* Since $n_1 \leq n_2$, we have $a + b \leq n$. Then we consider the following two cases:

1. *Case 3.1: $a + b < n$.* Let $m := n_1 + n_2$, $p := \frac{|x \wedge y|}{|x \oplus y|}$. Since

$$\begin{aligned} |\bar{x} \oplus y| &= |x \wedge y| + |\bar{x} \wedge \bar{y}| \\ &= |x \wedge y| + (n - (a + b - |x \wedge y|)) , \\ &= n - (a + b) + 2|x \wedge y| \end{aligned}$$

we have

$$p = \frac{|x \wedge y|}{n - (a + b) + 2|x \wedge y|} = \frac{1}{\frac{n - (a + b)}{|x \wedge y|} + 2} .$$

Notice that p is an increasing function with respect to $|x \wedge y|$. As a result, if $|x \wedge y| \leq c - g$, then $p \leq \frac{c-g}{m-2g}$; if $|x \wedge y| \geq c + g$, then $p \geq \frac{c+g}{m+2g}$. Let

$$\beta := \frac{1}{2} \left(\frac{c+g}{m+2g} + \frac{c-g}{m-2g} \right) = \frac{cm - 2g^2}{m^2 - 4g^2} \text{ and } \epsilon := \frac{1}{2} \left(\frac{c+g}{m+2g} - \frac{c-g}{m-2g} \right) = \frac{gm}{m^2 - 4g^2} .$$

Since $c - g \geq 0$, we have $\beta \leq \frac{c+g}{m+2g} \leq \frac{2c}{m}$ and

$$\begin{aligned} \epsilon &= \frac{1}{2} \left(\frac{c+g}{m+2g} - \frac{c}{m} \right) + \frac{1}{2} \left(\frac{c}{m} - \frac{c-g}{m-2g} \right) \\ &= \frac{1}{2} \left(\frac{g(m-2c)}{m(m+2g)} + \frac{g(m-2c)}{m(m-2g)} \right) \\ &= O\left(\frac{g}{m}\right) . \end{aligned}$$

For any $x \in \{0, 1\}^n$, we let $S_x := \{i : x_i = 1\}$. By Fact 23, Alice and Bob can sample i from $S_{\bar{x} \oplus y}$ uniformly using $O(\log n)$ bits communication. Since $i \in S_{\bar{x} \oplus y}$, if $x_i = y_i = 1$, then $i \in S_{x \wedge y}$; if $x_i = y_i = 0$, then $i \notin S_{x \wedge y}$. By Fact 22, using $O\left(\frac{\beta}{\epsilon^2}\right) = O\left(\frac{mc}{g^2}\right) = O\left(\frac{n_1 n_2}{g^2}\right)$ samples, Alice and Bob can decide whether $p \leq \beta - \epsilon$ or $p \geq \beta + \epsilon$ with bounded error. Equivalently, Alice and Bob can distinguish $|x \wedge y| \leq c - g$ from $|x \wedge y| \geq c + g$ with bounded error. By error reduction, using $O\left(\frac{n_1 n_2}{g^2} \log \log n\right)$ samples, they can decide whether $|x \wedge y| \geq c - g$ or $|x \wedge y| \leq c + g$ with success probability at least $1 - 1/(6 \log n)$. Thus, the communication complexity is $O\left(\frac{n_1 n_2}{g^2} \log n \log \log n\right)$.

2. *Case 3.2: $a + b = n$.* Alice and Bob generate new inputs $x' = x0$ and $y' = y0$ (pad a zero after the original input). We know

$$\text{SetInc}_{a,b,c,g}^n(x, y) = \text{SetInc}_{a,b,c,g}^{n+1}(x', y') .$$

Since $a + b < n + 1$, Alice and Bob perform the protocol in Case 3.1 in the new inputs, and the complexity analysis is similar to Case 3.1.

- *Case 4: $n_1 = n - a - b + c$ and $n_2 = c$, or $n_1 = a - c$ and $n_2 = b - c$, or $n_1 = b - c$ and $n_2 = a - c$.* A similar argument as in Case 3 works. ◀

4 Log-Rank Conjecture for Permutation-Invariant Functions

Theorem 6 states the Log-rank Conjecture for permutation-invariant functions. We argue for the lower bound (Lemma 24) and the upper bound (Lemma 25) separately.

► **Lemma 24.** *Fix $n \in \mathbb{N}$. Let $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1\}$ be a non-trivial total permutation-invariant function. For every $a, b \in [n]$ such that $f_{a,b}$ is not a constant function, we have*

$$\log \text{rank}(f) = \Omega(\max\{\log n, \min\{a, b, n - a, n - b\}\}),$$

where $f_{a,b}$ is defined as Equation (1).

► **Lemma 25.** *Fix $n \in \mathbb{N}$. Let $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1\}$ be a non-trivial total permutation-invariant function.*

$$D(f) = O\left(\max_{a,b \in [n]: f_{a,b} \text{ is not constant}} \min\{a, b, n - a, n - b\} \cdot \log n\right),$$

where $f_{a,b}$ is defined as Equation (1).

We prove Lemma 24 below, and the proof of Lemma 25 can be found in the full version [26].

Proof of Lemma 24. We rely on the following two claims to prove the lemma. Two claims show the lower bound on the rank of some special functions respectively.

► **Fact 26** ([15], merging Corollary 6 with Lemma 4). *Fix $n \in \mathbb{N}$. Let $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{-1, 1\}$ be defined as $f(x, y) := D(|x \wedge y|)$ for some predicate $D : \{0, 1, \dots, n\} \rightarrow \{-1, 1\}$. If t is the smallest integer such that $D(t) \neq D(t - 1)$, then $\log \text{rank}(f) = \Omega(\log(\sum_{i=t}^n \binom{n}{i}))$.*

39:14 Communication Complexity of Permutation-Invariant Functions

► **Fact 27.** Fix $n \in \mathbb{N}$. Let $\mathcal{X}, \mathcal{Y} := \{x \in \{0, 1\}^n : |x| = k\}$, where $k \leq n/2$. Let $\text{DISJ}_n^k : \mathcal{X} \times \mathcal{Y} \rightarrow \{-1, 1\}$ and $\text{EQ}_n^k : \mathcal{X} \times \mathcal{Y} \rightarrow \{-1, 1\}$ be defined as

$$\text{DISJ}_n^k(x, y) := \begin{cases} -1 & \text{if } |x \wedge y| = 0 \\ 1 & \text{if } |x \wedge y| \neq 0 \end{cases} \quad \text{and} \quad \text{EQ}_n^k(x, y) := \begin{cases} -1 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases} .$$

Then $\text{rank}(\text{DISJ}_n^k) \geq \binom{n}{k} - 1$ and $\text{rank}(\text{EQ}_n^k) \geq \binom{n}{k} - 1$.

We refer interesting readers to the full version [26] for the proof of Fact 27. Now we prove the lemma by casing on the values of a and b .

We can assume $a \leq b \leq n/2$ without loss of generality because the cases where $a > n/2$ or $b > n/2$ can be obtained by flipping each bit of Alice or Bob's input. Thus, it suffices to prove $\log \text{rank}(f) = \Omega(\max\{\log n, a\})$.

We prove the following two claims that directly lead to our result:

1. If $a \leq b \leq n/2$ and $a = o(\log n)$, then $\log \text{rank}(f) = \Omega(\log n)$.
2. If $a \leq b \leq n/2$ and $a = \Omega(\log n)$, then $\log \text{rank}(f) = \Omega(a)$.

We first prove Item 1. Suppose $a \leq b \leq n/2$ and $a = o(\log n)$. Since $f_{a,b}$ is not a constant function, there exists $c \in [0, a)$ such that $f_{a,b}(c) \neq f_{a,b}(c+1)$. Without loss of generality, we assume $f_{a,b}(c) = -1$. Let $n' := n - (a + b - c - 2)$. Since $b \leq n/2$ and $c \leq a = o(\log n)$, $n' = n - (a + b - c - 2) = \Omega(n)$. Let \mathcal{X} and \mathcal{Y} be the set $\{x \in \{0, 1\}^{n'} : |x| = 1\}$. For any $x \in \mathcal{X}, y \in \mathcal{Y}$,

$$\text{DISJ}_{n'}^1(x, y) = f_{a,b}(|x \wedge y| + c) = f(x', y') ,$$

where

$$x' := x \underbrace{1 \cdots 1}_c \underbrace{1 \cdots 1}_{a-c-1} \underbrace{10 \cdots 0}_{b-c-1} \quad \text{and} \quad y' := y \underbrace{1 \cdots 1}_c \underbrace{10 \cdots 0}_{a-c-1} \underbrace{1 \cdots 1}_{b-c-1} .$$

Thus, $\text{DISJ}_{n'}^1$ is a submatrix of f . By Fact 27, we have

$$\log \text{rank}(f) \geq \log \text{rank}(\text{DISJ}_{n'}^1) \geq \log(n' - 1) = \Omega(\log n) .$$

Now we prove Item 2. Suppose $a, b \leq n/2$ and $\min\{a, b\} = \Omega(\log n)$, we consider the following three cases:

- *Case 1:* There exists $c \in [4a/7, 3a/5)$ such that $f_{a,b}(c) \neq f_{a,b}(c+1)$. Let $k = \lfloor a/2 \rfloor$ and $k' = \lceil a/2 \rceil$. Let $g : \{0, 1\}^k \times \{0, 1\}^{k'} \rightarrow \{-1, 1\}$ be such that $g(x, y) = f_{a,b}(|x' \wedge y'|)$ for every $x, y \in \{0, 1\}^k$, where

$$x' := x \bar{x} \underbrace{0 \cdots 0}_k \underbrace{1 \cdots 1}_{k'} \underbrace{10 \cdots 0}_{b-a} \underbrace{0 \cdots 0}_{n-b-2k} \quad \text{and} \quad y' := y \underbrace{0 \cdots 0}_k \bar{y} \underbrace{1 \cdots 1}_{k'} \underbrace{1 \cdots 1}_{b-a} \underbrace{0 \cdots 0}_{n-b-2k} .$$

Observe that $x', y' \in \{0, 1\}^n$ and $|x'| = a, |y'| = b$. Moreover, $g(x, y) = D(|x' \wedge y'|)$ for predicate $D : \{0, 1, \dots, k\} \rightarrow \{-1, 1\}$ such that $D(z) = f_{a,b}(z + k')$ for every $z \in [0, k]$. Thus, we have $D(c - k') \neq D(c - k' + 1)$. By Fact 26, we have

$$\log \text{rank}(g) = \Omega \left(\log \left(\sum_{i=c-k'+1}^k \binom{k}{i} \right) \right) .$$

Since $c - k' + 1 < 3a/5 - \lceil a/2 \rceil + 1 \leq a/10 \leq k/2$, we conclude $\log \text{rank}(g) = \Omega(k) = \Omega(a)$.

- Case 2: There exists $c \in [0, 4a/7)$ such that $f_{a,b}(c) \neq f_{a,b}(c+1)$ and $f_{a,b}$ is a constant function in the range $[c, 3a/5)$. Without loss of generality, we assume $f_{a,b}(c) = -1$. Let $l := \lfloor 3a/5 \rfloor$, $l' := \lfloor 2a/5 \rfloor$, $m := n - (c + b - a + 2l')$. Since $a \leq b \leq n/2$ and $c < 4a/7$, we have

$$m = n - (c + b - a + 2l') \geq 2a - 2l' - c = 2l - c \geq 2(l - c) .$$

Let \mathcal{X} and \mathcal{Y} be the set $\{x \in \{0, 1\}^m : |x| = l - c\}$. For every $x \in \mathcal{X}, y \in \mathcal{Y}$, we have

$$\text{DISJ}_m^{l-c}(x, y) = f_{a,b}(|x' \wedge y'|) = f(x', y') ,$$

where

$$x' := x \underbrace{1 \cdots 1}_c \underbrace{0 \cdots 0}_{b-a} \underbrace{0 \cdots 0}_{l'} \underbrace{0 \cdots 1}_{l'} \text{ and } y' := y \underbrace{1 \cdots 1}_c \underbrace{1 \cdots 1}_{b-a} \underbrace{1 \cdots 1}_{l'} \underbrace{1 \cdots 0}_{l'} \cdots 0 .$$

Thus, DISJ_m^{l-c} is a submatrix of f . By Fact 27, we have

$$\log \text{rank}(f) \geq \log \text{rank}(\text{DISJ}_m^{l-c}) = \Omega \left(\log \binom{m}{l-c} \right) = \Omega(l - c) = \Omega(a) .$$

- Case 3: There exists $c \in [3a/5, a)$ such that $f_{a,b}(c) \neq f_{a,b}(c+1)$ and $f_{a,b}$ is a constant function in the range $[0, c)$. Without loss of generality, we assume $f_{a,b}(c) = -1$. Since $a \leq b \leq \frac{n}{2}$, we have $n - b + c \geq a + c \geq 2c$. Let \mathcal{X} and \mathcal{Y} be the set $\{x \in \{0, 1\}^{n-b+c} : |x| = c\}$. For every $x \in \mathcal{X}, y \in \mathcal{Y}$, we have

$$\text{EQ}_{n-b+c}^c(x, y) = f_{a,b}(|x' \wedge y'|) = f(x', y') ,$$

where

$$x' := x \underbrace{0 \cdots 0}_{b-a} \underbrace{0 \cdots 0}_{a-c} \text{ and } y' := y \underbrace{1 \cdots 1}_{b-a} \underbrace{1 \cdots 0}_{a-c} .$$

Thus, EQ_{n-b+c}^c is a submatrix of f . By Fact 27, we have

$$\log \text{rank}(f) \geq \log \text{rank}(\text{EQ}_{n-b+c}^c) = \Omega \left(\log \binom{n-b+c}{c} \right) = \Omega(c) = \Omega(a) . \quad \blacktriangleleft$$

5 Log-Approximate-Rank Conjecture for Permutation-Invariant Functions

We discuss Theorem 7. In particular, we use the following two lemmas (proved in the full version [26]) to prove Theorem 7. Additionally, we note that Lemmas 28 and 29 are variants of Lemmas 18 and 19.

► **Lemma 28.** Let $n, a, b, c, g \in \mathbb{Z}^+$. The following relations hold:

- $\widetilde{\text{rank}}(\text{ESetInc}_{a,b,c,g}^n) \leq \widetilde{\text{rank}}(\text{ESetInc}_{a+\ell_1+\ell_3, b+\ell_2+\ell_3, c+\ell_3, g}^{n+\ell})$ for $\ell_1, \ell_2, \ell_3 \geq 0$ such that $\ell_1 + \ell_2 + \ell_3 \leq \ell$;
- $\widetilde{\text{rank}}(\text{ESetInc}_{a,b,c,g}^n) = \widetilde{\text{rank}}(\text{ESetInc}_{a, n-b, a-c, g}^n) = \widetilde{\text{rank}}(\text{ESetInc}_{n-a, b, b-c, g}^n)$; and
- $\widetilde{\text{rank}}(\text{ESetInc}_{a,b,c,g}^n) \leq \widetilde{\text{rank}}(\text{ESetInc}_{ka, kb, kc, kg}^{kn})$ for $k \geq 1$.

► **Lemma 29.** Fix $k \in \mathbb{Z}$. Let l be a half-integer such that $l \leq k/2$. We have

$$\log \left(\widetilde{\text{rank}} \left(\text{ESetInc}_{2k, k, l, 1/2}^{4k} \right) \right) = \Omega \left(\sqrt{kl} \right) .$$

39:16 Communication Complexity of Permutation-Invariant Functions

Proof sketch of Theorem 7. We use a similar argument as in the proof of Lemma 15. Namely, for every $a, b \in [n]$ and jump $(c, g) \in \mathcal{J}(f_{a,b})$, let $n_1 := \min\{[a-c, c, b-c, n-a-b+c]\}$ and $n_2 := \min(\{[a-c, c, b-c, n-a-b+c]\} \setminus \{n_1\})$. We have

$$\log \widetilde{\text{rank}}(\text{ESetInc}_{a,b,c,g}^n) = \Omega\left(\frac{\sqrt{n_1 n_2}}{g}\right).$$

Since $\text{ESetInc}_{a,b,c,g}^n$ is a subfunction of f , we have

$$\log \widetilde{\text{rank}}(f) = \Omega\left(\max_{\substack{a,b \in [n] \\ (c,g) \in \mathcal{J}(f_{a,b})}} \frac{\sqrt{n_1 n_2}}{g}\right) = \Omega(m(f)).$$

Combining Theorem 14 and the above equation, we have Theorem 7 as desired. ◀

6 Conclusion

This paper proves that the randomized communication complexity of permutation-invariant Boolean functions is at most quadratic of the quantum communication complexity (up to a logarithmic factor). Our results suggest that symmetries prevent exponential quantum speedups in communication complexity, extending the analogous research on query complexity. Furthermore, we prove that the Log-rank Conjecture and Log-approximate-rank Conjecture hold for non-trivial permutation-invariant Boolean functions (up to a logarithmic factor). There are some interesting problems to explore in the future.

- *Permutation invariance over higher alphabets.* In this paper, the permutation-invariant function is a binary function. The interesting question is to generalize our results to larger alphabets, i.e., to permutation-invariant functions of the form $f : X^n \times Y^n \rightarrow R$ where X, Y and R are not necessarily binary sets.
- *Generalized permutation invariance.* It is possible to generalize our results for a larger class of symmetric functions. One candidate might be a class of functions that have graph-symmetric properties. Suppose $\mathcal{G}_A, \mathcal{G}_B$ are two sets of n -vertices graphs, and G_n is a group that acts on the edges of an n -vertices graph and permutes them in a way that corresponds to relabeling the vertices of the underlying graph. A function $f : \mathcal{G}_A \times \mathcal{G}_B \rightarrow \{0, 1\}$ is graph-symmetric if $f(x, y) = f(x \circ \pi, y \circ \pi)$, where $x \in \mathcal{G}_A, y \in \mathcal{G}_B$, and $\pi \in G_n$.

References



- 1 Scott Aaronson. How much structure is needed for huge quantum speedups? *arXiv preprint*, 2022. arXiv:2209.06930.
- 2 Scott Aaronson and Andris Ambainis. The need for structure in quantum speedups. *Theory of Computing*, 10:133–166, 2014. doi:10.4086/toc.2014.v010a006.
- 3 Scott Aaronson and Shalev Ben-David. Sculpting quantum speedups. In *Proceedings of the 31st Conference on Computational Complexity*, volume 50, pages 26:1–26:28, 2016. doi:10.4230/LIPIcs.CCC.2016.26.
- 4 Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of ACM*, 51(4):595–605, 2004. doi:10.1145/1008731.1008735.

- 5 Anurag Anshu, Naresh Goud Boddu, and Dave Touchette. Quantum log-approximate-rank conjecture is also false. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 982–994, 2019. doi:10.1109/FOCS.2019.00063.
- 6 Anurag Anshu, Dave Touchette, Penghui Yao, and Nengkun Yu. Exponential separation of quantum communication and classical information. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 277–288. ACM, 2017. doi:10.1145/3055399.3055401.
- 7 Ziv Bar-Yossef, T. S. Jayram, and Iordanis Kerenidis. Exponential separation of quantum and classical one-way communication complexity. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 128–137. ACM, 2004. doi:10.1145/1007352.1007379.
- 8 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. doi:10.1145/502090.502097.
- 9 Aleksandrs Belovs, Arturo Castellanos, François Le Gall, Guillaume Malod, and Alexander A. Sherstov. Quantum communication complexity of distribution testing. *Quantum Information and Computation*, 21(15&16):1261–1273, 2021. doi:10.26421/QIC21.15-16-1.
- 10 Shalev Ben-David. The structure of promises in quantum speedups. In *Proceedings of the 11th Conference on the Theory of Quantum Computation, Communication and Cryptography*, volume 61, pages 7:1–7:14, 2016. doi:10.4230/LIPIcs.TQC.2016.7.
- 11 Shalev Ben-David, Andrew M. Childs, András Gilyén, William Kretschmer, Supartha Podder, and Daochen Wang. Symmetries, graph properties, and quantum speedups. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, pages 649–660, 2020. doi:10.1109/FOCS46700.2020.00066.
- 12 Eli Ben-Sasson, Shachar Lovett, and Noga Ron-Zewi. An additive combinatorics approach relating rank to communication complexity. *J. ACM*, 61(4), 2014. doi:10.1145/2629598.
- 13 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.
- 14 Sergey Bravyi, David Gosset, Robert König, and Marco Tomamichel. Quantum advantage with noisy shallow circuits in 3D. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science*, pages 995–999, 2019. doi:10.1109/FOCS.2019.00064.
- 15 Harry Buhrman and Ronald de Wolf. Communication complexity lower bounds by polynomials. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity*, pages 120–130, 2001. doi:10.1109/CCC.2001.933879.
- 16 André Chailloux. A note on the quantum query complexity of permutation symmetric functions. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 19:1–19:7, 2019. doi:10.4230/LIPIcs.ITCS.2019.19.
- 17 Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, pages 51–60, 2011. doi:10.1145/1993636.1993644.
- 18 Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of Gap-Hamming-Distance. *SIAM Journal on Computing*, 41(5):1299–1317, 2012. doi:10.1137/120861072.
- 19 Arkadev Chattopadhyay, Nikhil S. Mande, and Suhail Sherif. The log-approximate-rank conjecture is false. *J. ACM*, 67(4), June 2020. doi:10.1145/3396695.
- 20 Sitan Chen, Jordan Cotler, Hsin-Yuan Huang, and Jerry Li. Exponential separations between learning with and without quantum memory. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science*, pages 574–585, 2021. doi:10.1109/FOCS52979.2021.00063.
- 21 Dmitry Gavinsky, Julia Kempe, Iordanis Kerenidis, Ran Raz, and Ronald de Wolf. Exponential separations for one-way quantum communication complexity, with applications to cryptography. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 516–525. ACM, 2007. doi:10.1145/1250790.1250866.



- 22 Dmitry Gavinsky and Pavel Pudlák. Exponential separation of quantum and classical non-interactive multi-party communication complexity. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity*, pages 332–339. IEEE Computer Society, 2008. doi:10.1109/CCC.2008.27.
- 23 Badih Ghazi, Pritish Kamath, and Madhu Sudan. Communication complexity of permutation-invariant functions. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1902–1921, 2016. doi:10.1137/1.9781611974331.ch134.
- 24 Daniel Grier and Luke Schaeffer. Interactive shallow clifford circuits: quantum advantage against NC^1 and beyond. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 875–888, 2020. doi:10.1145/3357713.3384332.
- 25 L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th IEEE Annual Symposium on Theory of Computing*, pages 212–219, 1996. doi:10.1145/237814.237866.
- 26 Ziyi Guan, Yunqi Huang, Penghui Yao, and Zekun Ye. Quantum and classical communication complexity of permutation-invariant functions. *arXiv preprint*, 2024. arXiv:2401.00454.
- 27 Yassine Hamoudi and Frédéric Magniez. Quantum chebyshev’s inequality and applications. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming*, volume 132, pages 69:1–69:16, 2019. doi:10.4230/LIPIcs.ICALP.2019.69.
- 28 Wei Huang, Yaoyun Shi, Shengyu Zhang, and Yufan Zhu. The communication complexity of the Hamming distance problem. *Information Processing Letter*, 99(4):149–153, 2006. doi:10.1016/j.ipl.2006.01.014.
- 29 John Kallaugher. A quantum advantage for a natural streaming problem. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science*, pages 897–908, 2021. doi:10.1109/FOCS52979.2021.00091.
- 30 Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Log-rank and lifting for and-functions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 197–208, 2021. doi:10.1145/3406325.3450999.
- 31 L. Lovasz and M. Saks. Lattices, mobius functions and communications complexity. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 81–90, 1988. doi:10.1109/SFCS.1988.21924.
- 32 Shachar Lovett. Communication is bounded by root of rank. *J. ACM*, 63(1), 2016. doi:10.1145/2724704.
- 33 Ashley Montanaro. A new exponential separation between quantum and classical one-way communication complexity. *Quantum Information and Computation*, 11(7&8):574–591, 2011. doi:10.26421/QIC11.7-8-3.
- 34 Ramamohan Paturi. On the degree of polynomials that approximate symmetric boolean functions (preliminary version). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 468–474. ACM, 1992. doi:10.1145/129712.129758.
- 35 Ran Raz. Exponential separation of quantum and classical communication complexity. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 358–367. ACM, 1999. doi:10.1145/301250.301343.
- 36 Alexander A Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya: Mathematics*, 67(1):145, 2003. doi:10.1070/IM2003v067n01ABEH000422.
- 37 Alexander A. Sherstov. The pattern matrix method. *SIAM Journal of Computing*, 40(6):1969–2000, 2011. doi:10.1137/080733644.
- 38 Alexander A. Sherstov. The communication complexity of gap Hamming distance. *Theory of Computing*, 8(1):197–208, 2012. doi:10.4086/toc.2012.v008a008.
- 39 Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, November 1994. doi:10.1109/SFCS.1994.365700.

- 40 Daniel R. Simon. On the power of quantum computation. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 116–123, November 1994. doi:10.1109/SFCS.1994.365701.
- 41 Makrand Sinha and Ronald de Wolf. Exponential separation between quantum communication and logarithm of approximate rank. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 966–981, 2019. doi:10.1109/FOCS.2019.00062.
- 42 Daiki Suruga. Matching upper bounds on symmetric predicates in quantum communication complexity. *arXiv preprint*, 2006. arXiv:2301.00370.
- 43 Thomas Vidick. A concentration inequality for the overlap of a vector on a large set, with application to the communication complexity of the Gap-Hamming-Distance problem. *Chicago Journal of Theoretical Computer Science*, 2012, 2012. URL: <http://cjtcs.cs.uchicago.edu/articles/2012/1/contents.html>.
- 44 Takashi Yamakawa and Mark Zhandry. Verifiable quantum advantage without structure. In *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science*, pages 69–74, 2022. doi:10.1109/FOCS54457.2022.00014.
- 45 Zhiqiang Zhang and Yaoyun Shi. Communication complexities of symmetric XOR functions. *Quantum Information and Computation*, 9(3&4):255–263, 2009. doi:10.26421/QIC9.3-4-5.

A Faster Algorithm for Vertex Cover Parameterized by Solution Size

David G. Harris  

Department of Computer Science, University of Maryland, College Park, MD, USA

N. S. Narayanaswamy  

Department of Computer Science and Engineering, Indian Institute of Technology Madras, India

Abstract

We describe a new algorithm for vertex cover with runtime $O^*(1.25284^k)$, where k is the size of the desired solution and O^* hides polynomial factors in the input size. This improves over the previous runtime of $O^*(1.2738^k)$ due to Chen, Kanj, & Xia (2010) standing for more than a decade. The key to our algorithm is to use a measure which simultaneously tracks k as well as the optimal value λ of the vertex cover LP relaxation. This allows us to make use of prior algorithms for Maximum Independent Set in bounded-degree graphs and Above-Guarantee Vertex Cover.

The main step in the algorithm is to branch on high-degree vertices, while ensuring that both k and $\mu = k - \lambda$ are decreased at each step. There can be local obstructions in the graph that prevent μ from decreasing in this process; we develop a number of novel branching steps to handle these situations.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases Vertex cover, FPT, Graph algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.40

Related Version *Full Version:* <https://arxiv.org/abs/2205.08022>

1 Introduction

For an undirected graph $G = (V, E)$, a subset $S \subseteq V$ is called a *vertex cover* if every edge has at least one endpoint in S . It is closely related to an *independent set*, since if S is an inclusion-wise minimal vertex cover then $V - S$ is an inclusion-wise maximal independent set, and vice-versa. Finding the size of the smallest vertex cover is a classic NP-complete problem [6]. In particular, G has a vertex cover of size at most k if and only if it has an independent set of size at least $n - k$.

There is natural LP formulation for vertex cover which we denote by $\text{LPVC}(G)$:

$$\begin{aligned} & \text{minimize} && \sum_{v \in V} \theta(v) \\ & \text{subject to} && \theta(u) + \theta(v) \geq 1 \quad \text{for all edges } e = (u, v) \in E \\ & && \theta(v) \in [0, 1] \quad \text{for all vertices } v \in V \end{aligned}$$

The optimal solution to $\text{LPVC}(G)$, denoted $\lambda(G)$ or just λ if G is clear from context, is a lower bound on the size of a minimum vertex cover of G . We also define $\mu(G) = k - \lambda(G)$, i.e. the gap between solution size and LP lower bound. This linear program has remarkable properties which have been exploited for a variety of algorithms [12, 13, 9]. For instance, an optimum basic solution is half-integral and can be found efficiently by a network flow computation.

Fixed-parameter tractable (FPT) and exact algorithms explore a landscape of parameters to understand the complexity of different problems [14, 4, 3, 2, 5]. VERTEX COVER was one of the first studied FPT problems with respect to the parameter k , the optimal solution size. Building on a long line of research, this culminated in an algorithm with $O^*(1.2738^k)$

runtime [1]. (Throughout, we write $O^*(T)$ as shorthand for $T \cdot \text{poly}(n)$.) This record has been standing for more than a decade. Assuming the Exponential Time Hypothesis, no algorithm with runtime $O^*(2^{o(k)})$ is possible [2].

1.1 Outline of our results

The main result of this paper is an improved algorithm for vertex cover parameterized by k .

► **Theorem 1.** *There is an algorithm for VERTEXCOVER with runtime $O^*(1.25284^k)$. Moreover, depending on the maximum vertex degree of the graph G , better bounds can be shown:*

If $\text{maxdeg}(G) \leq 3$, we get runtime $O^(1.14416^k)$.*

If $\text{maxdeg}(G) \leq 4$, we get runtime $O^(1.21131^k)$.*

If $\text{maxdeg}(G) \leq 5$, we get runtime $O^(1.24394^k)$.*

If $\text{maxdeg}(G) \leq 6$, we get runtime $O^(1.25214^k)$.*

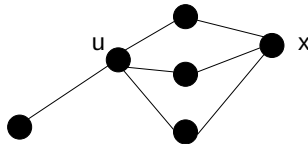
All of these algorithms use polynomial space and are deterministic.

The FPT algorithms for vertex cover, including our new algorithm, are built out of recursively branching on high-degree vertices or other structures, while tracking some “measure” of the graph. Our main new idea is to use a measure which is a piecewise-linear function of k and μ . To illustrate, consider branching on whether some degree- r vertex u is in the cover: we recurse on the subproblem $G_1 = G - u$ where k is reduced by one and on the subproblem $G_0 = G - u - N(u)$ where k is reduced by r . We must also show that μ is significantly reduced in the two subproblems.

Suppose that, wishfully speaking, $\frac{1}{2}$ remains the optimal solution to LPVC(G_0) and LPVC(G_1). This is what should happen in a “generic” graph. Then $\lambda(G_0) = |V(G_0)|/2 = (n - r - 1)/2$ and $\lambda(G_1) = |V(G_1)|/2 = (n - 1)/2$. Since $k_0 = k - r$ and $k_1 = k - 1$, this implies that μ is indeed significantly decreased:

$$\mu(G_0) = \mu(G) - (r - 1)/2, \quad \mu(G_1) = \mu(G) - 1/2$$

But suppose, on the other hand, that $\frac{1}{2}$ is not the optimal solution to LPVC(G_0) or LPVC(G_1). For a concrete example, suppose the neighborhood of x in G is a subset of that of u , so G_0 has an isolated vertex x and an optimal solution θ^* to LPVC(G_0) would set $\theta^*(x) = 0$. In this situation, we develop an alternate branching rule for G : rather than branching on u itself, we branch on the two subproblems where (i) both u, x are in the cover and (ii) x is not in the cover. See Figure 1 for an illustration.



■ **Figure 1** In the subgraph $G - N[u]$, the vertex x becomes isolated.

We emphasize that this is just one example of our branching analysis. We need to handle more general situations where $\frac{1}{2}$ is not the optimal solution to LPVC(G_0) or LPVC(G_1), which can require more complex branching rules. We emphasize that these branching rules are more powerful than simply using conventional branching rules along with the preprocessing rules; the latter may not be able to ensure a good reduction in μ .

In Section 6, we use these ideas for a relatively simple algorithm with runtime $O^*(1.2575^k)$, along with algorithms for bounded-degree graphs. This is already much better than [1].

In the full paper, we discuss how to improve the runtime by branching on a *targeted* vertex to create additional simplifications in the graph. Carrying out this secondary goal is much more complex. This would give the results stated in Theorem 1. Due to space limitations, we omit this analysis from the present version of the paper.

1.2 Related algorithms

Many different kinds of algorithms have been developed for the Vertex Cover problem. We cannot summarize this fully here; we describe some of the most relevant works for our own paper.

As we have mentioned, there is a long line of research into FPT algorithms parametrized by the solution size k . Currently, the fastest such algorithm has $O^*(1.2738^k)$ runtime [1]. In addition, [15] describes algorithms in graphs of maximum degree 3 and 4 with runtime respectively $O^*(1.1558^k)$ and $O^*(1.2403^k)$ respectively.

An important variant is ABOVE-GUARANTEE VERTEX COVER (AGVC), where the parameter is the difference between k and various lower bounds on vertex cover size [10, 11, 9, 7, 8]. Of these, the parameter $\mu(G) = k - \lambda(G)$ plays a particularly important role in this paper. We quote the following main result:

► **Theorem 2** ([9]). *Vertex cover can be solved in time $O^*(2.3146^\mu)$.*

Another natural choice is to measure runtime in terms of the graph size n . In this setting, the problem is more commonly referred to as Maximum Independent Set (MaxIS). Xiao and Nagamochi have developed a number of algorithms in this vein. These algorithms, and in particular their performance on bounded-degree graphs, will also play a crucial role in our analysis. We refer to the algorithm targeted for graphs of maximum degree Δ by the *MaxIS- Δ* algorithm.

► **Theorem 3.** *MaxIS-3 can be solved with runtime $O^*(1.083506^n)$ by [17].¹*

MaxIS-4 can be solved with runtime $O^(1.137595^n)$ by [16].*

MaxIS-5 can be solved with runtime $O^(1.17366^n)$ by [18].²*

MaxIS-6 can be solved with runtime $O^(1.18922^n)$ by [19].*

MaxIS-7 can be solved with runtime $O^(1.19698^n)$ by [19].*

MaxIS in graphs of arbitrary degree can be solved with runtime $O^(1.19951^n)$ by [19].*

1.3 Notation

We consider throughout a simple, undirected, unweighted graph $G = (V, E)$, and we write $n = |V|$. We write an ordered pair $\langle G, k \rangle$ for a graph G where we need to decide if there is a vertex cover of size at most k , and we say it is *feasible* if such a vertex cover exists. We say C is a *good cover* if it is a vertex cover of size at most k .

For a subset S of V , the subgraphs of G induced by S and $V \setminus S$ are denoted by $G[S]$ and $G - S$, respectively. We write $u \sim v$ if $(u, v) \in E$ and $u \not\sim v$ otherwise. For vertex sets X, Y , we write $X \sim Y$ if there exists $x \in X, y \in Y$ with $x \sim y$.

¹ This runtime is not claimed directly in [17], see [16] instead.

² This runtime is not claimed directly in [18], see [19].

For a vertex u , the neighborhood $N_G(u)$ is the set $\{u \in V(G) : u \sim v\}$ and the closed neighborhood $N_G[u]$ is the set $N_G(u) \cup \{u\}$. Extending this notation, for a vertex set $S \subseteq V(G)$, we write $N_G(S) = (\bigcup_{v \in S} N_G(v)) \setminus S$ and $N_G[S] = N_G(S) \cup S = \bigcup_{v \in S} N_G[v]$. For readability, we sometimes write $N(x, y)$ or $N[x, y]$ as shorthand for $N(\{x, y\})$ or $N[\{x, y\}]$.

The degree of vertex v , denoted by $\deg_G(v)$, is the size of $N_G(v)$. We call a vertex of degree i an i -vertex; for a vertex v , we refer to a neighbor u which has degree j as a j -neighbor of v . An isolated vertex is a 0-vertex. We say a vertex is *subquartic* if it has degree in $\{1, 2, 3, 4\}$ and *subcubic* if it has degree in $\{1, 2, 3\}$, and *subquadratic* if it has degree in $\{1, 2\}$. (We do not count isolated vertices).

We write $\max\deg(G)$ and $\min\deg(G)$ for the minimum and maximum vertex degrees in G , respectively. The graph is r -regular if $\max\deg(G) = \min\deg(G) = r$. We write V_i for the set of degree- i vertices and $n_i = |V_i|$.

We say that a pair of vertices u, v share neighbor y if $y \in N(u) \cap N(v)$. We denote by $\text{codeg}(u, v) = |N(u) \cap N(v)|$ the number of vertices shared by u, v .

An ℓ -cycle denotes a set of ℓ vertices x_1, \dots, x_ℓ with a path $x_1, x_2, \dots, x_\ell, x_1$.

2 Preliminaries

We review some basic facts about branching and preprocessing rules for vertex cover. Much of this material is standard, see e.g. [9]. We include proofs in Appendix A for completeness.

Our algorithm will heavily use the LP and its properties. This LP is closely related to properties of independent sets. For brevity, we refer to these as *indsets*, i.e. sets $X \subseteq V$ where no two vertices in X are adjacent. For an indset I in G , we define the *surplus* by $\text{surp}_G(I) = |N_G(I)| - |I|$. If G is clear from context, we write just $\text{surp}(I)$ or $N(I)$. We define $\text{minsurp}(G)$ to be the minimum value of $\text{surp}_G(I)$ over all *non-empty* indsets I in G , and a *min-set* to be any non-empty indset I achieving this minimum, i.e. $\text{surp}_G(I) = \text{minsurp}(G)$.

Since this comes up in a number of places, we define $\text{minsurp}^-(G) = \min\{0, \text{minsurp}(G)\}$.

► **Proposition 4.** $\lambda(G) = \frac{1}{2}(|V| + \text{minsurp}^-(G))$. Moreover, in a basic solution to LPVC(G), the set I of vertices with value zero forms an indset (possibly empty) with $\text{surp}_G(I) = \text{minsurp}^-(G)$.

We define a *critical-set* to be a non-empty indset I with $\text{surp}_G(J) \geq \text{surp}_G(I)$ for all non-empty subsets $J \subseteq I$. Clearly, any min-set or any singleton set is a critical-set.

► **Lemma 5.** For any critical-set I , there is a good cover C with either $I \subseteq C$ or $I \cap C = \emptyset$.
If $\text{surp}(I) \leq 0$, there is a good cover C with $I \cap C = \emptyset$.
If $\text{surp}(I) = 1$, there is a good cover C with either $N[I] \cap C = I$ or $N[I] \cap C = N(I)$.

► **Proposition 6.** For any non-empty indset I of G , there holds $\text{minsurp}(G) \leq \text{minsurp}^-(G - N[I]) + \text{surp}_G(I)$. The equality $\text{minsurp}(G) = \text{minsurp}^-(G - N[I]) + \text{surp}_G(I)$ holds if and only if I is contained in a min-set of G .

► **Proposition 7.** The value $\text{minsurp}(G)$ can be determined in polynomial time. Moreover, for any indset X (possibly $X = \emptyset$), we can efficiently find a min-set of G containing X (if any such exists).

2.1 Preprocessing and branching rules

Our algorithm uses three major preprocessing rules. For this, we need to define the following graph structure: a *funnel* is a vertex u with a neighbor x such that $G[N(u) \setminus \{x\}]$ forms a clique. Here, we call x the *out-neighbor* of u . For example, if a degree-3 vertex u has a

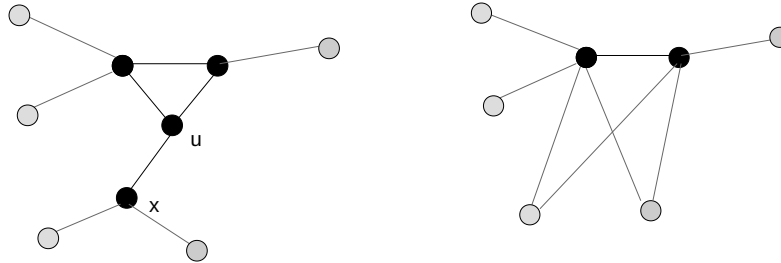
triangle with neighbors t_1, t_2 , and one other vertex x , then u is a funnel with out-neighbor x ; since this comes up so often, we refer to this as a *3-triangle*. A well-known result is that, for a funnel u with out-neighbor x , there exists a good cover C with either $u \notin C$ or $x \notin C$.

Preprocessing Rule 1 (P1): Given a critical-set I with $\text{surp}_G(I) \leq 0$, form graph G' with $k' = k - |N(I)|$ by deleting $N[I]$.

Preprocessing Rule 2 (P2): Given a critical-set I with $\text{surp}_G(I) = 1$, form graph G' with $k' = k - |I|$ by deleting $N[I]$ and adding a new vertex y adjacent to $N(N(I))$.

Preprocessing Rule 3 (P3): Given a funnel u with out-neighbor x , form graph G' with $k' = k - 1 - \text{codeg}(u, x)$ by removing vertices $N[u] \cap N[x]$ and adding edges between all vertices in $N(u) \setminus N[x]$ to all vertices in $N(x) \setminus N[u]$.

See Figure 2 for an illustration of rule (P3) applied to a funnel.

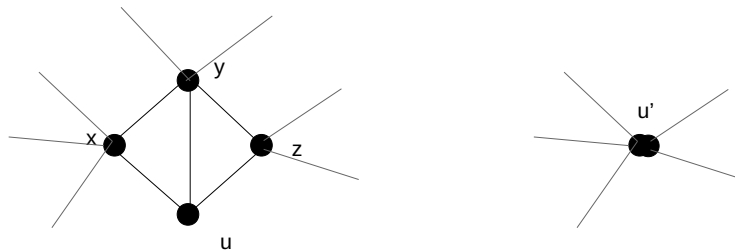


■ **Figure 2** A funnel (here, a 3-triangle) u before (left) and after (right) applying P3.

These rules can be applied to any eligible sets in any order. Note that if a vertex v has degree zero or one, then applying (P1) to $\{v\}$ removes v and its neighbor (if any). If a vertex v has degree two, then applying (P2) to $\{v\}$ removes v and contracts its neighbors x, y into a new vertex v' . When no further preprocessing rules can be applied, we say the graph G is *simplified*; the resulting graph has a number of nice properties, for instance, it has $\text{minsurp}(G) \geq 2$, and the solution $\frac{1}{2}$ is the unique optimal solution to $\text{LPVC}(G)$, and the minimum degree of any vertex is 3.

► **Proposition 8.** *After applying any preprocessing rule, we have $\mu(G') \leq \mu(G)$, and $\langle G, k \rangle$ is feasible if and only if $\langle G', k' \rangle$ is feasible.*

There is one situation where rule P3 is particularly powerful. Consider a 3-vertex u which has neighbors x, y, z where $x \sim y$ and $y \sim z$. We refer to this as a *kite*; see Figure 3.



■ **Figure 3** A kite u, x, y, z before (left) and after (right) applying P3.

► **Lemma 9.** *Suppose $\text{minsurp}(G) \geq 1$. Then applying (P3) to a kite in G yields a graph G' with $k' \leq k - 2$ and $\mu(G') \leq \mu(G) - 1/2$.*

2.2 Runtime and branching framework

Our algorithm follows the measure-and-conquer approach. Given an input graph G , it runs some preprocessing steps and then generates subproblems G_1, \dots, G_t such that G is feasible if and only if at least one G_i is feasible. It then runs recursively on each G_i . Such an algorithm has runtime $O^*(e^{\phi(G)})$ as long as the preprocessing steps have runtime $O^*(e^{\phi(G)})$ and it satisfies

$$\sum_{i=1}^t e^{\phi(G_i) - \phi(G)} \leq 1. \quad (1)$$

(Here and throughout $e = 2.718\dots$ is the base of the natural logarithm.) We refer to ϕ as the *measure* of the algorithm. For the most part, we will use measures of the form

$$\phi(G) = a\mu + bk \quad \text{for } a, b \geq 0; \quad (2)$$

We say a subproblem G' has *drop* $(\Delta\mu, \Delta k)$ if $k' \leq k - \Delta k$ and $\mu' \leq \mu - \Delta\mu$. We say a branching rule has *branch sequence* (or *branch-seq* for short) $B = [(\Delta\mu_1, \Delta k_1), \dots, (\Delta\mu_t, \Delta k_t)]$ if it generates subproblems G'_1, \dots, G'_t with the given drops. Given values of a, b , we define the *value* of B to be

$$\text{val}_{a,b}(B) = \sum_{i=1}^t e^{-a\Delta\mu_i - b\Delta k_i}. \quad (3)$$

We say branch-seq B' *dominates* B if $\text{val}_{a,b}(B') \leq \text{val}_{a,b}(B)$ for all $a, b \geq 0$, and G has a branch-seq B *available* if we can efficiently find a branching rule which has some branch-seq B' dominating B . In general, a “balanced” branch-seq dominates an “imbalanced” one; formally, if $\Delta k_1 \leq \Delta k_2$ and $\Delta\mu_1 \leq \Delta\mu_2$, then branch-seq $[(\Delta\mu_1, \Delta k_1), (\Delta\mu_2, \Delta k_2)]$ dominates $[(\Delta\mu_1 - f, \Delta k_1 - g), (\Delta\mu_1 + f, \Delta k_2 + g)]$ for any $f, g \geq 0$.

We use the following important compositional property throughout: if we have a branching rule B generating ℓ subproblems with drops $(\Delta\mu_i, \Delta k_i)$, and we then apply a branching rule with branch-seq B'_i to each subproblem $i = 1, \dots, \ell$, then, overall, we get

$$\text{val}_{a,b}(B) = \sum_{i=1}^{\ell} e^{-a\Delta\mu_i - b\Delta k_i} \text{val}_{a,b}(B'_i).$$

► **Lemma 10.** *Suppose that \mathcal{G} is a class of graphs closed under vertex deletion, and for which there are vertex cover algorithms with runtimes $O^*(e^{a\mu + bk})$ and $O(e^{cn})$ for $a, b, c \geq 0$. Then we can solve vertex cover in \mathcal{G} with runtime $O^*(e^{dk})$ where $d = \frac{2c(a+b)}{a+2c}$.*

To illustrate Lemma 10, consider graphs of maximum degree 3. We can combine the AGVC algorithm with runtime $O^*(2.3146^\mu)$ (corresponding to $a = \log(2.3146), b = 0$), and the MaxIS-3 algorithm with runtime $O^*(1.083506^n)$ (corresponding to $c = \log(1.083506)$), to get an algorithm with runtime $O^*(1.14416^k)$; this already gives us one of the results in Theorem 1.

3 Preprocessing rules and reductions in k

We define $S(G)$ to be the largest decrease in k obtainable from applying rules (P1) – (P3) exhaustively via some efficiently-computable sequence of operations.³ As a point of notation, we define $R(X) = S(G - X)$ for a vertex set X . For vertices x_1, \dots, x_ℓ and vertex sets X_1, \dots, X_r , we write $R(x_1, \dots, x_\ell, X_1, \dots, X_r)$ as shorthand for the more cumbersome $R(\{x_1, \dots, x_\ell\} \cup X_1 \cup \dots \cup X_r) = S(G - \{x_1, \dots, x_\ell\} - (X_1 \cup \dots \cup X_r))$. Note that we may have $\ell = 0$ or $r = 0$, e.g. $R(x_1, \dots, x_\ell)$ is shorthand for $R(\{x_1, \dots, x_\ell\})$.

We record a few observations on simplifications of various structures.

► **Observation 11.** *Suppose x, y are non-adjacent subquadratic vertices. Then $S(G) \geq 2$ if any of the following three conditions hold: (i) $\text{codeg}(x, y) = 0$ or (ii) $\text{deg}(x) + \text{deg}(y) \geq 3$ or (iii) $\text{minsurp}(G) \geq 0$.*

Proof. If $\text{deg}(x) = \text{deg}(y) = 1$ and $\text{codeg}(x, y) = 0$, then $\{x, y\}$ is a min-set with $\text{surp}(\{x, y\}) = 0$, and applying P1 reduces k by two. Otherwise, if $\text{deg}(x) \geq 2$, then we can apply (P2) to x , and vertex y remains subquadratic, and we can follow up by applying (P1) or (P2) again to y . Note that if $\text{minsurp}(G) \geq 0$, then at least one of the conditions (i) or (ii) must hold. ◀

► **Proposition 12.** *If G has 2-vertices x_1, \dots, x_ℓ which all have pairwise distance at least 3, then $S(G) \geq \ell$.*

Proof. We show it by induction on ℓ . The base case $\ell = 0$ is vacuous. For the induction step, we apply (P2) to x_ℓ , forming a graph G' where the neighbors y, z get contracted into a single new vertex t . By hypothesis, y, z are distinct from x_1, \dots, x_ℓ . We claim that $\text{dist}_{G'}(x_i, x_j) \geq 3$ and $\text{deg}_{G'}(x_i) \geq 2$ for any pair $i < j < \ell$. For, clearly $x_i \not\sim x_j$ in G' . If x_i, x_j share a neighbor in G' , it must be vertex t as no other vertices were modified. This is only possible if $\{x_i, x_j\} \sim \{y, z\}$; but this contradicts our hypothesis that $\text{codeg}(x_i, x_\ell) = \text{codeg}(x_j, x_\ell) = 0$. Finally, suppose that $\text{deg}_{G'}(x_i) \leq 1$. This is only possible if the two neighbors of x_i got merged together, i.e. $x_i \sim y$ and $x_i \sim z$. Again, this contradicts that $\text{codeg}(x_i, x_\ell) = 0$.

By induction hypothesis applied to G' , we have $S(G') \geq \ell - 1$ and hence $S(G) \geq \ell$. ◀

► **Lemma 13.** *If $\text{minsurp}(G) \geq 0$ and indset I has $\text{surp}_G(I) \leq 1$, then $S(G) \geq |I|$.*

Proof. Let $r = |I|$. If $\text{surp}_G(I) = 0$, or I is a critical-set, we can apply (P1) or (P2) to I . So suppose that $\text{surp}_G(I) = 1$ and $\text{surp}_G(X) = 0$ for a non-empty subset $X \subsetneq I$; among all such subsets X , choose one of maximum size, and let $s = |X|$. We apply (P1) to X , reducing k by s and obtaining a graph $G' = G - N[X]$. Now consider the indset $J = I \setminus X$ in G' , where $|J| = r - s$. We have $|N_{G'}(J)| = |N_G(I) \setminus N_G(X)| = r + 1 - s$ and so $\text{surp}_{G'}(J) = 1$.

We claim that J is a critical-set in G' . For, if some non-empty subset $X' \subseteq J$ has $|N_{G'}(X')| \leq |X'|$, then indset $I' = X \cup X' \subseteq I$ would have $|N_G(I')| \leq |N_G(X)| + |N_{G'}(X')| \leq |X| + |X'| = |I'|$. Hence $\text{surp}_G(I') \leq 0$, contradicting maximality of X . So, we can apply (P2) to J in G' , getting a further drop of $r - s$. Overall, we get a net drop of $s + (r - s) = r$ as desired. ◀

³ It is not clear how to calculate the *absolute* largest decrease in k via preprocessing rules, since applying some rules may prevent opportunities for other rules. We assume that we have fixed some polynomial-time computable sequences of potential preprocessing rules in order to reduce k as small as much as possible. At various points in our algorithm, we will describe simplifications available for intermediate graphs. We always assume that our preprocessing rules are chosen to find such simplifications.

4

 Branching rules

Most of our branching rules can be viewed in terms of guessing that a certain set of vertices X_1 is in the cover. In the graph $G - X_1$, a set of vertices X_0 may become isolated. We apply rule (P1) to remove X_0 ; the resulting subproblem $\langle G - (X_0 \cup X_1), k - |X_1| \rangle$ is called *principal*, and we define the *excess* to be $|X_0|$. Note that the graph $G' = G - (X_0 \cup X_1)$ may still contain isolated vertices, or have additional simplifications available. We say $\langle G', k - |X_1| \rangle$ has drop $(\Delta\mu', |X_1|)$ *directly*, and the final subproblem $G'' = \langle G'', k'' \rangle$ has drop $(\Delta\mu'', \Delta k'')$ *after simplification*.

Since this plays a central role in our analysis, we define the *shadow* of a vertex set X , denoted $\text{shad}(X)$, to be $\text{minsurp}(G - X)$. For readability, we write $\text{shad}(x_1, \dots, x_\ell, X_1, \dots, X_r)$ as shorthand for $\text{shad}(\{x_1, \dots, x_\ell\} \cup X_1 \cup \dots \cup X_r) = \text{minsurp}(G - \{x_1, \dots, x_\ell\} - (X_1 \cup \dots \cup X_r))$.

► **Proposition 14.** *If $\text{minsurp}(G) \geq 2$, then a principal subproblem $\langle G - X, k' \rangle$ with excess s has $\Delta\mu = \frac{1}{2}(\Delta k - s + \text{minsurp}^-(G')) = \frac{1}{2}(\Delta k - s + \min\{0, \text{shad}(X)\})$.*

Proof. Here Proposition 4 gives $\lambda(G) = n/2$. Let $G' = G - X$, where $|X| = \Delta k + s$. Then $\mu(G') = k' - \lambda(G') = k' - \frac{1}{2}(n' + \text{minsurp}^-(G'))$, where $n' = n - X = 2\lambda(G) - (s + \Delta k)$, which simplifies to $\mu(G') = \mu(G) - \frac{1}{2}(\Delta k - s + \text{minsurp}^-(G'))$ as desired. ◀

► **Observation 15.** *Suppose $\text{minsurp}(G) \geq 2$. Then for any indset I , there holds $\text{shad}(I) \geq 2 - |I|$ and $\text{shad}(N[I]) \geq 2 - \text{surp}_G(I)$.*

In particular, for a vertex u , there holds $\text{shad}(u) \geq 1$ and $\text{shad}(N[u]) \geq 3 - \text{deg}(u)$.

Proof. If J is a min-set of $G - I$, then $2 \leq \text{surp}_G(J) \leq \text{surp}_{G-I}(J) + |I|$. So $\text{surp}_{G-I}(J) \geq 2 - |I|$. Similarly, if J is a min-set of $G - N[I]$, then $2 \leq \text{surp}_G(I \cup J) = \text{surp}_{G-N[I]}(J) + \text{surp}_G(I)$. ◀

By combining Observation 11 and Observation 15, we immediately get the following simplification rule (which is used ubiquitously):

► **Observation 16.** *If G is simplified and $G - \{u, v\}$ has two non-adjacent subquadratic vertices, then $R(u, v) \geq 2$.*

Our bread-and-butter branching rule is to choose some vertex u , and branch on whether it is in the cover. We refer to this as *splitting on u* . This generates two principal subproblems $\langle G - u, k - 1 \rangle$ and $\langle G - N[u], k - \text{deg}(u) \rangle$. More generally, in light of Lemma 5, we know that for any critical-set I , there is either a good cover containing I or omitting I . We can branch on subproblems $\langle G - I, k - |I| \rangle$ and $\langle G - N[I], k - N[I] \rangle$, and we refer to this as *splitting on I* . Splitting on a vertex is a special case where I is a singleton.

Consider the effect of splitting on u . By Observation 15, the subproblem $\langle G - u, k - 1 \rangle$ has drop $(0.5, 1)$ and the subproblem $\langle G - N[u], k - \text{deg}(u) \rangle$ has drop $(1, \text{deg}(u))$. The latter bound, however, is very loose and should be improved to get an optimized analysis. In light of Proposition 14, it revolves around the value of $\text{shad}(N[u])$. If $\text{shad}(N[u]) \geq 0$, then λ drops by $\frac{\text{deg}(u)+1}{2}$ and so the subproblem has drop $(\frac{\text{deg}(u)-1}{2}, \text{deg}(u))$. This is the “generic” situation for splitting on u .

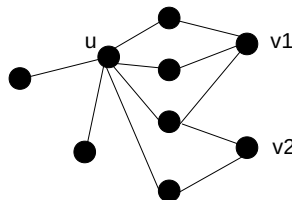
When $\text{shad}(N[u]) \leq 0$, we say that u is *blocked*. If x is a vertex in a min-set of $G - N[u]$, we say that x is a *blocker* of u . Necessarily, $x \approx u$. This motivates the following powerful branching rule:

(B) If x is a blocker of vertices u_1, \dots, u_ℓ , then branch on subproblems $\langle G - \{u_1, \dots, u_\ell, x\}, k - \ell - 1 \rangle$ and $\langle G - N[x], k - \text{deg}(x) \rangle$.

► **Proposition 17.** *Rule (B) is a valid branching rule.*

Proof. The subproblem $\langle G - N[x], k - \deg(x) \rangle$ is feasible if and only if G has a good cover omitting x . The subproblem $\langle G - \{u_1, \dots, u_\ell, x\}, k - \ell - 1 \rangle$ is feasible if and only if G has a good cover which includes all vertices u_1, \dots, u_ℓ, x . We claim that, if G is feasible, at least one of these cases holds. For, suppose a good cover of G omits vertex u_i . Then $G - N[u_i]$ has a cover of size at most $k - \deg(u_i)$. By Lemma 5, this implies that $G - N[u_i]$ has such a cover C' which omits any min-set I of $G - N[u_i]$, and in particular $x \notin C'$. Then G has a good cover $C \cup N(u_i)$ omitting x . ◀

In the vast majority of cases where we use (B), we have $\ell = 1$, that is, x is a blocker of a single vertex u . To provide intuition, keep in mind the picture that if $G - N[u]$ has s isolated vertices v_1, \dots, v_s , then $\{v_1, \dots, v_s\}$ would be an indset in $G - N[u]$ with zero neighbors and surplus $-s$, in particular, $\text{shad}(N[u]) \leq -s$. These vertices were hidden in the “shadow” of $N[u]$. See Figure 4.



■ **Figure 4** Here, vertices v_1, v_2 are both blockers of u ; in particular, we have $\text{shad}(N[u]) \leq -2$.

Also, note that if $\text{shad}(N[u]) = 0$ exactly, then we could still split on u if desired and get the ideal drop in μ . However, we can also apply rule (B) if desired, and this is usually more profitable. Vertices with $\text{shad}(N[u]) = 0$ share many properties with vertices with $\text{shad}(N[u]) < 0$ strictly. This is the reason for the somewhat unintuitive definition of *blocked vertex*.

5 Analysis of blockers and splitting

We will develop a series of branching rules (typically using rule (B)) to handle blocked vertices, along with a number of other related cases. The most significant consequence of these rules is that we are able to get near-ideal branch sequences for high-degree vertices. Specifically, we will show the following main result which is the heart of our branching algorithm:

► **Theorem 18.** *Suppose G is simplified, and let $r = \max\deg(G)$. Depending on r , then following branch-seqs are available:*

If $r \geq 4$: $[(1, 3), (1, 5)]$ or $[(0.5, 1), (1.5, 4)]$.

If $r \geq 5$: $[(1, 3), (1, 5)]$ or $[(0.5, 1), (2, 5)]$.

If $r \geq 6$: $[(1, 3), (1, 5)]$, $[(0.5, 2), (2, 5)]$, or $[(0.5, 1), (2.5, r)]$.

The proof of Theorem 18 has many cases which will require us to build up slowly from analysis of low-degree vertices. **For Section 5 only, we always assume that the starting graph G is simplified.** We begin with a few elementary observations.

► **Observation 19.** *Suppose u is blocked and let I be a min-set of $G - N[u]$. There is at least one vertex $x \in I$ with $\text{codeg}(x, u) \geq 1$.*

Proof. If not, we would have $\text{surp}_G(I) = \text{surp}_{G-N[u]}(I) \leq 0$, contradicting that G is simplified. \blacktriangleleft

► **Proposition 20.** *Suppose vertex u has $\text{shad}(N[u]) \leq 4 - \deg(u)$ and x is a blocker of u . If we apply (B) to vertices u, x , then subproblem $G' = \langle G - N[x], k - \deg(x) \rangle$ has drop (1, 4). Moreover, if $\deg(x) = 3$ and x is contained in a non-singleton min-set of $G - N[u]$, then G' has drop (1, 5).*

Proof. Since G is simplified, we have $\deg(x) \geq 3$ and $\text{minsurp}(G) \geq 2$. If $\deg(x) = 4$, then G' has drop (1, 4) directly. So, suppose $\deg(x) = 3$ exactly. Consider any min-set I of $G - N[u]$ with $x \in I$. The indset $J = I \cup \{u\} \setminus \{x\}$ has $\text{surp}_{G-N[x]}(J) \leq \text{surp}_{G-N[u]}(I) + (\deg(u) - \deg(x)) \leq 1$. On the other hand, $\text{shad}(N[x]) \geq \text{minsurp}(G) - \deg(x) + 1 \geq 0$. So Lemma 13 gives $R(N[x]) \geq |J| = |I|$.

By Proposition 14 we see that G' has drop (1, 3) directly. Since $|I| \geq 1$ trivially, we have $R(N[x]) \geq 1$ and G' has drop (1, 4) after simplification. If $|I| \geq 2$, then $R(N[x]) \geq 2$ and likewise G' has drop (1, 5) after simplification. \blacktriangleleft

5.1 Branching rules for indsets with surplus two

It will be very important to develop special branching rules for indsets with surplus two. Since G is assumed to be simplified, note that any such set I will be a critical set, and we can split on I to get subproblems $\langle G - I, k - |I| \rangle$ and $\langle G - N[I], k - |I| - 2 \rangle$. However, this branching rule is not powerful enough for us; we develop specialized branching rules for smaller indsets (indsets with cardinality two or three).

► **Proposition 21.** *Suppose G has an indset I with $\text{surp}_G(I) = 2$, and let $z \in N(I)$. If we split on z , the subproblem $\langle G - z, k - 1 \rangle$ has drop (0.5, 1 + |I|). If z has a blocker $x \notin I$ and we apply (B) to z, x , the subproblem $\langle G - \{z, x\}, k - 2 \rangle$ has drop (1, 2 + |I|).*

Proof. Subproblem $\langle G - z, k - 1 \rangle$ has drop (0.5, 1) directly. We have $\text{surp}_{G-z}(I) = \text{surp}_G(I) - 1 = 1$, and $\text{shad}(G - z) \geq \text{minsurp}(G) - 1 \geq 1$. So (P2) applied to I gives $R(z) \geq |I|$. Likewise, subproblem $\langle G - \{z, x\}, k - 2 \rangle$ has drop (1, 2) directly, and $\text{shad}(z, x) \geq \text{minsurp}(G) - 2 \geq 0$ and $\text{surp}_{G-\{z,x\}}(I) \leq \text{surp}_G(I) - 1 \leq 1$. So Lemma 13 gives $R(z, x) \geq |I|$. \blacktriangleleft

► **Lemma 22.** *Suppose G has an indset I with $\text{surp}_G(I) = 2, |I| \geq 3$. Then G has available branch-seq [(1, 4), (1, 5)] or [(0.5, 4), (2, 5)].*

Proof. If $|I| \geq 4$, then splitting on I gives branch-seq [(1, 4), (1, 6)]. So suppose $|I| = 3$. We first claim that there is some vertex z with $|N(z) \cap I| = 2$. For, consider the set of vertices $J = \{z : |N(z) \cap I| \geq 2\}$. The vertices in I are independent and have degree at least three, so there are at least 9 edges from I to $N(I)$. Since $|N(I)| = 5$, by the pigeonhole principle we must have $|J| \geq 2$. If $N(z) \not\subseteq I$ for all $z \in J$, then J would be an indset with surplus at most one, contradicting that G is simplified. Hence there is $z \in J$ with $|N(z) \cap I| = 2$ exactly.

Let z be any such vertex, and let $I = \{x_1, x_2, y\}$ where $z \sim x_1, z \sim x_2, z \not\sim y$ and $\deg(x_1) \leq \deg(x_2)$. If $\deg(z) \leq 4$, then we split on I , generating subproblems $\langle G - I, k - 3 \rangle$ and $\langle G - N[I], k - 5 \rangle$ with drops (1, 3) and (1, 5) directly, where $G - I$ has a subquadratic vertex z so it has drop (1, 4) after simplification. If $\deg(z) \geq 5$ and $\text{shad}(N[z]) \geq 5 - \deg(z)$, then we split on z ; by Proposition 21, subproblem $\langle G - z, k - 1 \rangle$ has drop (0.5, 4), and subproblem $\langle G - N[z], k - \deg(z) \rangle$ has drop (2, 5) directly. If $\deg(z) \geq 5$ and z has a blocker $t \neq y$, then we apply (B) to z, t ; by Proposition 20, subproblem $\langle G - N[t], k - \deg(t) \rangle$ has drop (1, 4) and by Proposition 21, subproblem $\langle G - \{z, t\}, k - 2 \rangle$ has drop (1, 5).

The only remaining possibility is if $\deg(z) \geq 5$ and y is the only blocker of z , i.e. $\deg(z) = 5$, $\deg(y) = 3$ and $N(y) \subseteq N(z)$. Each vertex x_i has $\deg(x_i) + \deg(y) - \text{codeg}(x_i, y) \leq |N(I)|$, i.e. $\text{codeg}(x_i, y) \geq \deg(x_i) - 2$. If $\deg(x_1) = 3$, this implies that x_1, y share some neighbor u ; but in this case G has a 3-triangle x_1, u, z , contradicting that G is simplified.

So we suppose $\deg(x_2) \geq \deg(x_1) \geq 4$ and then $\text{codeg}(x_1, y) \geq 2$ and $\text{codeg}(x_2, y) \geq 2$. Equivalently, $\deg_{G-N[x_1]}(y) \leq 1$ and $\deg_{G-N[x_2]}(y) \leq 1$. So y is a blocker of all three vertices x_1, x_2, z . We apply (B) giving subproblems $\langle G-N[y], k-\deg(y) \rangle$ and $\langle G-\{x_1, x_2, z, y\}, k-4 \rangle$, with drops (1, 3) and (1, 4) directly. Furthermore, $\text{surp}_{G-N[y]}(\{x_1, x_2\}) \leq 2 - |N_{G-N[y]}(I)| = 2 - 2 = 0$ and neither x_1 or x_2 is isolated in $G - N[y]$ (they retain neighbor z). So we can apply (P1) to indset $\{x_1, x_2\}$ in $G - N[y]$, giving $R(N[y]) \geq 2$. So $G - N[y]$ has drop (1, 5) after simplification. ◀

► **Lemma 23.** *Suppose G has an indset I with $\text{surp}_G(I) = 2, |I| \geq 2$. Then G has available branch-seq $[(1, 4), (1, 4)]$ or $[(0.5, 3), (2, 5)]$.*

Proof. If $|I| \geq 3$, then we apply Lemma 22. So let $I = \{x, y\}$, where $\deg(x) \leq \deg(y)$ and let $A = N(x, y)$. Note that $|A| = \deg(x) + \deg(y) - 4 \geq \deg(x) - 1$. The vertices in A cannot form a clique, as then x would have a funnel. So consider vertices $z_1, z_2 \in A$ with $z_1 \not\sim z_2$.

If z_1 and z_2 are both subquartic, then we split on I ; the subproblem $\langle G-N[I], k-|N[I]|-2 \rangle$ has drop (1, 4) directly. The subproblem $G - I = G - \{x, y\}$ has non-adjacent subquadratic vertices z_1, z_2 , so by Observation 16, we have $R(x, y) \geq 2$ and subproblem $\langle G - I, k - |I| \rangle$ has drop (1, 4) after simplification.

So suppose $\deg(z_1) \geq 5$. If $\text{shad}(N[z_1]) \geq 0$, then split on z_1 ; by Proposition 21, the subproblem $\langle G - z_1, k - 1 \rangle$ has drop (0.5, 3), while the subproblem $\langle G - N[z_1], k - \deg(z_1) \rangle$ has drop (2, 5) directly. Otherwise, suppose $\text{shad}(N[z_1]) \leq -1$ and t is a blocker of z_1 . Note that $t \notin I$ since $t \not\sim z_1$ whereas $z_1 \in N(x) \cap N(y)$. We then apply (B) to t, z_1 ; the subproblem $\langle G - N[z_1], k - \deg(z_1) \rangle$ has drop (1, 5) directly, and by Proposition 21, the subproblem $\langle G - \{t, z_1\}, k - 2 \rangle$ has drop (1, 4). ◀

5.2 Branching rules for blocked vertices

We now turn to developing branching rules for blocked vertices of various types. We need to work our way from low to high degree.

► **Proposition 24.** *Suppose vertex u has $\text{shad}(N[u]) \leq 5 - \deg(u)$ and u has a blocker x of degree at least 4. Then G has available branch-seq $[(1, 4), (1, 4)]$ or $[(0.5, 2), (2, 5)]$.*

Proof. If $\text{shad}(N[x]) \leq 3 - \deg(x)$ and J is a min-set of $G - N[x]$, then $\text{surp}_G(J \cup \{x\}) \leq (3 - \deg(x)) + (\deg(x) - 1) \leq 2$ and we can apply Lemma 23 to indset $J \cup \{x\}$.

So suppose $\text{shad}(N[x]) \geq 4 - \deg(x)$. In this case, we apply (B) to u, x . Subproblem $\langle G - \{u, x\}, k - 2 \rangle$ has drop (1, 2) directly. If $\deg(x) \geq 5$, then subproblem $\langle G - N[x], k - \deg(x) \rangle$ has drop (1.5, 5) directly. If $\deg(x) = 4$, then consider a min-set I of $G - N[u]$ with $x \in I$. We have $\text{surp}_{G-N[x]}(I \cup \{u\} \setminus \{x\}) \leq \text{surp}_{G-N[u]}(I) + \deg(u) - \deg(x) \leq 1$. So $R(N[x]) \geq 1$ and G' has drop (1.5, 5) after simplification. In either case, $\langle G - \{u, x\}, k - 2 \rangle$ and $\langle G - N[x], k - \deg(x) \rangle$ give drops (1, 2), (1.5, 5) respectively. This branch-seq $[(1, 2), (1.5, 5)]$ dominates $[(0.5, 2), (2, 5)]$. ◀

► **Lemma 25.** *Suppose G has a vertex of degree at least 5 which has a 3-neighbor. Then G has available branch-seq $[(1, 3), (1, 5)]$ or $[(0.5, 2), (2, 5)]$.*

Proof. Let u be a vertex of degree at least 5 and let z be a 3-neighbor of u . There are a number of cases to consider.

Case I: $\text{shad}(N[u]) \geq 5 - \deg(u)$. Splitting on u gives subproblems $G - u, G - N[u]$ with drops $(0.5, 1), (2, 5)$ directly; the former has drop $(0.5, 2)$ after simplification due to its 2-neighbor z .

Case II: u has a blocker x with $\deg(x) \geq 4$. We suppose that $\text{shad}(N[u]) \leq 4 - \deg(u)$ since otherwise it would be covered in Case I. Then the result follows from Proposition 24, noting that $[(1, 4), (1, 4)]$ dominates $[(1, 3), (1, 5)]$.

Case III: u has a blocker x with $R(u, x) \geq 2$. We apply (B) to vertices u, x . Subproblem $\langle G - \{u, x\}, k - 2 \rangle$ has drop $(1, 4)$ after simplification, and by Proposition 20 subproblem $\langle G - N[x], k - \deg(x) \rangle$ has drop $(1, 4)$. We get branch-seq $[(1, 4), (1, 4)]$, which dominates $[(1, 3), (1, 5)]$.

Case IV: $G - N[u]$ has some non-singleton min-set I . Let $x \in I$. Necessarily $\deg(x) = 3$, else it would be covered in Case II. We apply (B) to u, x ; subproblem $\langle G - \{u, x\}, k - 2 \rangle$ has drop $(1, 3)$ after simplification due to subquadratic vertex z . By Proposition 20 subproblem $\langle G - N[x], k - \deg(x) \rangle$ has drop $(1, 5)$ after simplification.

Case V: No previous cases apply. Suppose that no vertices in the graph are covered by any of the previous cases. Let U denote the non-empty set of vertices u with $\deg(u) \geq 5$ and u having a 3-neighbor. We claim that every vertex $u \in U$ has degree exactly 5. For, suppose $\deg(u) \geq 6$, and since we are not in Case I we have $\text{shad}(N[u]) \leq 4 - \deg(u) \leq -2$. Then necessarily a min-set of $G - N[u]$ would have size at least two, and it would be covered in Case IV.

So every $u \in U$ has degree exactly 5, and is blocked by a singleton 3-vertex x , i.e. $N(x) \subseteq N(u)$. Let us say in this case that x is *linked* to u ; we denote by X the set of all such 3-vertices linked to any vertex in U . We claim that each $x \in X$ has at least two neighbors of degree at least 5, which must be in U due to their 3-neighbor x . For, suppose that x has two subquartic neighbors z_1, z_2 ; necessarily $z_1 \not\sim z_2$ since G is simplified and x has degree 3. Then $R(u, x) \geq 2$ by Observation 16, which would have been covered in Case III.

On the other hand, we claim that each vertex $u \in U$ has at most one neighbor in X . For, suppose u is linked to x and u has two 3-neighbors $x_1, x_2 \in X$, which must be non-adjacent since G is simplified. Then Observation 16 gives $R(u, x) \geq 2$ due to subquadratic vertices x_1, x_2 . This would have been covered in Case III.

Thus, we see that each vertex in X has at least two neighbors in U and each vertex in U has at most one neighbor in X . Hence $|U| \geq 2|X|$, and so by the pigeonhole principle, there must be some vertex $x \in X$ which is linked to two vertices $u_1, u_2 \in U$. We apply (B) to u_1, u_2, x , getting subproblems $\langle G - \{u_1, u_2, x\}, k - 3 \rangle$ and $\langle G - N[x], k - \deg(x) \rangle$. By Proposition 20, the latter subproblem has drop $(1, 4)$. Furthermore, if z_1, z_2 are two 5-neighbors of x , then $z_1 \not\sim z_2$ else x would have a 3-triangle. So $G - \{u_1, u_2, x\}$ has non-adjacent 2-vertices z_1, z_2 , and subproblem $\langle G - \{u_1, u_2, x\}, k - 3 \rangle$ has drop $(1, 5)$ after simplification. ◀

► **Proposition 26.** *Suppose a vertex u has $\deg(u) \geq 4$ and $\text{shad}(N[u]) \leq 4 - \deg(u)$. Then G has available branch-seq $[(1, 3), (1, 5)]$ or $[(0.5, 2), (2, 5)]$.*

Proof. Let I be a min-set of $G - N[u]$. By Observation 19, there is a vertex $x \in I$ which shares some neighbor t with u . If $\deg(x) \geq 4$, then we apply Proposition 24, noting that $[(1, 4), (1, 4)]$ dominates $[(1, 3), (1, 5)]$. If $\deg(x) = 3$ and $\deg(t) \geq 5$, we apply Lemma 25 to

t. If $\deg(x) = 3, \deg(t) \leq 4, |I| \geq 2$, we apply (B) to u, x ; subproblem $\langle G - \{u, x\}, k - 2 \rangle$ has drop (1, 3) after simplification (due to subquadratic vertex t) and by Proposition 20 subproblem $\langle G - N[x], k - \deg(x) \rangle$ has drop (1, 5).

So finally take $I = \{x\}$ where $\deg(x) = 3$ and x shares two neighbors t_1, t_2 with u . Necessarily $t_1 \not\sim t_2$ since G has no 3-triangles. If either t_1 or t_2 has degree 5 or more, we can apply Lemma 25. If they are both subquartic, then we apply (B) to u, x ; by Proposition 20, subproblem $\langle G - N[x], k - \deg(x) \rangle$ has drop (1, 4) and by Observation 16 there holds $R(u, x) \geq 2$ and subproblem $\langle G - \{u, x\}, k - 2 \rangle$ has drop (1, 4) after simplification. Here, $[(1, 4), (1, 4)]$ dominates $[(1, 3), (1, 5)]$. ◀

► **Lemma 27.** *Suppose vertex u has $\deg(u) \geq 5$ and $\text{shad}(N[u]) \leq 5 - \deg(u)$.*

- *If $\deg(u) = 5$, then G has available branch-seq $[(1, 3), (1, 4)]$ or $[(0.5, 2), (2, 5)]$.*
- *If $\deg(u) \geq 6$, then G has available branch-seq $[(1, 3), (1, 5)]$ or $[(0.5, 2), (2, 5)]$.*

Proof. Let I be a min-set of $G - N[u]$ of smallest size. If any vertex in I has degree at least 4, the result follows from Proposition 24 (noting that $[(1, 4), (1, 4)]$ dominates $[(1, 3), (1, 5)]$). So suppose that all vertices in I have degree 3. By Observation 19, we may select some $x \in I$ with $\text{codeg}(x, u) \geq 1$. For this vertex x , define $Z = N(x) \cap N(u)$ and $Y = N_{G-N[u]}(x)$, where $|Z| + |Y| = \deg(x) = 3$ and $|Z| \geq 1$.

If any vertex $z \in Z$ has degree at least 5, we apply Lemma 25 to z with its 3-neighbor x . If any vertex $z \in Z$ has degree 3, we apply Lemma 25 to u with its 3-neighbor z . If $\text{codeg}(x, x') \geq 2$ for any vertex $x' \in I \setminus \{x\}$, then we apply Lemma 23 to the surplus-two indset $\{x, x'\}$. So we suppose that $\text{codeg}(x, x') \leq 1$ for all $x' \in I \setminus \{x\}$ and that all vertices in Z have degree exactly four.

The vertices in Z must be independent, as otherwise G would have a 3-triangle with x . If $\text{codeg}(z, z') \geq 3$ for any vertices $z, z' \in Z$, then $\text{shad}(N[z]) \leq 0$ (since $G - N[z]$ has a 1-vertex z'), and we can apply Proposition 26. So we suppose the vertices in Z share no neighbors besides u and x . In this case, in the graph $G - \{u, x\}$, the vertices in Z become subquadratic and share no common neighbors. By Proposition 12, we thus have $R(u, x) \geq |Z|$.

We next claim that if $|I| \geq 2$, then $R(u, x) \geq |Y|$. For, every vertex $y \in Y$ must have a neighbor $\sigma(y) \in I \setminus \{x\}$, as otherwise we would have $\text{surp}_{G-N[u]}(I') \leq \text{surp}_{G-N[u]}(I)$ for non-empty ind-set $I' = I \setminus \{x\}$, contradicting minimality of I . The vertices $\sigma(y) : y \in Y$ must be distinct, as a common vertex $x' = \sigma(y) = \sigma(y')$ would have $\text{codeg}(x, x') \geq 2$ which we have already ruled out. So, in the graph $G - N[x]$, each vertex $\sigma(y) \in I$ loses neighbor y . By Observation 15, we have $\text{shad}(N[x]) \geq 3 - \deg(x) = 0$. So $G - N[x]$ has $|Y| \leq 2$ non-adjacent subquadratic vertices. Observation 16 implies that $R(N[x]) \geq |Y|$.

At this point, our strategy is to apply (B) to u, x . If $|I| = 1$, then either $\deg(u) = 5, |Z| \geq 2$ or $\deg(u) \geq 6, |Z| = 3$. Subproblem $\langle G - N[x], k - \deg(x) \rangle$ has drop (1, 3) directly and subproblem $\langle G - \{u, x\}, k - 2 \rangle$ has drop $(1, 2 + |Z|)$ after simplification; that is, it has drop (1, 4) if $\deg(u) = 5$ and drop (1, 5) otherwise.

Otherwise, if $|I| \geq 2$, then subproblem $\langle G - \{u, x\}, k - 2 \rangle$ has drop $(1, 2 + |Z|)$ after simplification and subproblem $\langle G - N[x], k - \deg(x) \rangle$ has drop $(1, 3 + |Y|)$ after simplification. Since $|Z| + |Y| = 3$ and $|Z| \geq 1$, this always gives drops (1, 3), (1, 5) or (1, 4), (1, 4). ◀

We can finally prove Theorem 18 (restated for convenience).

► **Theorem 18.** *Suppose G is simplified, and let $r = \max\deg(G)$. Depending on r , then following branch-seqs are available:*

If $r \geq 4$: $[(1, 3), (1, 5)]$ or $[(0.5, 1), (1.5, 4)]$.

If $r \geq 5$: $[(1, 3), (1, 5)]$ or $[(0.5, 1), (2, 5)]$.

If $r \geq 6$: $[(1, 3), (1, 5)], [(0.5, 2), (2, 5)],$ or $[(0.5, 1), (2.5, r)]$.

Proof. Consider an r -vertex u . If $r = 4$ and $\text{shad}(N[u]) \geq 0$, then split on u with branch-seq $[(0.5, 1), (1.5, 4)]$. If $r = 4$ and $\text{shad}(N[u]) \leq -1$, then use Proposition 26. If $r = 5$ and $\text{shad}(N[u]) \geq 0$, then split on u with branch-seq $[(0.5, 1), (2, 5)]$. If $r = 5$ and $\text{shad}(N[u]) \leq -1$, then apply Proposition 26. If $r \geq 6$ and $\text{shad}(N[u]) \geq 6 - r$, then split on u , getting branch-seq $[(0.5, 1), (2.5, r)]$. If $r \geq 6$ and $\text{shad}(N[u]) \leq 5 - r$, then apply Lemma 27. ◀

6 A simple branching algorithm

We now describe our first branching algorithm for vertex cover. As discussed earlier, the measure is a piecewise-linear function of k and μ . It is more convenient to describe it in terms of multiple self-contained “mini-algorithms,” with *linear* measure functions. As a starting point, we can describe the first algorithm:

■ **Algorithm 1** Function `Branch4Simple`(G, k).

-
- 1 Simplify G .
 - 2 If $\text{maxdeg}(G) \leq 3$, then run either the algorithm of Theorem 2 or the MaxIS-3 algorithm, whichever is cheaper, and return.
 - 3 Otherwise, apply Theorem 18 to an arbitrary vertex of degree at least 4, and run `Branch4Simple` on the two resulting subproblems.
-

To clarify, despite the name `Branch4Simple`, the graph is allowed to have vertices of either higher or lower degrees. What we mean is that, depending on the current values of k and μ , it is advantageous to branch on a vertex of degree 4 or higher. Our algorithm is *looking* for such a vertex; if it is not present (i.e. the graph has maximum degree 3) then an alternate algorithm should be used instead. Specifically, here we use either the AGVC algorithm with runtime depending on μ or the MaxIS-3 algorithm with runtime depending on $n = 2(k - \mu)$.

Henceforth we describe our branching algorithms more concisely. Whenever we apply a branching rule, we assume that we recursively use the algorithm under consideration (here, `Branch4Simple`) on the resulting subproblems and return. Likewise, when we list some other algorithms for the problem, we assume they are dovetailed (effectively running the cheapest of them).

► **Proposition 28.** `Branch4Simple` has measure $a\mu + bk$ for $a = 0.71808, b = 0.019442$.

Proof. Simplifying the graph in Line 1 can only reduce $\phi(G) = a\mu + bk$. If Theorem 18 is used, it gives a branch-seq B with drops dominated by $[(1, 3), (1, 5)]$ or $[(0.5, 1), (1.5, 4)]$. To satisfy Eq. (3), we thus need:

$$\text{val}_{a,b}(B) \leq \max\{e^{-a-3b} + e^{-a-5b}, e^{-0.5a-b} + e^{-1.5a-4b}\} \leq 1 \quad (4)$$

Otherwise, suppose G has maximum degree 3. Since it is simplified, it has $n = 2\lambda = 2(k - \mu)$, so the MaxIS-3 algorithm has runtime $O^*(1.083506^{2(k-\mu)})$. Then, it suffices to show that

$$\min\{\mu \log(2.3146), 2(k - \mu) \log(1.083506)\} \leq a\mu + bk. \quad (5)$$

This can be verified mechanically. ◀

By Lemma 10, this combined with the MaxIS-4 algorithm (with $a = 0.71808, b = 0.019442, c = \log 1.137595$) immediately gives runtime $O^*(1.2152^k)$ for degree-4 graphs.

Although Proposition 28 is easy to check directly, the choices for the parameters a and b may seem mysterious. The explanation is that the most constraining part of the algorithm is dealing with the lower-degree graphs, i.e. solving the problem when the graph has maximum degree 3. The inequality (5), which governs this case, should be completely tight: there should be a “triple point” with respect to `Branch4Simple`, the `MaxIS-3` algorithm, and the algorithm of Theorem 2. That is, for chosen parameters a, b , we should have

$$\mu \log(2.3146) = 2(k - \mu) \log(1.083506) = a\mu + bk$$

This allows us to determine b in terms of a . Then our goal is to minimize a whilst respecting the branching constraints of Eq. (4). This same reasoning will be used for parameters in all our algorithms. For example, we can define algorithms for branching on degree 5 and degree 6 vertices.

■ **Algorithm 2** Function `Branch5Simple`(G, k).

-
- 1 Simplify G
 - 2 If $\max\deg(G) \leq 4$, then run either the `MaxIS-4` algorithm or `Branch4Simple`
 - 3 Otherwise, apply Theorem 18 to an arbitrary vertex of degree at least 5.
-

■ **Algorithm 3** Function `Branch6Simple`(G, k).

-
- 1 Simplify G
 - 2 If $\max\deg(G) \leq 5$, then run either the `MaxIS-5` algorithm or `Branch5Simple`
 - 3 Otherwise, apply Theorem 18 to an arbitrary vertex of degree at least 6.
-

Along similar lines, we immediately obtain the results:

- **Proposition 29.** `Branch5Simple` has measure $a\mu + bk$ for $a = 0.44849, b = 0.085297$.
`Branch6Simple` has measure $a\mu + bk$ for $a = 0.20199, b = 0.160637$.

Combined with the `MaxIS-5` algorithm using Lemma 10, `Branch5Simple` immediately gives an algorithm for degree-5 graphs with runtime $O^*(1.2491^k)$. The final algorithm is very similar, but we only keep track of runtime in terms of k , not μ or n .

■ **Algorithm 4** Function `Branch7Simple`(G, k).

-
- 1 If $\max\deg(G) \leq 6$, then use Lemma 10 with algorithms of `Branch6Simple` and `MaxIS-6`
 - 2 Otherwise, split on an arbitrary vertex of degree at least 7.
-

- **Theorem 30.** Algorithm `Branch7Simple`(G, k) runs in time $O^*(1.2575^k)$.

Proof. If we split on a vertex of degree at least 7, then we generate subproblems with vertex covers of size at most $k - 1$ and $k - 7$. These have cost 1.2575^{k-1} and 1.2575^{k-7} by induction. Since $1.2575^{-1} + 1.2575^{-7} < 1$, the overall cost is $O^*(1.2575^k)$. Otherwise the two algorithms in question have measure $a\mu + bk$ and cn respectively; where $a = 0.20199, b = 0.160637, c = \log(1.1893)$; by applying Lemma 10, we get a combined algorithm with cost $O^*(1.2575^k)$ as desired. ◀

References

- 1 J. Chen, I.A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010.
- 2 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 3 R.G. Downey and M.R. Fellows. *Parameterized complexity*. Springer, 1999.
- 4 J. Flum and M. Grohe. *Parameterized complexity theory*. Springer, 2006.
- 5 F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Springer-Verlag Berlin Heidelberg, 2010.
- 6 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman and Company, 1979.
- 7 Shivam Garg and Geevarghese Philip. Raising the bar for vertex cover: Fixed-parameter tractability above a higher guarantee. In *Proc. 26th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1152–1166, 2016.
- 8 Leon Kellerhals, Tomohiro Koana, and Pascal Kunz. Vertex cover and feedback vertex set above and below structural guarantees. In *Proc. 17th International Symposium on Parameterized and Exact Computation (IPEC)*, 2022.
- 9 D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014.
- 10 M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31:335–354, 1999.
- 11 M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *Journal of Computer and System Sciences*, 75(2):137–153, 2009.
- 12 G. L. Nemhauser and L. E. Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6(1):48–61, 1974.
- 13 G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975.
- 14 R. Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford University Press, 2006.
- 15 Dekel Tsur. Above guarantee parameterization for vertex cover on graphs with maximum degree 4. *Journal of Combinatorial Optimization*, 45(1):34, 2023.
- 16 Mingyu Xiao and Hiorshi Nagamochi. A refined algorithm for maximum independent set in degree-4 graphs. *Journal of Combinatorial Optimization*, 34(3):830–873, 2017.
- 17 Mingyu Xiao and Hiroshi Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theoretical Computer Science*, 469:92–104, 2013.
- 18 Mingyu Xiao and Hiroshi Nagamochi. An exact algorithm for maximum independent set in degree-5 graphs. *Discrete Applied Mathematics*, 199:137–155, 2016.
- 19 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

A Proofs for basic results

Proof of Proposition 4. Consider any basic solution $\theta \in \{0, \frac{1}{2}, 1\}^V$ to LPVC(G). Let $I_0 = \theta^{-1}(0)$ and $I_1 = \theta^{-1}(1)$; note that I_0 is an indset (possibly empty) and $N(I_0) \subseteq I_1$. So $\sum_v \theta(v) = |I_1| + \frac{1}{2}(n - |I_0| - |I_1|) \geq (n + |N(I_0)| - |I_0|)/2 = (n + \text{surp}(I_0))/2 \geq (n + \text{minsurp}^-(G))/2$. On the other hand, if I_0 is a min-set of G , then setting $\theta(v) = 1$ for $v \in N(I_0)$ and $\theta(v) = 0$ for $v \in I_0$ and $\theta(v) = 1/2$ for all other vertices v gives a fractional vertex cover with $\sum_v \theta(v) = (n + \text{minsurp}^-(G))/2$. ◀

Proof of Lemma 5. Let I be a critical-set; we first claim there is a good cover C with $I \subseteq C$ or $I \cap C = \emptyset$. For, suppose that $I \cap C = J$ where $\emptyset \neq J \subsetneq I$. In this case, we must also have $N(I \setminus J) \subseteq C$, so overall $|C \cap N[I]| \geq |J| + |N(I \setminus J)| = |J| + |I \setminus J| + \text{surp}(I \setminus J)$; since $I \setminus J$ is a non-empty subset of I , we have $\text{surp}(I \setminus J) \geq \text{surp}(I)$, hence this is at least $|J| + |I \setminus J| + \text{surp}(I) = |I| + \text{surp}(I) \geq N(I)$. Thus, replacing $C \cap N[I]$ by $N(I)$ would give a vertex cover C' with $C' \cap I = \emptyset$ and $|C'| \leq |C|$.

Furthermore, if $\text{surp}(I) \leq 0$, then consider a good cover C with $I \subseteq C$. Then replacing I with $N(I)$ would give another cover C' with $|C'| \leq |C| + |N(I)| - |I| \leq |C|$ and $C' \cap I = \emptyset$.

Finally, suppose $\text{surp}(I) = 1$ and let $J = N(I)$. If G has a good cover with $I \cap C = \emptyset$, then $J \subseteq C$ as desired. Otherwise, suppose G has a good cover with $I \subseteq C$. Then, $C' = (C \setminus I) \cup J$ would be another cover of size $|C| - |I| + (|J| - |C \cap J|)$; since $|J| = |I| + 1$, this is $|C| + 1 - |C \cap J|$. If $C \cap J \neq \emptyset$, this would be a good cover with $I \cap C = \emptyset$. ◀

Proof of Proposition 6. Let J be an indset with $\text{surp}_{G-N[I]}(J) = \text{minsurp}^-(G - N[I])$. Then $J \cup I$ is a non-empty indset of G with surplus $\text{surp}_{G-N[I]}(J) + \text{surp}_G(I) = \text{minsurp}^-(G - N[I]) + \text{surp}_G(I)$; in particular, if $\text{minsurp}^-(G - N[I]) + \text{surp}_G(I) = \text{minsurp}(G)$, then I is a min-set of G . On the other hand, if I is contained in a min-set J of G , then $\text{surp}_{G-N[I]}(J \setminus I) = \text{surp}_G(J) + |I| - |N(I)| = \text{surp}_G(J) - \text{surp}_G(I)$. So $\text{minsurp}^-(G - N[I]) \leq \text{minsurp}(G) - \text{surp}_G(I)$. ◀

Proof of Proposition 7. By applying Proposition 6 to indsets $I = \{x\}$, we get $\text{minsurp}(G) = \min_{x \in V} (\text{minsurp}^-(G - N[x]) + \deg(x) - 1)$. We can use Proposition 4 to compute each value $\text{minsurp}^-(G - N[x])$ in polynomial time (by solving the LP). This gives an algorithm to compute $\text{minsurp}(G)$. Likewise, given any vertex-set X , we can use Proposition 6 to determine if X is contained in a min-set of G and, if so, we can use Proposition 4 to find a vertex set J attaining $\text{surp}_{G-N[X]}(J) = \text{minsurp}^-(G - N[X])$; then $J \cup X$ is a min-set of G containing X . ◀

Proof of Proposition 8. Let us first consider P1. If G' has a good cover C' , then $C' \cup N(I)$ is a cover for G of size k . Conversely, by Lemma 5, there is a good cover C of G with $C \cap I = \emptyset$, and then $C \setminus N(I)$ is a cover of G' of size $k - |N(I)| = k'$. Similarly, if G' has a fractional cover θ , then we can extend to G by setting $\theta(v) = 0$ for $v \in I$ and $\theta(v) = 1$ for $v \in N(I)$, with total weight $\lambda(G') + (k - k')$. So $\mu(G') = k' - \lambda(G') \leq k' - (\lambda(G) - (k - k')) = k - \lambda(G) = \mu(G)$.

Next, for rule P2, let $J = N(I)$. Since I is a critical-set with surplus one, it cannot contain any isolated vertex. Given any good cover C of G with $J \subseteq C$, observe that $(C \setminus J) \cup \{y\}$ is a cover of G' of size $k - |J| + 1 = k'$. Likewise, given a good cover C of G with $J \cap C = \emptyset$, we have $N(J) \subseteq C$; in particular, $I \subseteq C$ since I has no isolated vertices, so $C \setminus I$ is a cover of G' of size $k - |I| = k'$. Conversely, consider a good cover C' of G' ; if $y \in C'$, then $(C' \cup J) \setminus \{y\}$ is a cover of G , of size $k' + |J| - 1 = k$. If $y \notin C'$, then $C' \cup I$ is a cover of G , of size $k' + |I| = k$. Similarly, if G has an optimal fractional cover θ , it can be extended to G by setting $\theta(v) = \theta(y)$ for $v \in J$, and $\theta(v) = 1 - \theta(y)$ for $v \in I$. This has weight $\lambda(G') + |I|(1 - \theta(y)) + |J|\theta(y) - \theta(y) = \lambda(G') + |I| \geq \lambda(G)$. So $\mu(G') = k' - \lambda(G') \leq (k - |I|) - (\lambda(G) - |I|) = \mu(G)$.

Now for P3, let $A = N_G(u) \cap N_G(x)$ and $B_x = N_G(x) \setminus N_G[u]$ and $B_u = N_G(u) \setminus N_G[x]$. To show the validity of the preprocessing rule, suppose $\langle G, k \rangle$ is feasible, and let C be a good cover of G with either $|C \cap \{u, x\}| = 1$; then $C \setminus (\{u, x\} \cup A)$ is a cover of G' of size k' . Conversely, suppose $\langle G', k' \rangle$ is feasible with a good cover C' . So $|B_u \setminus C'| \leq 1$. If $B_u \setminus C' = \emptyset$, then $C = C' \cup A \cup \{x\}$ is a cover of G . If $|B_u \setminus C'| = 1$, then necessarily $B_x \subseteq C'$ and $C = C' \cup A \cup \{u\}$ is a cover of G .

40:18 A Faster Algorithm for Vertex Cover Parameterized by Solution Size

Finally, we claim that (P3) satisfies $\mu(G') \leq \mu(G)$; since $k' = k - 1 - |A|$, it suffices to show that $\lambda(G) \leq \lambda(G') + 1 + |A|$. For, take an optimal fractional cover θ for G' . To extend it to a fractional cover for G , we set $\theta(v) = 1$ for $v \in A$ and there are a few cases to determine the values for $\theta(u)$ and $\theta(x)$. If $B_u = \emptyset$, we set $\theta(u) = 0, \theta(x) = 1$. Otherwise, if $B_u \neq \emptyset$, let $b_u = \min_{v \in B_u} \theta(v)$ and then set $\theta(u) = 1 - b_u, \theta(x) = b_u$; since G' has a biclique between B_u and B_x , this covers any edge between x and $v \in B_x$ with $\theta(x) + \theta(v) = b_u + \theta(v) \geq 1$. Overall, θ is a fractional vertex cover of G of weight $\sum_{v \in G'} \theta(v) + \theta(u) + \theta(x) + |A| = \lambda(G') + 1 + \text{codeg}(u, x)$. \blacktriangleleft

Proof of Lemma 9. Consider a kite u, x, y, z . We can view the application of (P3) to funnel u as a two-part process. First, we remove the shared neighbor $y \in N(u) \cap N(x)$; then, we merge vertices z, x into a new vertex u' . In the first step, we obtain a graph $G'' = G - y$ with $k'' = k - 1$ and $n'' = n - 1$. So, by Proposition 4, we have $\lambda(G'') = \lambda(G) - 1/2$ and hence $\mu(G'') \leq \mu(G) - 1/2$ and $k'' = k - 1$. By Proposition 8, the second step gives $\mu(G') \leq \mu(G'')$ and $k' = k'' - 1$. \blacktriangleleft

Proof of Lemma 10. First, exhaustively apply rule (P1) to G ; the resulting graph G' has $n' \leq n$ and $k' \leq k$, and $\text{minsurp}(G') \geq 0$. In particular, $\mu(G') = k' - \lambda(G') = k' - n'/2$. Note that $G' \in \mathcal{G}$ since (P1) just deletes vertices. Now dovetail the two algorithms for G' , running both simultaneously and returning the output of whichever terminates first. This has runtime $O^*(\min\{e^{cn'}, e^{a(k'-n'/2)+bk'}\})$. For fixed k' , this is maximized at $n' = \frac{2k'(a+b)}{a+2c}$, at which point it takes on value $e^{dk'} \leq e^{dk}$. \blacktriangleleft

Linear Loop Synthesis for Quadratic Invariants

S. Hitarth  

Hong Kong University of Science and Technology, Hong Kong

George Kenison  

Liverpool John Moores University, UK

Laura Kovács  

TU Wien, Austria

Anton Varonka  

TU Wien, Austria

Abstract

Invariants are key to formal loop verification as they capture loop properties that are valid before and after each loop iteration. Yet, generating invariants is a notorious task already for syntactically restricted classes of loops. Rather than generating invariants for given loops, in this paper we synthesise loops that exhibit a predefined behaviour given by an invariant. From the perspective of formal loop verification, the synthesised loops are thus correct by design and no longer need to be verified.

To overcome the hardness of reasoning with arbitrarily strong invariants, in this paper we construct simple (non-nested) **while** loops with linear updates that exhibit polynomial equality invariants. Rather than solving arbitrary polynomial equations, we consider loop properties defined by a single quadratic invariant in any number of variables. We present a procedure that, given a quadratic equation, decides whether a loop with affine updates satisfying this equation exists. Furthermore, if the answer is positive, the procedure synthesises a loop and ensures its variables achieve infinitely many different values.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Computing methodologies → Equation and inequality solving algorithms; Computing methodologies → Algebraic algorithms

Keywords and phrases program synthesis, loop invariants, verification, Diophantine equations

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.41

Related Version *Extended Version*: <https://doi.org/10.48550/arXiv.2310.05120> [15]

Funding The project leading to this publication has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101034440. The project was also partially supported by the ERC consolidator grant ARTIST 101002685 and by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT19018].



1 Introduction

Linear loops, in their simplicity, constitute a convenient and yet expressive model. From an algebraic point of view, a linear loop corresponds to a system of recurrence relations; solutions of such systems form a robust class in algorithmic combinatorics and algebraic number theory [12, 20]. Linear loops are particularly common in control and digital signal processing software [19]. Note also that the problem of studying the functional behaviour of affine loops (loops with update polynomials of degree 1) can be reduced to that of studying linear loops [28]. Moreover, linear loops can be used to overapproximate the behaviour of more expressive numerical programs, including those with unrestricted control flow and recursive procedures [22].



© S. Hitarth, George Kenison, Laura Kovács, and Anton Varonka;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 41; pp. 41:1–41:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Loop Invariants. While variable updates of linear loops are restricted to linear assignments, it is quite common that linear loops exhibit intricate polynomial properties in the form of polynomial invariants. Non-linear polynomial invariant assertions might come in handy for the verification of safety properties; by approximating the program’s behaviour more accurately, they admit fewer false positives. That is, a program verifier using polynomial loop invariants infers less frequently that a true assertion can be violated [7].

Loop Synthesis. Generating invariants, in particular polynomial invariants, is a notorious task, shown to be undecidable for loops with arbitrary polynomial arithmetic [16]. Rather than generating invariants for loops, *in this paper we work in the reverse direction: generating loops from invariants*. Thus we ensure that the constructed loops exhibit intended invariant properties and are thus correct by design. Loop synthesis therefore provides an alternative approach for proving program correctness. If intermediate assertions of an involved program are written in terms of polynomial equalities, automated loop synthesis can provide a code fragment satisfying that assertion, while being correct by construction with respect to the specification.

To overcome hardness of polynomial reasoning and solving arbitrary polynomial equations, we restrict our attention to linear loops, and provide a decision procedure for computing linear loops from (quadratic) polynomial invariants (Algorithm 1).

Linear loop synthesis showcases how a simple model (a linear loop) can express complicated behaviours (quadratic invariants), as also witnessed in sampling algorithms of real algebraic geometry [2, 11]. A non-trivial linear loop for a polynomial invariant allows to sample infinitely many points from the algebraic variety defined by the polynomial. Moreover, the computational cost to generate a new sample point only involves a matrix-vector multiplication. We give further comment on why we do not accept trivial loops in the synthesis process in Remark 2.8.

Thus the result of a loop synthesis process for a polynomial equation (invariant) is an infinite family of solutions defined by recurrence relations. This family is parameterised by n , the number of loop iterations: n th terms of the synthesised recurrence sequences yield a solution of the polynomial equation. Whether the solution set of an equation admits a parameterisation of a certain kind is, in general, an open problem [34, 36].

Our Contributions. The main contributions of this work are as follows:

1. We present a procedure that, given a quadratic equation $P(x_1, \dots, x_d) = 0$ with an arbitrary number of variables and rational coefficients, generates an affine loop such that $P = 0$ is invariant under its execution; i.e., the equality holds after any number of loop iterations. If such a loop does not exist, the procedure returns a negative answer. The values of the loop variables are rational. Moreover, the state spaces of the loops synthesised by this procedure are infinite and, notably, the same valuation of loop variables is never reached twice. The correctness of this procedure is established in Theorem 5.4.
2. If the equation $Q(x_1, \dots, x_d) = c$ under consideration is such that Q is a quadratic form, we present a stronger result: a procedure (Algorithm 1) that generates a *linear* loop with d variables satisfying the invariant equation.

Paper Outline. Section 2 introduces relevant preliminary material. We defer the discussion of *polynomial equation solving*, a key element of loop synthesis, to Section 3. Then, in Section 4, we provide a method to synthesise *linear loops* for invariants, where the invariants restricted to be equations with *quadratic forms*. We extend these results in Section 5 and

present a procedure that synthesises *affine loops*, and hence also linear loops, for invariants that are *arbitrary quadratic* equations. We discuss aspects of our approach and propose further directions in Section 6, in relation to known results.

An extended version of this paper, containing further details on our approach, is available online [15]. In Appendix A of [15], we summarise the procedure for finding isotropic solutions to quadratic forms (which we employ in our synthesis procedure). The abstract arithmetic techniques contained therein are beyond the scope of this short paper and detail the contributions of many sources [18, 9, 33, 32, 26, 6]. In Appendix B of [15], we summarise the synthesis procedure underlying Theorem 5.4.

2 Preliminaries

2.1 Linear and quadratic forms

► **Definition 2.1** (Quadratic form). *A d -ary quadratic form over the field \mathbb{K} is a homogeneous polynomial of degree 2 with d variables:*

$$Q(x_1, \dots, x_d) = \sum_{i \leq j} c_{ij} x_i x_j,$$

where $c_{ij} \in \mathbb{K}$. It is convenient to associate a quadratic form Q with the symmetric matrix:

$$A_Q := \begin{pmatrix} c_{11} & \frac{1}{2}c_{12} & \dots & \frac{1}{2}c_{1d} \\ \frac{1}{2}c_{12} & c_{22} & \dots & \frac{1}{2}c_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{2}c_{1d} & \frac{1}{2}c_{2d} & \dots & c_{dd} \end{pmatrix}.$$

We note that since A_Q is symmetric, its eigenvalues are all real-valued. Further, $Q(\mathbf{x}) = \mathbf{x}^\top A_Q \mathbf{x}$ for a vector $\mathbf{x} = (x_1, \dots, x_d)$ of variables.

We consider quadratic forms over the field \mathbb{Q} of rational numbers by default. Therefore, a quadratic form has a rational quadratic matrix associated with it.

A quadratic form Q is *non-degenerate* if its matrix A_Q is not singular; that is, $\det A_Q \neq 0$. A quadratic form Q over \mathbb{Q} *represents* the value $a \in \mathbb{Q}$ if there exists a vector $\mathbf{x} \in \mathbb{Q}^d$ such that $Q(\mathbf{x}) = a$. A quadratic form Q over \mathbb{Q} is called *isotropic* if it represents 0 non-trivially; i.e., there exists a non-zero vector $\mathbf{x} \in \mathbb{Q}^d$ with $Q(\mathbf{x}) = 0$. The vector itself is then called *isotropic*. If no isotropic vector exists, the form is *anisotropic*. A quadratic form Q is called *positive* (resp. *negative*) *definite* if $Q(\mathbf{x}) > 0$ (resp. $Q(\mathbf{x}) < 0$) for all $\mathbf{x} \neq \mathbf{0}$. Note that definite forms are necessarily anisotropic.

► **Definition 2.2.** *Let Q_1 and Q_2 be d -ary quadratic forms. The forms Q_1 and Q_2 are equivalent, denoted by $Q_1 \sim Q_2$, if there exists $\sigma \in \text{GL}_d(\mathbb{Q})$ such that $Q_2(\mathbf{x}) = Q_1(\sigma \cdot \mathbf{x})$.*

From the preceding definition, there exists an (invertible) linear change of variables over \mathbb{Q} under which representations by Q_2 are mapped to the representations by Q_1 . It is clear that two equivalent quadratic forms represent the same values. In terms of matrices, we have $(\sigma \mathbf{x})^\top A_{Q_1} \sigma \mathbf{x} = \mathbf{x}^\top A_{Q_2} \mathbf{x}$, and hence $A_{Q_2} = \sigma^\top A_{Q_1} \sigma$.

► **Definition 2.3** (Linear form). *A linear form in d variables over the field \mathbb{Q} is a homogeneous polynomial $L(x_1, \dots, x_d) = \sum_{i=1}^d b_i x_i$ of degree 1, where $b_1, \dots, b_d \in \mathbb{Q}$.*

Note that each linear form admits a vector interpretation: $L(\mathbf{x}) = \mathbf{b}^\top \mathbf{x}$, where $\mathbf{b} = (b_1, \dots, b_d)^\top \in \mathbb{Q}^d$ is a non-zero vector of the linear form.

2.2 Loops and Loop Synthesis

Linear loops are a class of single-path loops whose update assignments are determined by a homogeneous system of linear equations in the program variables.

► **Definition 2.4** (Linear loop). *A linear loop $\langle M, \mathbf{s} \rangle$ is a loop program of the form*

$$\mathbf{x} \leftarrow \mathbf{s}; \text{ while } \star \text{ do } \mathbf{x} \leftarrow M\mathbf{x},$$

where \mathbf{x} is a d -dimensional column vector of program variables, \mathbf{s} is an *initial* d -dimensional vector, and M is a $d \times d$ update matrix. For the procedures, which we introduce here, to be effective, we assume that the entries of M and \mathbf{s} are rational.

We employ the notation \star , instead of using `true` as loop guard, as our focus is on loop synthesis rather than proving loop termination.

► **Definition 2.5** (Affine loop). *An affine loop $\langle M, \mathbf{s}, \mathbf{t} \rangle$ is a loop program of the form*

$$\mathbf{x} \leftarrow \mathbf{s}; \text{ while } \star \text{ do } \mathbf{x} \leftarrow M\mathbf{x} + \mathbf{t},$$

where, in addition to the previous definition, $\mathbf{t} \in \mathbb{Q}^d$ is a translation vector.

► **Remark 2.6** (Linear and Affine Loops). A standard observation permits the simulation of affine loops by linear ones at a cost of one additional variable constantly set to 1. An *augmented matrix* of an affine loop with d variables is a matrix $M' \in \mathbb{Q}^{(d+1) \times (d+1)}$ of the form

$$M' := \left(\begin{array}{c|c} 1 & 0_{1,d} \\ \hline \mathbf{t} & M \end{array} \right).$$

It follows that a linear loop $\langle M', (1, \mathbf{s})^\top \rangle$ simulates the affine loop in its last d variables.

A linear (or affine) loop with variables $\mathbf{x} = (x_1, \dots, x_d)$ generates d sequences of numbers. For each loop variable x_j , let $\langle x_j(n) \rangle_{n=0}^\infty \subseteq \mathbb{Q}$ denote the sequence whose n th term is given by the value of x_j after the n th loop iteration. Similarly, define the sequence of vectors $\langle \mathbf{x}(n) \rangle_n \subseteq \mathbb{Q}^d$. For a given loop, we refer to its reachable set of states in \mathbb{Q}^d as the loop's *orbit*. A loop with variables x_1, \dots, x_d is *non-trivial* if the orbit

$$\mathcal{O}_{\mathbf{x}} := \{(x_1(n), \dots, x_d(n)) : n \geq 0\} \subseteq \mathbb{Q}^d$$

is infinite. A *polynomial invariant* of a loop is a polynomial $P \in \mathbb{Q}[\mathbf{x}]$ such that

$$P(x_1(n), \dots, x_d(n)) = 0$$

holds for all $n \geq 0$.

► **Problem 2.7** (Loop Synthesis). *Given a polynomial invariant $P \in \mathbb{Q}[x_1, \dots, x_d]$, find a non-trivial linear (affine) loop with vector sequence $\langle \mathbf{x}(n) \rangle_n$ such that*

$$P(x_1(n), \dots, x_d(n)) = 0$$

holds for any $n \geq 0$.

We emphasise that, unless stated otherwise, the objective of the loop synthesis process from Problem 2.7 is to find a loop with the same number of variables d as in the input invariant. That is, $\langle \mathbf{x}(n) \rangle_n = (\langle x_1(n) \rangle_n, \dots, \langle x_d(n) \rangle_n)$

Note that $P = 0$ in Problem 2.7 does not need to be an *inductive invariant* for the synthesised loop: We do not require the matrix M to preserve the equality for all vectors \mathbf{x} . There might still exist a vector \mathbf{s}' such that $P(\mathbf{s}') = 0$ but $P(M \cdot \mathbf{s}') \neq 0$. Observe that the search space only expands when we allow non-inductive invariants, thus making our loop synthesis procedures more general.

In summary, the search for an update matrix M (or the augmented matrix M' in the affine loop version of Problem 2.7), is integrally linked to the search of \mathbf{s} , a solution of the polynomial $P = 0$.

► **Remark 2.8 (Loop Synthesis and Polynomial Equation Solving).** We note that Problem 2.7, Loop Synthesis, relies on, but it is not equivalent to, solving polynomial equations. Indeed, we focus on *non-trivial* loops in Problem 2.7. Allowing loops with finite orbits would mean that a loop with an identity matrix update I_d is accepted as a solution:

$$\mathbf{x} \leftarrow \mathbf{s}; \text{ while } \star \text{ do } \mathbf{x} \leftarrow I_d \cdot \mathbf{x}.$$

Then, the loop synthesis problem would be equivalent to the problem of finding a rational solution of a polynomial equation $P = 0$ (see Problem 3.1). The problem, as we define it in Problem 2.7, neglects loops that satisfy a desired invariant but reach the same valuation of variables twice. Due to this, the Problem 2.7 of loop synthesis is different from the Problem 3.1 of solving polynomial equations.

3 Solving Quadratic Equations

As showcased in Problem 2.7 and discussed in Remark 2.8, loop synthesis for a polynomial invariant $P = 0$ is closely related to the problem of solving a polynomial equation $P = 0$.

► **Problem 3.1 (Solving Polynomial Equations).** *Given a polynomial $P \in \mathbb{Q}[x_1, \dots, x_d]$, decide whether there exists a rational solution $(s_1, \dots, s_d) \in \mathbb{Q}^d$ to the equation $P(x_1, \dots, x_d) = 0$.*

We emphasise that determining whether a given polynomial equation has a rational solution, is a fundamental open problem in number theory [29], see also Section 6.1.

Clearly, this poses challenges to our investigations of loops satisfying arbitrary polynomial invariants. In light of this, it is natural to restrict Problem 2.7 to loop invariants given by *quadratic* equations. Given a single equation $P(\mathbf{x}) = 0$ of degree 2, the challenge from now on is to find a rational solution \mathbf{s} and an update matrix M such that iterative application of M to \mathbf{s} of the equation does not violate the invariant: $P(M^n \mathbf{s}) = 0$ for all $n \geq 0$.

In this section, we recall well-known methods for solving quadratic equations. In the sequel, we will employ said methods in the novel setting of loop synthesis for quadratic polynomial invariants (Sections 4 and 5).

► **Problem 3.2 (Solving Quadratic Equations).** *Given a quadratic equation in d variables with rational coefficients, decide whether it has rational solutions. If it does, generate one of the solutions.*

3.1 Solutions of Quadratic Equations in Two Variables

We first prove two lemmas that discuss the solutions of binary quadratic forms in preparation for Section 4.

► **Lemma 3.3.** *For all $a, b \in \mathbb{Q} \setminus \{0\}$, Pell's equation $x^2 + \frac{b}{a}y^2 = 1$ has a rational solution (α, β) with $\alpha \notin \{\pm 1, \pm \frac{1}{2}, 0\}$ and $\beta \neq 0$.*

Proof. So long as $a \neq -b$, it is easy to see that $\left(\frac{b-a}{a+b}, \frac{2a}{a+b}\right)$ is a rational solution to Pell's equation. Recall that $a \neq 0$, hence $\beta \neq 0$ and $\alpha \neq \pm 1$. However, the generic solution might have $\alpha = 0$ or $|\alpha| = \frac{1}{2}$. We thus explicitly pick alternative solutions for the cases when it occurs: (i) $x^2 + y^2 = 1$ has another rational point, e.g., $(\frac{3}{5}, \frac{4}{5})$; (ii) $x^2 + 3y^2 = 1$ has a rational point $(-\frac{11}{13}, \frac{4}{13})$; (iii) $x^2 + \frac{1}{3}y^2 = 1$ has a rational point $(\frac{1}{7}, \frac{12}{7})$.

Finally, if $a = -b$, we can take a rational point $(\frac{5}{3}, \frac{4}{3})$ on the hyperbola $x^2 - y^2 = 1$. ◀

► **Lemma 3.4.** *An equation $ax^2 + by^2 = c$ with $a, b \in \mathbb{Q} \setminus 0$ has either no rational solutions different from $(0, 0)$, or infinitely many rational solutions different from $(0, 0)$.*

Proof. Define $R := \begin{pmatrix} \alpha & -\frac{b}{a}\beta \\ \beta & \alpha \end{pmatrix}$ where $(\alpha, \beta) \in \mathbb{Q}^2 \setminus \mathbf{0}$ satisfies $\alpha^2 + \frac{b}{a}\beta^2 = 1$ (is a solution to Pell's equation) for which $\alpha \notin \{\pm 1, \pm \frac{1}{2}, 0\}$ (as in Lemma 3.3). What follows can be viewed as an application of the multiplication principle for the generalised Pell's equation [1]. Observe that if $\mathbf{v} = (x, y)^\top$ is a solution to $ax^2 + by^2 = c$, then so is $R\mathbf{v}$.

We now show how to generate infinitely many rational solutions to $ax^2 + by^2 = c$ from a single rational solution. Assume, towards a contradiction, that $R^{n+k}\mathbf{v} = R^n\mathbf{v}$ holds for some $n \geq 0, k \geq 1$. Therefore, there exists an integer k such that 1 is an eigenvalue of R^k . Equivalently, there exists a root of unity ω which is an eigenvalue of R . We proceed under this assumption.

By construction, the eigenvalues of R are ω and ω^{-1} . Let φ be the argument of ω . Then the real part of ω , $\cos(\varphi)$, is equal to α (and thus rational). Since ω is a root of unity, φ is a rational multiple of 2π . By Niven's theorem [27], the only rational values for $\cos(\varphi)$ are 0, $\pm \frac{1}{2}$ and ± 1 . We arrive at a contradiction, as α was carefully picked to avoid these values.

In summary, we have shown that R has no eigenvalues that are roots of unity, from which we deduce the desired result. ◀

3.2 Solving Isotropic Quadratic Forms

We next present an approach to solving Problem 3.2 that uses the theory of representations of quadratic forms. First, we prove a lemma concerning the representations of 0.

► **Lemma 3.5.** *Let $Q(x_1, \dots, x_n) = a_1x_1^2 + \dots + a_nx_n^2$ be an isotropic quadratic form with $a_1, \dots, a_n \neq 0$. There exists a representation $(\alpha_1, \dots, \alpha_n)$ of 0; i.e., $a_1\alpha_1^2 + \dots + a_n\alpha_n^2 = 0$ such that $\alpha_1, \dots, \alpha_n \neq 0$.*

Proof. Let $(\beta_1, \dots, \beta_n) \in \mathbb{Q}^n$ be a representation of 0 by Q . We further assume that $\beta_1, \dots, \beta_r \neq 0$ while $\beta_{r+1} = \dots = \beta_n = 0$, and $r < n$. Moreover, let $\lambda := a_r\beta_r^2 + a_{r+1}\beta_{r+1}^2$.

Consider the equation $x^2 + \frac{a_{r+1}}{a_r}y^2 = 1$. From Lemma 3.3, it has a rational solution (α, β) such that $\alpha, \beta \neq 0$. This implies $a_r\alpha^2 + a_{r+1}\beta^2 = a_r$. The pair $(\beta_r, 0)$ is one solution to $a_rx_r^2 + a_{r+1}x_{r+1}^2 = \lambda$. Following the steps in the proof of Lemma 3.4, we can construct a matrix R for which $R \cdot (\beta_r, 0)^\top = (\alpha\beta_r, \beta\beta_r)^\top$ where $(\alpha\beta_r, \beta\beta_r)$ is a solution of $a_rx_r^2 + a_{r+1}x_{r+1}^2 = \lambda$ with both components being non-zero. Therefore, $(\beta_1, \dots, \beta_{r-1}, \alpha\beta_r, \beta\beta_r, \beta_{r+2}, \dots, \beta_n)$ is an isotropic vector of Q with fewer zero entries. By repeating the process, we obtain an isotropic vector $(\alpha_1, \dots, \alpha_n)$ as desired. ◀

We emphasise that the process of eliminating zeros from the isotropic vector is effective. A similar proof is given in [3, p.294, Theorem 8].

In this discussion, we focus on solving equations of the form $Q(x_1, \dots, x_d) = c$, where Q is a quadratic form. As it will be shown later in Section 4, it is always possible to find an equivalent diagonal quadratic form $D \sim Q$. Therefore, we restrict our attention to equations

of the form $a_1x_1^2 + \dots + a_dx_d^2 = c$. Assuming $c \neq 0$, we start by homogenising the equation, and so consider the solutions of

$$a_1x_1^2 + \dots + a_dx_d^2 - cx_{d+1}^2 = 0. \quad (1)$$

In other words, we are searching for a rational isotropic vector of a quadratic form.

► **Proposition 3.6.** *An equation*

$$a_1x_1^2 + \dots + a_dx_d^2 = c \quad (2)$$

has a rational solution different from $(0, \dots, 0)$ if and only if the quadratic form $Q = a_1x_1^2 + \dots + a_dx_d^2 - cx_{d+1}^2$ has an isotropic vector.

Proof. For $c = 0$, the statement is a recitation of a definition. We continue under the assumption $c \neq 0$. Recall from Lemma 3.5 that if the form Q is isotropic, then there is an isotropic vector $(\alpha_1, \dots, \alpha_{d+1})$ with $\alpha_i \neq 0$ for all $i \in \{1, \dots, d+1\}$. Therefore, we can find a non-zero solution $(\alpha_1/\alpha_{d+1}, \dots, \alpha_d/\alpha_{d+1})$ to Equation (2). Conversely, if (2) has a non-trivial solution $(\beta_1, \dots, \beta_d)$, it follows that $(\beta_1, \dots, \beta_d, 1)$ is an isotropic vector for Q . ◀

3.3 Finding Isotropic Vectors

Proposition 3.6 implies that solving Problem 3.1, and hence also loop synthesis in Problem 2.7, requires detecting whether a certain quadratic form is isotropic. Effective isotropy tests are known for quadratic forms $Q(x_1, \dots, x_{d+1})$ as in Equation (1). A more difficult task is the problem of finding an isotropic vector for such a form.

The abstract arithmetic techniques employed in finding an isotropic vector are beyond the scope of this paper; however, we give a brief overview of the computational task and a number of references to the literature in the extended version [15, Appendix A]. Our takeaways from the theory, summarised there [18, 9, 33, 32, 26, 6], are the following functions:

- **ISISOTROPIC**: a function that, given an indefinite quadratic form over the rationals as an input, determines whether the input is isotropic and duly returns the answers YES and NO (as appropriate).
- **FINDISOTROPIC**: a function that accepts isotropic quadratic forms over the rationals as inputs and returns an isotropic vector for each such form.
- **SOLVE**: a function that takes Equation (2) as an input and returns a non-zero solution if the form $a_1x_1^2 + \dots + a_dx_d^2 - cx_{d+1}^2$ is isotropic; otherwise SOLVE returns “NO SOLUTIONS”. The function SOLVE calls both ISISOTROPIC and FINDISOTROPIC, see [15] for details.

We note the SOLVE subroutine in the sequel: the function LINLOOP defined in Algorithm 1, calls on SOLVE; and, in turn, the function LINLOOP is called by the procedure in Section 5.

4 Quadratic Forms: Linear Loops

The core of this section addresses equations, and hence loop invariants, that involve quadratic forms. The equations (invariants) of this section do not have a linear part; they are quadratic forms equated to constants; that is, equations of the form

$$Q(x_1, \dots, x_d) = c, \quad (3)$$

where Q is an arbitrary d -ary quadratic form with rational coefficients, c is a rational number.

The main result of this section is the following theorem, which establishes a decision procedure that can determine if a given quadratic invariant admits a linear loop and, if so, constructs that loop.

► **Theorem 4.1** (Linear Loops for Quadratic Forms). *There exists a procedure that, given an equation $Q(x_1, \dots, x_d) = c$ of the form (3), decides whether a non-trivial linear loop satisfying $Q(x_1, \dots, x_d) = c$ exists and, if so, synthesises a loop.*

We prove Theorem 4.1 in several steps. The first of them is to diagonalise the quadratic form Q and thus reduce to Equation (3) without mixed terms on the left-hand side.

4.1 Rational Diagonalisation

A rational quadratic form can be diagonalised by an invertible change of variables with only rational coefficients.

► **Proposition 4.2.** *Let Q be a (possibly degenerate) d -ary quadratic form. There exists an equivalent quadratic form D with a diagonal matrix $A_D \in \mathbb{Q}^{d \times d}$, i.e., $Q \sim D$. Furthermore, $A_D = \sigma^T A_Q \sigma$ holds with $\sigma \in \text{GL}_d(\mathbb{Q})$.*

A diagonalisation algorithm is described in [23, Algorithm 12.1], see also “diagonalisation using row/column operations” in [35, Chapter 7, 2.2]. The idea, as presented in [35], is to perform row operations on the matrix Q . Different from the usual Gauss–Jordan elimination, the analogous column operations are performed after each row operation. We emphasise that the change-of-basis matrix σ is invertible as a product of elementary matrices.

► **Remark 4.3** (Degeneracy). Let $A_D := \text{diag}(a_1, \dots, a_d)$ be the diagonal matrix of the quadratic form D as in Proposition 4.2. The product $a_1 \cdots a_d$ is zero if and only if the initial quadratic form Q is degenerate.

► **Proposition 4.4.** *Let Q_1 and Q_2 be two equivalent d -ary quadratic forms. If there exists a linear loop $\mathcal{L} = \langle M, \mathbf{s} \rangle$ with invariant $Q_2 = c$ for a constant $c \in \mathbb{Q}$, then $Q_1 = c$ is an invariant of the linear loop $\mathcal{L}' = \langle \sigma M \sigma^{-1}, \sigma \mathbf{s} \rangle$. Here, $\sigma \in \text{GL}_d(\mathbb{Q})$ is a change-of-basis matrix such that $Q_2(\mathbf{x}) = Q_1(\sigma \cdot \mathbf{x})$.*

Proof. If $(M^n \mathbf{s})^T A_{Q_2} (M^n \mathbf{s}) = c$ for all $n \geq 0$, then

$$\begin{aligned} ((\sigma M \sigma^{-1})^n \sigma \mathbf{s})^T A_{Q_1} ((\sigma M \sigma^{-1})^n \sigma \mathbf{s}) &= (\sigma M^n \mathbf{s})^T A_{Q_1} (M^n \mathbf{s}) \\ &= \mathbf{s}^T (M^n)^T \sigma^T A_{Q_1} \sigma M^n \mathbf{s} = \mathbf{s}^T (M^n)^T A_{Q_2} M^n \mathbf{s} = (M^n \mathbf{s})^T A_{Q_2} M^n \mathbf{s} = c \end{aligned}$$

for all $n \geq 0$ as well. We emphasise that σ is a bijection from \mathbb{Q}^d to itself, so the reduction described here preserves the infiniteness of loop orbits. ◀

We conclude from Propositions 4.2 and 4.4 that for a general quadratic form Q , a linear loop with an invariant $Q(\mathbf{x}) = c$ exists if and only if a linear loop exists for an invariant $D(\mathbf{x}) = c$, where D is an equivalent diagonal form.

4.2 Diagonal Quadratic Forms

In this subsection we consider diagonal quadratic forms $a_1 x_1^2 + \cdots + a_d x_d^2 = c$, where $a_1, \dots, a_d, c \in \mathbb{Q}$ as in Equation (2). If the equation is homogeneous; that is, $c = 0$, then loop synthesis reduces to the problem of searching for a rational solution $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_d)$. Indeed, a loop with a matrix $\lambda \cdot I_d$ (scaling each variable by $\lambda \in \mathbb{Q} \setminus \{-1, 0, 1\}$) and the initial vector $\boldsymbol{\alpha}$ is a non-trivial linear loop satisfying the invariant $Q(\mathbf{x}) = 0$.

From Section 3, we know how to generate a solution (or prove there is no solution) to Equation (2) in its general form, also with $c \neq 0$. The bottleneck of loop synthesis in Problem 2.7 is thus finding an update matrix M for the linear loop. En route to solving this issue, we state the following corollary of Lemma 3.4.

► **Corollary 4.5.** *If an equation $ax^2 + by^2 = c$ with $a, b \in \mathbb{Q} \setminus 0$ has infinitely many rational solutions different from $(0, 0)$, then there exists a non-trivial linear loop with polynomial invariant $ax^2 + by^2 = c$.*

Proof. We use the construction in the proof of Lemma 3.4, which demonstrates that the orbit of the linear loop $\langle R, \mathbf{v} \rangle$ is infinite with polynomial invariant $ax^2 + by^2 = c$. ◀

Proof of Theorem 4.1. Due to Proposition 4.4, we can consider an equation of the form (2):

$$a_1x_1^2 + \cdots + a_dx_d^2 = c.$$

We describe the loop synthesis procedure in this case. If $d = 1$, the equation only has finitely many solutions, hence any loop for Equation (2) is trivial. Hereafter we assume that $d \geq 2$.

In order to generate an initial vector of the loop for Equation (2), we exploit the results of Section 3. Either Equation (2) has no rational solutions and hence no loop exists, or we effectively construct a solution $\alpha = (\alpha_1, \dots, \alpha_d) \in \mathbb{Q}^d$ using procedure SOLVE. Recall that we can guarantee $\alpha_i \neq 0$ for all $i \in \{1, \dots, d\}$ due to Lemma 3.5.

Note that some of the coefficients a_i , $i \in \{1, \dots, d\}$, may be zero if the original quadratic form Q is degenerate. We have to consider the case when all coefficients but one are 0, separately. That is, $a_1x_1^2 + 0x_2^2 + \cdots + 0x_d^2 = c$. For this form, a solution exists if and only if c/a_1 is a square of a rational number. Subsequently, if a solution α is found, set $M := \text{diag}(1, 2, \dots, 2)$ to be a diagonal update matrix. Since $d \geq 2$, we guarantee that the orbit of the linear loop $\langle M, \alpha \rangle$ is infinite.

Without loss of generality, we now assume $a_1 \neq 0$ and $a_2 \neq 0$. Define $\gamma := a_1\alpha_1^2 + a_2\alpha_2^2$, then the equation $a_1x_1^2 + a_2x_2^2 = \gamma$ has a non-trivial solution (α_1, α_2) .

From Corollary 4.5, there exists a matrix $R \in \mathbb{Q}^{2 \times 2}$ that preserves the value of the quadratic form $a_1x_1^2 + a_2x_2^2$. This matrix can be constructed as in the proof of Corollary 4.5 by considering the equation $x_1^2 + \frac{a_2}{a_1}x_2^2 = 1$. Let M be the matrix given by the direct sum

$$R \oplus I_{d-2} = \left(\begin{array}{c|c} R & 0 \\ \hline 0 & I_{d-2} \end{array} \right)$$

where I_n is an identity matrix of size n .

A desired loop is (M, α) as for each $n \geq 0$, $M^n \alpha$ satisfies Equation (2). The loop is non-trivial because its orbit, restricted to x_1, x_2 , is infinite. ◀

The process of synthesising a loop for the quadratic invariant $Q(x_1, \dots, x_n) = c$ is summarised in Algorithm 1. The algorithm starts with a diagonalisation step, proceeds with finding a loop for an equation of the form (2), and applies the inverse transformation to obtain a linear loop for the initial invariant. Whenever Algorithm 1 returns a loop, this loop is linear.

5 Arbitrary Quadratic Equations: Affine Loops

In this section, we leave the realm of quadratic forms and consider general quadratic invariants that may have a linear part. Any quadratic equation can be written in terms of a quadratic form Q , a linear form L , and a constant term c :

$$Q(x_1, \dots, x_d) + L(x_1, \dots, x_d) = c. \quad (4)$$

On our way to a complete solution of Problem 2.7 for arbitrary quadratic equations, we carefully analyse Equation (4). A standard technique (see e.g. [14, Proposition 1]) allows to reduce Equation (4) with a non-degenerate quadratic form Q to Equation (3) considered

41:10 Linear Loop Synthesis for Quadratic Invariants

■ **Algorithm 1** Synthesise a linear loop satisfying a given quadratic form equation.

Input: quadratic form Q in d variables and $c \in \mathbb{Q}$. Assert $d \geq 2$.

```

1: function LINLOOP( $Q, s$ )
2:    $\langle M, s \rangle :=$  UNDEFINED.
3:   compute a rational diagonalisation for  $Q(\mathbf{x})$ : a  $\sigma \in \text{GL}_d(\mathbb{Q})$  such that  $Q'(\sigma\mathbf{x}) = Q(\mathbf{x})$ 
   with  $Q'$  a diagonal quadratic form
4:   rewrite the equation  $Q' = c$  as  $a_1x_1^2 + \dots + a_dx_d^2 = c$  with  $a_1, \dots, a_r \neq 0$  and
    $a_{r+1} = \dots = a_d = 0$ 
5:   let  $\alpha := (\alpha_1, \dots, \alpha_r, 1, \dots, 1)^\top \in \mathbb{Q}^d$ , where  $(\alpha_1, \dots, \alpha_r) := \text{SOLVE}(a_1, \dots, a_r, c)$ 
   ▷ for SOLVE see [15, Algorithm 2]
6:   if  $r = 1$  and  $\alpha \neq$  “NO SOLUTIONS” then
7:      $M := \text{diag}(1, 2, \dots, 2)$ .
8:   else if  $\alpha =$  “NO SOLUTIONS” then
9:     return “NO LOOP”.
10:  else
11:    compute a solution  $(y_1, y_2)$  of  $x_1^2 + \frac{a_2}{a_1}x_2^2 = 1$ . ▷ see Lemma 3.3
12:     $M := R \oplus I_{d-2}$ , where  $R = \begin{pmatrix} y_1 & -\frac{a_2}{a_1}y_2 \\ y_2 & y_1 \end{pmatrix}$ .
13:  end if
14:  return  $\langle \sigma^{-1}M\sigma, \sigma^{-1}\alpha \rangle$ .
15: end function

```

in Section 4. We now give the details of this reduction and describe how to synthesise an affine loop for an invariant (4) in the non-degenerate case. Subsequently, we close the gap by discussing the case when Q is degenerate. Using Remark 2.6, our results on affine loop synthesis imply then linear loop synthesis.

5.1 Non-Degenerate Quadratic Forms

For convenience, we rewrite the equation in the matrix-vector form: $\mathbf{x}^\top A_Q \mathbf{x} + \mathbf{b}^\top \mathbf{x} - c = 0$. Here, A_Q is the non-singular matrix of the quadratic form Q , and \mathbf{b} is the vector of the linear form. Let $\delta := \det A_Q \neq 0$ and C be the cofactor matrix of A_Q , i.e., $A_Q \cdot C = C \cdot A_Q = \delta \cdot I_d$. We further define $\mathbf{h} := C \cdot \mathbf{b}$ and $\tilde{c} = 4\delta^2 c + Q(\mathbf{h})$. It can be checked directly that

$$Q(2\delta \cdot \mathbf{x} + \mathbf{h}) = \tilde{c} \Leftrightarrow Q(\mathbf{x}) + L(\mathbf{x}) = c. \quad (5)$$

In words, every equation of the form Equation (4) can be reduced to an equation of the form $Q(\mathbf{y}) = \tilde{c}$ by an affine transformation f that maps each $\mathbf{x} \in \mathbb{Q}^d$ to $2\delta \cdot \mathbf{x} + \mathbf{h} \in \mathbb{Q}^d$. As such, this means that solutions of Equation (4) under the non-degeneracy assumption are in a one-to-one correspondence with representations of \tilde{c} for Q .

► **Proposition 5.1.** *Let Q be a non-degenerate quadratic form and L a linear form, both in $d \geq 2$ variables. Define $\delta := \det(A_Q)$, \mathbf{h} and \tilde{c} , as in the discussion above. The following are equivalent:*

1. *There exists a linear loop $\langle M, \mathbf{s} \rangle$ satisfying the invariant $Q(\mathbf{x}) = \tilde{c}$.*
2. *There exists an affine loop*

$$\langle M, \frac{1}{2\delta}(\mathbf{s} - \mathbf{h}), \frac{1}{2\delta}(M - I_d)\mathbf{h} \rangle,$$

satisfying the invariant $Q(\mathbf{x}) + L(\mathbf{x}) = c$.

Proof. Start with the first assumption. For all $n \geq 0$, it holds $Q(M^n \mathbf{s}) = \tilde{c}$. Equivalently,

$$Q(f^{-1}(M^n \mathbf{s})) + L(f^{-1}(M^n \mathbf{s})) = c, \text{ or } Q\left(\frac{1}{2\delta}(M^n \mathbf{s} - \mathbf{h})\right) + L\left(\frac{1}{2\delta}(M^n \mathbf{s} - \mathbf{h})\right) = c$$

for all $n \geq 0$.

On the other hand, let $\mathbf{x}(n)$ be the variable vector after the n th iteration of an affine loop from the statement. We prove by induction that $\mathbf{x}(n) = \frac{1}{2\delta}(M^n \mathbf{s} - \mathbf{h})$. The base case is true since the initial vector of the affine loop is $\frac{1}{2\delta}(\mathbf{s} - \mathbf{h}) = \frac{1}{2\delta}(M^0 \mathbf{s} - \mathbf{h})$. Now, assume that $\mathbf{x}(k) = \frac{1}{2\delta}(M^k \mathbf{s} - \mathbf{h})$ for an arbitrary $k \geq 0$. Then, by applying the loop update once, we have

$$\begin{aligned} \mathbf{x}(k+1) &= M \cdot \left(\frac{1}{2\delta}(M^k \mathbf{s} - \mathbf{h})\right) + \frac{1}{2\delta}(M - I_d) \mathbf{h} \\ &= \frac{1}{2\delta}(M^{k+1} \mathbf{s} - M\mathbf{h} + M\mathbf{h} - \mathbf{h}) = \frac{1}{2\delta}(M^{k+1} \mathbf{s} - \mathbf{h}), \end{aligned}$$

and the inductive step has been shown. By the above work, we conclude that $Q(\mathbf{x}(n)) + L(\mathbf{x}(n)) = c$ holds for all $n \geq 0$. \blacktriangleleft

► **Example 5.2.** Consider an invariant $p(x, y) := x^2 + y^2 - 3x - y = 0$. After an affine change of coordinates $f(x, y) = (2x - 3, 2y - 1)$, it becomes $x^2 + y^2 = 10$ (that corresponds to $\delta = 1$, $\mathbf{h} = (-3, -1)^\top$, $\tilde{c} = 10$). There exists a linear loop for this equation:

$$M = \begin{pmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{pmatrix} \text{ and } \mathbf{s} = \begin{pmatrix} 1 \\ -3 \end{pmatrix}.$$

Next, compute the components of an affine loop. The update matrix is M , whilst the initial and translation vectors are

$$\frac{1}{2} \left[\begin{pmatrix} 1 \\ -3 \end{pmatrix} - \begin{pmatrix} -3 \\ -1 \end{pmatrix} \right] = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \quad \text{and} \quad \frac{1}{2} \left[\begin{pmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right] \begin{pmatrix} -3 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix},$$

respectively. The resulting affine loop is non-trivial with invariant $p(x, y) = 0$ due to Proposition 5.1:

$$\begin{pmatrix} x \\ y \end{pmatrix} \leftarrow \begin{pmatrix} 2 \\ -1 \end{pmatrix}; \text{ while } \star \text{ do } \begin{pmatrix} x \\ y \end{pmatrix} \leftarrow \begin{pmatrix} \frac{3}{5}x - \frac{4}{5}y + 1 \\ \frac{4}{5}x + \frac{3}{5}y - 1 \end{pmatrix}.$$

5.2 Degenerate Quadratic Forms

Let $r < d$ be the rank of A_Q . There exist $k := d - r$ linearly independent vectors $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{Q}^d$ such that $A_Q \cdot \mathbf{v}_i = \mathbf{0}$. Construct a matrix $\tau \in \text{GL}_d(\mathbb{Q})$ such that $\mathbf{v}_1, \dots, \mathbf{v}_k$ constitute its first columns. It follows that every non-zero entry $(M)_{ij}$ of a matrix $M := \tau^\top A_Q \tau$ is located in the bottom right corner, that is, $i > k$ and $j > k$. We rewrite $Q(\tau \mathbf{x}) = \tilde{Q}(x_{k+1}, \dots, x_d)$ and $L(\tau \mathbf{x}) = \tilde{L}(x_{k+1}, \dots, x_d) + \lambda_1 x_1 + \dots + \lambda_k x_k$ in Equation (4). Now we have:

$$\tilde{Q}(x_{k+1}, \dots, x_d) + \tilde{L}(x_{k+1}, \dots, x_d) = c - \lambda_1 x_1 - \dots - \lambda_k x_k, \tag{6}$$

where \tilde{Q} is a non-degenerate quadratic form of r variables.

In the rest of this subsection, we are concerned with finding an affine loop satisfying Equation (6). We emphasise that such a loop $\langle M, \mathbf{s}, \mathbf{t} \rangle$ exists if and only if $\langle \tau M \tau^{-1}, \tau \mathbf{s}, \tau \mathbf{t} \rangle$ satisfies Equation (4). The proof is due to τ inducing an automorphism of \mathbb{Q}^d , cf. Proposition 4.4.

41:12 Linear Loop Synthesis for Quadratic Invariants

If $\lambda_1 = \dots = \lambda_k = 0$, we have arrived at an instance of Equation (4) with a non-degenerate quadratic form and fewer variables. Let δ be the determinant of $\tilde{Q}(x_{k+1}, \dots, x_d)$ and, as in the non-degenerate setting, define an affine transformation f on the subset of variables $\{x_{k+1}, \dots, x_d\}$. The constant \tilde{c} and the vector $\mathbf{h} \in \mathbb{Q}^r$ are defined similarly to their non-degenerate setting counterparts.

After the change of coordinates that corresponds to f , we have

$$0x_1^2 + \dots + 0x_k^2 + \tilde{Q}(x_{k+1}, \dots, x_d) = \tilde{c}. \quad (7)$$

Recall (e.g. from the proof of Theorem 4.1) that once Equation (7) with $k \geq 1$ has a solution, there is a non-trivial linear loop satisfying the polynomial invariant defined by the equation. Now, let $\langle M, \mathbf{s} \rangle$ be a linear loop for Equation (7), where $\mathbf{s} = (s_1, \dots, s_d)^\top$. In fact, one can assume $M := \text{diag}(2, \dots, 2, 1, \dots, 1)$ with k twos and r ones. Define $\mathbf{s}' := \frac{1}{2\delta} (\mathbf{s} - \begin{pmatrix} 0 \\ \mathbf{h} \end{pmatrix})$. It is not hard to see that a non-trivial *linear* loop $\langle M, \mathbf{s}' \rangle$ satisfies $\tilde{Q}(x_{k+1}, \dots, x_d) + \tilde{L}(x_{k+1}, \dots, x_d) = c$ if and only if $\tilde{Q}(x_{k+1}, \dots, x_d) = \tilde{c}$ has a solution (s_{k+1}, \dots, s_d) .

From now on, we assume that $k \geq 1$ is the number of non-zero λ_i 's on the right-hand side of Equation (6). We show next that the loop synthesis question has a positive answer.

► **Proposition 5.3** (Affine Loops for Quadratic Forms). *Given a quadratic equation of the form (6), there exists a non-trivial affine loop in variables x_1, \dots, x_d for which said equation is a polynomial invariant.*

Proof. Since $k \geq 1$ and $\lambda_1 \neq 0$, the right-hand side $c - \sum_{i=1}^k \lambda_i x_i$ represents every rational number. Set the values of x_{k+1}, \dots, x_d to some fixed values $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_{d-k})$ such that $\boldsymbol{\alpha} \neq \mathbf{0}$ and solve the equation for x_1, \dots, x_k attaining a vector of values $\boldsymbol{\beta} = (\beta_1, \dots, \beta_k)$. We have $\tilde{Q}(\boldsymbol{\alpha}) + \tilde{L}(\boldsymbol{\alpha}) = A(\boldsymbol{\beta})$, where $A(x_1, \dots, x_k) := c - \lambda_1 x_1 - \dots - \lambda_k x_k$.

We introduce the following case distinction.

Case 1. $k > 1$;

Case 2. $r > 1$ and so \tilde{Q} is a non-degenerate quadratic form of at least 2 variables;

Case 3. $r = 1$ and $k = 1$; that is, Equation (6) has the form $ax^2 + bx = c - dy$, $d \neq 0$.

In the rest of the proof, we show that for all these cases, a non-trivial affine loop satisfies Equation (6) and hence, the invariant of Equation (4). Moreover, in Cases 1 and 2 there exist *linear* loops of this sort.

In Case 1, we focus on the vector $\boldsymbol{\beta}$ computed in the previous step. Without loss of generality, $(\beta_1, \beta_2) \neq (0, 0)$. We construct a linear loop that preserves the values of all variables but β_1, β_2 . To this end, it suffices to notice that a linear transformation of \mathbb{Q}^2 defined by $(x_1, x_2) \mapsto (2x_1, -\frac{\lambda_1}{\lambda_2}x_1 + x_2)$ preserves the value of $\lambda_1 x_1 + \lambda_2 x_2$. The desired linear loop has initial vector $\mathbf{s} = (\boldsymbol{\beta}, \boldsymbol{\alpha})^\top$ and an update matrix $M = \begin{pmatrix} 2 & 0 \\ -\frac{\lambda_1}{\lambda_2} & 1 \end{pmatrix} \oplus I_{d-2}$.

Let us turn to Case 2 and focus on vector $\boldsymbol{\alpha}$. Clearly, we can now assume $k = 1$. Without loss of generality, we shall assume that $\beta_1 \neq 0$. Consider the equation

$$\tilde{Q}(\mathbf{x}) + \tilde{L}(\mathbf{x}) = A(\beta_1)$$

over the variables \mathbf{x} , with $A(y) = c - \lambda_1 y$. Using Equation (5), we argue that its solutions are related to the representations of a certain number \tilde{c} by \tilde{Q} . We compute δ , \tilde{c} , \mathbf{h} for the non-degenerate quadratic form \tilde{Q} , linear form \tilde{L} and constant $A(\beta_1)$ such that $\tilde{Q}(2\delta \cdot \mathbf{x} + \mathbf{h}) = \tilde{c}$. From Theorem 4.1 and, more specifically, its proof, observe that there exists a non-trivial

linear loop satisfying $\tilde{Q}(\cdot) = \tilde{c}$. Indeed, there exists at least one solution of this equation, namely $f(\boldsymbol{\alpha})$. Let $\langle M, \mathbf{s} \rangle$ be a linear loop satisfying $\tilde{Q}(\cdot) = \tilde{c}$ with matrix $M \in \mathbb{Q}^{r \times r}$. Proposition 5.1 shows that an affine loop

$$\mathcal{A} := \langle M, \frac{1}{2\delta}(\mathbf{s} - \mathbf{h}), \frac{1}{2\delta}(M - I_r)\mathbf{h} \rangle,$$

satisfies the invariant $\tilde{Q}(\mathbf{x}) + \tilde{L}(\mathbf{x}) = A(\beta_1)$. The sequence $\langle \mathbf{x}(n) \rangle_{n=0}^\infty$ of \mathcal{A} 's variable vectors can be expressed in terms of an augmented matrix (see M' in Section 2) associated with the affine transformation $\mathbf{x} \mapsto M\mathbf{x} + \mathbf{t}$, where $\mathbf{t} = \frac{1}{2\delta}(M - I_r)\mathbf{h}$ and $\mathbf{s}' = \frac{1}{2\delta}(\mathbf{s} - \mathbf{h})$:

$$\begin{pmatrix} 1 \\ \mathbf{x}(n) \end{pmatrix} = \left(\begin{array}{c|c} 1 & 0_{1,r} \\ \mathbf{t} & M \end{array} \right)^n \begin{pmatrix} 1 \\ \mathbf{s}' \end{pmatrix},$$

satisfies $\tilde{Q}(\mathbf{x}) + \tilde{L}(\mathbf{x}) = A(\beta_1)$ for all $n \geq 0$. Then,

$$\begin{pmatrix} y(n) \\ \mathbf{x}(n) \end{pmatrix} = \left(\begin{array}{c|c} 1 & 0_{1,r} \\ \frac{1}{\beta_1}\mathbf{t} & M \end{array} \right)^n \begin{pmatrix} \beta_1 \\ \mathbf{s}' \end{pmatrix}$$

satisfies $\tilde{Q}(\mathbf{x}(n)) + \tilde{L}(\mathbf{x}(n)) = A(y(n))$ as in Equation (6) for all $n \geq 0$. We denote by M_β the d -dimensional square matrix in the preceding displayed equation. Observe that $\langle M_\beta, (\beta_1, \mathbf{s}')^\top \rangle$ is a linear loop satisfying the invariant of Equation (4).

Finally, we come to the special case, Case 3, that considers quadratic equations of the form $ax^2 + bx = c - dy$ where $d \neq 0$. It suffices to observe that an affine transformation of \mathbb{Q}^2 defined by $(x, y) \mapsto (2x, 2\frac{b}{d}x + 4y - 3\frac{c}{d})$ preserves the equation $ax^2 + bx = c - dy$. We conclude that $ax^2 + bx = c - dy$ is a polynomial invariant of the affine loop with initial vector $(1, \frac{c-a-b}{d})^\top$, translation vector $(0, -3\frac{c}{d})^\top$, and update matrix $\begin{pmatrix} 2 & 0 \\ 2\frac{b}{d} & 4 \end{pmatrix}$. ◀

5.3 The Procedure: Affine Loop Synthesis for Quadratic Invariants

► **Theorem 5.4** (Affine Loops for Quadratic Equations). *There exists an effective procedure that, given a quadratic equation (i.e. invariant)*

$$Q(x_1, \dots, x_d) + L(x_1, \dots, x_d) = c,$$

decides whether a non-trivial affine loop satisfying it exists and, if so, synthesises a loop.

The theorem is essentially proved in Propositions 5.1 and 5.3. If the quadratic form is non-degenerate, Proposition 5.1 reduces the search for an affine loop to the search for a linear loop satisfying $Q(x_1, \dots, x_d) = \tilde{c}$. The solution of this problem was given in Theorem 4.1. If the quadratic form is degenerate, we consider Equation (6). If at least one of the λ_i 's is non-zero, a loop exists, as shown by the ad hoc constructions of Proposition 5.3. In two of the three cases there, the loop is not just affine, but linear. Otherwise, if all of the λ_i 's are zero, we obtain a linear loop by essentially testing whether a solution to an equation $\tilde{Q}(x_1, \dots, x_d) = \tilde{c}$ exists. Finally, in order to obtain an affine loop satisfying the original equation, we apply transformation τ to the loop synthesised for Equation (6).

The synthesis procedure is summarised in the extended version, see [15, Appendix B]. By analysing the algorithm, one can argue that a negative output implies that Equation (4) has no solutions. The problem of *deciding* whether a loop exists for a given invariant, as in Problem 2.7 and opposed to the synthesis of numerical values, is thus solved as follows.

41:14 Linear Loop Synthesis for Quadratic Invariants

► **Corollary 5.5.** Let Q be a quadratic form, L a linear form over variables $\mathbf{x} = (x_1, \dots, x_d)$.

1. A non-trivial affine loop satisfying the quadratic equation $Q(\mathbf{x}) + L(\mathbf{x}) = c$ exists if and only if the equation has a rational solution different from $\mathbf{x} = \mathbf{0}$.
2. A non-trivial linear loop satisfying the equation $Q(\mathbf{x}) = c$ exists if and only if the equation has a rational solution different from $\mathbf{x} = \mathbf{0}$.

► **Example 5.6.** Let $-11x^2 + y^2 - 3z^2 + 2xy - 12xz + x + z = -1$ be a quadratic invariant in 3 variables. The quadratic form $Q(x, y, z) = -11x^2 + y^2 - 3z^2 + 2xy - 12xz$ is degenerate with rank $r = 2$ and so we can compute $\tau = \begin{pmatrix} -1 & 0 & 0 \\ \frac{1}{2} & \frac{3}{0} & \frac{0}{3} \end{pmatrix}$ such that $\tau^\top A_Q \tau = \text{diag}(0, 9, -27)$ is the matrix of an equivalent form. We have $Q(\tau\mathbf{x}) = \tilde{Q}(y, z) = 9y^2 - 27z^2$. For the linear part, $L(x, y, z) = x + z$, the change of coordinates results in $L(\tau\mathbf{x}) = \tilde{L}(y, z) + x = 3z + x$. Continue with the equation of the form (6): $9y^2 - 27z^2 + 3z = -1 - x$. Here, $\lambda_1 = 1$, and so we set (y, z) to $(\alpha_1, \alpha_2) = (\frac{1}{3}, 0)$ and find a solution for x : $\beta_1 = -2$. Next, find an affine transformation f associated with $9y^2 - 27z^2 + 3z = 1$. We have $\delta = 243$, $\mathbf{h} = (0, 27)^\top$ and $\tilde{c} = 216513$. The solutions of $9y^2 - 27z^2 + 3z = 1$ are exactly the solutions of $9y^2 - 27z^2 = 216513$ under the action of f .

Using the LINLOOP procedure, we find a linear loop $\langle M, \mathbf{s} \rangle$ for the invariant $9y^2 - 27z^2 = 216513$ with $M = \begin{pmatrix} 2 & 3 \\ 1 & 2 \end{pmatrix}$ and $\mathbf{s} = (-162, 27)^\top$. Therefore, an affine loop

$$\mathcal{A} := \langle M, \frac{1}{2\delta}(\mathbf{s} - \mathbf{h}), \frac{1}{2\delta}(M - I_2)\mathbf{h} \rangle;$$

that is, an affine loop with augmented matrix M' and initial vector \mathbf{s}' given by

$$M' = \left(\begin{array}{c|c} 1 & 0_{1,r} \\ \hline \frac{1}{2\delta}(M - I_2)\mathbf{h} & M \end{array} \right) = \begin{pmatrix} 1 & 0 & 0 \\ -1/6 & 2 & 3 \\ -1/18 & 1 & 2 \end{pmatrix} \quad \text{and} \quad \mathbf{s}' = \frac{1}{2\delta}(\mathbf{s} - \mathbf{h}) = \begin{pmatrix} \frac{1}{3} \\ 0 \\ 0 \end{pmatrix},$$

satisfies the invariant $9y^2 - 27z^2 + 3z = 1$. Consequently, a *linear* loop with update matrix

$$M_\beta := \begin{pmatrix} 1 & 0 & 0 \\ 1/12 & 2 & 3 \\ 1/36 & 1 & 2 \end{pmatrix}$$

and initial vector $(-2, 1/3, 0)^\top$ satisfies the invariant $9y^2 - 27z^2 + 3z = -1 - x$. We conclude by applying transformation τ : a linear loop with matrix

$$\tau M_\beta \tau^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 27/4 & 2 & 3 \\ 35/12 & 1 & 2 \end{pmatrix} \quad \text{and initial vector} \quad \tau \begin{pmatrix} -2 \\ 1/3 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ -4 \end{pmatrix}$$

satisfies the original invariant $-11x^2 + y^2 - 3z^2 + 2xy - 12xz + x + z = -1$.

6 Conclusion

6.1 Related Work

Loop Synthesis. Work by Humenberger et al. on loop synthesis employs an approach based on algebraic reasoning about linear recurrences and translating loop synthesis into an SMT (Satisfiability Modulo Theory) solving task in non-linear arithmetic [17]. Their approach is relatively complete in the sense that every loop with algebraic values is captured as one

of the solutions to the system of constraints. At the same time, no method is known to decide whether such a system has a *rational* solution. In contrast, our approach gives a characterisation of quadratic invariants that have linear loops with rational values.

Another SMT-based algorithm for template-based synthesis of general polynomial programs is given in work by Goharshady et al. [13]. However, loops generated for an invariant $P = 0$ using the latter approach necessarily have $P = 0$ as an inductive invariant and are not guaranteed to have infinite orbits. Recent work by Kenison et al. addresses the loop synthesis problem for multiple polynomial invariants, where each of the polynomials is a binomial of a certain type [21]. In our work, we restrict not the number of monomials in an invariant, but its degree, and thus achieve a complete solution for a single quadratic invariant.

Solving Polynomial Equations. As noted in Remark 2.8, one of the fundamental challenges towards loop synthesis arises from the study of integer and rational solutions to polynomial equations. A *Diophantine equation* $F(x_1, x_2, \dots, x_d) = 0$ is a polynomial equation with rational coefficients in at least two variables. A general decision procedure for the existence of rational solutions to a Diophantine equation (Problem 3.1) is not known. Over the ring of integers, this is Hilbert’s 10th Problem, proven undecidable by Matiyasevich in 1970 [25]. Furthermore, there does not exist an algorithm that for an arbitrary Diophantine equation, decides whether it has infinitely many integer solutions [10].

In contrast to the algorithmic unsolvability of Hilbert’s 10th Problem and the open status of Problem 3.1, algorithms exist that allow finding rational solutions for special classes of equations. For instance, there exist procedures [14, 30, 24] completely solving the specialisation of the problem to quadratic equations. Masser introduced an approach based on the effective search bound for rational solutions [24]. A further improvement of this approach for $d \geq 5$ is provided in [5]. An alternative procedure to decide whether an arbitrary quadratic equation has a rational solution is described in [14] (see Corollary, pg. 2 therein). Determining the existence of integer solutions to a *system* of quadratic equations is, however, undecidable [4].

6.2 Discussion

We conclude by sketching some observations and pointing out the directions for future work.

Multiple loops. The approach of Algorithm 1 can be adapted to generate multiple linear loops satisfying a given invariant. Different solutions of the quadratic equation can be found in line 5 (Algorithm 1) and subsequently used as an initial vector. Moreover, in line 11, it is possible to pick two variables (x_1, x_2) in different ways, thus obtaining different matrices M in line 12. Each of the matrices synthesised so is an element of the orthogonal group $\Gamma(Q')$ ¹ of the quadratic form Q' . Therefore, all possible products of these matrices also preserve the value of Q' and can be used as updates. Other than the default matrix selected by the algorithm, some of these matrices alter more than two variables non-trivially, intuitively making the synthesised loop more specific to the polynomial invariant.

¹ The orthogonal group $\Gamma(Q)$ of a quadratic form Q is the group of all linear automorphisms $M \in \text{GL}_d(\mathbb{K})$ such that $Q(\mathbf{x}) = Q(M\mathbf{x})$ for all $\mathbf{x} \in \mathbb{K}^d$.

Number of loop variables. Let $P(x_1, \dots, x_d) = 0$ be a quadratic invariant in d variables. Note that Theorem 5.4 can be interpreted in terms of linear loops with variables x_0, x_1, \dots, x_d . Specifically, we can redefine the loop synthesis problem (Problem 2.7) by searching for linear loops with $s = d + 1$ variables. To this end, update the procedure in Section 5 as follows: if the output of the original algorithm is an affine loop $\langle M, \mathbf{s}, \mathbf{t} \rangle$, then output the linear loop

$$\left\langle \left(\begin{array}{c|c} 1 & 0_{1,d} \\ \mathbf{t} & M \end{array} \right), \begin{pmatrix} 1 \\ \mathbf{s} \end{pmatrix} \right\rangle.$$

Due to Corollary 5.5, the updated procedure solves the problem of loop synthesis with one additional variable. What follows is a reinterpretation of Theorem 5.4:

► **Corollary 6.1.** *There exists an effective procedure for the following problem: given a quadratic equation*

$$Q(x_1, \dots, x_d) + L(x_1, \dots, x_d) = c,$$

decide whether there exists a non-trivial linear loop in $d + 1$ variables $\{x_0, x_1, \dots, x_d\}$ that satisfies it. Furthermore, the procedure synthesises a loop, if one exists.

Increasing the number of variables in the loop template leads to the following question, also raised in [17]:

► **Question.** *Let P be an arbitrary polynomial in d variables. Does there exist an upper bound N such that if a non-trivial linear loop satisfying $P = 0$ exists, then there exists a non-trivial linear loop with at most N variables satisfying the same invariant?*

Corollary 6.1 (together with Corollary 5.5) shows that, for quadratic polynomials, N is at most $d + 1$. Moreover, we show in Section 4 that in the class of polynomial equations $Q(\mathbf{x}) - c$, where Q is a quadratic form, the bound $N = d$ is tight. A full characterisation of quadratic equations for which linear loops with d variables exist would also be of interest.

Sufficient conditions. The results of Sections 4 and 5 witness another class of polynomial invariants for which non-trivial linear (or affine) loops always exist. Similar to the setting of equations with pure difference binomials in [21], we can claim this for invariants $Q(x_1, \dots, x_d) = c$ with isotropic quadratic forms Q . In particular, for every equation of the form $a_1x_1^2 + \dots + a_dx_d^2 + c = 0$ with $d \geq 4$ and a_1, \dots, a_d, c not all possessing the same sign, there exists a non-trivial linear loop with d variables. This fact is due to Meyer's Theorem on isotropy of indefinite forms [26] and Corollary 5.5(2).

Beyond quadratic. One future work direction concerns loop synthesis from invariants that are polynomial equalities of higher degrees, and, in particular, algebraic forms. However, we are limited by the hardness of Problem 3.1, as before. For Diophantine equations defined with homogeneous polynomials of degree 3, the loop synthesis is related to the study of rational points on elliptic curves, a central topic in computational number theory [31, 8].

References

- 1 T. Andreescu and D. Andrica. *Quadratic Diophantine Equations*. Developments in Mathematics. Springer New York, 2016.
- 2 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

- 3 Zenon I. Borevich and Igor R. Shafarevich. *Number Theory*. Pure and Applied Mathematics. Academic Press, New York, NY, 1966.
- 4 J. L. Britton. Integer solutions of systems of quadratic equations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 86(3):385–389, 1979. doi:10.1017/S0305004100056218.
- 5 T. D. Browning and R. Dietmann. On the representation of integers by quadratic forms. *Proceedings of the London Mathematical Society*, 96(2):389–416, 2008. doi:10.1112/plms/pdm032.
- 6 Pierre Castel. Solving quadratic equations in dimension 5 or more without factoring. In *ANTS X—Proceedings of the Tenth Algorithmic Number Theory Symposium*, volume 1 of *Open Book Ser.*, pages 213–233. Math. Sci. Publ., Berkeley, CA, 2013. doi:10.2140/obs.2013.1.213.
- 7 Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Ehsan Kafshdar Goharshady. Polynomial invariant generation for non-deterministic recursive programs. In *PLDI 2020*, pages 672–687, 2020. doi:10.1145/3385412.3385969.
- 8 H. Cohen. *Number Theory: Volume I: Tools and Diophantine Equations*. Graduate Texts in Mathematics. Springer New York, 2007.
- 9 J. E. Cremona and D. Rusin. Efficient solution of rational conics. *Math. Comp.*, 72(243):1417–1441, 2003. doi:10.1090/S0025-5718-02-01480-1.
- 10 Martin Davis. On the number of solutions of diophantine equations. *Proceedings of the American Mathematical Society*, 35(2):552–554, 1972.
- 11 Emilie Dufresne, Parker B. Edwards, Heather A. Harrington, and Jonathan D. Hauenstein. Sampling real algebraic varieties for topological data analysis, 2018. arXiv:1802.07716.
- 12 G. Everest, A. van der Poorten, I. Shparlinski, and T. Ward. *Recurrence sequences*, volume 104 of *Math. Surveys Monogr.* Amer. Math. Soc., Providence, RI, 2003. doi:10.1090/surv/104.
- 13 Amir Kafshdar Goharshady, S. Hitarth, Fatemeh Mohammadi, and Harshit Jitendra Motwani. Algebraic-geometric algorithms for template-based synthesis of polynomial programs. *Proc. ACM Program. Lang.*, 7(OOPSLA1), April 2023. doi:10.1145/3586052.
- 14 Fritz J. Grunewald and Daniel Segal. How to solve a quadratic equation in integers. *Mathematical Proceedings of the Cambridge Philosophical Society*, 89(1):1–5, 1981. doi:10.1017/S030500410005787X.
- 15 S. Hitarth, George Kenison, Laura Kovács, and Anton Varonka. Linear loop synthesis for quadratic invariants, 2023. Extended version. arXiv:2310.05120.
- 16 Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. On strongest algebraic program invariants. *J. ACM*, August 2023. Just Accepted. doi:10.1145/3614319.
- 17 Andreas Humenberger, Daneshvar Amrollahi, Nikolaj Bjørner, and Laura Kovács. Algebra-based reasoning for loop synthesis. *Form. Asp. Comput.*, 34(1), July 2022. doi:10.1145/3527458.
- 18 K. Ireland and M.I. Rosen. *A Classical Introduction to Modern Number Theory*. Graduate Texts in Mathematics. Springer, 1990.
- 19 Bertrand Jeannet, Peter Schrammel, and Sriram Sankaranarayanan. Abstract acceleration of general linear loops. *SIGPLAN Not.*, 49(1):529–540, January 2014. doi:10.1145/2578855.2535843.
- 20 Manuel Kauers and Peter Paule. *The Concrete Tetrahedron - Symbolic Sums, Recurrence Equations, Generating Functions, Asymptotic Estimates*. Texts & Monographs in Symbolic Computation. Springer, 2011.
- 21 George Kenison, Laura Kovács, and Anton Varonka. From polynomial invariants to linear loops. In *Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation, ISSAC '23*, pages 398–406, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3597066.3597109.
- 22 Zachary Kincaid, Jason Breck, John Cyphert, and Thomas W. Reps. Closed Forms for Numerical Loops. In *Proc. of POPL*, pages 55:1–55:29, 2019.
- 23 Seymour Lipschutz and Marc Lipson. *Schaum's Outline of Linear Algebra*. Schaum's Outline Series. McGraw-Hill, New York, fourth edition, 2009.

41:18 Linear Loop Synthesis for Quadratic Invariants

- 24 D. W. Masser. How to solve a quadratic equation in rationals. *Bulletin of the London Mathematical Society*, 30(1):24–28, 1998. doi:10.1112/S0024609397003913.
- 25 Yuri Matiyasevich. Enumerable sets are Diophantine. *Soviet Math. Dokl.*, 11:354–358, 1970.
- 26 A. Meyer. Mathematische Mittheilungen. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 29:209–222, 1884.
- 27 Ivan Niven. *Irrational numbers*. Mathematical Association of America; distributed by John Wiley & Sons, Inc., New York, N.Y.,, 1956.
- 28 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *ACM SIGLOG News*, 2(2):4–13, April 2015. doi:10.1145/2766189.2766191.
- 29 Alexandra Shlapentokh. Defining integers. *The Bulletin of Symbolic Logic*, 17(2):230–251, 2011.
- 30 Carl Ludwig Siegel. Zur theorie der quadratischen formen. *Nachrichten der Akademie der Wissenschaften in Göttingen, Mathematisch-Physikalische Klasse*, 3:21–46, 1972.
- 31 Joseph H. Silverman and John T. Tate. *Rational Points on Elliptic Curves*. Springer Publishing Company, Incorporated, 2nd edition, 2015.
- 32 Denis Simon. Quadratic equations in dimensions 4, 5 and more. 2005.
- 33 Denis Simon. Solving quadratic equations using reduced unimodular quadratic forms. *Math. Comp.*, 74(251):1531–1543, 2005. doi:10.1090/S0025-5718-05-01729-1.
- 34 T. Skolem. *Diophantische Gleichungen*. Ergebnisse der Mathematik und ihrer Grenzgebiete. J. Springer, 1938.
- 35 Sergei Treil. Linear algebra done wrong, 2004.
- 36 Leonid Vaserstein. Polynomial parametrization for the solutions of diophantine equations and arithmetic groups. *Annals of Mathematics*, 171:979–1009, 2010.

Algorithms for Claims Trading

Martin Hoefer  

Goethe University Frankfurt, Germany

Carmine Ventre  

King's College London, UK

Lisa Wilhelmi  

Goethe University Frankfurt, Germany

Abstract

The recent banking crisis has again emphasized the importance of understanding and mitigating systemic risk in financial networks. In this paper, we study a market-driven approach to rescue a bank in distress based on the idea of *claims trading*, a notion defined in Chapter 11 of the U.S. Bankruptcy Code. We formalize the idea in the context of the seminal model of financial networks by Eisenberg and Noe [5]. For two given banks v and w , we consider the operation that w takes over some claims of v and in return gives liquidity to v (or creditors of v) to ultimately rescue v (or mitigate contagion effects). We study the structural properties and computational complexity of decision and optimization problems for several variants of claims trading.

When trading incoming edges of v (i.e., claims for which v is the creditor), we show that there is no trade in which *both banks v and w strictly* improve their assets. We therefore consider *creditor-positive* trades, in which v profits strictly and w remains indifferent. For a given set C of incoming edges of v , we provide an efficient algorithm to compute payments by w that result in a creditor-positive trade and maximal assets of v . When the set C must also be chosen, the problem becomes weakly NP-hard. Our main result here is a bicriteria FPTAS to compute an approximate trade, which allows for slightly increased payments by w . The approximate trade results in nearly the optimal amount of assets of v in any exact trade. Our results extend to the case in which banks use general monotone payment functions to settle their debt and the emerging clearing state can be computed efficiently.

In contrast, for trading outgoing edges of v (i.e., claims for which v is the debtor), the goal is to maximize the increase in assets for the creditors of v . Notably, for these results the characteristics of the payment functions of the banks are essential. For payments ranking creditors one by one, we show NP-hardness of approximation within a factor polynomial in the network size, in both problem variants when the set of claims C is part of the input or not. Instead, for payments proportional to the value of each debt, our results indicate more favorable conditions.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains; Networks → Network algorithms

Keywords and phrases Financial Networks, Claims Trade, Systemic Risk

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.42

Related Version *Full Version:* <https://arxiv.org/pdf/2402.13627.pdf>

Funding Martin Hoefer and Lisa Wilhelmi gratefully acknowledge support by DFG Research Unit ADYN under grant DFG 411362735.

Acknowledgements This paper was initiated at the Dagstuhl seminar 22452 “Computational Social Dynamics”. The authors thank the organizers and participants of the seminar.

1 Introduction

The global banking crisis of March 2023 caused turmoil in a market fearful of the repeat of the Great Financial Crisis of 2007. These recent events serve as a stark reminder of the paramount importance of the study of systemic risk in financial networks. In this growing



© Martin Hoefer, Carmine Ventre, and Lisa Wilhelmi;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 42; pp. 42:1–42:17



Leibniz International Proceedings in Informatics

LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



body of work, the focus is mainly on the complexity of computing *clearing states*, known to measure the exposure of the different banks in the network to insolvencies within, see, e.g., [5, 10, 23], and strategic aspects of the banks' behavior, cf. [1, 8, 11, 17]. However, to calm the market and prevent contagion, regulators and central banks are more interested in finding ways to rescue banks in distress, reassure investors that the system is stable and avoid further bank runs. In fact, Silicon Valley Bank, Signature Bank and Credit Suisse – the three banks at the heart of the crisis last March – were all acquired by other banks in the network, and, by modifying the network, this has seemingly mitigated systemic risk.

A line of research in financial networks on interventions in the network is recently discussed in [7, 18], the main idea being that banks can swap debt contracts. In particular, the authors of [7] study the extent to which a sequence of debt swaps can reduce the risk in the network, in the sense that bank assets Pareto-improve. Notably, swaps can occur anywhere in the network, even if the focus is strict improvement of the assets of a given bank.

In this work, we build on this idea and initiate research on the computation of a network-based “rescue package” deal for a given bank with the objective of making it solvent. This is exactly the problem that regulators faced in March 2023 for the aforementioned banks. However, *acquisitions* do not seem to be the right operations in these instances since they have two main drawbacks from a societal perspective (as also witnessed by the reactions to recent deals). Firstly, the acquiring bank rarely has enough time or freedom to evaluate the purchase and make a sensible business decision. Secondly, and consequently, it often requires a security for bailout from the central bank, in the form of significant protection against potential losses from risks associated with the transaction. For example, in the acquisition of Credit Suisse, UBS had little choice in the matter, as reported by Bloomberg news [2], and received a guarantee worth CHF 9 billion, as confirmed by the Swiss Federal Council [3].

We instead study a market-driven approach to rescue banks in distress based on the idea of *claims trading*. Claims trading is defined in Chapter 11 of the U.S. Bankruptcy Code. We formalize the idea and analyze the consequences of such trades in the context of financial networks. When a company is in financial distress, its creditors can assert their rights to repayment by submitting a claim. At this point, a creditor can either wait for the positions to unwind and get (a part of) the claim once the bankruptcy is settled, or she can sell her credit claim to a willing buyer for some immediate liquidity. The former approach is equivalent to the mainstream work on systemic risk since the insolvency of a bank can directly cascade through the network via lower payments to its creditors. We want to explore ways to find interested buyers that purchase the claims of an insolvent bank v and give liquidity to the network that ultimately rescues v . Ideally, the buyers should avoid any loss so that the cash invested in buying the claim will return via increased payments within the network; this way incentives of buyers are aligned, and systemic risk is reduced at no extra cost to the network.

We design efficient algorithms to compute claims trades or settle the inherent complexity status of the problems. The importance of algorithms computing claim trades that resolve complicated systemic issues in finance cannot be underestimated. In practice, deals are concocted when markets are closed, and algorithms that efficiently compute solutions in these pressurised situations become essential.

Related Work. Much of the work on systemic risk in financial networks, including ours, builds upon [5]. In this seminal work, the authors propose a model and prove existence and properties of clearing states. Moreover, they also provide a polynomial-time algorithm for their computation. The model in [5] has been extended along many dimensions by follow-up work; for example, the authors of [20] add default costs whereas financial derivatives are

considered in [22]. Computation of clearing states for the latter model is studied in [9, 10, 23] for different notions of approximation and payment schemes adopted. The solution space of clearing states for financial networks with derivatives is studied in [19].

The study of strategic behavior in financial networks was initiated in [1], where banks are assumed to strategize in the way they allocate money to their creditors. A similar approach is used in [8, 11]. A different model, featuring derivatives, and banks strategically donating money or cancelling debts is studied in [17]. The idea of cancelling debts is further explored in [12]. The authors of [12, 16] consider computational complexity of computing optimal or approximate bailout policies from the central bank external to the network. In contrast, in our work all transfers of assets are intrinsic to the network, and the bank providing the assets must not be harmed. In [13], the authors study computational complexity of strategic changes to the underlying network via debt transfers.

Debt swapping is introduced in [18] – the authors focus more on the existence and properties of swaps with and without shocks to the system. As discussed above, the authors of [7] share goals that are somewhat similar to ours but use a different operation to update the network. A related line of work considers portfolio compression, an accounting operation by means of which all the cycles in the network are deleted. The effects on systemic risk of portfolio compression are studied in [21, 24]. However, it is important to note that portfolio compression can lead to a worse outcome for banks that are not contained in the cycle [21] and consequently it is not clear why banks should accept to modify their balance sheets in this way, as argued empirically in [15].

To the best of our knowledge, ours is the first work to study claims trading in the analysis of financial networks. Claims trading in bankruptcy has been studied by law scholars, who for example argue that its effects in that context are variegated and nuanced in general [14] but do not concern the governance of the bankruptcy process [6].

Contribution. We focus on the elementary setting with one given bank v to save (e.g., Credit Suisse) and one bank w that may rescue it (e.g., UBS). We consider the following problem: Are there claims of v that can be sold to w so that v becomes solvent (i.e., after the claims trade, v can fully pay all its liabilities)? This problem gives rise to a suite of algorithmic questions, depending on the remit of the algorithmic decision, such as: How many claims are we allowed to trade? Which claims of v should we trade? What are the payments that must be transferred from w to v to make the trade worthwhile for w ? Our treatment is steered by the following structural insight: We prove that it is impossible for *both* v and w to *strictly* profit from the claims trade. Accordingly, we restrict our attention to creditor-positive trades that strictly improve v without harming w .

For our first set of algorithmic results in Section 3, we fix one claim with creditor v to be traded with w . Does this represent a feasible (i.e., creditor-positive) trade? This can be decided by simply computing clearing states that determine the payment towards each debt in the network. The problem becomes interesting if we also determine the *haircut rate* $\alpha \in [0, 1]$ of the trade – in order to provide liquidity, w may be willing to pay an α -fraction of the claim’s liability to v . Depending on the payment functions used by banks to distribute money to their debts, we design different polynomial-time algorithms that determine feasibility of the trade and also α^* (if any), the value of α maximizing the assets of v (or a close approximation of α^* if the payment functions are too granular vis-a-vis the input size). Let us highlight that these results also apply to the case in which v is the *debtor* of the claim to trade; in fact, we prove that every creditor-positive trade Pareto-improves the clearing state – each bank in the network is (weakly) better off after the trade. By maximizing the assets of the creditor of the traded claim we, thus, also maximize assets of the debtor.

We consider trading multiple claims with creditor v in Section 4. For a fixed set of claims our results from Section 3 extend rather directly. The picture becomes less benign when we also have to choose the subset of claims to be traded. Indeed, in Section 4.2 we show that it is weakly NP-hard to decide if there is a subset of claims along with suitable haircut rates to obtain a creditor-positive claims trade that makes v solvent. In our most technical contribution, we show that there exists a bicriteria FPTAS for deciding this problem. If an exact trade exists that yields total assets of A^* for v , we find an approximate trade with assets at least $A^* - \delta$ for exponentially small δ , which allows haircut rates of at most $1 + \varepsilon$. The FPTAS applies to all financial networks with general monotone payment functions for which a clearing state can be computed efficiently. On a technical level, we fix a desired value A for the total assets of v . Using a subroutine we determine if there is an approximate trade that yields this asset level for v . En route, we discover an intricate monotonicity property – if there is an exact trade that yields assets A^* for v , then for every $A \leq A^*$ there is an approximate trade with assets A for v . Notably, monotonicity can break above A^* . Still, we can apply binary search to find an approximate trade with assets at least $A^* - \delta$.

Finally, for trading multiple claims with debtor u in Section 5 – rather than trying to save u – our goal is improve conditions for the creditors of u to minimize the contagion effects by u 's bankruptcy. Interestingly, the results here depend significantly on the choice of the payment functions. For payments based on a ranking of the creditors, we show that the problem becomes NP-hard to approximate within a factor polynomial in the network size. In contrast, for payments proportional to the value of each debt, we can solve the problem for a given set of claims, but it becomes strongly NP-hard when having to choose the set of claims.

All missing proofs are deferred to the full version of this paper.

2 Model and Preliminaries

A financial network $\mathcal{F} = (G, \ell, \mathbf{a}^x, \mathbf{f})$ is expressed as a directed multigraph¹ $G = (V, E, \text{de}, \text{cr})$ without self loops. We denote $n = |V|$. Every node $v \in V$ in the graph represents a financial institution or *bank*. Every edge $e \in E$ represents a *debt contract* or *claim* involving two banks. For each edge $e \in E$, $\text{de}(e)$ specifies the debtor (i.e., the source) and $\text{cr}(e)$ the creditor (i.e., target). Edge $e \in E$ has a weight $\ell_e \in \mathbb{N}_{>0}$. In other words, in the context of debt contract e , bank $\text{de}(e)$ owes $\text{cr}(e)$ an amount of ℓ_e . We denote the set of outgoing and incoming edges of a bank v by $E^+(v) = \{e \in E \mid v = \text{de}(e)\}$ and $E^-(v) = \{e \in E \mid v = \text{cr}(e)\}$. Since we allow multi-edges, several debt contracts with possibly different liabilities could exist between the same pair of banks. The *total liabilities* L_v of v are the sum of weights of all outgoing edges of v , i.e., $\sum_{e \in E^+(v)} \ell_e = L_v$. Furthermore, every bank v holds *external assets* $a_v^x \in \mathbb{N}$. They can be interpreted as an amount of money the bank receives from outside the network.

Let $b_v \in [a_v^x, a_v^x + \sum_{e \in E^-(v)} \ell_e]$ be the total funds of bank v . Bank v distributes her total funds according to a given *payment function* $\mathbf{f}_v = (f_e)_{e \in E^+(v)}$, where $f_e : \mathbb{R} \rightarrow [0, \ell_e]$. For every outgoing edge, the function $f_e(b_v)$ defines the amount of money v pays towards e . We follow previous literature and assume the following conditions for every payment function:

- (1) Every function $f_e(b_v)$ is non-decreasing and bounded by $0 \leq f_e(b_v) \leq \ell_e$.
 - (2) Every bank pays all funds until all liabilities are settled: $\sum_{e \in E^+(v)} f_e(b_v) = \min\{b_v, L_v\}$.
 - (3) The sum of payments of a bank is limited by the total funds: $\sum_{e \in E^+(v)} f_e(b_v) \leq b_v$.
- Here (2) implies (3), and we mention (3) explicitly for clarity. For a *monotone* function \mathbf{f}_v , v weakly increases the payment on every outgoing edge when receiving additional funds.

¹ Claims trades in simple graphs can result in graphs with multi-edges. This can sometimes be avoided by analyzing the trades in equivalent simple graphs with suitable auxiliary banks. Since all our arguments can also be applied in the context of multigraphs, we discuss the more general model.

Clearing States. Let $\mathbf{p} = (p_e)_{e \in E}$ be the arising payments in the network when every bank v distributes the funds according to her payment functions \mathbf{f}_v . The *incoming payments* of v are given by $\sum_{e \in E^-(v)} p_e$. The *total assets* a_v are defined as the external assets plus the incoming payments, i.e., $a_v = a_v^x + \sum_{e \in E^-(v)} p_e$. Observe that the above conditions (1), (2) and (3) are fixed-point constraints. A vector of total assets $\mathbf{a} = (a_v)_{v \in V}$ is called *feasible* if it satisfies all fixed-point constraints. More formally, for every feasible \mathbf{a} it holds that $a_v = a_v^x + \sum_{e \in E^-(v)} f_e(a_{\text{de}(e)})$. The payments \mathbf{p} corresponding to a feasible vector \mathbf{a} are called a *clearing state*. For fixed payment functions, multiple clearing states may exist. We assume throughout that every payment function \mathbf{f}_v is *monotone*, i.e., $f_e(x) \geq f_e(y)$ for all $x \geq y \geq 0$ and every $e \in E^+(v)$. This implies that all clearing states form a complete lattice [1, 4]. Thus, the point-wise minimal and maximal clearing states are unique. We follow previous literature and assume that the *maximal* clearing state arises in the network.

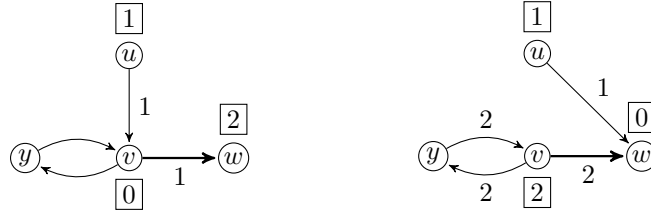
Payment Functions. In the seminal work of Eisenberg and Noe [5] and the majority of subsequent works, all banks are assumed to allocate their assets using *proportional* payment functions. The *recovery rate* $r_v = \min\{a_v/L_v, 1\}$ is the fraction of total liabilities v can pay off, and the payments on edge $e \in E^+(v)$ are defined proportionally by $f_e(a_v) = r_v \cdot \ell_e$. Hence, if $r_v = 1$, then v will fully settle all liabilities. Otherwise, v is in default, $r_v < 1$, and the liabilities are settled partially in proportion to their weight. These payments are often used when all debt contracts fall due at the same date. If, on the other hand, different debt contracts are assigned different priorities or maturity dates, payments are more suitably expressed by *edge-ranking* payment functions. Then, the debt contracts in $E^+(v)$ are ordered by a permutation π_v . First, v makes payments towards the highest ranked edge $\pi_v(1)$ until the edge is saturated or v has no remaining assets. Once $\pi_v(1)$ is fully paid off, v pays off the second highest ranked edge $\pi_v(2)$ until the edge is saturated or v has no remaining assets. The process continues and ends when either all liabilities are settled or v exhausted all assets.

Both proportional and edge-ranking payments are monotone. For proportional payments, the clearing state can be computed in polynomial time [5]; for edge-ranking payments in strongly polynomial time [1].

In this paper, we obtain some results explicitly for networks with proportional and edge-ranking payment functions. Most of our results, however, generalize to arbitrary monotone payment functions when there is an efficient *clearing oracle*, i.e., there exists an algorithm that receives a network \mathcal{F} as input and outputs the clearing state \mathbf{p} in polynomial time.

Claims Trades. When a bank u is in default and unable to settle all debt, this introduces risk into the network. In particular, the creditors of u do not receive their full liabilities. This could lead to further defaults of the creditor banks. In order to reduce the risk of spreading default, the creditors of u can sell claims they make towards u . More in detail, consider banks u, v and w with edge e with $\text{de}(e) = u$ and $\text{cr}(e) = v$, $\ell_e \geq 0$. Suppose u is in default. If v and w perform a claims trade, w becomes the new creditor of bank u with the same liability. Consequently, any payment from u towards the claim will be received by w . In return for the traded claim, v receives a *return* ρ from w , i.e., an immediate payment of $\rho = \alpha \cdot \ell_e$, for some $\alpha \in [0, 1]$. We call α the *haircut rate*. To separate the return from the payments in the clearing state, we model the return by a transfer of external assets from w to v . Note that w can invest at most her external assets as a return, so every trade must satisfy $\alpha \ell_e \leq a_w^x$. After a trade the external assets of v and w might no longer be integer values.

We proceed to define three variants of claims trades. We are given a financial network with distinct banks $u, v, w \in V$, an edge $e \in E$ with $(\text{de}(e), \text{cr}(e)) = (u, v)$ and haircut rate $\alpha \in [0, 1]$. For a (*single*) *claims trade* of e to w we perform the following adjustments to



■ **Figure 1** The network from Example 1 before the trade is depicted left, and right after the trade. All liabilities equal 2. Edges are labeled with positive payments (if any) in the clearing state.

the network: (1) change the creditor of e to $\text{cr}'(e) = w$, (2) change external assets of v to $a_v^x + \alpha \cdot \ell_e$ and (3) change external assets of w to $a_w^x - \alpha \cdot \ell_e \geq 0$. We denote the resulting *post-trade network* by $\mathcal{F}' = (G', \ell, \mathbf{a}'^x, \mathbf{f})$, and the resulting clearing state in \mathcal{F}' by \mathbf{p}' . For a given trade of e to w , we call v the *creditor* and w the *buyer*. Observe that the total assets of v after the trade are given by $a'_v = a_v^x + \alpha \cdot \ell_e + \sum_{e' \in E' - (v)} p'_{e'}$. Similarly, the total assets of w after the trade are $a'_w = a_w^x - \alpha \cdot \ell_e + \sum_{e' \in E' - (w)} p'_{e'}$.

The claims trade operation can be directly extended to a trade of multiple edges. As outlined in the introduction, we are interested in the effects when a single bank (in default) trades claims with another bank w (such as a central bank). We study the differences when trading incoming or outgoing edges. Observe that both generalize single claims trades.

For a *multi-trade of incoming edges*, there are distinct banks v, w in a network \mathcal{F} , a set C of k distinct incoming edges $e_1, \dots, e_k \in E^-(v)$, and haircut rates $\alpha_1, \dots, \alpha_k$, such that $\text{de}(e_i) \neq w$, for all i . After the trade, a new network \mathcal{F}' emerges: We change $\text{cr}'(e_i) = w$, for all $i = 1, \dots, k$, adjust external assets for v to $a_v^x + \sum_{i=1}^k \alpha_i \ell_{e_i}$, and for w to $a_w^x - \sum_{i=1}^k \alpha_i \ell_{e_i} \geq 0$.

For a *multi-trade of outgoing edges*, there are distinct banks u, w in a network \mathcal{F} , a set C of k distinct outgoing edges $e_1, \dots, e_k \in E^+(v)$ with $\text{cr}(e_i) = v_i$, and haircut rates $\alpha_1, \dots, \alpha_k$, such that $\text{cr}(e_i) \neq w$, for all i . After the trade, a new network \mathcal{F}' emerges: We change $\text{cr}'(e_i) = w$, for all $i = 1 \dots, k$, adjust external assets for each v_i to $a_{v_i}^x + \alpha_i \ell_{e_i}$, and for w to $a_w^x - \sum_{i=1}^k \alpha_i \ell_{e_i} \geq 0$.

We proceed with a small example of trading a single claim.

► **Example 1.** Consider the example network depicted in Figure 1 (left) on a simple directed graph. The liability of every edge is 2. The only banks with non-zero external assets are u and w , where $a_u^x = 1$ and $a_w^x = 2$. $a_v^x = 0$ is also explicitly displayed for convenience. Banks u, w and y each have at most one outgoing edge. They pay all their assets (if any) to the unique outgoing edge until it is saturated. This implies payments of 1 on edge (u, v) . v is the only bank with a non-trivial payment function – suppose it uses an edge-ranking function with priority $\pi_v(1) = (v, w)$ and $\pi_v(2) = (v, y)$. Then, v pays the incoming assets of 1 to w , and there are no payments on the cycle of v and y . To see this, assume $p_{(y,v)} = x > 0$. By the edge-ranking function, from these additional assets v allocates a portion of $\min(x, 1)$ to (v, w) and the rest to (v, y) . Hence, the total assets of y are $\max(x - 1, 0)$ while the outgoing payments are x , which contradicts the feasibility constraint (3). Overall, in the clearing state, the total assets are $a_u = a_v = 1, a_w = 3$ and $a_y = 0$.

Suppose we perform the trade of edge $e = (u, v)$ to w with $\alpha = 1$, see Fig. 1 (right) for the resulting network. w buys edge e and pays the full liability ℓ_e to v . The external assets of v increase to 2 and allow v to settle all debt. The total assets become 1 for u , 2 for y , 3 for w and 4 for v . The total assets of u and w are unaffected by the trade, the total assets of v and y strictly increase. Overall, the clearing state is point-wise non-decreasing.

A similar observation can be made when v uses proportional payments. Before the trade, v pays 1 to y and w . After the trade, both edges can be paid fully. ◻

Properties of Claims Trades. In Example 1, v strictly benefits from the trade while w is indifferent. Interestingly, it is impossible for both creditor and buyer to strictly profit from a single trade. This property holds true for the more general class of multi-trades of incoming edges, and it applies in any network \mathcal{F} with monotone payment functions.

The proof builds on a connection with *debt swaps* studied in [7,18]. A debt swap exchanges the creditors of two edges with the same liabilities. We show that a claims trade and the resulting payments can be represented by a debt swap in an auxiliary network.

► **Definition 2 (Debt Swap).** *Consider a financial network \mathcal{F} with four distinct nodes $u_1, u_2, v_1, v_2 \in V$ and edges $e_1, e_2 \in E$, where $u_1 = de(e_1), v_1 = cr(e_1)$ and $u_2 = de(e_2), v_2 = cr(e_2)$. Suppose the liabilities are $\ell_{e_1} = \ell_{e_2}$. A debt swap σ of e_1 and e_2 creates a new network \mathcal{F}^σ with $G^\sigma = (V, E, de, cr^\sigma)$ where $cr^\sigma(e_1) = v_2$, $cr^\sigma(e_2) = v_1$ and $cr^\sigma(e) = cr(e)$ for all $e \in E \setminus \{e_1, e_2\}$.*

► **Proposition 3.** *For every financial network with monotone payment functions, there exists no multi-trade of incoming edges such that both creditor v and buyer w strictly improve their total assets.*

Proof. For a given network \mathcal{F} , consider a multi-trade of incoming edges and construct a new network $\hat{\mathcal{F}}$ by adding an auxiliary bank \hat{v} to \mathcal{F} without external assets. \hat{v} serves as an “accumulator” for the payments along the edges e_i . We change the targets of the edges in C to $cr(e_i) = \hat{v}$. We add an edge \hat{e} with $de(\hat{e}) = \hat{v}$ and $cr(\hat{e}) = v$ and liability $\ell_{\hat{e}} = \sum_{i=1}^k \ell_{e_i}$. Every payment that gets paid to v via e_i in \mathcal{F} now first goes to \hat{v} , and then gets forwarded from \hat{v} to v , since \hat{e} has sufficiently high liability. Consider the clearing state $\hat{\mathbf{p}}$ in the resulting network $\hat{\mathcal{F}}$. Obviously, for the new edge $\hat{p}_{\hat{e}} = \sum_{i=1}^k \hat{p}_{e_i}$. As such, every (non-auxiliary) bank from \mathcal{F} receives the same external assets and eventually the same incoming and outgoing payments in $\hat{\mathcal{F}}$. Consequently, both \mathcal{F} and $\hat{\mathcal{F}}$ give rise to the same clearing state, i.e., $p_e = \hat{p}_e$, for all $e \in E$, and the same assets for every (non-auxiliary) bank.

The new network $\hat{\mathcal{F}}$ allows to conveniently route all payments along edges in C to w by trading the single accumulator edge \hat{e} to w . Thus, the multi-trade of incoming edges C in \mathcal{F} is equivalent to trading the single claim \hat{e} to w in $\hat{\mathcal{F}}$, for a suitably chosen haircut rate $\hat{\alpha}$ such that $\hat{\alpha} \cdot \sum_{i=1}^k \ell_{e_i} = \sum_{i=1}^k \alpha_i \cdot \ell_{e_i}$.

Now let us further adjust $\hat{\mathcal{F}}$ to $\tilde{\mathcal{F}}$ by introducing an auxiliary bank \tilde{w} . Intuitively, we “outsource” parts of external assets from w to \tilde{w} . Formally, external assets of w are reduced to $a_w^x - \sum_{i=1}^k \alpha_i \cdot \ell_{e_i} \geq 0$, external assets of \tilde{w} are $\sum_{i=1}^k \alpha_i \cdot \ell_{e_i}$. We add an edge \tilde{e} with $de(\tilde{e}) = \tilde{w}$ and $cr(\tilde{e}) = w$ as well as liabilities $\ell_{\tilde{e}} = \ell_{\hat{e}} = \sum_{i=1}^k \ell_{e_i}$. The clearing state $\tilde{\mathbf{p}}$ in the resulting network $\tilde{\mathcal{F}}$ is $\tilde{p}_{\tilde{e}} = \sum_{i=1}^k \alpha_i \ell_{e_i}$, since \tilde{e} is the only outgoing edge of \tilde{w} and $\ell_{\tilde{e}} \geq a_{\tilde{w}}^x$. Hence, w and (consequently) every non-auxiliary bank from \mathcal{F} receives the same total assets in $\tilde{\mathbf{p}}$. Indeed, \mathcal{F} , $\hat{\mathcal{F}}$ and $\tilde{\mathcal{F}}$ yield equivalent clearing states with $p_e = \hat{p}_e = \tilde{p}_e$, for all $e \in E$.

In $\tilde{\mathcal{F}}$ we can implement the return payments from w to v by re-routing the “outsource” edge \tilde{e} to v instead of w . Thus, the claim trade of \hat{e} in $\hat{\mathcal{F}}$ can be expressed by a swap of creditors of \hat{e} and \tilde{e} in $\tilde{\mathcal{F}}$. Now since $\ell_{\tilde{e}} = \ell_{\hat{e}}$, this swap of creditors represents a debt swap. Thus, the multi-trade in \mathcal{F} is equivalent to single trade in $\hat{\mathcal{F}}$ and a debt swap in $\tilde{\mathcal{F}}$. No debt swap can strictly improve both creditor banks [7, Corollary 6]. Thus, no multi-trade of incoming edges can strictly improve both creditor and buyer. ◀

The above proof implies a structural equivalence. Using the network $\hat{\mathcal{F}}$, we reduced a multi-trade of incoming edges to a single claim trade.

► **Corollary 4.** *For every multi-trade of incoming edges in a network \mathcal{F} , there is an adjusted network $\hat{\mathcal{F}}$ such that the result of the multi-trade in \mathcal{F} is the result of a single trade in $\hat{\mathcal{F}}$.*

Our motivation is to analyze claims trades to improve the situation of a creditor in default by trading claims with a buyer. Since it is impossible to strictly improve the conditions of both banks, we focus on strictly improving the creditor and weakly improving the buyer. Note that the trade performed in Example 1 satisfies this property.

► **Definition 5** (Creditor-positive trade). *A multi-edge trade of incoming edges of bank v to bank w is called creditor-positive if $a'_v > a_v$ and $a'_w \geq a_w$.*

For the proof of Proposition 3, we express the multi-trade by a debt swap in an auxiliary network. For a creditor-positive trade, the associated debt swap satisfies the same property, i.e., it is a so-called *semi-positive* debt swap. In any network \mathcal{F} with monotone payment functions, a semi-positive debt swap Pareto-improves the clearing state and, hence, the total assets of *every* bank [7]. This directly implies the next corollary.

► **Corollary 6.** *In every financial network with monotone payments, every creditor-positive trade Pareto-improves the clearing state.*

A creditor-positive trade reduces the impact of a defaulting debtor on the creditor. No bank in the entire network suffers. Hence, these trades contribute to the stabilization of the entire financial network. We focus on creditor-positive trades for the remainder of the paper.

3 Trading a Single Claim

In this section, we study a given single creditor-positive trade and optimize the effects on the assets in the network. For exposition, we mostly focus on financial networks with proportional or edge-ranking payments.

The choice of α affects the external assets of v and, thus, payments throughout the network. If a given trade is creditor-positive for some $\alpha \in [0, 1]$, we say that α is creditor-positive. Can we efficiently decide the existence of a creditor-positive α ? What is the *optimal* α to maximize the improvement $a'_v - a_v$ of v ? Clearly, a trade with optimal α maximizes the total assets a'_v . Since $a'_w = a_w$ in every creditor-positive trade, maximizing a'_v also maximizes the payments of v , the incoming payments of v 's creditors, and, inductively, the payments and assets of every edge and bank in the network. A creditor-positive α that maximizes a'_v also simultaneously maximizes (1) the Pareto-improvement of payments for each edge in the network, and (2) the return ρ by w . This holds for all networks with monotone payment functions.

To answer the above questions, we modify \mathcal{F} into a *return network* \mathcal{F}^{ret} defined as follows. We switch edge e to $\text{cr}(e) = w$ and add a *return edge* e_r with $\text{de}(e_r) = w$ and $\text{cr}(e_r) = v$. The payment on this edge models the return from w to v , so the liability is $\ell_{e_r} = \min\{\ell_e, a_w^x\}$. Since we consider creditor-positive trades, we modify the payment function of w as follows. For all $e' \in E^+(w) \setminus \{e_r\}$, we set $f_{e'}^{\text{ret}}(x) = f_{e'}(x)$ for all $x \leq a_w$ and $f_{e'}^{\text{ret}}(x) = f_{e'}(a_w)$ for all $x \geq a_w$. For e_r we set $f_{e_r}^{\text{ret}}(x) = 0$ for all $x \leq a_w$ and $f_{e_r}^{\text{ret}}(x) = x - a_w$ for all $x \geq a_w$. Similarly, we modify the liabilities to $\ell_{e'} = f_{e'}(a_w)$. Intuitively, in \mathcal{F}^{ret} w maintains its payments up to a total outgoing assets of a_w . It forwards any assets exceeding a_w as return via e_r to v .

► **Lemma 7.** *Consider the clearing state \mathbf{p}^{ret} in \mathcal{F}^{ret} .*

- (a) *Suppose there is an optimal creditor-positive α with return $\rho = \alpha \ell_e$. Then \mathcal{F}^{ret} has $a_w^x > p_e$. In \mathbf{p}^{ret} we obtain assets of $a_w^{\text{ret}} \in (a_w, a_w + \ell_{e_r}]$ and $a_v^{\text{ret}} > a_v$, and $p_{e_r}^{\text{ret}} = \rho$.*
- (b) *If $a_w^x > p_e$ and \mathbf{p}^{ret} yields assets of $a_w^{\text{ret}} \in (a_w, a_w + \ell_{e_r}]$ and $a_v^{\text{ret}} > a_v$, then payment p_{e_r} represents a return of an optimal creditor-positive trade.*

Proof. We first show (a). Suppose there is an optimal creditor-positive α . It results in a return $\rho = \alpha \ell_e \leq \min\{\ell_e, a_w^x\} = \ell_{e_r}$, assets of a_w for w , and $a'_v > a_v$ for v . When we assign payments $\hat{p}_e = p'_e$ for all $e' \in E$ and set the payment on e_r to $\hat{p}_{e_r} = \rho$, we obtain a vector of payments $\hat{\mathbf{p}}$ in \mathcal{F}^{ret} that satisfies all fixed-point conditions.

We first show that this implies $a_w^x > p_e$, the payment on e in \mathbf{p} before the trade. Consider the assets of w . We have $\hat{a}_v = a'_v > a_v$. Recall $\mathbf{p}' \geq \mathbf{p}$ by Corollary 6, so

$$\begin{aligned} \hat{a}_v &= a_v^x + \rho + \sum_{e' \in E^-(v) \setminus \{e\}} \hat{p}_{e'} = a_v^x + \rho + \sum_{e' \in E^-(v) \setminus \{e\}} p'_{e'} \\ &\geq a_v^x + \rho + \sum_{e' \in E^-(v) \setminus \{e\}} p_{e'} = a_v + \rho - p_e \end{aligned}$$

Hence $a_w^x \geq \rho > p_e$, as desired.

For the other conditions, consider the clearing state \mathbf{p}^{ret} in \mathcal{F}^{ret} . Due to maximality of the clearing state, $\mathbf{p}^{\text{ret}} \geq \hat{\mathbf{p}}$. Thus, $a_w^{\text{ret}} > a_w$, $a_v^{\text{ret}} > a_v$ and $p_{e_r}^{\text{ret}} \geq \rho$. We show that, indeed, $\hat{\mathbf{p}} = \mathbf{p}^{\text{ret}}$, and that the condition $a_w + \ell_{e_r} \geq \hat{a}_w$ is satisfied.

Case 1: The clearing state satisfies $a_w + \ell_{e_r} \geq a_w^{\text{ret}}$. Then we prove below that \mathbf{p}^{ret} is equivalent to a creditor-positive trade with payments that Pareto-dominate $\hat{\mathbf{p}}$ and, consequently, higher assets for v with $\hat{a}_v \geq a_v^{\text{ret}}$. As such, \mathbf{p}^{ret} represents a better creditor-positive trade, a contradiction to $\hat{\mathbf{p}}$ stemming from an optimal one.

Case 2: The clearing state satisfies $a_w + \ell_{e_r} < a_w^{\text{ret}}$. Then $a_v^{\text{ret}} > a_v$, and w is solvent in \mathcal{F}^{ret} . Indeed, w could transfer even more assets to the edges of $E^+(w) \setminus \{e_r\}$. This implies that with return ℓ_{e_r} , there is a clearing state in \mathcal{F}' that can *strictly improve both* v and w . This is a contradiction to Corollary 3.

To prove (b), suppose \mathbf{p}^{ret} fulfills the conditions. Then, clearly, the payment $p_{e_r}^{\text{ret}}$ represents a feasible return. The payments $p_{e'}^{\text{ret}}$ on the other edges $e' \in E \setminus \{e_r\}$ fulfill the fixed-point conditions in \mathcal{F}' . Now for contradiction assume that $p_{e'}^{\text{ret}} > p_{e'}^{\text{ret}}$ for some e' . Then $e' \neq e_r$, since we assume $p_{e_r}^{\text{ret}}$ is the return used to construct \mathcal{F}' . Hence, any strict increase in \mathbf{p}' could be manifested in \mathbf{p}^{ret} as well, which contradicts the maximality of \mathbf{p}^{ret} in \mathcal{F}^{ret} . ◀

► **Corollary 8.** *Consider a given single claims trade of e to w .*

- (a) *A return of $a_w^x \geq \rho > p_e$ is necessary to make the trade creditor-positive. For $\rho = p_e$, we obtain $\mathbf{p}' = \mathbf{p}$.*
- (b) *Consider all creditor-positive α . A value α with return $\rho = \alpha \ell_e$ maximizes the assets of v if and only if it maximizes the payment on every single edge in \mathcal{F}' , the assets of every single bank, as well as the value of ρ and α .*

► **Proposition 9.** *For a given financial network with edge-ranking payments and a single claims trade, there is an efficient algorithm to compute an optimal creditor-positive $\alpha^* \in [0, 1]$ or decide that none exists.*

Proof. We construct network \mathcal{F}^{ret} as described above. Observe that the adjusted payment function $\mathbf{f}_w^{\text{ret}}$ is again an edge-ranking function – it first fills edges according to \mathbf{f}_w until assets a_w are paid. Thus, at most one edge $e' \in E^+(w)$ is paid partially. For this edge, the liabilities are reduced to $f_{e'}(a_w)$. For all other edges, the liabilities either remain untouched or are decreased to 0. Then the additional assets are directed to e_r . Thus, $\mathbf{f}_w^{\text{ret}}$ can be represented by the same ranking as \mathbf{f}_w up to (and including) edge e' , and then using e_r as the next (and last) edge in the order. Hence, we can compute \mathcal{F}^{ret} in strongly polynomial time. By checking the conditions of Lemma 7, we can verify in polynomial time whether or not a creditor-positive trade exists and obtain the optimal return as the payment on e_r . ◀

► **Proposition 10.** *For a given financial network with proportional payments and a single claims trade, there is an efficient algorithm to compute an optimal creditor-positive $\alpha^* \in [0, 1]$ or decide that none exists.*

Finally, our main result in this section shows that for general monotone payments with efficient clearing oracle, we can obtain an approximately optimal solution via binary search.

► **Theorem 11.** *Consider a given financial network with monotone payment functions and efficient clearing oracle. For a given single claims trade, there exists an additive FPTAS for approximating the optimal improvement of v from any creditor-positive α .*

Our algorithm uses binary search. Towards this end, we first show, for a given target value $A \geq a_v$, how to verify the existence of a trade that achieves at least a value A for the total assets of v . For intuition, we use a *split network* \mathcal{F}^{SP} obtained from \mathcal{F}' after the trade as follows: We replace v and w by source and sink banks v_{in}, v_{out} , and w_{in}, w_{out} . v_{in} has the incoming edges of v , w_{in} the ones of w (including e). The outgoing edges of v (w) are attached to v_{out} (w_{out}). We set the external assets of v_{out} and w_{out} to A and a_w , and these banks use the payment functions of v and w , respectively. As such, the clearing state \mathbf{p}^{SP} in \mathcal{F}^{SP} can be computed using the clearing oracle.

Consider the incoming payments of \mathbf{p}^{SP} at v_{in} and w_{in} . These payments shall exactly recover the expenses at v_{out} and w_{out} – modulo external assets and the return payment from w to v . We define the *budget difference* by

$$d_w^{\text{SP}} = \left(a_w^x + \sum_{e' \in E^-(w_{in})} p_{e'}^{\text{SP}} \right) - a_w \quad \text{and} \quad d_v^{\text{SP}} = A - \left(a_v^x + \sum_{e' \in E^-(v_{in})} p_{e'}^{\text{SP}} \right).$$

d_w^{SP} is the surplus money earned by w_{in} that shall be invested into the return, d_v^{SP} is the excess money spent by v_{out} that must be recovered through the return.

► **Lemma 12.** *For a given single claims trade and a given target value $A > a_v$, there is a creditor-positive trade with a value at least A for v if and only if $d_v^{\text{SP}} = d_w^{\text{SP}} > 0$.*

Proof. We first show that if $d_v^{\text{SP}} = d_w^{\text{SP}} > 0$, then there exists a creditor-positive trade with asset at least A for v . Suppose we consider \mathbf{p}^{SP} in the network \mathcal{F}' using return $\rho = d_v^{\text{SP}} = d_w^{\text{SP}}$. This exactly equilibrates the budgets of v and w – v receives d_v^{SP} , the money needed to obtain total assets of A . Also, w spends exactly d_w^{SP} , the money needed to obtain total assets of a_w . Hence, \mathbf{p}^{SP} satisfies all fixed-point conditions in \mathcal{F}' . As such, $\mathbf{p}' \geq \mathbf{p}^{\text{SP}}$ coordinate-wise due to maximality of the clearing state. This implies that using return ρ , the clearing state \mathbf{p}' yields $a'_v \geq A > a_v$ and $a'_w \geq a_w$. A creditor-positive trade with return ρ exists.

Now for the other direction, consider an optimal creditor-positive trade, which yields the highest asset level A^* and consider any $A \in (a_v, A^*]$. We show that in this case $d_v^{\text{SP}} = d_w^{\text{SP}} > 0$ holds in the clearing state \mathbf{p}^{SP} of \mathcal{F}^{SP} with external assets A for v_{out} .

Consider the optimal trade, its return $\rho^* > 0$ and the emerging payments \mathbf{p}^* in \mathcal{F}' after this trade. Now in the corresponding split network $\mathcal{F}^{*,\text{SP}}$ with external assets of A^* for v_{out} , the payments \mathbf{p}^* yield $d_v^* = d_w^* = \rho^*$, by definition of \mathbf{p}^* . The previous paragraph and maximality of A^* then imply that \mathbf{p}^* must also be the clearing state $\mathbf{p}^{*,\text{SP}} = \mathbf{p}^*$ of $\mathcal{F}^{*,\text{SP}}$.

Now suppose in $\mathcal{F}^{*,\text{SP}}$ we reduce the external assets of v_{out} by $\varepsilon = A^* - A > 0$. Then \mathcal{F}^{SP} evolves. Since we reduce the assets of a single source v_{out} by ε , we obtain $\mathbf{p}^{*,\text{SP}} \geq \mathbf{p}^{\text{SP}}$. Moreover, by non-expansivity [7, Lemma 33], the total incoming assets of all sinks must reduce by *at most* ε . For the sinks v_{in} and w_{in} we set

$$\varepsilon_v = a_{v_{in}}^{*,\text{SP}} - a_{v_{in}}^{\text{SP}} = \sum_{e' \in E^-(v_{in})} p_{e'}^* - p_{e'}^{\text{SP}} \quad \varepsilon_w = a_{w_{in}}^{*,\text{SP}} - a_{w_{in}}^{\text{SP}} = \sum_{e' \in E^-(w_{in})} p_{e'}^* - p_{e'}^{\text{SP}}$$

and, thus, $d_v^{\text{SP}} = d_v^* - (\varepsilon - \varepsilon_v)$ and $d_w^{\text{SP}} = d_w^* - \varepsilon_w$. Non-expansivity implies $\varepsilon_v + \varepsilon_w \leq \varepsilon$.

First, we observe that $\varepsilon_v + \varepsilon_w < \varepsilon$ is impossible. Then $\varepsilon_w < \varepsilon - \varepsilon_v$, so $d_w^{\text{SP}} > d_v^{\text{SP}}$, i.e., w has more excess money in \mathbf{p}^{SP} than required by v . Consider a return of $\rho = d_v^{\text{SP}}$ and \mathbf{p}^{SP} as payment vector in the resulting network \mathcal{F}' . Then all banks are feasible w.r.t. the fixed-point conditions, except for w which has strictly more income than outgoing assets. Hence, the clearing state satisfies $\mathbf{p}' \geq \mathbf{p}^{\text{SP}}$, $a'_v \geq A > a_v$, and $a'_w > a_w$, a contradiction to Proposition 3.

Second, suppose that $\varepsilon_v + \varepsilon_w = \varepsilon$, then $d_v^{\text{SP}} = d_w^{\text{SP}}$. Then the clearing state \mathbf{p}^{SP} exactly fulfills the fixed-point conditions for all banks in \mathcal{F}' and yields assets $A > a_v$ for v and a_w for w with $\rho = d_v^{\text{SP}}$. Note that $\rho > 0$, since otherwise we contradict the maximality of the initial clearing state \mathbf{p} . Therefore, the existence of a creditor-positive trade with assets $A^* > A$ for v implies that $d_v^{\text{SP}} = d_w^{\text{SP}} > 0$ for \mathbf{p}^{SP} in \mathcal{F}^{SP} emerging from A . ◀

We are now ready to prove Theorem 11.

Proof of Theorem 11. Our algorithm works by testing different target values A for the total assets of v . For a given target value A , we then use Lemma 12 to verify existence of a return ρ achieving at least assets A for v . The maximum achievable assets for v are $M_v = \sum_{e' \in E^-(v)} \ell_{e'} + a_v^x + \min\{a_w^x, \ell_e\}$. We determine the maximal achievable A using binary search on the interval $(a_v, M_v]$.

More formally, we choose $\delta > 0$ and apply binary search over the set $\{a_v + \delta, a_v + 2\delta, \dots, M_v\}$. Verifying the condition in Lemma 12 can be done in polynomial time via a call to the clearing oracle in \mathcal{F}^{SP} . If the algorithm discovers that the condition does not hold for all tested values, then no creditor-positive trade with asset level at least $a_v + \delta$ for v exists. Otherwise, let \hat{A} be largest discovered value for which the test is positive. Then, any value of at least $\hat{A} + \varepsilon$ cannot be achieved for any return ρ . Hence, the optimal achievable total assets of v in any creditor-positive trade are bounded by $A^* \in [\hat{A}, \hat{A} + \delta]$, and the additive approximation follows $\hat{A} - a_v \geq (A^* - a_v) - \delta$.

For the running time, we require at most $\lceil \log_2(1 + (M_v - a_v)/\delta) \rceil$ oracle calls, which is polynomial in the input size and $1/\delta$. ◀

Since the number of possible (single) claims trades in a network is limited by $|E| \cdot |V|$, we can use the algorithm above to compute every creditor-positive trade with an (approximately) optimized haircut rate for a given network in polynomial time.

4 Multi-Trades of Incoming Edges

4.1 Fixed Set of Claims

In this section, we are interested in multi-trades of incoming edges of a creditor bank v to a buyer bank w . This arises naturally, for example, when a high fraction of v 's debtors are in default or v is “too big to fail”. Then bankruptcy of v would cause significant damage to the entire network.

We are given a financial network \mathcal{F} with two distinct banks v and w , and a set C of k incoming edges of v . Suppose the haircut rates α_i can be chosen individually for each $e_i \in C$ as part of the trade. We call a vector $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_k)$ of haircut rates *creditor-positive* if the resulting multi-trade is creditor-positive. Our goal is to select creditor-positive $\alpha_i \in [0, 1]$, for every $i \in [k]$, in order to maximize the improvement of v , i.e., $a'_v - a_v = \sum_{i=1}^k \alpha_i \cdot \ell_{e_i} + \sum_{e' \in E^-(v)} p'_{e'} - \sum_{e' \in E^-(v)} p_{e'}$. Observe that we can restrict our attention to vectors with uniform $\alpha_i = \alpha'$ for all $i \in [k]$ and some $\alpha' \in [0, 1]$ – given any $\boldsymbol{\alpha}$, choose α' with $\alpha'_i = \alpha'$ such that $\alpha' \cdot \sum_{i=1}^k \ell_{e_i} = \sum_{i=1}^k \alpha'_i \cdot \ell_{e_i}$. This results in $\alpha' \in [0, 1]$, the same return, and the same assets of v as for $\boldsymbol{\alpha}$.

Our result is a reduction to single trades.

► **Proposition 13.** *Consider a financial network with monotone payment functions and efficient clearing oracle. For a given multi-trade of incoming edges, there is an additive FPTAS for approximating the optimal improvement of v from any creditor-positive α .*

Proof. Consider a financial network \mathcal{F} with banks v and w and edges C , where $|C| = k$. By Corollary 4, the multi-trade in \mathcal{F} can be modeled by a single claims trade with edge \hat{e} in adjusted network $\hat{\mathcal{F}}$. Invoke the FPTAS to compute a haircut rate α for the single claim in $\hat{\mathcal{F}}$. This results in assets of $\alpha \cdot \sum_{i=1}^k \ell_{e_i} + \sum_{e \in E^-(v)} p'_e$ for v in $\hat{\mathcal{F}}$. Clearly, the same value is obtained with the multi-trade when all haircut rates are set to α , i.e., $\alpha_i = \alpha \forall i \in [k]$. Clearly, this choice of haircut rates also yields an (approximately) optimal solution for the multi-trade. ◀

Combining the insight with Propositions 10 and 9, we obtain the following corollary.

► **Corollary 14.** *Consider a financial network with proportional or edge-ranking payments. For a given multi-trade of incoming edges, there are efficient algorithms to compute an optimal creditor-positive α^* or decide that none exists.*

4.2 Choosing Subsets of Claims

For a fixed pair of creditor v and buyer w , the incoming edges of v yield an exponential number of different edge sets C that might be used for a multi-trade. Thus, a creditor-positive multi-trade *cannot* be derived trivially by checking feasibility for all sets C . For improving the assets of v by a multi-trade with buyer w , the selection of claims to be traded is critical. How can we compute a (near-)optimal set of incoming edges $C \subseteq (E^-(v) \setminus E^+(w))$ for a creditor-positive multi-trade with w such that we maximize the improvement of v ?

The challenge is to find a set of claims C with creditor v and appropriate individual haircut rates α_i , for $e_i \in C$. The resulting multi-trade shall be creditor-positive and yield the maximal improvement for v (over all creditor-positive multi-trades of incoming edges of v).

We show that this problem is NP-hard, for every set of monotone payment functions. Formally, we show it is NP-hard to decide whether creditor v can be *saved* by a creditor-positive multi-trade of incoming edges, i.e., whether total assets of L_v can be achieved. We call this problem INCOMINGSAVE-VR (for variable haircut rates).

In the class of networks we construct for the reduction, every bank has at most one outgoing edge. Hence, all payments will be independent of the payment function that is used. Moreover, once a set of claims C is chosen, finding optimal haircut rates for the multi-trade of C to w is trivial in this class of networks. Hardness arises from the choice of C .

► **Theorem 15.** *INCOMINGSAVE-VR is weakly NP-hard.*

4.2.1 Approximate Claims Trades

Contrasting NP-hardness, we show that the problem to compute a multi-trade improving v by a given amount can be solved efficiently when slightly relaxing the liability condition.

► **Definition 16** (ε -Approximate Multi-Trade). *A multi-trade C with creditor v , buyer w , return $\rho > 0$ and haircut rates $\alpha_i \in [0, 1 + \varepsilon]$, for all $e_i \in C$, is called ε -approximate if $\rho \leq (1 + \varepsilon) \cdot \sum_{e_i \in C} \ell_{e_i}$.*

Consider a *creditor-positive* ε -*approximate* trade. Such a trade (1) strictly increases the assets of v and exactly maintains the ones of w , (2) is affordable by w , i.e., $\rho = \sum_{e_i \in C} \alpha_i \ell_{e_i} \leq a_w^x$, and (3) satisfies exact fixed-point conditions in the emerging clearing state. It is approximate only in the liability condition of the trade.

We construct a bicriteria FPTAS to compute a creditor-positive multi-trade of incoming edges. Suppose $\varepsilon, \delta > 0$ such that $1/\varepsilon$ is polynomial and $1/\delta$ is exponential in the representation size of \mathcal{F} . Our FPTAS guarantees that the computed trade is ε -approximate and yields assets of at least $A^* - \delta$ for v , where A^* are the assets of v resulting from an *optimal exact* creditor-positive trade. The FPTAS uses a connection to the KNAPSACK problem.

We proceed in several steps: First, we consider computing an (exact) trade that achieves a target asset value A for the creditor. For this problem, we derive KNAPSACK-style constraints capturing a set of three necessary and sufficient conditions of a valid creditor-positive trade. We then adapt the dynamic program for KNAPSACK to construct an FPTAS to compute an ε -approximate multi-trade with assets value at least A for v in polynomial time. Finally, we show how to use binary search to find a trade with asset level at least $A^* - \delta$.

Necessary and Sufficient Conditions. As a first step, we consider exact trades that achieve assets of at least A for v . Suppose there is such a trade with a set C of traded edges, and let $k = |C|$. Consider \mathcal{F}' after trade C has occurred with a suitably chosen return. Since C is fixed, by Corollary 4 we can express the outcome of the trade using a single claims trade. Now apply the split network \mathcal{F}^{SP} and Lemma 12. Hence, using

$$d_v^{\text{SP}} = A - \left(a_v^x + \sum_{e' \in (E^-(v) \setminus C)} p'_{e'} \right) \quad \text{and} \quad d_w^{\text{SP}} = \left(a_w^x + \sum_{e' \in (E^-(w) \cup C)} p'_{e'} \right) - a_w \quad ,$$

a creditor-positive trade with set C and asset value at least A exists if and only if $d_v^{\text{SP}} = d_w^{\text{SP}} > 0$. This implies that

$$(A - a_v^x) + (a_w - a_w^x) = \sum_{e \in E^-(v) \cup E^-(w)} p'_e \quad (1)$$

must hold. This condition is *independent* of the set C of traded edges. As such (1) is a necessary condition that any creditor-positive trade with asset level at least A for v can exist.

With return $\rho = d_v^{\text{SP}}$ for the given set C , we satisfy the fixed-point conditions in \mathcal{F}' . By Lemma 12 the optimal trade using the given set C only yields a larger return, i.e., $\rho \geq d_v^{\text{SP}} > 0$. Moreover, the liabilities of the traded edges must be high enough to allow the return ρ , i.e., $\sum_{e_i \in C} \ell_{e_i} \geq \rho$. Using $P' = \sum_{e \in E^-(v)} p'_e$ this necessary condition is expressed by

$$\sum_{e \in C} \ell_e \geq d_v^{\text{SP}} = A - a_v^x - P' + \sum_{e \in C} p'_e \quad \iff \quad \sum_{e \in C} (\ell_e - p'_e) \geq A - a_v^x - P' \quad . \quad (2)$$

w must be able to pay the required return. Since the return is solely funded by external assets, we obtain the necessary condition

$$a_w^x \geq \rho = d_v^{\text{SP}} \quad \iff \quad \sum_{e \in C} p'_e \leq a_w^x - A + a_v^x + P' \quad . \quad (3)$$

While each condition (1)-(3) is necessary, it is easy to see that in combination they are sufficient. Indeed, if they hold, then there is a creditor-positive trade of set C with return $\rho \geq d_v^{\text{SP}} = d_w^{\text{SP}} > 0$ that respects the exact liabilities of traded edges, is affordable by w , and achieves asset level at least A for v . We summarize the argument in the following lemma:

► **Lemma 17.** For a given set C of incoming edges of v , the following are equivalent:

1. There is a multi-trade of C to w that achieves an asset level at least A for v .
2. Equation (1) holds, and the set C satisfies (2) and (3).

Knapsack-Style FPTAS. While condition (1) can be checked directly after computing the clearing state \mathbf{p}' , determining the existence of a set C that satisfies conditions (2) and (3) can be cast as a KNAPSACK decision problem: For each edge $e \in E^-(v)$ the payments p'_e are the non-negative *weight* of e , and the residual $\ell_e - p'_e$ is the non-negative *value* of e . Decide the existence of a subset of edges with total value lower bounded by (2) and total weight upper bounded by (3).

We next adapt the standard FPTAS for KNAPSACK to compute an *approximate* multi-trade. We *round up* the residual $\ell_e - p'_e$ of every edge to the next multiple of a parameter s . This can be interpreted as increasing the liabilities ℓ_e by a small amount. We then determine if (2) and (3) allow a feasible solution by using the standard dynamic program for KNAPSACK in polynomial time. We term this procedure the Level-FPTAS.

► **Lemma 18 (Level-FPTAS).** For a given number A , suppose there exists a creditor-positive multi-trade of incoming edges that yields assets at least A for v . Then, for every $\varepsilon > 0$, there is an algorithm to compute an ε -approximate creditor-positive trade with assets at least A for v in time polynomial in the size of \mathcal{F} and $1/\varepsilon$.

Proof. First, check feasibility of condition (1) since otherwise the desired trade does not exist. Then, consider all incoming edges $e \in E^-(v)$ of v and define $m = |E^-(v)|$. We denote by $r_e = \ell_e - p'_e$ the residual of e . Let r_{max} be the maximal residual capacity with respect to \mathbf{p}' of any edge that satisfies the weight constraint, i.e., $r_{max} = \max\{r_e \mid e \in E^-(v), p'_e \leq a_w^x - A + a_v^x + P'\}$. Round the residual capacities *up* using a scaling factor $s = \frac{\varepsilon \cdot r_{max}}{m}$. Then determine the optimal solution C^* for the rounded values $\tilde{r}_e = s \cdot \lceil (r_e)/s \rceil$ via the standard dynamic program for KNAPSACK. The running time is bounded by $O(m^3/\varepsilon)$.

Using a return of $\rho = A - \left(a_v^x + \sum_{e \in E^-(v) \setminus C^*} p'_e\right)$ the trade of C^* yields a clearing state in \mathcal{F}' with assets at least A for v . By definition, C^* satisfies (3), and since the instance satisfies (1), $\rho \leq a_w^x$ is also guaranteed.

Regarding the liabilities, note that

$$\sum_{e \in C^*} \tilde{r}_e \leq \sum_{e \in C^*} r_e + s \leq \varepsilon r_{max} + \sum_{e \in C^*} r_e \leq (1 + \varepsilon) \sum_{e \in C^*} r_e \leq \sum_{e \in C^*} \ell_e (1 + \varepsilon) - p'_e .$$

There exists an exact trade C with assets at least A for v , so the trade with C satisfies (2) and (3). As such,

$$\sum_{e \in C^*} \tilde{r}_e \geq \sum_{e \in C} \tilde{r}_e \geq \sum_{e \in C} r_e \geq A - a_v^x - P' ,$$

so the optimal solution C^* satisfies (2) using the rounded residuals. Hence,

$$\begin{aligned} \rho &= A - \left(a_v^x + \sum_{e \in E^-(v) \setminus C^*} p'_e \right) = A - a_v^x - P' + \sum_{e \in C^*} p'_e \leq \sum_{e \in C^*} \tilde{r}_e + p'_e \\ &\leq (1 + \varepsilon) \sum_{e \in C^*} \ell_e , \end{aligned}$$

so the return generated by C^* violates the liability condition by at most a factor of $1 + \varepsilon$. ◀

The lemma gives rise to an efficient algorithm computing an ε -approximate multi-trade with assets at least A , whenever an exact trade with assets at least A exists. Similar to Theorem 11, we use this test to construct a bicriteria FPTAS.

Bicriteria-FPTAS. As our main result for multi-trades of incoming edges, we obtain a bicriteria FPTAS. Suppose assets A^* for v are achievable by an exact creditor-positive multi-trade of incoming edges. We will compute an ε -approximate one resulting in assets at least $A^* - \delta$ for v , for any $\varepsilon, \delta > 0$. We say such a trade is δ -optimal.

► **Theorem 19.** *Consider a financial network with monotone payment functions and efficient clearing oracle, creditor v and buyer w . If there exists a creditor-positive multi-trade of incoming edges, then an ε -approximate δ -optimal trade can be computed in time polynomial in the size of \mathcal{F} , $1/\varepsilon$ and $\log 1/\delta$, for every $\varepsilon, \delta > 0$,*

Proof. We use the binary search idea put forward in Theorem 11. We choose $\delta > 0$ and apply binary search over the set $\{a_v + \delta, a_v + 2\delta, \dots, M_v\}$ of potential asset values for v . Recall that M_v is an upper bound for maximal achievable assets of v . For multi-trades, M_v is upper bounded by $a_v^x + a_w^x + \sum_{e \in E^-(v)} \ell_e$. The goal is to find an asset value that is as large as possible.

Running the Level-FPTAS with any value from the interval $A' \in (a_v, A^*]$, we are guaranteed to receive an approximate multi-trade with asset level at least A' for v in polynomial time. As such, the binary search will never terminate with a value of $A' \leq A^* - \delta$. The search terminates in at most $\lceil \log_2(1 + (M_v - a_v)/\delta) \rceil$ steps. ◀

When called with an asset level $A' > A^*$, the Level-FPTAS might or might not return a corresponding multi-trade – rounding up the residuals can introduce non-monotone behavior. As such, using binary search our algorithm does not necessarily return an optimal asset value of v for any creditor-positive ε -approximate multi-trade. However, since the Level-FPTAS never fails to return a multi-trade for any asset level $A' \leq A^*$, we are guaranteed that assets of more than $A^* - \delta$ for v are achieved.

Ranking Payments. For edge-ranking payments, the set of meaningful values to be tested for A^* in the binary search can be restricted to a grid of at most exponential precision in the input size. This allows to compute an ε -approximate multi-trade with assets at least A^* , i.e., such a trade is δ -optimal with $\delta = 0$.

► **Corollary 20.** *Consider a financial network with edge-ranking functions, creditor v and buyer w . If there exists a creditor-positive multi-trade of incoming edges, then an ε -approximate 0-optimal trade can be computed in time polynomial in the size of \mathcal{F} and $1/\varepsilon$.*

Proof. Consider an optimal exact multi-trade C with return ρ that achieves optimal assets of A^* for v . Recall that all liabilities and external assets are integers, and so is M_v . If A^* is integral, then we can run the binary search with $\delta = 1$ and obtain an approximate trade with assets of (more than $A^* - 1$ and, thus) at least A^* for v .

To show that A^* is integral, consider an optimal creditor-positive trade C achieving assets A^* . We resort to the equivalent representation as a single claims trade (Corollary 4). For this single trade, consider the return network \mathcal{F}^{ret} . An optimal return ρ for a trade of edge set C evolves as the payment $p_{e_r}^{\text{ret}}$ on e_r in the clearing state \mathbf{p}^{ret} . Recall that the payment function of w in the return network is also an edge-ranking function (c.f. Proposition 9). For edge-ranking payments, if all liabilities and external assets are integers, then the clearing state has integral payments [1]. The assets of every bank in \mathbf{p}^{ret} (and A^*) are integral. ◀

5 Multi-Trades of Outgoing Edges

In this section, we study multi-trades of *outgoing* edges of a bankrupt bank u . We strive to improve the assets of u 's creditors directly (and not indirectly via u through trades of incoming edges). It might not be feasible to save a particular bank u , e.g., when its debt is too high in relation to the claims. In such cases, we attempt to minimize the *contagion* of bankruptcy from u to her creditors by conducting multi-trades of outgoing edges of u . We execute multi-trades that *maximize* the total profit of all creditors, not just those involved in the trade. No creditor nor buyer w should be harmed by the trade.

► **Definition 21** (Pareto-positive trade). *Let v_1, v_2, \dots, v_l be u 's creditors. A multi-trade of outgoing edges of u to w is called Pareto-positive, if $a'_{v_i} > a_{v_i}$ for at least one creditor v_i , $a'_{v_i} \geq a_{v_i}$ for all creditors and $a'_w \geq a_w$.*

Suppose we are given a financial network with banks u, w and a set C of k outgoing edges of u . Denote the creditors of edges C by $V_C = \{v_i \mid e_i \in C, \text{cr}(e_i) = v_i\}$. A collection of haircut rates $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$ is called *Pareto-positive* if C together with α forms a Pareto-positive multi-trade. The objective is to derive the optimal values of α which maximize the sum of profit of creditors v_1, v_2, \dots, v_l , i.e., $\max \sum_{i=1}^l a'_{v_i} - a_{v_i}$.

Consider the problem where set C is not given as part of the input but is chosen as part of the solution. The goal is to select a subset of u 's outgoing edges $C \subseteq E^+(u)$ together with a vector of haircut rates $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_{|C|})$ such that the multi-trade is Pareto-positive and maximizes the improvement of u 's creditors, i.e., $\sum_{i=1}^l a'_{v_i} - a_{v_i}$.

In the previous section, we obtained a bicriteria FPTAS for this problem when we trade incoming edges of an insolvent bank. Interestingly, the results hold for all monotone payment functions for which there is an efficient clearing oracle (e.g., edge-ranking or proportional payments). Our results here show a strong contrast – depending on the payment functions trading outgoing edges can be much harder. For edge-ranking functions (and variable haircut rates), we denote the problem by OUTGOINGER-VR.

► **Theorem 22.** *OUTGOINGER-VR is strongly NP-hard. For any constant $\varepsilon > 0$ there exists no efficient $n^{1/2-\varepsilon}$ -approximation algorithm for OUTGOINGER-VR unless $P = NP$.*

Now suppose the set of traded edges C is given as part of the input. Interestingly, the hardness for edge-ranking payments continues to apply when C is fixed a priori.

► **Corollary 23.** *Consider a financial network with banks u, w , a set of outgoing edges C of u and edge-ranking payment rules. It is strongly NP-hard to determine Pareto-positive haircut rates that maximize the sum of profits of u 's creditors. For any constant $\varepsilon > 0$, there exists no efficient $n^{1/2-\varepsilon}$ -approximation algorithm unless $NP = P$.*

Finally, we briefly observe that these problems for outgoing edges depend crucially on the set of payment functions. For proportional payments (and variable α), the problem for a given set C can be solved efficiently (even if the set C of edges involves different debtors). When the set C of outgoing edges is chosen as part of the solution, we refer to the problem as OUTGOINGPROP-VR and obtain NP-hardness. The approximability status of these problems for different payment functions is an interesting direction for future work.

► **Proposition 24.** *For a given financial network with proportional payments and a set C of k edges, there exists an efficient algorithm that computes an optimal Pareto-positive $\alpha^* \in [0, 1]^k$ or decides that none exists.*

► **Theorem 25.** *OUTGOINGPROP-VR is strongly NP-hard.*

References

- 1 Nils Bertschinger, Martin Hoefler, and Daniel Schmand. Strategic payments in financial networks. In *Proc. 11th Symp. Innov. Theoret. Comput. Sci. (ITCS)*, pages 46:1–46:16, 2020.
- 2 Bloomberg. URL: <https://www.bloomberg.com/news/features/2023-03-20/credit-suisse-ubs-takeover-how-a-166-year-old-bank-collapsed>.
- 3 Swiss Federal Council. URL: <https://www.admin.ch/gov/en/start/documentation/media-releases.msg-id-93793.html>.
- 4 Péter Csóka and Jean-Jacques Herings. Decentralized clearing in financial networks. *Manag. Sci.*, 64(10):4681–4699, 2018.
- 5 Larry Eisenberg and Thomas Noe. Systemic risk in financial systems. *Manag. Sci.*, 47(2):236–249, 2001.
- 6 Jared Ellias. Bankruptcy claims trading. *J. Empirical Legal Stud.*, 15(4):772–799, 2018.
- 7 Henri Froese, Martin Hoefler, and Lisa Wilhelmi. The complexity of debt swapping. *arXiv preprint arXiv:2302.11250*, 2023.
- 8 Martin Hoefler and Lisa Wilhelmi. Seniorities and minimal clearing in financial network games. In *Proc. 15th Symp. Algorithmic Game Theory (SAGT)*, pages 187–204, 2022.
- 9 Stavros Ioannidis, Bart de Keijzer, and Carmine Ventre. Financial networks with singleton liability priorities. In *Proc. 15th Symp. Algorithmic Game Theory (SAGT)*, pages 205–222, 2022.
- 10 Stavros Ioannidis, Bart de Keijzer, and Carmine Ventre. Strong approximations and irrationality in financial networks with derivatives. In *Proc. 49th Int. Colloq. Autom. Lang. Programming (ICALP)*, pages 76:1–76:18, 2022.
- 11 Panagiotis Kanellopoulos, Maria Kyropoulou, and Hao Zhou. Financial network games. In *Proc. 2nd Int. Conf. AI in Finance (ICAIF)*, pages 26:1–26:9, 2021.
- 12 Panagiotis Kanellopoulos, Maria Kyropoulou, and Hao Zhou. Forgiving debt in financial network games. In *Proc. 31st Int. Joint Conf. Artif. Intell. (IJCAI)*, pages 335–341, 2022.
- 13 Panagiotis Kanellopoulos, Maria Kyropoulou, and Hao Zhou. Debt transfers in financial networks: Complexity and equilibria. In *Proc. 22nd Conf. Auton. Agents and Multi-Agent Syst. (AAMAS)*, pages 260–268, 2023.
- 14 Adam Levitin. Bankruptcy markets: Making sense of claims trading. *Brooklyn J. Corporate, Financial & Commercial Law*, 4(1), 2009.
- 15 Katherine Mayo and Michael P. Wellman. A strategic analysis of portfolio compression. In *Proc. 2nd Int. Conf. AI in Finance (ICAIF)*, pages 20:1–20:8, 2021.
- 16 Marios Papachristou and Jon Kleinberg. Allocating stimulus checks in times of crisis. In *Proc. 31th World Wide Web Conf. (WWW)*, pages 16–26, 2022.
- 17 Pál András Papp and Roger Wattenhofer. Network-aware strategies in financial systems. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. 47th Int. Colloq. Autom. Lang. Programming (ICALP)*, pages 91:1–91:17, 2020.
- 18 Pál András Papp and Roger Wattenhofer. Debt swapping for risk mitigation in financial networks. In *Proc. 22nd Conf. Econ. Comput. (EC)*, pages 765–784, 2021.
- 19 Pál András Papp and Roger Wattenhofer. Default ambiguity: Finding the best solution to the clearing problem. In *Proc. 17th Conf. Web and Internet Econ. (WINE)*, pages 391–409, 2021.
- 20 Leonard Rogers and Luitgard Veraart. Failure and rescue in an interbank network. *Manag. Sci.*, 59(4):882–898, 2013.
- 21 Steffen Schuldenzucker and Sven Seuken. Portfolio compression in financial networks: Incentives and systemic risk. In *Proc. 21st Conf. Econ. Comput. (EC)*, page 79, 2020.
- 22 Steffen Schuldenzucker, Sven Seuken, and Stefano Battiston. Clearing payments in financial networks with credit default swaps. In *Proc. 17th Conf. Econ. Comput. (EC)*, page 759, 2016.
- 23 Steffen Schuldenzucker, Sven Seuken, and Stefano Battiston. Finding clearing payments in financial networks with credit default swaps is ppad-complete. In *Proc. 8th Symp. Innov. Theoret. Comput. Sci. (ITCS)*, pages 32:1–32:20, 2017.
- 24 Luitgard Veraart. When does portfolio compression reduce systemic risk? *Math. Finance*, 32(3):727–778, 2022.

A Faster Algorithm for Constructing the Frequency Difference Consensus Tree

Jesper Jansson ✉

Kyoto University, Japan

Wing-Kin Sung ✉

The Chinese University of Hong Kong, China

Hong Kong Genome Institute, Hong Kong Science Park, China

Seyed Ali Tabatabaee ✉

Dept. of Computer Science, University of British Columbia, Vancouver, Canada

Yutong Yang ✉

Hong Kong Genome Institute, Hong Kong Science Park, China

Abstract

A consensus tree is a phylogenetic tree that summarizes the evolutionary relationships inferred from a collection of phylogenetic trees with the same set of leaf labels. Among the many types of consensus trees that have been proposed in the last 50 years, the frequency difference consensus tree is one of the more finely resolved types that retains a large amount of information. This paper presents a new deterministic algorithm for constructing the frequency difference consensus tree. Given k phylogenetic trees with identical sets of n leaf labels, it runs in $O(kn \log n)$ time, improving the best previously known solution.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Applied computing → Bioinformatics

Keywords and phrases phylogenetic tree, frequency difference consensus tree, tree algorithm, centroid path decomposition, max-Manhattan Skyline Problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.43

Supplementary Material *Software*: https://github.com/tswddd2/FDCT_new
archived at `swh:1:dir:7cb120cbc61221f740e9e824e93bce0bbf64270a`

Funding This work was partially funded by JSPS KAKENHI grant 22H03550 and NSERC Discovery Grants.

Acknowledgements The authors would like to thank Varun Gupta for some ideas employed in the procedure `Fast_Label_Trees`.

1 Introduction

In phylogenetic analysis, variations in datasets, algorithms, and models of evolution typically yield different phylogenetic trees. Hence, researchers often need to analyze a collection of phylogenetic trees with the same set of leaf labels but different branching structures, and to this end, they use consensus trees. A consensus tree is a single phylogenetic tree that represents a collection of phylogenetic trees, aiming to highlight the commonly agreed-upon parts of the evolutionary history. Consensus trees have applications across various fields of science, including biology, evolutionary studies, epidemiology, and ecology. Many alternative consensus trees, each with its strengths and limitations, have been proposed; see, e.g., [7].

The *frequency difference consensus tree* (FDCT) [16] has garnered interest among researchers in recent years [15, 17, 20]. Given k phylogenetic trees with identical sets of n leaf labels, the FDCT is a phylogenetic tree consisting of each cluster that occurs more frequently



© Jesper Jansson, Wing-Kin Sung, Seyed Ali Tabatabaee, and Yutong Yang;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 43; pp. 43:1–43:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



in the input trees than any single cluster incompatible with it. In this context, a cluster refers to any nonempty subset of the leaf label set and is said to occur in a phylogenetic tree iff it corresponds to the set of all leaf labels descending from a single node of the tree. Furthermore, two clusters are deemed incompatible if they cannot simultaneously occur in the same phylogenetic tree. The advantage of the FDCT compared to some other popular types of consensus trees, such as the strict consensus tree [30] and the majority rule consensus tree [25], is that it captures more of the shared branching information and has more clusters.

It is evident that $\Omega(kn)$ serves as a lower bound for the running time of any algorithm aiming to build the FDCT, given that it corresponds to the input size. Unlike certain other types of consensus trees such as the strict consensus tree and the majority rule consensus tree, there has been no algorithm proposed to construct the FDCT that can achieve a running time matching this lower bound. Before this paper, the $O(kn \log^2 n)$ -time algorithm by Gawrychowski et al. [15] was the asymptotically fastest algorithm for constructing the FDCT. In this paper, we present an $O(kn \log n)$ -time algorithm for constructing the FDCT, thus reducing the gap between the upper and lower bounds for the running time of the fastest algorithm to solve this problem.

The overarching structure of our new algorithm follows the framework proposed by Jansson et al. [20] for computing the FDCT. By improving the methods for solving two subproblems in [20], our algorithm achieves a running time of $O(kn \log n)$. First, our algorithm incorporates a novel divide-and-conquer solution that runs in $O(kn \log n)$ time for the weighting step, where the number of phylogenetic trees in which each cluster occurs is calculated. We remark that algorithms for computing other types of consensus trees such as the *greedy consensus tree* [7, 13] involve the same weighting step [15, 21, 32] and may derive advantages from our improved method. Second, the running time of the procedure `Filter_Clusters` is improved to $O(n \log n)$ by solving instances of the MAX-MANHATTAN SKYLINE PROBLEM [9] to identify the clusters that should be removed at each recursive stage of the algorithm (refer to Sections 3 and 5 for the detailed explanation).

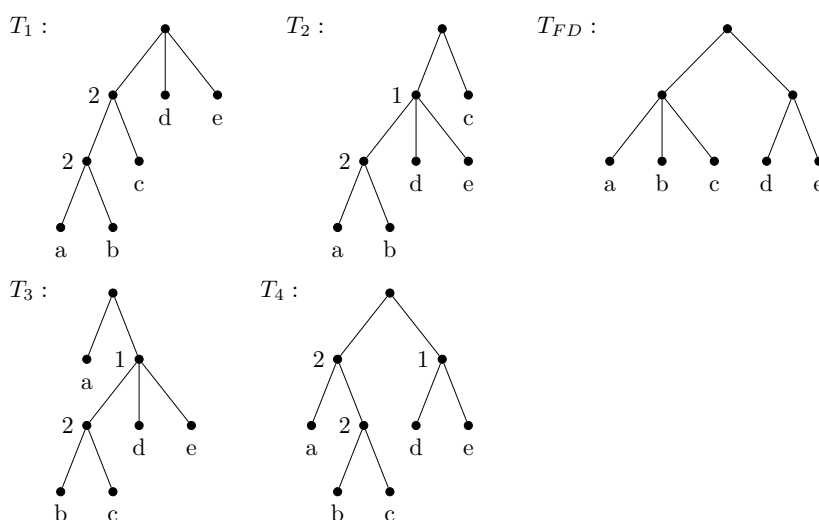
1.1 Definitions and Notation

A *phylogenetic tree* is a rooted tree that represents the evolutionary relationships among different organisms. Every internal node of a phylogenetic tree has at least two unordered children and every leaf has a distinct label. The term *trees* will be employed as a shorthand for *phylogenetic trees* in the remainder of this paper. Let T be some tree. The set of nodes of T is denoted by $V(T)$. Let $\Lambda(T)$ be the set of leaf labels of T . Non-empty subsets of $\Lambda(T)$ are called *clusters*. Clusters with cardinality 1 or $|\Lambda(T)|$ are *trivial clusters*. For any node $u \in V(T)$, $T[u]$ is the subtree of T rooted at u and $\Lambda(T[u])$ is the set of leaf labels of $T[u]$, called the cluster *associated* with u . The *cluster collection* of T , denoted by $\mathcal{C}(T)$, is the set $\bigcup_{u \in V(T)} \{\Lambda(T[u])\}$. Any cluster $C \subseteq \Lambda(T)$ *occurs* in T iff $C \in \mathcal{C}(T)$. For any nodes $u, v \in V(T)$, we denote the lowest common ancestor of u and v in T by $\text{lca}^T(u, v)$. Further, for any non-empty set of nodes $U \subseteq V(T)$, we refer to the lowest common ancestor of all these nodes in T by $\text{lca}^T(U)$.

Any two clusters $C_1, C_2 \subseteq \Lambda(T)$ are said to be *compatible*, written as $C_1 \smile C_2$, iff $C_1 \subseteq C_2$, $C_2 \subseteq C_1$, or $C_1 \cap C_2 = \emptyset$. If C_1 and C_2 satisfy none of the preceding properties, then they are *incompatible*, denoted as $C_1 \not\smile C_2$. Similarly, given trees T_1 and T_2 with identical leaf label sets, and nodes $u \in V(T_1)$ and $v \in V(T_2)$, we say u is compatible with v , denoted as $u \smile v$, if the clusters associated with u and v are compatible. We now extend the notion of compatibility to trees. A cluster $C \subseteq \Lambda(T)$ is *compatible* with T (denoted as $C \smile T$) iff for every $C' \in \mathcal{C}(T)$, we have $C \smile C'$. Further, two trees T_1 and T_2 with

identical leaf label sets are *compatible* (denoted as $T_1 \sim T_2$) iff for every $C \in \mathcal{C}(T_1)$, $C \sim T_2$, i.e. iff every cluster in T_1 is compatible with T_2 . This also means that every cluster in T_2 is compatible with T_1 .

The *frequency difference consensus tree* (FDCT) is defined as follows. Let \mathcal{S} be a set of k trees with identical sets of n leaf labels, i.e. $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ and $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$ (where $|L| = n$). For any cluster $C \subseteq L$, let the *weight* of C , denoted as $w(C)$, be $|\{T : T \in \mathcal{S} \text{ and } C \in \mathcal{C}(T)\}|$, i.e., the number of trees in \mathcal{S} in which C occurs. For any tree $T \in \mathcal{S}$ and any node $u \in V(T)$, we define the weight of node u as $w(u) = w(\Lambda(T[u]))$. Then, the FDCT of \mathcal{S} is the tree T_{FD} , where $\mathcal{C}(T_{FD}) = \{C : C \subseteq L \text{ and } w(C) > \max(\{w(C') : C' \subseteq L \text{ and } C \not\sim C'\})\}$. Thus, T_{FD} contains every cluster that occurs more frequently than any cluster incompatible with it (we refer to such clusters as *frequency difference clusters*). By Proposition 3 in [31], this tree always exists and is unique for a given \mathcal{S} . Figure 1 illustrates the FDCT for a collection of four phylogenetic trees.



■ **Figure 1** Let $\mathcal{S} = \{T_1, T_2, T_3, T_4\}$. T_{FD} is the FDCT of \mathcal{S} . In each T_i , the number beside each non-root internal node u indicates the weight $w(u)$. In this example, T_{FD} does not contain the clusters $\{a, b\}$ and $\{b, c\}$ with weight 2 because they are incompatible with each other. On the other hand, T_{FD} contains the cluster $\{d, e\}$ with weight 1. Furthermore, T_{FD} contains the cluster $\{a, b, c\}$ with weight 2, even though that cluster is incompatible with two input trees.

1.2 Previous Work

A variety of types of consensus trees have been developed over the last half-century, starting with the Adams consensus tree [2] in 1972. Some of these consensus trees are summarized in [7]. Here, we describe two well-studied types of consensus trees: the strict consensus tree [30] and the majority-rule consensus tree [25]. The strict consensus tree keeps only the clusters that occur in all input trees and is easily computed in optimal $O(kn)$ running time [10]. However, some potentially important clusters might be discarded from this consensus tree, if one of the input trees does not contain them. The *majority-rule consensus tree* is a generalization of the previous consensus tree and contains all clusters that occur in more than half of the trees. The majority-rule consensus tree can also be computed in optimal $O(kn)$ time [21].

The *frequency difference consensus tree* (FDCT) was introduced by Goloboff et al. [16] as a more informative alternative that contains not only the clusters that occur in the majority of trees but also the other frequency difference clusters. Dong et al. [11] provided a comparison of the FDCT and a few other types of consensus trees. Barrett et al. [3] employed the idea of using the frequency difference metric while analyzing angiosperm phylogeny. Steel and Velasco [31] investigated a generalization of the FDCT to *supertrees*, i.e. consensus trees built from input trees that do not necessarily have the same leaf label sets. They showed that, unlike some other commonly used definitions, the FDCT easily generalizes to a viable supertree definition. Moreover, the FDCT has been utilized in various other studies over the years [14, 18, 19, 23, 24, 26, 27, 28, 29].

Several research works have focused on computing the FDCT of a set of k trees, each labeled by the same set of n leaf labels. An implementation of the FDCT can be found in the free software package TNT [17]; however, the algorithm employed by TNT and its complexity remain undisclosed. Jansson et al. [20] gave a deterministic $\min\{O(kn^2), O(kn(k + \log^2 n))\}$ -time algorithm for constructing the FDCT. This algorithm was implemented in the open-source FACT package [21] and experimentally shown [20] to be significantly faster than TNT's implementation. Subsequently, Gawrychowski et al. [15] developed a faster method for the weighting step in [20], yielding an improved running time of $O(kn \log^2 n)$ for constructing the FDCT.

1.3 Organization of the Article

This paper is organized as follows. Section 2 contains some results from previous works that are utilized later in this paper. Section 3 gives the framework of the $O(kn \log n)$ -time algorithm for computing the FDCT. Sections 4 and 5 present algorithms for solving the subproblems of the FDCT construction. Finally, Section 6 provides the concluding remarks.

2 Preliminaries

2.1 The *delete* Operation

The *delete* operation on a non-root internal node u in a tree T makes all children of u become children of u 's parent and then removes u along with all edges connected to it. After applying this operation on u , the cluster $\Lambda(T[u])$ is removed from the cluster collection of T . If c denotes the number of u 's children, then the *delete* operation on u takes $O(c)$ time.

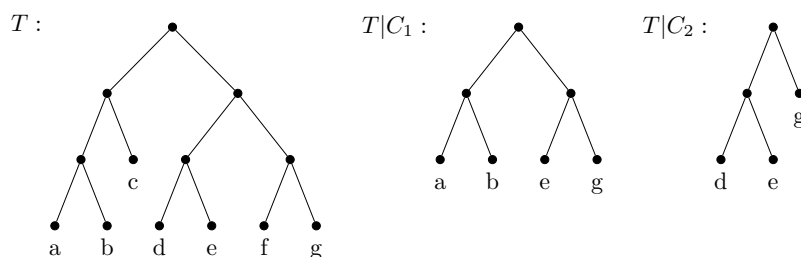
2.2 The Lowest Common Ancestor

We restate the following lemma outlining the *lowest common ancestor* (*lca*) operation from [4]:

► **Lemma 1.** *Given any tree T , the *lca* data structure can be constructed in $O(n)$ time, where $n = |V(T)|$. Then, for any nodes $u, v \in V(T)$, the query $\text{lca}^T(u, v)$ can be answered in constant time.*

2.3 Restriction of Trees

For any tree T and any cluster $C \subseteq \Lambda(T)$, we define $T|C$ (called *T restricted to C*) as the tree T' with $V(T') = \{\text{lca}^T(u, v) : u, v \in C\}$ such that $\text{lca}^T(C') = \text{lca}^{T'}(C')$ for all $C' \subseteq C$. Intuitively, T' has the leaf label set C and consists of all nodes in T that are *lca*'s of the leaves in C , and the ancestral relationships between the nodes in T' are the same as they were in T . Figure 2 provides an example of restricting a tree to different clusters.



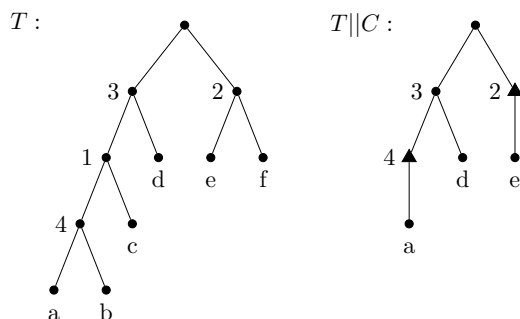
■ **Figure 2** Illustration of restricting a tree to different clusters, where $C_1 = \{a, b, e, g\}$ and $C_2 = \{d, e, g\}$.

2.4 Expanded Restriction of Trees

Recall that every node in the tree T is associated with a weight. For any subset $C \subseteq L$, when we compute the restricted tree $T|C$, some nodes in T are deleted. This leads to losing information about the weights of the clusters associated with these nodes. To address this, we extend the concept of restricted trees following the definition given in [20] and allow some nodes to be marked as *spoiled* (see below for details). For any $C \subseteq \Lambda(T)$, we obtain the weighted tree $T||C$ (called the *expanded restriction of T to C*) as follows:

1. Let $T' = T|C$.
2. For every node u in T' , set the weight of u equal to its weight in T and mark u as spoiled if $\Lambda(T'[u]) \neq \Lambda(T[u])$ or if u is a spoiled node in T .
3. For every edge (u, v) in T' , let P be the path in T between u and v , excluding u and v . If P contains at least one node, then create a new node z in T' (referred to as a *path node*), replace the edge (u, v) with the two edges (u, z) and (z, v) , assign the weight of z to the highest weight among all nodes in P , and mark z as spoiled.
4. Let $T||C = T'$.

Intuitively, a node u in $T|C$ that was not already spoiled in T becomes spoiled in $T||C$ if at least one leaf label in $\Lambda(T[u])$ is not in C . It follows that if a node becomes spoiled in $T||C$, then all of its ancestors become spoiled. Furthermore, every path node is a spoiled node (but not vice versa). The purpose of the path nodes in $T||C$ is to compactly represent the weights of the clusters that were lost when building the restriction of T to C and that may conflict with clusters that are subsets of C . Figure 3 shows an example of the expanded restriction of trees.



■ **Figure 3** Illustration of the expanded restriction of a tree, where $C = \{a, d, e\}$. Internal nodes are labeled with their weights. Nodes represented by triangles are path nodes.

We extend the definition of compatibility to spoiled nodes. Suppose that $C_1, C_2 \subseteq \Lambda(T)$ and that u is a spoiled node in $T|C_1$. We have $C_2 \sim u$ iff C_2 and $\Lambda((T|C_1)[u])$ are disjoint or $C_2 \subseteq \Lambda((T|C_1)[u])$. Observe that if $\Lambda((T|C_1)[u]) \subsetneq C_2$, then $C_2 \not\sim u$, i.e., the set inclusion relations in the definition of compatibility are asymmetric for spoiled nodes.

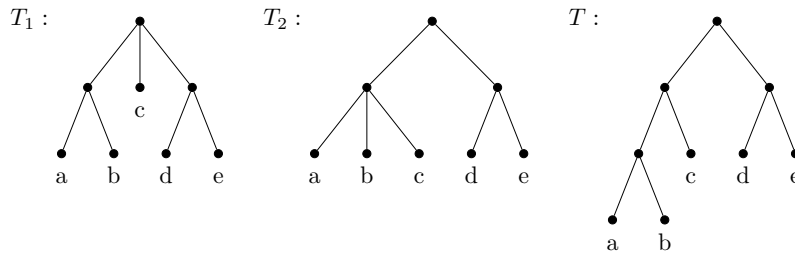
The expanded restrictions can be computed efficiently according to the following lemma:

► **Lemma 2.** *Let T be a weighted phylogenetic tree with n leaves. After $O(n \log n)$ time preprocessing, for any partition C_1, C_2, \dots, C_q of $\Lambda(T)$, the trees $T|C_i$ for all $i \in \{1, 2, \dots, q\}$ can be constructed in a total of $O(n)$ time.*

Proof. By Lemma 5.2 of [12], the trees $T|C_i$ for all $i \in \{1, 2, \dots, q\}$ can be constructed in a total of $O(n)$ time. By Theorem 2 of [22], after an $O(n \log n)$ -time preprocessing of T , the maximum weight of all nodes along the path between any two nodes u and v in T can be found in $O(1)$ time. Furthermore, after an $O(n)$ -time preprocessing of T , the second node on the path from any node u in T to any other node v in T can be found in $O(1)$ time (this can be achieved by finding level ancestors [5, 6]). Hence, for every edge (u, v) in $T|C_i$ (for any $i \in \{1, 2, \dots, q\}$), we can compute the maximum weight of all nodes along the path between u and v in T , excluding u and v , in $O(1)$ time. Consequently, $T|C_i$ can be constructed in $O(|C_i|)$ time from $T|C_i$ for any $i \in \{1, 2, \dots, q\}$, which completes the proof. ◀

2.5 Merging Trees

Given two trees T_1 and T_2 where $\Lambda(T_1) = \Lambda(T_2) = L$ and $T_1 \sim T_2$, `Merge_Trees`(T_1, T_2) returns a tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T) = \mathcal{C}(T_1) \cup \mathcal{C}(T_2)$. Jansson et al. [21] showed that `Merge_Trees` can be computed in $O(|L|)$ time. Figure 4 gives an example for `Merge_Trees`.



■ **Figure 4** Illustration of merging trees where $T = \text{Merge_Trees}(T_1, T_2)$.

2.6 The Max-Manhattan Skyline Problem

Given a set \mathcal{S} of $O(n)$ subintervals of $[1, n - 1]$ with positive integer heights of size $O(n)$, the MAX-MANHATTAN SKYLINE PROBLEM asks for a table f such that for $t \in [1, n - 1]$, $f[t] = \max\{\text{height}([i, j]) : t \in [i, j], [i, j] \in \mathcal{S}\}$. Crochemore *et al.* [9] gave an $O(n)$ -time algorithm for the MIN-MANHATTAN SKYLINE PROBLEM, defined in a similar way as the MAX-MANHATTAN SKYLINE PROBLEM, except that it seeks the minimum height instead of the maximum in the definition of the output table f . Their algorithm first sorts the intervals according to their heights in non-decreasing order. Then, for each interval $[i, j]$ in this order, it sets the values of $f[t]$ for all positions $t \in [i, j]$ that have not yet been assigned a value to $\text{height}([i, j])$. By modifying Crochemore *et al.*'s algorithm [9] to sort the intervals in non-increasing order, we immediately have:

► **Lemma 3.** *The MAX-MANHATTAN SKYLINE PROBLEM can be solved in $O(n)$ time.*

2.7 Centroid Paths and Side Trees

A *centroid path* [8] of a tree T is a path of the form $\pi = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$, where p_α is any node in T , the node p_{i-1} for every $i \in \{2, \dots, \alpha\}$ is any child of p_i with the maximum number of leaf descendants (ties are broken arbitrarily), and p_1 is a leaf. Suppose that π is a centroid path of T . For any $u \in V(T)$ such that u does not belong to π but the parent of u does, $T[u]$ is called a *side tree* of π . From these definitions, we can derive the following lemma:

► **Lemma 4.** *Let T be a tree and τ a side tree of a centroid path that starts at the root of T . Then $|\Lambda(\tau)| \leq |\Lambda(T)|/2$.*

By computing a centroid path π starting at the root of T and recursively applying this procedure to the side trees of π , we obtain a *centroid path decomposition* of T . It follows from Lemma 4 that the number of recursion levels needed to complete such a decomposition is $O(\log |\Lambda(T)|)$.

3 Algorithm Fast_Frequency_Difference

The pseudocode of our new algorithm, named `Fast_Frequency_Difference`, is shown in Algorithm 1. Its overall structure follows the framework developed in [20] for constructing the FDCT, which we review next.

■ **Algorithm 1** The algorithm `Fast_Frequency_Difference` for constructing the FDCT, which maintains the overall structure established by the algorithm `Frequency_Difference` in [20].

Algorithm `Fast_Frequency_Difference`

Input: A set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$.

Output: The frequency difference consensus tree of \mathcal{S} .

```

/* Preprocessing */
1 Fast_Compute_Weights( $\mathcal{S}$ )

/* Main algorithm */
2  $T := T_1$ 
3 for  $j := 2$  to  $k$  do
     $A := \text{Fast_Filter_Clusters}(T, T_j)$ 
     $B := \text{Fast_Filter_Clusters}(T_j, T)$ 
     $T := \text{Merge_Trees}(A, B)$ 
endfor
4 for  $j := 1$  to  $k$  do
     $T := \text{Fast_Filter_Clusters}(T, T_j)$ 
5 return  $T$ 
End Fast_Frequency_Difference

```

Suppose that the input is $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$. The basic idea of the algorithm in [20] is to initially let T be a copy of T_1 and then consider the other trees one by one while updating the clusters of T accordingly. More precisely, when considering any such tree T_j , the algorithm deletes every cluster in T that is incompatible with a cluster in T_j of equal or higher weight, and also inserts every cluster from T_j into T that could potentially be a frequency cluster but is not already in T . This strategy produces a tree T whose set of clusters is a superset of the set of the frequency difference clusters, so the algorithm applies a final postprocessing step to delete all non-frequency difference clusters from T .

To update T in each iteration, the procedure `Merge_Trees`, described in Section 2.5, and a procedure called `Filter_Clusters` are used. The latter takes as input two trees T_A and T_B with identical leaf label sets and outputs a copy of T_A from which every cluster that is incompatible with a cluster in T_B of equal or higher weight has been deleted.

Our new algorithm `Fast_Frequency_Difference` improves the time complexity of the previous algorithm from [20] by replacing the preprocessing step for computing the weights of all clusters occurring in \mathcal{S} (Step 1) and the procedure `Filter_Clusters` (used in Steps 3 and 4) by more efficient solutions, referred to as `Fast_Compute_Weights` and `Fast_Filter_Clusters` below.

In Section 4, we will prove the following theorem concerning the correctness and time complexity of the procedure `Fast_Compute_Weights`:

► **Theorem 5.** *Given a set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with identical sets of n leaf labels, the procedure `Fast_Compute_Weights`(\mathcal{S}) calculates the weights of all clusters occurring in \mathcal{S} in $O(kn \log n)$ time.*

Moreover, Section 5 contains the proof for the following theorem regarding the correctness and time complexity of the procedure `Fast_Filter_Clusters`:

► **Theorem 6.** *Given two weighted trees T_A and T_B with identical sets of n leaf labels, the procedure `Fast_Filter_Clusters`(T_A, T_B) filters out clusters as needed in $O(n \log n)$ time.*

On the grounds of the two theorems stated above, we can prove the main theorem of this paper:

► **Theorem 7.** *Given a set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with identical sets of n leaf labels, the algorithm `Fast_Frequency_Difference`(\mathcal{S}) constructs the FDCT of \mathcal{S} in $O(kn \log n)$ time.*

Proof. Assuming that the improved procedures function as intended, the correctness of `Fast_Frequency_Difference` follows from that of `Frequency_Difference` proved by [20].

Now, we analyze the time complexity of the algorithm. Step 1 makes a call to the procedure `Fast_Compute_Weights`, which takes $O(kn \log n)$ time, according to Theorem 5. Furthermore, Steps 3 and 4 make $O(k)$ calls to the procedures `Fast_Filter_Clusters` and `Merge_Trees`. By Theorem 6, each call to the procedure `Fast_Filter_Clusters` takes $O(n \log n)$ time. Furthermore, the procedure `Merge_Trees` runs in $O(n)$ time [21]. Hence, Steps 3 and 4 take $O(kn \log n)$ time. Consequently, the running time of the algorithm is $O(kn \log n)$. ◀

4 Procedure `Fast_Compute_Weights`

We break down the improved procedure `Fast_Compute_Weights` into two phases called labeling and counting, similar to the strategy used in [15]. The procedure `Fast_Label_Trees` is responsible for the labeling phase. This procedure assigns an integer label to each node u in every tree in \mathcal{S} , denoted by $id(u)$, such that $id(u) \in \{1, \dots, 2kn\}$ and that for any other node u' in any tree in \mathcal{S} , $id(u) = id(u')$ iff the clusters associated with u and u' are the same. Next, during the counting phase, the labels are sorted using counting sort, the count of each distinct label is determined, and the weight of each cluster is obtained from the count of the label of its associated node. Algorithm 2 presents the pseudocode for the procedure `Fast_Compute_Weights`.

■ **Algorithm 2** The procedure `Fast_Compute_Weights`.

Algorithm `Fast_Compute_Weights`

Input: A set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$.

Output: Compute the weight of each cluster C occurring in \mathcal{S} , i.e., the number of trees in \mathcal{S} in which C occurs.

```

/* Labeling phase */
1 Fast_Label_Trees( $\mathcal{S}$ )

/* Counting phase */
2 Sort the obtained labels using counting sort.
3 Determine the count of each distinct label.
4 Set the weight of each cluster to the count of the label of its associated node.

End Fast_Compute_Weights

```

The procedure `Fast_Label_Trees` uses a divide-and-conquer approach to carry out the labeling phase. Let L' and L'' form a partition of L such that the difference between $|L'|$ and $|L''|$ is at most one. `Fast_Label_Trees` recursively calls `Fast_Label_Trees` on $\{T_1|L', \dots, T_k|L'\}$ and $\{T_1|L'', \dots, T_k|L''\}$ to obtain the labels for all nodes in $T_i|L'$ and $T_i|L''$ for all $i \in \{1, 2, \dots, k\}$. Then, for each node u in any tree $T_i \in \mathcal{S}$, the pair $(id(\varphi_u^{L'}), id(\varphi_u^{L''}))$ is assigned to u , where φ_u^C for any $C \subseteq L$ denotes the node in $T_i|C$ that corresponds to u (if such node does not exist in $T_i|C$, then φ_u^C is set to Φ , a special node with $id(\Phi) = 0$). Next, all pairs are sorted using radix sort, and a positive integer rank is assigned to each unique pair. Finally, for each node u in any tree $T_i \in \mathcal{S}$, $id(u)$ is set to the rank of the pair $(id(\varphi_u^{L'}), id(\varphi_u^{L''}))$. The pseudocode for the procedure `Fast_Label_Trees` is given in Algorithm 3. Moreover, Figure 5 illustrates one iteration of `Fast_Label_Trees`.

■ **Algorithm 3** The procedure `Fast_Label_Trees`.

Algorithm `Fast_Label_Trees`

Input: A set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$.

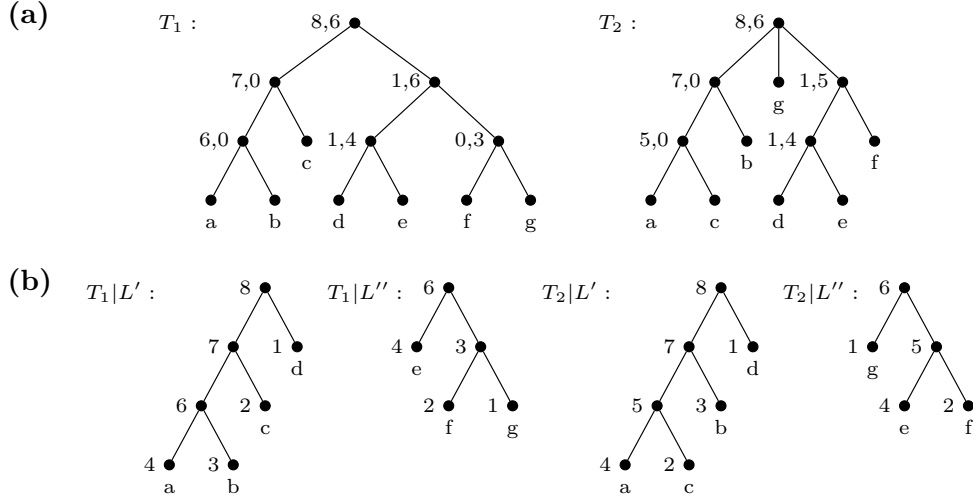
Output: Label each node u in a tree in \mathcal{S} with $id(u) \in \{1, \dots, 2k|L|\}$ such that two nodes in different trees receive the same label iff the clusters associated with them are the same.

```

1 if  $|L| = 1$  then
  /* Base case (each tree has only one node) */
  For each node  $u$  in any tree in  $\mathcal{S}$ , set  $id(u) = 1$ .
  return
endif
2 Partition  $L$  into  $L'$  and  $L''$ , such that the difference between  $|L'|$  and  $|L''|$  is at most one.
3 For all  $i \in [1 \dots k]$ , let  $T'_i = T_i|L'$  and  $T''_i = T_i|L''$ .
4 Fast_Label_Trees( $\{T'_1, T'_2, \dots, T'_k\}$ ).
5 Fast_Label_Trees( $\{T''_1, T''_2, \dots, T''_k\}$ ).
6 For each node  $u$  in any tree  $T_i \in \mathcal{S}$ , assign the pair  $(id(\varphi_u^{L'}), id(\varphi_u^{L''}))$  to  $u$ .
7 Sort all the obtained pairs using radix sort, remove duplicates, and assign a rank to each unique pair.
8 For each node  $u$  in any tree  $T_i \in \mathcal{S}$ , set  $id(u)$  to the rank of the pair  $(id(\varphi_u^{L'}), id(\varphi_u^{L''}))$ .

End Fast_Label_Trees

```



■ **Figure 5** Illustration of one iteration of `Fast_Label_Trees`($\{T_1, T_2\}$), where $L' = \{a, b, c, d\}$ and $L'' = \{e, f, g\}$. Part (a) shows the trees T_1 and T_2 , where the pair assigned to each node is presented beside it (except for the leaves). Part (b) shows the recursively labeled trees $T_1|L'$, $T_1|L''$, $T_2|L'$, and $T_2|L''$, where the labels are presented beside the nodes.

The following lemma proves the correctness of the procedure `Fast_Label_Trees`:

► **Lemma 8.** *Given a set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with identical sets of n leaf labels, the following statements hold after running the procedure `Fast_Label_Trees`(\mathcal{S}):*

1. *For any node $u \in V(T_i)$ where $T_i \in \mathcal{S}$, we have $id(u) \in \{1, \dots, 2k|L|\}$.*
2. *For any two nodes $u \in V(T_i)$ and $v \in V(T_j)$ where $T_i, T_j \in \mathcal{S}$, we have $id(u) = id(v)$ iff $\Lambda(T_i[u]) = \Lambda(T_j[v])$.*

Proof. We start by showing that the first statement holds. It can be easily seen that $|V(T_i)| < 2|L|$ for any $T_i \in \mathcal{S}$. Thus, the total number of nodes in all trees is less than $2k|L|$. Consequently, the label of each node is in $\{1, \dots, 2k|L|\}$.

To prove the second statement, we use induction on $|L|$. The base case of $|L| = 1$ holds because all nodes have the same cluster associated with them and receive the same label. The induction hypothesis states that if $|L| \leq k$ for some $k \geq 1$, then we have $id(u) = id(v)$ iff $\Lambda^{T_i}(u) = \Lambda^{T_j}(v)$. Now, we want to prove the statement for $|L| = k + 1$.

If $\Lambda(T_i[u]) = \Lambda(T_j[v])$, we have $\Lambda((T_i|L')[\varphi_u^{L'}]) = \Lambda((T_j|L')[\varphi_v^{L'}])$ and $\Lambda((T_i|L'')[\varphi_u^{L''}]) = \Lambda((T_j|L'')[\varphi_v^{L''}])$. Hence, considering that $|L'| \leq k$ and $|L''| \leq k$, we can apply the induction hypothesis to state that $id(\varphi_u^{L'}) = id(\varphi_v^{L'})$ and $id(\varphi_u^{L''}) = id(\varphi_v^{L''})$. Consequently, we have $(id(\varphi_u^{L'}), id(\varphi_u^{L''})) = (id(\varphi_v^{L'}), id(\varphi_v^{L''}))$ and thereby, $id(u) = id(v)$.

On the other hand, if $id(u) = id(v)$, we have $(id(\varphi_u^{L'}), id(\varphi_u^{L''})) = (id(\varphi_v^{L'}), id(\varphi_v^{L''}))$. As a result, we have $id(\varphi_u^{L'}) = id(\varphi_v^{L'})$ and $id(\varphi_u^{L''}) = id(\varphi_v^{L''})$. Therefore, considering that $|L'| \leq k$ and $|L''| \leq k$, we can use the induction hypothesis to deduce that $\Lambda((T_i|L')[\varphi_u^{L'}]) = \Lambda((T_j|L')[\varphi_v^{L'}])$ and $\Lambda((T_i|L'')[\varphi_u^{L''}]) = \Lambda((T_j|L'')[\varphi_v^{L''}])$. Thus, we have $\Lambda(T_i[u]) = \Lambda(T_j[v])$. ◀

Next, we analyze the time complexity of the procedure `Fast_Label_Trees`:

► **Lemma 9.** *Given a set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with identical sets of n leaf labels, the procedure `Fast_Label_Trees`(\mathcal{S}) runs in $O(kn \log n)$ time.*

Proof. Let $T(n)$ be the running time of `Fast_Label_Trees`(\mathcal{S}). By Lemma 5.2 of [12], the construction of T'_i and T''_i takes $O(n)$ time for each $T_i \in \mathcal{S}$, and thereby, a total of $O(kn)$ time for all of the trees. Computing $id(\varphi_u^{L'})$ and $id(\varphi_u^{L''})$ for each node u in some tree $T_i \in \mathcal{S}$ can be done by a bottom up traversal of T_i , along with T'_i and T''_i , in a total of $O(kn)$ time. The number of obtained pairs is $O(kn)$. Furthermore, we can deduce from Lemma 8 that all values in the pairs are in $\{0, 1, \dots, O(kn)\}$. Thus, sorting these pairs using radix sort and assigning labels to the nodes take $O(kn)$ time. Therefore, we have $T(n) = 2T(n/2) + O(kn)$, and consequently, $T(n) = O(kn \log n)$. ◀

Now, we can prove Theorem 5, regarding the correctness and time complexity of the procedure `Fast_Compute_Weights`:

▶ **Theorem 5.** *Given a set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with identical sets of n leaf labels, the procedure `Fast_Compute_Weights`(\mathcal{S}) calculates the weights of all clusters occurring in \mathcal{S} in $O(kn \log n)$ time.*

Proof. We start by proving that the procedure `Fast_Compute_Weights` works correctly. The correctness of Step 1, making a call to the procedure `Fast_Label_Trees`, follows from Lemma 8. In the following steps, the weight of each cluster is set to the count of the label of its associated node, indicating the number of trees in \mathcal{S} in which that cluster occurs.

Now, we analyze the time complexity. As shown in Lemma 9, assigning labels to each node in Step 1 takes $O(kn \log n)$ time. Considering that there are $O(kn)$ labels in total and each label is in $\{1, \dots, 2kn\}$, Step 2 (counting sort) takes $O(kn)$ time. Furthermore, it is easy to see that Steps 3 and 4 take $O(kn)$ time each. Therefore, the running time of the procedure is $O(kn \log n)$. ◀

5 Procedure `Fast_Filter_Clusters`

On a high level, our new procedure `Fast_Filter_Clusters`, with an improved running time of $O(n \log n)$, follows a similar approach as the $O(n \log^2 n)$ -time procedure `Filter_Clusters` in [20]. The objective is to build a tree T whose cluster collection is $\mathcal{C}(T) = \{\Lambda(T_A[u]) : u \in V(T_A) \text{ and } w(u) > w(x) \text{ for every } x \in V(T_B) \text{ with } \Lambda(T_A[u]) \not\prec \Lambda(T_B[x])\}$, i.e., to delete every cluster u in T_A that conflicts with at least one cluster in T_B with a weight greater than or equal to that of u . To do this, both procedures apply the centroid path decomposition technique to divide T_A into a centroid path $\pi = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$, where p_α is the root of T_A , and the set $\sigma(\pi)$ of side trees of π . Since each cluster in T_A is either located inside a side tree of π or associated with a node belonging to π , the cluster collection $\mathcal{C}(T_A)$ may be expressed recursively as:

$$\mathcal{C}(T_A) = \bigcup_{\tau \in \sigma(\pi)} \mathcal{C}(\tau) \cup \bigcup_{p_i \in \pi} \{\Lambda(T_A[p_i])\}. \quad (1)$$

Following this key observation, to check all clusters of T_A in order to decide which ones to delete, the procedures handle the side trees of π recursively and the clusters associated with π directly.

The main difference between `Filter_Clusters` from [20] and `Fast_Filter_Clusters` presented here is how they handle the clusters $\bigcup_{p_i \in \pi} \{\Lambda(T_A[p_i])\}$. The former uses a dynamic data structure to keep track of the currently conflicting nodes from T_B while traversing π upwards and retrieving the heaviest conflicting cluster at each step, taking $O(n \log n)$ time to process all the clusters associated with π . In addition, the time taken to set up the recursive calls to the side trees is $O(n)$. Since the total time spent on each recursion level is $O(n \log n)$ and there are $O(\log n)$ recursion levels, the time complexity of `Filter_Clusters` is

43:12 A Faster Algorithm for Constructing the Frequency Difference Consensus Tree

$O(n \log^2 n)$. In contrast, `Fast_Filter_Clusters` detects conflicts between clusters associated with π and clusters in T_B by representing clusters as suitably defined integer intervals. It then solves an instance of the MAX-MANHATTAN SKYLINE PROBLEM to identify the heaviest conflicting clusters. We will show that this method requires $O(n)$ time to handle all of the clusters associated with π . Thus, each recursion level takes $O(n)$ time, and the total running time becomes $O(n \log n)$.

The pseudocode of `Fast_Filter_Clusters` is presented in Algorithm 4.

■ **Algorithm 4** The procedure `Fast_Filter_Clusters`.

Algorithm `Fast_Filter_Clusters`

Input: Two weighted trees T_A and T_B with $\Lambda(T_A) = \Lambda(T_B) = L$ such that every $u \in V(T_A) \cup V(T_B)$ has a positive integer weight $w(u)$, and that some nodes in T_B may be spoiled.

Output: A tree T with $\Lambda(T) = L$ and $\mathcal{C}(T) = \{\Lambda(T_A[u]) : u \in V(T_A) \text{ and } w(u) > w(x) \text{ for every } x \in V(T_B) \text{ with } \Lambda(T_A[u]) \not\prec \Lambda(T_B[x])\}$.

1 Compute a centroid path $\pi = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$ of T_A , where p_α is the root of T_A and p_1 is a leaf, and compute the set $\sigma(\pi)$ of side trees of π .

 /* Handling the side trees */

2 **for** each side tree $\tau \in \sigma(\pi)$ **do**

 Construct $T_B || \Lambda(\tau)$.

 Temporarily change the node weights in τ and $T_B || \Lambda(\tau)$ by sorting them in nondecreasing order and setting each node weight equal to its rank.

 Let $\tau' := \text{Fast_Filter_Clusters}(\tau, T_B || \Lambda(\tau))$.

 Replace τ by τ' in T_A and restore the node weights in τ' .

endfor

 /* Handling the centroid path */

3 **for** $i = 1$ **to** α **do**

 Compute $n_i := |\Lambda(T_A[p_i])|$.

4 Temporarily relabel the leaf labels in L by the positive integers $\{1, 2, \dots, n_\alpha\}$ in a way that makes π become a stratifying path in T_A .

5 Compute and store, for every $v \in V(T_B)$, the values $m(v) = \min\{\Lambda(T_B[v])\}$ and $M(v) = \max\{\Lambda(T_B[v])\}$. For any $v \in V(T_B)$, if v is a spoiled node, then set $M(v) = n_\alpha + 1$.

6 Compute and store, for every $v \in V(T_B)$, the value $\text{filled}(v)$ equivalent to the largest integer x such that $\{1, 2, \dots, x\} \subseteq \Lambda(T_B[v])$.

7 Create a set I of weighted intervals over $\{1, 2, \dots, n_\alpha + 1\}$ as follows:

 For each $v \in V(T_B)$, make an interval $[v_\ell, v_r]$ with weight $w(v)$, where $v_\ell = 2 \cdot \max\{\text{filled}(v) + 1, m(v)\}$ and $v_r = 2 \cdot M(v)$. Insert the weighted interval into I .

8 Solve the MAX-MANHATTAN SKYLINE PROBLEM on I and let f be the solution.

9 **for** $i = \alpha$ **downto** 2 **do**

if $w(p_i) \leq f[2 \cdot n_i + 1]$ **then**

 Apply a *delete* operation on p_i in T_A .

endif

endfor

10 Restore the leaf labels of L to the values that they had before Step 4.

11 **return** T_A

End `Fast_Filter_Clusters`

Before describing the correctness of `Fast_Filter_Clusters`, we introduce a lemma that can be used to speed up the detection of conflicting clusters. Let T be a phylogenetic tree such that $\Lambda(T)$ is a set of positive integers $\{1, 2, \dots, n\}$. Moreover, let $\pi = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$ denote a path in T , where p_α is the root of T and p_1 is a leaf. For each $i \in \{1, 2, \dots, \alpha\}$, define $n_i := |\Lambda(T[p_i])|$. Note that $n_1 = 1$ and $n_\alpha = n$. The path π is called a *stratifying path* if $\Lambda(T[p_i]) = \{1, 2, \dots, n_i\}$ for every $i \in \{1, 2, \dots, \alpha\}$. Furthermore, for any $C \subseteq \Lambda(T)$, define $\text{filled}(C)$ as the largest integer x such that $\{1, 2, \dots, x\} \subseteq C$. In the following lemma, we determine whether a specified cluster C and the cluster associated with a specified node p_i on a stratifying path π are compatible:

► **Lemma 10.** *Let T be a phylogenetic tree with $\Lambda(T) = \{1, 2, \dots, n\}$. Also, let $\pi = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$ be a stratifying path in T . For any $C \subseteq \Lambda(T)$ and $i \in \{1, 2, \dots, \alpha\}$, we have $C \not\sim \Lambda(T[p_i])$ iff $\max\{\text{filled}(C) + 1, \min(C)\} < n_i + 0.5 < \max(C)$ holds, where $\Lambda(T[p_i]) = \{1, 2, \dots, n_i\}$.*

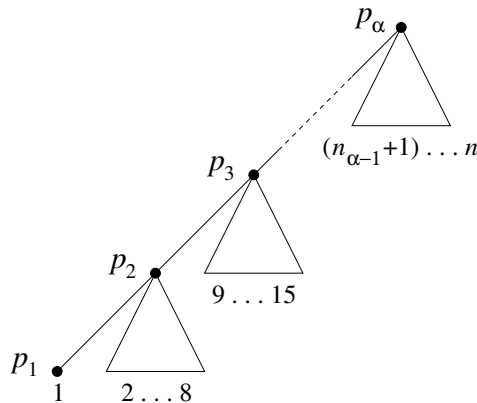
Proof. First suppose that $C \not\sim \Lambda(T[p_i])$. This means that there exist $x, y, z \in \Lambda(T)$ such that $x, y \in C$, $z \notin C$, $x, z \in \Lambda(T[p_i])$, and $y \notin \Lambda(T[p_i])$. Then:

- $x \in \Lambda(T[p_i])$ implies $x \leq n_i$. Since $x \in C$, we have $\min(C) < n_i + 0.5$.
- $y \notin \Lambda(T[p_i])$ gives $y > n_i$. Since $y \in C$ is an integer, we deduce that $n_i + 0.5 < \max(C)$.
- $z \in \Lambda(T[p_i])$ implies $z \leq n_i$. Furthermore, $z \notin C$ gives $z \geq \text{filled}(C) + 1$. Combining the two inequalities, we get $\text{filled}(C) + 1 < n_i + 0.5$.

On the other hand, suppose that $C \sim \Lambda(T[p_i])$. By the definition of cluster compatibility, at least one of the following three cases holds:

- $C \subseteq \Lambda(T[p_i])$: Then $x \leq n_i$ for all $x \in C$, i.e., $\max(C) \leq n_i$. Thus, the inequality $n_i + 0.5 < \max(C)$ is false.
- $\Lambda(T[p_i]) \subseteq C$: Then $\text{filled}(C) \geq n_i$, and hence $\text{filled}(C) + 1 < n_i + 0.5$ is false.
- $C \cap \Lambda(T[p_i]) = \emptyset$: Then $x > n_i$ for all $x \in C$, i.e., $\min(C) > n_i$. Thus, $\min(C) < n_i + 0.5$ is false. ◀

Figure 6 provides an illustration of Lemma 10.



■ **Figure 6** Illustration of Lemma 10. Let T be the tree with a stratifying path $\langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$ shown above. Consider the node p_2 and a cluster $C = \{5, 6, 8, 9\}$. Since $\min(C) = 5$, $\max(C) = 9$, $\text{filled}(C) = 0$, and $n_2 = 8$, Lemma 10 implies $C \not\sim \Lambda(T[p_2])$. Next, consider p_2 and $C' = \{1, 2, \dots, 12\}$. Since $\min(C') = 1$, $\max(C') = 12$, $\text{filled}(C') = 13$, and $n_2 = 8$, the inequality in Lemma 10 does not hold. Thus, $C' \sim \Lambda(T[p_2])$.

Now, we prove the correctness of the procedure `Fast_Filter_Clusters`:

► **Lemma 11.** *Given two weighted trees T_A and T_B with identical sets of n leaf labels, the procedure `Fast_Filter_Clusters`(T_A, T_B) works correctly.*

Proof. In Step 1, the procedure computes a centroid path $\pi = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$, where p_α is the root of T_A , and the set $\sigma(\pi)$ of side trees of π . According to Equation (1), any cluster C in T_A that should be removed is in either π or one of its side trees. In the former case, C will be removed in Steps 3–10, and in the latter case, C will be removed during some recursive call in Step 2.

Step 2 handles the side trees in $\sigma(\pi)$ by recursively calling the procedure for each $\tau \in \sigma(\pi)$ and replacing τ in T_A by the obtained tree τ' . Before each recursive call, the procedure normalizes the weights of the nodes in τ and $T_B||\Lambda(\tau)$ to make them positive integers of size $O(|\Lambda(\tau)|)$. This is achieved by sorting the weights in non-decreasing order and then setting the weight of each node equal to its rank in this order, where equally ranked nodes get identical weights.

Steps 3–10 handle the centroid path π as follows. After doing a bottom-up traversal of T_A to compute $n_i := |\Lambda(T_A[p_i])|$ for all $p_i \in \pi$, the leaf labels are modified to make π a stratifying path in T_A . As a consequence, for any node p_i on the centroid path π , its associated cluster $\Lambda(T_A[p_i])$ makes an integer interval of the form $[1, n_i]$. According to Lemma 10, $\Lambda(T_A[p_i])$ and any cluster C associated with a non-spoiled node in T_B conflict with each other iff $\max\{\text{filled}(C) + 1, \min(C)\} < n_i + 0.5 < \max(C)$. Therefore, one can determine whether $\Lambda(T_A[p_i]) \not\prec C$ by constructing an interval whose left endpoint is $2 \cdot \max\{\text{filled}(C) + 1, \min(C)\}$ and whose right endpoint is $2 \cdot \max(C)$, and then checking if it contains the point $2 \cdot n_i + 1$. Otherwise, if C is associated with a spoiled node in T_B , the condition for a conflict becomes $\max\{\text{filled}(C) + 1, \min(C)\} < n_i + 0.5$, and the corresponding interval's right endpoint is set to $2 \cdot (n_\alpha + 1)$. Steps 5–9 determine conflicts simultaneously between all clusters associated with π and all clusters in T_B by solving an instance of the MAX-MANHATTAN SKYLINE PROBLEM. By the above, its solution f has the property that $f[2 \cdot n_i + 1]$ is the weight of the heaviest cluster in T_B that conflicts with any cluster of the form $\Lambda(T_A[p_i])$. If this number is greater than or equal to $w(p_i)$, then the procedure deletes p_i from T_A . ◀

Next, we show that `Fast_Filter_Clusters` runs in $O(n \log n)$ time:

► **Lemma 12.** *Given two weighted trees T_A and T_B with identical sets of n leaf labels, the procedure `Fast_Filter_Clusters`(T_A, T_B) runs in $O(n \log n)$ time.*

Proof. Step 1 can be completed in $O(n)$ time [8]. Step 2 uses $O(n)$ time to construct the $T_B||\Lambda(\tau)$ -trees according to Lemma 2 and by applying radix sort to normalize the node weights. Step 2 also makes a recursive call for each τ . Next, bottom-up traversals of T_A and T_B are used to implement Steps 3–6, taking an additional $O(n)$ time. Creating the intervals that represent clusters in Step 7 takes $O(n)$ time. Also, solving the MAX-MANHATTAN SKYLINE PROBLEM in Step 8 takes $O(n)$ time, according to Lemma 3. This is because there are $O(n)$ intervals and their weights are positive integers of size $O(n)$. The necessary *delete* operations on π are carried out in top-down order, which means that the parent of any node in T_A is changed at most once and thereby, Step 9 takes $O(n)$ time in total. Finally, Step 10 restores the original leaf labels in $O(n)$ time.

The time complexity of `Fast_Filter_Clusters`(T_A, T_B) is thus $g(n) + \sum_{\tau \in \sigma(\pi)} h(\tau)$, where $g(n)$ is the execution time, excluding any recursive calls, and $h(\tau)$ is the running time of `Fast_Filter_Clusters`($\tau, T_B||\Lambda(\tau)$) for any side tree τ of π . According to the discussion above, $g(n) = O(n)$. For any recursion level j , let σ_j denote the set of all side trees that are computed for all the centroid paths on this level. The total time taken on the recursion

level $j + 1$ for the non-recursive parts is $\sum_{\tau \in \sigma_j} g(|\Lambda(\tau)|)$, and since the trees in σ_j are disjoint, $\sum_{\tau \in \sigma_j} g(|\Lambda(\tau)|) = g(n) = O(n)$. By Lemma 4, every τ satisfies $|\Lambda(\tau)| \leq n/2$, and hence there are $O(\log n)$ recursion levels. Therefore, the total running time is $O(n \log n)$. ◀

Combining Lemmas 11 and 12 provides the proof of Theorem 6:

► **Theorem 6.** *Given two weighted trees T_A and T_B with identical sets of n leaf labels, the procedure `Fast_Filter_Clusters`(T_A, T_B) filters out clusters as needed in $O(n \log n)$ time.*

6 Concluding Remarks

In this paper, we introduced an $O(kn \log n)$ -time algorithm for computing the FDCT, leading to an asymptotically faster approach compared to the best previously known algorithm (Gawrychowski et al. [15]). The improved procedure `Fast_Compute_Weights`, presented as part of our new algorithm, can also be employed in algorithms for building the greedy consensus tree [15, 21, 32], replacing the slower versions of the procedure. Closing the gap between the upper bound of $O(kn \log n)$ and the lower bound of $\Omega(kn)$ for the running time of the fastest FDCT construction algorithm remains an important open problem.

We implemented our $O(kn \log n)$ -time algorithm for constructing the FDCT. The source code can be found at https://github.com/tswddd2/FDCT_new. We achieved this implementation by adding approximately 2000 lines of C++ code to the source code of the $\min\{O(kn^2), O(kn(k + \log^2 n))\}$ -time algorithm [20] for the same problem, available at <https://github.com/Mesh89/FACT2> and also included in the FACT package [21], which was previously the fastest implementation. To implement the new $O(kn \log n)$ algorithm, we followed the descriptions in this article and used `dynamic_bitset` from the Boost libraries [1]. Preliminary experiments to evaluate the performance of our new algorithm indicate that it is faster in practice than the $O(kn^2)$ -time and $O(kn(k + \log^2 n))$ -time algorithms from [20]. The detailed results will be reported in the journal version of this paper.

References

- 1 The Boost C++ Libraries. <https://www.boost.org/>. Accessed on September 14, 2023.
- 2 Edward N Adams III. Consensus techniques and the comparison of taxonomic trees. *Systematic Biology*, 21(4):390–397, 1972.
- 3 Craig F Barrett, Jerrold I Davis, Jim Leebens-Mack, John G Conran, and Dennis W Stevenson. Plastid genomes and deep relationships among the commelinid monocot angiosperms. *Cladistics*, 29(1):65–87, 2013.
- 4 Michael A Bender and Martin Farach-Colton. The LCA problem revisited. In *Latin American Symposium on Theoretical Informatics*, pages 88–94. Springer, 2000.
- 5 Michael A Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321(1):5–12, 2004.
- 6 Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *Journal of Computer and System Sciences*, 48(2):214–230, 1994.
- 7 David Bryant. A classification of consensus methods for phylogenetics. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 61:163–184, 2003.
- 8 Richard Cole, Martin Farach-Colton, Ramesh Hariharan, Teresa Przytycka, and Mikkel Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30(5):1385–1404, 2000.

- 9 Maxime Crochemore, Costas S Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014.
- 10 William HE Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2:7–28, 1985.
- 11 Jianrong Dong, David Fernández-Baca, FR McMorris, and Robert C Powers. Majority-rule (+) consensus trees. *Mathematical Biosciences*, 228(1):10–15, 2010.
- 12 Martin Farach and Mikkel Thorup. Fast comparison of evolutionary trees. *Information and Computation*, 123(1):29–37, 1995.
- 13 J Felsenstein. Phylip version 3.6. *Software package, Department of Genome Sciences, University of Washington, Seattle, USA*, 2005.
- 14 Nicolás García, Alan W Meerow, Douglas E Soltis, and Pamela S Soltis. Testing deep reticulate evolution in Amaryllidaceae tribe Hippeastreae (Asparagales) with ITS and chloroplast sequence data. *Systematic Botany*, 39(1):75–89, 2014.
- 15 Paweł Gawrychowski, Gad M Landau, Wing-Kin Sung, and Oren Weimann. A faster construction of greedy consensus trees. *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, 2018.
- 16 Pablo A Goloboff, James S Farris, Mari Källersjö, Bengt Oxelman, Martín J Ramírez, and Claudia A Szumik. Improvements to resampling measures of group support. *Cladistics*, 19(4):324–332, 2003.
- 17 Pablo A Goloboff, James S Farris, and Kevin C Nixon. TNT, a free program for phylogenetic analysis. *Cladistics*, 24(5):774–786, 2008.
- 18 Jon Gorrie. *Does culture evolve? Testing evolutionary theories of culture through a case study of El Khiam points from three sites in the Pre Pottery Neolithic A of the Southern Levant*. PhD thesis, Oxford Brookes University, 2021.
- 19 Gang Han, Luis M Chiappe, Shu-An Ji, Michael Habib, Alan H Turner, Anusuya Chinsamy, Xueling Liu, and Lizhuo Han. A new raptorial dinosaur with exceptionally long feathering provides insights into dromaeosaurid flight performance. *Nature Communications*, 5:4382, 2014.
- 20 Jesper Jansson, Ramesh Rajaby, Chuanqi Shen, and Wing-Kin Sung. Algorithms for the majority rule (+) consensus tree and the frequency difference consensus tree. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 15(1):15–26, 2018.
- 21 Jesper Jansson, Chuanqi Shen, and Wing-Kin Sung. Improved algorithms for constructing consensus trees. *Journal of the ACM*, 63(3):28, 2016.
- 22 Sung Kwon Kim, Jung-Sik Cho, and Soo-Cheol Kim. Path maximum query and path maximum sum query in a tree. *IEICE TRANSACTIONS on Information and Systems*, 92(2):166–171, 2009.
- 23 Max Cardoso Langer, Blair Wayne McPhee, Júlio César de Almeida Marsola, Lúcio Roberto-da Silva, and Sérgio Furtado Cabreira. Anatomy of the dinosaur *Pampadromaeus barberenai* (Saurischia-Sauropodomorpha) from the Late Triassic Santa Maria Formation of southern Brazil. *PLOS ONE*, 14(2):e0212543, 2019.
- 24 Charlotte Lindqvist, Jan De Laet, Robert R Haynes, Lone Aagesen, Brian R Keener, and Victor A Albert. Molecular phylogenetics of an aquatic plant lineage, Potamogetonaceae. *Cladistics*, 22(6):568–588, 2006.
- 25 Timothy Margush and Fred R McMorris. Consensus n -trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.
- 26 Carlos Molineri. A cladistic revision of *Tortopus* Needham & Murphy with description of the new genus *Tortopsis* (Ephemeroptera: Polymitarcyidae). *Zootaxa*, 2481:1–36, 2010.
- 27 Carlos Molineri and Frederico F Salles. Phylogeny and biogeography of the ephemeral *Campsurus* Eaton (Ephemeroptera, Polymitarcyidae). *Systematic Entomology*, 38(2):265–277, 2013.

- 28 Carlos Molineri, Frederico F Salles, and Janice G Peters. Phylogeny and biogeography of Asthenopodinae with a revision of *Asthenopus*, reinstatement of *Asthenopodes*, and the description of the new genera *Hubbardipes* and *Priasthenopus* (Ephemeroptera, Polymitaarcyidae). *ZooKeys*, 478:45–128, 2015.
- 29 Martín O Pereyra, Boris L Blotto, Diego Baldo, Juan C Chaparro, Santiago R Ron, Agustín J Elias-Costa, Patricia P Iglesias, Pablo J Venegas, Maria Tereza C Thome, Jhon Jairo Ospina-Sarria, et al. Evolution in the genus *Rhinella*: a total evidence phylogenetic analysis of Neotropical true toads (Anura: Bufonidae). *Bulletin of the American Museum of Natural History*, 447(1):1–156, 2021.
- 30 Robert R Sokal and F James Rohlf. Taxonomic congruence in the Leptopodomorpha re-examined. *Systematic Zoology*, 30(3):309–325, 1981.
- 31 Mike Steel and Joel D Velasco. Axiomatic opportunities and obstacles for inferring a species tree from gene trees. *Systematic Biology*, 63(5):772–778, 2014.
- 32 Hongxun Wu. Near-optimal algorithm for constructing greedy consensus tree. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

Decremental Sensitivity Oracles for Covering and Packing Minors

Lawqueen Kanesh ✉ 

Indian Institute of Technology Jodhpur, India

Fahad Panolan ✉ 

School of Computing, University of Leeds, UK

M. S. Ramanujan ✉ 

University of Warwick, UK

Peter Strulo ✉

University of Warwick, UK

Abstract

In this paper, we present the first decremental fixed-parameter sensitivity oracles for a number of basic covering and packing problems on graphs. In particular, we obtain the first decremental sensitivity oracles for VERTEX PLANARIZATION (delete k vertices to make the graph planar) and CYCLE PACKING (pack k vertex-disjoint cycles in the given graph). That is, we give a sensitivity oracle that preprocesses the given graph in time $f(k, \ell)n^{O(1)}$ such that, when given a set of ℓ edge deletions, the data structure decides in time $f(k, \ell)$ whether the updated graph is a positive instance of the problem. These results are obtained as a corollary of our central result, which is the first decremental sensitivity oracle for TOPOLOGICAL MINOR DELETION (cover all topological minors in the input graph that belong to a specified set, using k vertices).

Though our methodology closely follows the literature, we are able to produce the first explicit bounds on the preprocessing and query times for several problems. We also initiate the study of fixed-parameter sensitivity oracles with so-called structural parameterizations and give sufficient conditions for the existence of fixed-parameter sensitivity oracles where the parameter is just the treewidth of the graph. In contrast, all existing literature on this topic and the aforementioned results in this paper assume a bound on the solution size (a weaker parameter than treewidth for many problems). As corollaries, we obtain decremental sensitivity oracles for well-studied problems such as VERTEX COVER and DOMINATING SET when only the treewidth of the input graph is bounded. A feature of our methodology behind these results is that we are able to obtain query times independent of treewidth.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Dynamic graph algorithms

Keywords and phrases Sensitivity oracles, Data Structures, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.44

Funding *M. S. Ramanujan*: Supported by the Engineering and Physical Sciences Research Council (grant numbers EP/V007793/1 and EP/V044621/1).

Acknowledgements We thank anonymous reviewers for the pointers to [18, 37].

1 Introduction

The study of basic graph problems on dynamic inputs has been a central aspect of algorithmics for several decades. A well-studied model in this line of research is the “fault tolerance model”. In this model, one assumes that the network at hand is susceptible to a bounded number of faulty network components (i.e., failing nodes or links) at any given time. The goal is to efficiently preprocess the network and produce a sufficiently small data structure so



© Lawqueen Kanesh, Fahad Panolan, M. S. Ramanujan, and Peter Strulo;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 44; pp. 44:1–44:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



that once the set of faulty nodes or links is given (or equivalently, the corresponding vertices or edges in the graph are deleted), one can recover various properties of the network from the stored data structure without recomputing these from scratch. The fault tolerance model has been a hugely successful setting for various advances on fundamental data structures such as spanners [39] and distance sensitivity oracles [21]. The “dimensions” of interest in such data structures are: the time needed by the preprocessing algorithm, the space complexity of the data structure, the time required to query the data structure in order to recover various properties of the input graph minus the set of failed elements and in some cases, the time required to update the data structure to reflect the failures.

The primary focus of research in the fault tolerance model has been on polynomial-time solvable problems. However, in a recent paper, Bilò et al. [6] extended the fault tolerance model to NP-complete graph problems by introducing a notion of decremental *fixed-parameter sensitivity oracles* (FSO). For an edge (respectively, vertex) decremental sensitivity oracle for a fixed-parameter tractable (FPT) problem Π , the input is an instance (G, k) of Π , where G is an n -vertex input graph and k is the parameter and a number ℓ , and the goal is to develop a preprocessing algorithm \mathcal{A} that builds a data-structure (i.e., the oracle) that, when queried on a set F of at most ℓ edges (respectively, vertices), decides whether $(G - F, k)$ is a positive instance of the problem, using a query algorithm \mathcal{Q} . The goal here is to ensure that the preprocessing time is $f(k, \ell)n^{\mathcal{O}(1)}$ and the query time is $g(k, \ell)n^{\mathcal{O}(1)}$ for some functions f and g . Unless otherwise specified, one allows both edge and vertex failures. Using this framework, Bilò et al. [6] gave the first edge decremental FSO for several problems including LONG PATH and VERTEX COVER. Subsequently, Alman and Hirsch [3] extended the work of Bilò et al. [6] to also account for edge additions, by introducing a *fully dynamic* notion of sensitivity oracles. Moreover, Alman and Hirsch [3] define a notion of *efficient* sensitivity oracles, where the preprocessing time is $f(k)n^{\mathcal{O}(1)}$, and the query time is $\ell^{\mathcal{O}(1)}g(k)n^{\mathcal{O}(1)}$. That is, the dependence on ℓ in both the preprocessing and query time is polynomial. By developing a dynamic variant of the extensor coding method [12], they show that LONG PATH has a fully dynamic efficient sensitivity oracle even on directed graphs.

These advances made by Bilò et al. [6] and Alman and Hirsch [3] for individual problems pose some natural questions: Could we prove general statements that provide a unified explanation of the existence of fixed-parameter sensitivity oracles (FSOs) for families of problems? Could we obtain *efficient* FSOs for these problems and give explicit bounds on the preprocessing and query times? These questions at the intersection of data structures and parameterized algorithms are our main motivation.

In this paper, we make significant progress towards answering these questions by presenting meta-theorems from which decremental FSOs for a number of basic covering and packing problems on graphs can be derived.

1.1 Our contributions I: FSOs for vertex deletion problems

Many important NP-hard graph optimization problems can be phrased as a vertex deletion problem to a graph class satisfying some property P . Here, the input is a graph G on n vertices and the task is to find a minimum size vertex subset S such that the graph $G - S$ obtained from G by removing S and its incident edges has the property P . By the well-known result of Lewis and Yannakakis [40] such problems are NP-complete for hereditary properties. For this reason the study of these problems is an integral part of the areas of approximation algorithms, exact-exponential algorithms and parameterized complexity, and has been responsible for the development of many classic algorithmic techniques.

Hence, the family of vertex deletion problems provide a natural candidate for us to develop meta-theorems. This brings us to the TOPOLOGICAL MINOR DELETION (TM-DELETION) problem which is a vertex deletion problem that directly generalizes numerous well-known problems vertex deletion problems including VERTEX COVER, FEEDBACK VERTEX SET (delete at most k vertices to obtain a forest) and VERTEX PLANARIZATION, to name a few. In TM-DELETION, the input is an undirected graph G , a family \mathcal{F} of undirected graphs such that every graph in \mathcal{F} has at most $h(\mathcal{F})$ vertices, and an integer k . The parameter is $k + h(\mathcal{F})$ and the goal is to decide whether there exists $S \subseteq V(G)$ of size at most k such that $G - S$ contains no graph from \mathcal{F} as a topological minor. A graph H is a *topological minor* of G if H can be obtained from G by deleting vertices or edges, and then contracting edges as long as each such edge is incident to at least one vertex of degree precisely 2. The expressive power of TM-DELETION naturally implies that an FSO for this problem would enable us to obtain as a consequence, FSOs for a host of other problems.

► **Theorem 1.** TOPOLOGICAL MINOR DELETION has a decremental fixed-parameter sensitivity oracle.

As a consequence of Theorem 1, we obtain decremental FSOs¹ for many well-studied parameterized problems, thus extending the scope of sensitivity oracles for NP-complete graph problems significantly beyond the state of the art. We refer the reader to the appendix for the definitions of problems not defined here and to Section 2 for the formal definition of treewidth and (topological) minors.

► **Corollary 2.** The following problems have FSOs as a consequence of Theorem 1.

1. FEEDBACK VERTEX SET (FVS). Or more generally, η -TREEWIDTH MODULATOR, i.e., decide whether we can delete at most k vertices from the input graph to obtain a graph with treewidth at most η .
2. VERTEX PLANARIZATION. Or more generally, MINOR DELETION, i.e., for a set \mathcal{F} of graphs, decide whether we can delete at most k vertices from the input graph to obtain a graph that excludes every graph in \mathcal{F} as a minor.
3. CYCLE PACKING. Or more generally, TOPOLOGICAL MINOR PACKING, i.e., for a set \mathcal{F} of graphs, decide whether the input graph contains k vertex disjoint topological-minor models of graphs in \mathcal{F} .
4. LONG PATH and LONG CYCLE. That is, decide whether there is a path (or a cycle, respectively) of length at least k in the input graph.

A useful feature of our proof techniques is that it allows for easy (albeit rough) estimations of the preprocessing and query times of most of the FSOs in the above statement. As a result, without much additional effort, one can prove the following bounds on specific FSOs in Corollary 2.

► **Theorem 3.** The following bounds can be obtained.²

1. FEEDBACK VERTEX SET has an FSO with preprocessing time $\text{tow}(3, \mathcal{O}((k + \ell)^{11}))n^4$ and query time $\text{tow}(2, \mathcal{O}((k + \ell)^{11}))$.
2. CYCLE PACKING has an FSO with preprocessing time $\text{tow}(3, \mathcal{O}((k + \ell)^{20}))n^4$ and query time $\text{tow}(2, \mathcal{O}((k + \ell)^{20}))$.
3. LONG PATH and LONG CYCLE have FSOs with preprocessing time $\text{tow}(2, \mathcal{O}(k \log(\ell)))n^4$ and query time $\text{tow}(2, \mathcal{O}(k \log(\ell)))$.

¹ Since we only deal with the decremental setting in this paper, we drop the explicit reference to this term in the rest of the paper and simply say, FSO.

² The notation $\text{tow}(p, q)$ indicates a runtime that is exponential in q , where q is on top of a tower of iterated exponentials of height p .

Note that these are the first concrete bounds for CYCLE PACKING. However, the bounds for LONG PATH implied by our meta-theorem are significantly worse than that of Bilò et al. [6] and Alman and Hirsch [3], which is not surprising since we obtain these bounds by instantiating a general-purpose theorem. For instance, the former get query time upper bounded by $\mathcal{O}(\ell(\ell + k))$ and the latter, $\ell^2 2^k k^{\mathcal{O}(1)}$. However, we prove the bounds in the above theorem in order to illustrate how to use our methodology to obtain explicit bounds for specific problems.

1.2 Our contributions II: A meta-theorem for efficient FSOs

Algorithmic meta-theorems are general algorithmic results applicable to a whole range of problems. Many prominent algorithmic meta-theorems are about model checking; such theorems state that for certain kinds of logic L , and all classes of structures \mathcal{C} that have a certain property, there is an algorithm that takes as input a formula $\phi \in L$ and a structure $S \in \mathcal{C}$ and efficiently determines whether $S \models \phi$. One of the most famous results in this direction is the seminal theorem of Courcelle [16, 14, 15] for model checking of Monadic Second Order Logic (MSO) on graphs of bounded treewidth (see also [1, 5, 11, 17, 22]). Courcelle’s theorem (which also extends to a fragment called Counting Monadic Second Order Logic or CMSO) is a crucial component of the parameterized complexity toolbox because numerous well-studied graph problems can be expressed in this particular fragment of logic. Classic examples of CMSO-definable graph properties are Hamiltonicity and 3-Colorability. We refer the reader to Section 2.2 for a formal description of CMSO-definability. Consequently a natural question arises – “Does an analogue of Courcelle’s theorem hold in the fault-tolerant setting?” An affirmative answer is implied by existing results in the literature on query testing MSO formulas on bounded-treewidth graphs (see, for instance, Theorem 6.1.3 in [37]). These results build upon Courcelle’s approach of reducing model checking MSO sentences on bounded-treewidth graphs to model checking MSO sentences on labelled trees. However, in the quest for *efficient* FSO (recall that we want preprocessing and query time *polynomial* in the number of failures), this approach does not yield a positive outcome since it involves a reduction to MSO model checking on graphs, where the formula size now depends on the number of failures and so, the query algorithm may take time exponential in the number of failures.

In this paper, we prove the following meta-theorem giving a sufficient condition for the existence of *efficient* FSOs with the additional property that the query times are actually independent of input size n (although the definition allows for sublinear dependence on n).

► **Theorem 4.** *Every CMSO-definable graph problem has an efficient non-uniform FSO with query time independent of input size, when parameterized by the treewidth of the input graph and size of the CMSO-sentence defining it.*

A *non-uniform* FSO is simply an FSO where one is allowed to have, for every value of the parameter k and number ℓ of permitted failures, a distinct preprocessing algorithm $\mathcal{A}_{k,\ell}$ and query algorithm $\mathcal{Q}_{k,\ell}$.

Theorem 4 forms a crucial component of our proof of Theorem 1. Essentially, we use Theorem 4 to handle “low-treewidth” instances of TM-DELETION. However, notice that Theorem 4 only guarantees a non-uniform FSO whereas Theorem 1 has no such caveat. Hence, a few remarks are in order here. Often, in the literature on non-uniform FPT algorithms, it has been demonstrated that the non-uniformity can be omitted either through self-reducibility arguments or by a case-by-case understanding of the combinatorics behind each problem. We are able to provide such arguments regarding Theorem 4 that essentially

suggest that as long as one could solve the CMSO-definable problem under consideration using an explicit dynamic programming algorithm on bounded-treewidth graphs, i.e., the vast majority of natural CMSO-definable problems in the literature, then one can actually infer an FSO for the problem on bounded-treewidth graphs *without* the caveat of non-uniformity. Moreover, this approach can lead to obtaining explicit running time bounds. This is also why we avoid resorting to the aforementioned black-box results on query testing in the literature to handle the low treewidth case. In this paper, we formally exemplify our strategy of eliminating the non-uniformity resulting from the invocation of Theorem 4 for the special case of TM-DELETION, which enables us to prove Theorem 1. Though we only deal with TM-DELETION, our arguments can be easily seen to extend to other problems, which we use to obtain explicit bounds for some of them. We believe that Theorem 4 will be a crucial component of designing FSOs for more problems, especially in conjunction with techniques such as irrelevant vertex removal [25].

In this context, it is important to mention the work of Courcelle and Vanicat [18]. They prove a meta-theorem that implies an efficient FSO for all CMSO-definable problems when parameterized by the treewidth of the input graph and size of the CMSO-sentence defining it. We note that their query times have a logarithmic dependence on n . It is known in the community (although not explicitly published to the best of our knowledge) that the $\log(n)$ dependence can be removed with appropriate preprocessing. However, we believe that the methodology behind Theorem 4 is useful as it enables us to easily obtain concrete bounds in our applications, which does not appear to be straightforward from the result of [18].

1.3 Our contributions III: Edge FSOs parameterized only by treewidth

We demonstrate the further applicability of the proof technique behind Theorem 4 to obtain a meta-theorem that gives sufficient conditions on a problem to have an edge FSO parameterized by the treewidth alone. Notice that all our results and those in the literature up to this point have the solution size as the parameter either explicitly or implicitly. In particular, in Theorem 4 the parameter also includes the size of the MSO formula, which in turn often depends on the solution size in the case of specific problems. Moreover, we highlight the fact that the oracles in this section have query time with a polynomial dependence on ℓ (in fact, only $\mathcal{O}(\ell^2)$). Moreover, the query times are independent of the treewidth. However, they are not *efficient* oracles as the preprocessing algorithm has exponential dependence on ℓ .

We give the following (non-exhaustive) exemplifications of our meta-theorem.

► **Theorem 5.** *The following hold.*

1. VERTEX COVER admits an edge FSO parameterized by the treewidth k with preprocessing time $\ell^{\mathcal{O}(2^k)} \cdot n^{\mathcal{O}(1)}$, and query time $\mathcal{O}(\ell^2)$.
2. DOMINATING SET admits an edge FSO parameterized by the treewidth k with preprocessing time $\ell^{\mathcal{O}(3^k)} \cdot n^{\mathcal{O}(1)}$, and query time $\mathcal{O}(\ell^2)$.

Note that since the treewidth of a graph is at most the size of the minimum vertex cover, the first statement directly implies an edge FSO for VERTEX COVER parameterized by solution size.

Here also, the work of Courcelle and Vanicat [18] is relevant as they prove an optimization version of their meta-theorem. However, their query time depends on the treewidth whereas we are able to obtain FSOs with query times independent of the treewidth.

1.4 Related work

Alman and Hirsch [3] note that the work of van den Brand and Saranurak [46] (see full version [47]) on distance sensitivity oracles in combination with standard color-coding techniques also imply a fully dynamic sensitivity oracle for LONG PATH on directed graphs, but with a worse dependence on k and ℓ . We note that though a $n^{o(1)}$ multiplicative factor in the query time is permitted in the definition of FSO, this is not exploited in their results and similar to our results, the queries of both Alman and Hirsch [3] as well the alternate oracle implied by Brand and Saranurak [46] run in time independent of the input size.

In other recent work, Pilipczuk et al. [44] gave a sensitivity oracle that answers s - t connectivity in constant time if a constant number of vertex failures occur. Interestingly, Pilipczuk et al. [44] show that the techniques they use to obtain their result can be used to design a model checking algorithm for the recently introduced *separator logic* [10] which is more expressive than First Order Logic but less expressive than MSO. This is a promising sign that advances on sensitivity oracles can have a much broader impact beyond the specific problem for which they are developed. We also note that the Arxiv version of [44] contains the tools (MSO query testing on trees) required to prove Kazana's result [37] on MSO query testing on bounded-treewidth graphs.

In recent years, spurred by the first systematic exploration of the intersection of parameterized and dynamic graph algorithms by Alman et al. [4], there has been a significant amount of work combining techniques from these two areas. Of special interest in the context of our paper is the work of Dvorak et al. [23] (improved upon by Chen et al. [13]) and Majewski et al. [42], who gave fully dynamic data structures that are able to maintain CMSO properties. That is, they obtain a data structure that is stronger than just a sensitivity oracle, but at the cost of weaker parameters than the one we use (i.e., treewidth).

Finally, on the topic of intersecting parameterized complexity and fault-tolerant data structures, Lochet et al. [41], in a work preceding the work of Bilò et al. [6], studied fault-tolerant spanners in directed graphs by choosing parameters expressing certain types of structure. Recently, Misra [43] initiated the study of computing fault-tolerant *solutions* (e.g., a solution that remains a feedback vertex set of the graph even if one vertex is removed from the solution) for NP-hard problems, with follow up work by Blazej et al. [7].

2 Preliminaries

2.1 Graphs

Given a graph G , let $V(G)$ and $E(G)$ denote the vertex and edge set of G , respectively. We only deal with simple graphs in this paper. When G is clear from the context, let n and m denote $|V(G)|$ and $|E(G)|$, respectively. For a graph G , $\text{paths}(G)$ denotes the set of all simple paths in G . For a set $A \subseteq V(G)$, we denote by $E(A)$ the set of those edges in G with both endpoints in A .

Formally, the treewidth of a graph is defined as follows.

► **Definition 6** (Tree decomposition). *A tree decomposition of a graph G is a pair (T, β) of a tree T and $\beta : V(T) \rightarrow 2^{V(G)}$, such that: (i) $\bigcup_{t \in V(T)} \beta(t) = V(G)$, (ii) for any edge $e \in E(G)$, there exists a node $t \in V(T)$ such that both endpoints of e belong to $\beta(t)$, and (iii) for any vertex $v \in V(G)$, the subgraph of T induced by the set $T_v = \{t \in V(T) : v \in \beta(t)\}$ is a tree. We say a tree decomposition is nice if it additionally satisfies the conditions on page 161 of [19]. The width of (T, β) is $\max_{v \in V(T)} \{|\beta(v)|\} - 1$. The treewidth of G is the minimum width of a tree decomposition of G .*

Let (T, β) be a tree decomposition of a graph G . We refer to the vertices of the tree T as *nodes*. We always assume that T is a rooted tree and so, we have a natural parent-child and ancestor-descendant relationship among nodes in T . The set $\beta(t)$ is called the bag at t . For two nodes $u, t \in T$, we say that u is a *descendant* of t , denoted $u \preceq t$, if t lies on the unique path connecting u to the root. Note that every node is its own descendant. If $u \preceq t$ and $u \neq t$, then we write $u \prec t$. For a tree decomposition (T, β) we also have a mapping $\gamma : V(T) \rightarrow 2^{V(G)}$ defined as $\gamma(t) = \bigcup_{u \prec t} \beta(u)$. For every $t \in V(T)$, we also define $\widehat{\beta}(t) = \beta(t) \cup E(\beta(t))$ and $\widehat{\gamma}(t) = \gamma(t) \cup E(\gamma(t))$. Recall that for a vertex set S , $E(S)$ denotes the set of all edges with both endpoints in S . We call a tree decomposition *nice* if it satisfies the conditions in section 7.2 of [20].

There is an algorithm that, given a graph G on n vertices and an integer w , runs in time $\mathcal{O}(f(w)n^3)$ and either correctly answers that G has treewidth more than w or outputs a tree decomposition of G of optimal width [8].

We next recall the classic notions of minors and topological minors.

► **Definition 7 (Minors).** *A graph H is a minor of G if there exists a function $\phi : V(H) \rightarrow 2^{V(G)}$ with the following properties: (i) for every $h \in V(H)$, $G[\phi(h)]$ is a connected graph, (ii) for all distinct $h, h' \in V(H)$, $\phi(h) \cap \phi(h') = \emptyset$, and (iii) for all $\{h, h'\} \in E(H)$, there exist $u \in \phi(h)$ and $v \in \phi(h')$ such that $\{u, v\} \in E(G)$. The function ϕ is called a minor model of H in G .*

► **Definition 8 (Topological minors).** *Let G and H be two graphs. We say that H is a topological minor of G if there exist injective functions $\phi : V(H) \rightarrow V(G)$ and $\varphi : E(H) \rightarrow \text{paths}(G)$ such that*

- for every $e = \{h, h'\} \in E(H)$, the endpoints of $\varphi(e)$ are $\phi(h)$ and $\phi(h')$,
- for every distinct $e, e' \in E(H)$, the paths $\varphi(e)$ and $\varphi(e')$ are internally vertex-disjoint,
- there does not exist a vertex v in the image of ϕ and an edge $e \in E(H)$ such that v is an internal vertex on $\varphi(e)$.

We say that (ϕ, φ) is a topological-minor model of H in G .

Note that if H is a topological minor of G , then it is also a minor of G . However, the converse does not hold.

Boundaried graphs. Roughly speaking, a boundaried graph is a graph where some vertices are labeled. A formal definition is as follows.

► **Definition 9 (Boundaried graph).** *A boundaried graph is a graph G with a set $\partial(G) \subseteq V(G)$ of distinguished vertices called boundary vertices, and an injective labeling $\lambda_G : \partial(G) \rightarrow \mathbb{N}$. The set $\partial(G)$ is the boundary of G , and the label set of G is $\Lambda(G) = \{\lambda_G(v) \mid v \in \partial(G)\}$.*

Given a finite set $I \subseteq \mathbb{N}$, \mathcal{G}_I denotes the class of all boundaried graphs whose label set is I , and $\mathcal{G}_{\subseteq I} = \bigcup_{I' \subseteq I} \mathcal{G}_{I'}$. A boundaried graph in $\mathcal{G}_{\subseteq [t]}$ is called a *t-boundaried* graph. Note that if G is a boundaried graph and $x \in V(G)$ is a vertex in the boundary, then $G - x$ is a boundaried graph that inherits its boundary and labeling from G in the natural way. That is, we simply remove x and preserve the labeling of the remaining vertices.

2.2 Counting Monadic Second Order Logic

The syntax of Monadic Second Order Logic (MSO) of graphs includes the logical connectives $\vee, \wedge, \neg, \Leftrightarrow, \Rightarrow$, variables for vertices, edges, sets of vertices and sets of edges, the quantifiers \forall and \exists , which can be applied to these variables, and the binary relations: (i) $u \in U$, where

u is a vertex variable and U is a vertex set variable; (ii) $d \in D$, where d is an edge variable and D is an edge set variable; (iii) $\mathbf{inc}(d, u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident to u ; (iv) equality of variables representing vertices, edges, vertex sets and edge sets.

An MSO sentence is an MSO formula without free variables. *Counting Monadic Second Order Logic* (CMSO) extends MSO by including atomic sentences testing whether the cardinality of a set is equal to q modulo r , where q and r are integers such that $0 \leq q < r$ and $r \geq 2$. That is, CMSO is MSO with the following atomic sentence: $\mathbf{card}_{q,r}(S) = \mathbf{true}$ if and only if $|S| \equiv q \pmod{r}$, where S is a set. We refer to [5, 16, 15] for a detailed introduction to CMSO. We note that what we refer to as CMSO in this paper is sometimes called CMSO₂ in the literature to indicate that quantifying over edge sets is permitted.

► **Definition 10** (Property). *A property is a function σ from the set of all graphs to $\{\mathbf{true}, \mathbf{false}\}$. For a CMSO sentence ψ , the property σ_ψ is defined as follows. Given a graph G , $\sigma_\psi(G)$ equals \mathbf{true} if and only if $G \models \psi$.*

► **Definition 11** (CMSO-definable property). *A property σ is CMSO-definable if there exists a CMSO sentence ψ such that $\sigma = \sigma_\psi$. In this case, we say that ψ defines σ .*

We next recall an implication of the classic Courcelle’s Theorem [16, 14, 15] proof (see also [17]). This fact, which is a central component in the proof of Theorem 4, says that a certain canonical equivalence relation over boundaried graphs has finite index. We first need to identify precisely those pairs of graphs that could potentially be related by the canonical equivalence and so, we define the *compatibility* equivalence relation \equiv_c on boundaried graphs as follows. We write $G_\alpha \equiv_c G_\beta$ and say that G_α is *compatible* with G_β if $\Lambda(G_\alpha) = \Lambda(G_\beta)$. Now, we define the *canonical equivalence relation* \equiv_σ on boundaried graphs.

► **Definition 12** (Canonical equivalence). *Given a property σ of graphs, the canonical equivalence relation \equiv_σ on boundaried graphs is defined as follows. For two boundaried graphs G_α and G_β , we say that $G_\alpha \equiv_\sigma G_\beta$ if (i) $G_\alpha \equiv_c G_\beta$, and (ii) for every boundaried graphs G_γ compatible with G_α (and thus also with G_β), we have: $\sigma(G_\alpha \oplus G_\gamma) = \mathbf{true} \Leftrightarrow \sigma(G_\beta \oplus G_\gamma) = \mathbf{true}$.*

Here, the gluing operator \oplus identifies equally-labeled vertices of the two boundaried graphs.

A property σ of graphs has *finite state* if $\forall I \subseteq \mathbb{N}$, the set of equivalence classes of \equiv_σ when restricted to $\mathcal{G}_{\subseteq I}$ is finite. Given a CMSO sentence ψ , the canonical equivalence relation associated with ψ is \equiv_{σ_ψ} , and for simplicity, we denote this relation by \equiv_ψ .

We are now ready to state the required consequence of Courcelle’s Theorem (see, for example, [16, 14, 15, 9]).

► **Proposition 13.** *Every CMSO-definable property on graphs has finite state.*

We use the RAM model (see Harel and Tarjan [29]) with addition and uniform cost measure. Each word holds $\mathcal{O}(\log n)$ bits and each basic operation on a word is assumed to take constant time.

3 Technical overview

In this section, we give an overview of our techniques, omitting details where necessary due to space constraints.

Since we effectively use Theorem 4 in our proof of Theorem 1, we first describe our proof of this result and how it implies a (uniform) FSO for TM-DELETION parameterized by the treewidth of the input graph in addition to the standard parameterization comprising the deletion set size k and the size of the largest graph in the family \mathcal{F} that we want to exclude as topological minors.

3.1 The FSO for CMSO-definable problems on bounded-treewidth graphs

We first give an overview of our proof of Theorem 4. For every CMSO formula ψ , one can define a canonical equivalence relation (see Definition 12) over the set of all boundaried graphs. A boundaried graph is simply a graph where some vertices are called boundary vertices and are assigned labels from a finite label set. A graph is t -boundaried if the label set has size at most t . The Myhill-Nerode equivalence for ψ on boundaried graphs says that if we take two t -boundaried graphs G_1 and G_2 then they are equivalent if and only if their boundary labels are equal and moreover, for any t -boundaried graph whose boundary labels are the same as those of G_1 (and hence also of G_2), if we glue the graphs G_1 and H along the boundaries by identifying equally labeled vertices and similarly, if we glue the graphs G_2 and H along the boundaries in this way, either both resulting graphs model ψ or neither one does. It is known (e.g., from the proof of Courcelle’s theorem itself) that for every CMSO formula ψ and every t , the number of equivalence classes induced by this relation over t -boundaried graphs is a function of ψ and t alone (a property called finite state). This implies that there is an $r \in \mathbb{N}$ that depends only on ψ and t (in our context, t will be 1 plus the treewidth of the graph), such that both the number of equivalence classes and the length of the encoding of a smallest boundaried graph in each equivalence class is upper bounded by r . If ψ and t are fixed, then r is constant. Now, suppose that \mathcal{R} is the set comprising a smallest boundaried graph from each of the equivalence classes (having fixed ψ and t) and suppose we know \mathcal{R} .

Our key insight is the following. Suppose we take a nice tree decomposition of the input graph and pick a bag. Now, consider the boundaried graph obtained by taking the graph induced by those vertices that appear in this bag or below it and making the vertices in the bag the boundary. Then, we observe that (a) this boundaried graph is equivalent to one of the graphs in \mathcal{R} and (b) regardless of the element failures in this boundaried graph, the resulting graph will still be equivalent to one of the graphs in \mathcal{R} . The only catch here is that before the query is given, one cannot know the representative of the “future” equivalence class. Hence, our preprocessing strategy aims to keep track, for each boundaried subgraph of the input graph obtained in the way we described above, *all* possible canonical equivalence classes that this graph can fall into, upon the removal of the failed vertices or edges given by the query in future. Our querying strategy, on the other hand, is a dynamic-programming algorithm. Depending on the at-most- ℓ queried edges and vertices, we identify a set of $\mathcal{O}(\ell)$ boundaried graphs that we have preprocessed and by examining the possible different ways in which the equivalence classes of only these specific boundaried graphs can be impacted by the failures, we are able to produce a correct answer to the query. This gives us Theorem 4. The non-uniformity comes from the assumption of knowing \mathcal{R} .

To overcome the non-uniformity aspect while applying Theorem 4, one must avoid the requirement of knowing \mathcal{R} . Instead, it is sufficient if, for every bounded-treewidth boundaried graph, we could efficiently compute an equivalent (under the Myhill-Nerode equivalence) boundaried graph whose size is bounded by some computable function of ψ and treewidth.

This approach, which was first introduced in order to obtain constructive versions of meta-algorithmic results on kernelization [26], has proved useful in several other instances in the literature [24, 25]. We show that this approach is indeed applicable to TM-DELETION

and in fact our arguments suggest that it is generally applicable as long as one has an explicit dynamic programming algorithm on bounded-treewidth graphs. Hence, we are able to obtain an FSO for TM-DELETION parameterized by the deletion set size k , the treewidth of the input graph, and the size of the largest graph in the family \mathcal{F} that we want to exclude as topological minors.

The same idea is also used to obtain the bounds in Theorem 3. That is, we essentially reuse the non-uniform oracles given by the proof of Theorem 4 (while avoiding the only source of non-uniformity) to handle low-treewidth instances of these specific problems. We then use a win-win argument to extend to FSOs with explicit bounds. In the win-win argument, we use the fact that if the treewidth of the input graph is already high, we get a trivial oracle that always answers either yes or no depending on the problem.

As an illustration, we describe the proof of the consequence for CYCLE PACKING in Corollary 2. That is, CYCLE PACKING has an FSO parameterized by the solution size k . Towards this, we first show that there is an FSO for this problem parameterized by k and treewidth. Let us assume that the treewidth bound is the max of treewidth and k , and let it be w . Then, we show that there is an FSO for this problem parameterized by w with preprocessing time $\text{tow}(3, \mathcal{O}(w^2)) \cdot n^4$ and query time $\text{tow}(2, \mathcal{O}(w^2)) + \ell^{\mathcal{O}(1)}$. This is done by showing that the size of the set \mathcal{R} and maximum size of a graph in \mathcal{R} are computable and then using Theorem 4. Intuitively, the bound on the size of the set \mathcal{R} comes from the size of the table at a bag when performing the standard dynamic programming over tree decompositions. The same fact can be used to also obtain a bound on the size of the graphs in \mathcal{R} . That is, if the subgraph “rooted” at a bag is larger than some computable function of w , then one can find a pair of ancestor-descendant bags that have the same dynamic programming tables, implying that these two graphs are equivalent. Now, a standard replacement operation of “cutting” the subgraph below the ancestor and “pasting” the subgraph below the descendant gives a strictly smaller equivalent graph and this process can be repeated.

Now, consider a general graph G . If G has treewidth greater than $s(k, \ell)$ for some function s (from grid-minor theorem), we can conclude that G has a sufficiently large grid, implying sufficiently many vertex-disjoint cycles. One can argue that after removing at most ℓ vertices or edges we will still have at least k vertex-disjoint cycles and hence we will have a positive instance of CYCLE PACKING. So we simply output an oracle that always answers yes to any query. If G has a smaller treewidth, then we can apply the above argument for low treewidth graphs, giving us a preprocessing time of $\text{tow}(3, \mathcal{O}((k + \ell)^{\mathcal{O}(1)}))$ and a query time of $\text{tow}(2, \mathcal{O}((k + \ell)^{\mathcal{O}(1)}))$.

3.2 The FSO for TOPOLOGICAL MINOR DELETION in general graphs

We now give an overview of the proof of Theorem 1 assuming that we have an FSO for TM-DELETION parameterized by k , treewidth of the input graph and the size of the largest graph in the family \mathcal{F} that we want to exclude as topological minors. Recall that, here the input is (G, k, \mathcal{F}) and an integer ℓ , and we want our preprocessing algorithm to essentially encode the answers to the input instances $(G - F, k, \mathcal{F})$ of TM-DELETION for any $F \subseteq V(G) \cup E(G)$ of size at most ℓ such that these answers can be retrieved efficiently by the query algorithm. Towards the preprocessing of (G, k, \mathcal{F}) we use the results about “irrelevant vertices” for the TM-DELETION problem by Fomin et al. [25]. As these irrelevant vertex results are for the vertex deletion problem, as a first step we construct an “equivalent” instance (G', k, \mathcal{F}') for the FSO, where all the edge failures in the original instance can be replaced with appropriate vertex failures in the new instance. This allows to restrict ourselves to handling vertex failures alone. More formally, for each $H \in \mathcal{F}$, we define H' to be the graph obtained from

H by adding three pendant (i.e., degree-1) vertices adjacent to each vertex v of H . Then, we set $\mathcal{F}' = \{H' : H \in \mathcal{F}\}$. The graph G' is constructed from G as follows. First we subdivide each edge in G . For each $e \in E(G)$, let u_e be the subdivision vertex in G' corresponding to e . Then for each vertex $v \in V(G)$ we add three pendant vertices adjacent to v . Then, each edge e failing in G can be thought of as a vertex u_e failing in G' . We prove that it is enough to give a vertex FSO for the instance (G', k, \mathcal{F}') . Then, the preprocessing algorithm \mathcal{A} uses the template of Fomin et al. [25] which in turn was built on the approach introduced by Robertson and Seymour for DISJOINT PATHS [45]. Their approach has found applications in many significant results in the area [28, 30, 34, 31, 33, 36, 35, 32, 27]. In the template of Fomin et al. [25], we have three exhaustive cases.

Case 1. The treewidth of the input graph G is upper bounded by a function of k, ℓ , and \mathcal{F}' .

In this case we use our fault-tolerance oracle for TM-DELETION parameterized by k, \mathcal{F}' and the treewidth which we have outlined in the previous subsection.

Case 2. The input graph G has a clique minor whose size is lower bounded by a computable function of k, ℓ , and \mathcal{F}' . In this case Fomin et al. [25] gave an algorithm to find an irrelevant vertex v with respect to “any” vertex deletion set of size $k + \ell$. That is, for any vertex subset $S \subseteq V(G')$, the topological minors of size at most δ in $G - S$ and $G - S - v$ are same. Here, we set δ to be the maximum size of a graph in \mathcal{F}' .

Case 3. Case 1 and 2 are not applicable. In this case the “weak structure theorem” [45] implies that the graph G contains a “large flat wall”. Here, large means that its size is lower bounded by a function of k, ℓ , and \mathcal{F}' . In this case as well Fomin et al. [25] gave an algorithm to find an irrelevant vertex v with respect to “any” vertex deletion set of size $k + \ell$.

In our preprocessing algorithm, as long as Case 2 or Case 3 is applicable, we delete the irrelevant vertices computed and finally we end up with a graph with bounded treewidth, which places us in Case 1, which we have described how to handle.

Corollaries of Theorem 1

We now argue that the assertions made in Corollary 2 hold.

► **Proposition 14** ([2, 38]). *For every $\eta \in \mathbb{N}$, there is a set \mathcal{F}_η of graphs such that a graph has treewidth at most η if and only if it excludes the graphs in \mathcal{F}_η as a minor.*

We also require the following simple fact.

► **Proposition 15.** *For every family \mathcal{F} of graphs, there is a set \mathcal{F}' of graphs such that any graph contains a minor model of a graph from \mathcal{F} if and only if it contains a topological-minor model of a graph from \mathcal{F}' .*

We note that the families \mathcal{F}_η in Proposition 14 and \mathcal{F}' in Proposition 15 are constructive. In combination with these two propositions, Theorem 1 implies the first two statements of Corollary 2. That is, η -TREEWIDTH MODULATOR is precisely TM-DELETION where the family \mathcal{F}' of forbidden topological minors is obtained by first using η as “input” to Proposition 14 to obtain \mathcal{F}_η , and then plugging in \mathcal{F}_η as input to Proposition 15 to obtain the required family \mathcal{F}' . Similarly, MINOR DELETION where \mathcal{F} is the family of graphs to exclude as minors can be written as TM-DELETION, where the forbidden family of topological minors is obtained by plugging \mathcal{F} in to Proposition 15. We now proceed to the remaining statements of Corollary 2.

Let us now consider the dual problem to TM-DELETION, i.e., TOPOLOGICAL MINOR PACKING (TM-PACKING), which is formally defined as follows.

TOPOLOGICAL MINOR PACKING (TM-PACKING)

Input: An undirected graph G , a family of undirected graphs \mathcal{F} such that every graph in \mathcal{F} has at most h^* vertices, and an integer k .
Parameter: $k + h^*$.
Problem: Does there exist k vertex-disjoint topological-minor models of graphs in \mathcal{F} ?

► **Theorem 16.** TM-PACKING has an FSO.

Proof. Let $(G, \mathcal{F}, k, \ell)$ be the input of TM-PACKING. Now we define a family \mathcal{F}' as follows.

$$\mathcal{F}' = \{H \mid \exists H_1, \dots, H_k \in \mathcal{F} : H \text{ is the disjoint union of } H_1, \dots, H_k\}$$

Then, notice that any subgraph of G contains k vertex-disjoint topological-minor models from \mathcal{F} if and only if the same subgraph of G contains a graph from \mathcal{F}' as topological minor. Moreover, $|\mathcal{F}'| \leq |\mathcal{F}|^k$ and the largest graph in \mathcal{F}' has size at most $k \cdot h(\mathcal{F})$.

Now, we are ready to give an FSO $(\mathcal{A}, \mathcal{Q})$ for TM-PACKING by using an FSO $(\mathcal{A}', \mathcal{Q}')$ for TM-DELETION (Theorem 1) as a subroutine.

The preprocessing algorithm \mathcal{A} : Let the input to \mathcal{A} be an instance (G, \mathcal{F}, k) of TM-PACKING and $\ell \in \mathbb{N}_0$.

Step 1: Construct the family \mathcal{F}' defined above.

Step 2: Run the algorithm \mathcal{A}' on input $(G, \mathcal{F}', 0)$ and ℓ and return its output.

This completes the description of the preprocessing algorithm.

The query algorithm \mathcal{Q} : Let the input be $F \subseteq V(G) \cup E(G)$.

Step 1: Use the query algorithm \mathcal{Q}' to decide whether $(G - F, \mathcal{F}', 0)$ is a yes-instance of TM-DELETION.

Step 2: If \mathcal{Q}' answers YES, then \mathcal{Q} answers NO. Else, \mathcal{Q} answers YES.

This completes the description of the query algorithm.

Notice that the family \mathcal{F}' can be computed in time bounded by a function of $h(\mathcal{F}) + k$. Moreover, since $(\mathcal{A}', \mathcal{Q}')$ is an FSO for TM-DELETION, it follows that \mathcal{A}' is an FPT algorithm parameterized by $h(\mathcal{F}') + k \leq k \cdot (h(\mathcal{F}) + 1)$. Since these are the only two steps in \mathcal{A} , it follows that \mathcal{A} is also an FPT algorithm parameterized by $h(\mathcal{F}) + k$. Similarly, since \mathcal{Q}' runs in time $f'(h(\mathcal{F}') + k)$ for some computable function f' , it follows that \mathcal{Q} runs in time $f(h(\mathcal{F}') + k)$ for some computable function f . Finally, the correctness of the pair $(\mathcal{A}, \mathcal{Q})$ follows from the fact that for every $F \subseteq V(G) \cup E(G)$, the instance $(G - F, \mathcal{F}, k)$ of TM-PACKING is a yes-instance if and only if the instance $(G - F, \mathcal{F}', 0)$ of TM-DELETION is a no-instance. ◀

Finally, notice that LONG PATH and LONG CYCLE are both special cases of TM-PACKING. Hence, the final statement of Corollary 2 can also be obtained as a consequence of Theorem 1.

3.3 Edge FSOs parameterized by treewidth alone

We employ the same high-level approach as that used for Theorem 4 (see Section 3.1). However, instead of considering equivalence among boundaried graphs, we aim to identify equivalent sets of failure edges for the boundaried graph obtained at each bag. This idea can

be summarized as follows. First, assume that the problem satisfies certain properties that are typically satisfied by problems for which there is an explicit dynamic programming algorithm over a given tree decomposition. This includes well-known problems such as VERTEX COVER and DOMINATING SET. Further, suppose that when any set of ℓ edges are deleted below a bag B and we were to re-run the dynamic programming algorithm on the graph induced by the vertices in this bag and its descendants, then the dynamic programming table computed at the bag B is only a “small” perturbation of the dynamic programming table that was initially computed at this bag. In our context, “small” simply means that the entries in the cells change (either increase or decrease) by at most ℓ . For instance, in the case of VERTEX COVER, deleting ℓ edges will not change any of the partial solutions by more than ℓ . Now, since the size of the bag is bounded by the treewidth, this implies a bounded number of equivalence classes for the set of all edge failures of size at most ℓ . This fact is then used to keep a representative of each equivalence class and also a solution corresponding to them. Finally, we show how the query algorithm can identify the equivalence class of the given query efficiently.

Formally, we prove the result below. The terms in the theorem statement build upon the notation of Garnero et al. [26] and are explained in the subsequent paragraphs.

► **Theorem 17.** *Consider a subset problem Π . If Π has an encoder ξ that admits a gap function g , a gap signature computation algorithm, an FPT exact algorithm, and Π is DP effective, then Π admits an edge FSO parameterized by the treewidth.*

Here, a parameterized graph problem Π is called a *subset problem* if there exists a language L_Π associated with Π that comprises of pairs (G, S) where, for every graph G and $S \subseteq V(G)$, $(G, S) \in L_\Pi$ if and only if S is a solution to Π on G .

The following definitions encapsulate the idea of the standard dynamic programming algorithms on treewidth. The boundary of G will be a bag of the tree decomposition so the vertices of the boundary will be the only way that some solution S can “interact” with the rest of the graph. $\mathcal{C}(|\Lambda(G)|)$ represents the space of possible interactions and S is compatible with some encoding R from this space if it does in fact interact as specified by R .

► **Definition 18 (Encoder).** *Consider a subset problem Π . An encoder ξ of Π is a pair (\mathcal{C}, L_C) , where:*

1. $\mathcal{C} : \mathbb{N}_0 \rightarrow 2^{\Sigma^*}$ is a computable function, , with $\mathcal{C}(0) = \varepsilon$. Here, Σ is some finite alphabet depending on Π .
2. L_C is a language that comprises of triples (G, S, R) , where G is a boundaried graph, $S \subseteq V(G)$, and $R \in \mathcal{C}(|\Lambda(G)|)$. If $(G, S, R) \in L_C$, then we say that S is compatible with R under ξ .
3. For every 0-boundaried graph G and $S \subseteq V(G)$, the triple $(G, S, \varepsilon) \in L_C$ if and only if $(G, S) \in L_\Pi$.

For example, an encoder of VERTEX COVER sets $\mathcal{C}(k)$ to the power set of $[k]$ representing subsets of the boundary and $(G, S, R) \in L_C$ iff S is a vertex cover of G and $S \cap \partial(G) \supseteq R$. That is S is compatible with R when S “agrees” with R on the boundary. Thus, the encoder describes the space that the dynamic programming is over.

We now define the function that such a dynamic programming algorithm would calculate.

► **Definition 19 (Family of nice functions associated to an encoder).** *Consider a subset problem Π with an encoder $\xi = (\mathcal{C}, L_C)$. For a boundaried graph G , we define a nice function $\eta_G^\xi : \mathcal{C}(|\Lambda(G)|) \rightarrow \mathbb{N}_0$ as follows: For every $R \in \mathcal{C}(|\Lambda(G)|)$,*

$$\eta_G^\xi(R) = \begin{cases} |V(G)| + 1 & \text{if } \{S : (G, S, R) \in L_C\} = \emptyset \\ \min\{|S| : (G, S, R) \in L_C\} & \text{otherwise.} \end{cases}$$

Note that since $\min\{|S| : (G, S, \varepsilon) \in L_C\} = \min\{|S| : (G, S) \in L_\Pi\}$, this means that for all $p \in \mathbb{N}$, $\eta_G^\xi(\varepsilon) \leq p$ iff $(G, p) \in \Pi$.

Continuing our VERTEX COVER example, if ξ is the encoder described above, then $\eta_G^\xi(R)$ is the size of the minimum vertex cover of G that contains every vertex from R . In the standard dynamic programming algorithm for VERTEX COVER parameterized by treewidth, the table entry indexed by some $x \in V(T)$ and $R \subset \beta(x)$ is exactly $\eta_{G_x}^\xi(R)$.

We now wish to bound how much the answer can change due to edge failures. For example, in VERTEX COVER the size of the solution will never decrease by more than the number of edge failures.

► **Definition 20** (Gap function for encoder). *Consider a subset problem Π with an encoder $\xi = (C, L_C)$. We say that ξ admits a gap function $g : \mathbb{N} \rightarrow \mathbb{N}$, if for every boundaried graph G , for every $F \subseteq E(G)$, and for every $R \in \mathcal{C}(|\Lambda(G)|)$, we have*

$$|\eta_G^\xi(R) - \eta_{G-F}^\xi(R)| \leq g(|F|).$$

From now onwards we will consider a subset problem Π and assume it has an encoder ξ that admits a gap function g .

► **Definition 21** (Gap signature). *For all boundaried graphs G , and sets $F \subseteq E(G)$, the gap signature of F in G is the function $\sigma_G^F : \mathcal{C}(|\Lambda(G)|) \rightarrow \{-g(|F|), \dots, g(|F|)\}$, defined as follows: for every $R \in \mathcal{C}(|\Lambda(G)|)$,*

$$\sigma_G^F(R) = \eta_G^\xi(R) - \eta_{G-F}^\xi(R).$$

Bounding the amount the answer changes allows us to keep track of exactly how much the answer changes for each failure set with the gap signature. However, notice that the number of possible gap signatures for any set $F \subseteq E(G)$ is $(2g(|F|) + 1)^{|\mathcal{C}(|\Lambda(G)|)|}$, that is it only depends on the size of boundary of G and the size of F . Being able to calculate the gap signature will be a key part of our algorithm.

► **Definition 22** (Gap signature computation algorithm). *A gap signature computation algorithm takes as an input (i) a graph G , (ii) a nice tree decomposition of G , (T, β) with width k , (iii) a node $x \in V(T)$, and (iv) a set of edges, $F \subseteq E(G_{x,T}^\downarrow)$ of size ℓ , runs in time $f_{gs}(k, \ell)n^{\mathcal{O}(1)}$ and outputs $\sigma_{G_{x,T}^\downarrow}^F$, where f_{gs} is a computable function.*

Since we are mostly interested in the gap signature of a given failure set we introduce the following equivalence relation.

► **Definition 23** (Equivalence relation \equiv_G). *For all boundaried graphs G , and sets $F_1, F_2 \subseteq E(G)$, we say that $F_1 \equiv_G F_2$ if and only if all of the following statements hold.*

1. $\sigma_G^{F_1} = \sigma_G^{F_2}$. That is, for every $R \in \mathcal{C}(|\Lambda(G)|)$, $\sigma_G^{F_1}(R) = \sigma_G^{F_2}(R)$.
2. $F_1 \cap \binom{\partial(G)}{2} = F_2 \cap \binom{\partial(G)}{2}$, that is, F_1 and F_2 coincide on the set of edges with both endpoints in $\partial(G)$.
3. $|F_1| = |F_2|$.

Since there are only a small number of possible gap signatures, there are also not many equivalence classes. More precisely, if the sets F_1 and F_2 are of size at most ℓ then there are at most $\mathcal{O}(\ell|\partial(G)|^2)$ equivalence classes for each gap signature. We would like to exploit this by pre-calculating solutions on one element from each equivalence class. To this end we define a set consisting of exactly one element from each equivalence class.

In the following, for a set X and $\ell \in \mathbb{N}$, by $\mathcal{P}_\ell(X)$, we denote the set of all the subsets of X of size at most ℓ . Given a nice tree decomposition (T, β) , for all $(u, v) \in E(G)$ let $\text{Highest}((u, v))$ be the unique highest node t in T such that $\beta(t)$ contains both u and v . For each $x \in V(T)$, we define $\chi(x) = \{e \in E(G) : x = \text{Highest}(e)\}$ and $\chi^\downarrow(x) = \cup_{y \prec x} \chi(y)$. Note that $\chi(x) \subseteq E(G[\beta(x)])$ and $\chi^\downarrow(x) \subseteq E(G_x^\downarrow)$. Also χ is a partition of $E(G)$, that is $\chi(x) \cap \chi(y) = \emptyset$ for all $x \neq y$ and $\chi^\downarrow(\text{root}(T)) = \cup_{x \in V(T)} \chi(x) = E(G)$.

► **Definition 24** (Type representative family). *Consider a graph G , a nice tree decomposition (T, β) of G , and $\ell \in \mathbb{N}$. For every node $x \in V(T)$, we define a type representative family $\mathfrak{R}(x) \subseteq \mathcal{P}_\ell(\chi^\downarrow(x))$ such that:*

1. For every $F_1, F_2 \in \mathfrak{R}(x)$, if $F_1 \equiv_{G_x^\downarrow} F_2$ then $F_1 = F_2$, and
2. For every $F \in \mathcal{P}_\ell(\chi^\downarrow(x))$, there exists $F' \in \mathfrak{R}(x)$ such that $F' \equiv_{G_x^\downarrow} F$.

Note that since $\chi^\downarrow(\text{root}(T)) = E(G_{\text{root}(T)}^\downarrow) = E(G)$, $\mathfrak{R}(\text{root}(T))$ contains a representative from each equivalence class of failure sets. Also note that if $F_1 \subseteq \chi(x)$ then, for all $F_2 \in \mathcal{P}_\ell(\chi^\downarrow(x))$, we have $F_1 \equiv_{G_x^\downarrow} F_2$ if and only if $F_1 = F_2$ so $\mathcal{P}_\ell(\chi(x)) \subseteq \mathfrak{R}(x)$. Let

$$f_R(k, \ell) = \ell \cdot k^2 \cdot (2g(\ell) + 1)^{|C(k)|}$$

then the size of $\mathfrak{R}(x)$ is at most $\mathcal{O}(f_R(k, \ell))$ where k is the width of (T, β) . However there are $\mathcal{O}(n^\ell)$ possible failure sets that could be in $\mathfrak{R}(x)$ which is too many to calculate a representative from each equivalence class by brute force. So we will split the failure set into smaller subproblems and use the following property to calculate the representatives by dynamic programming. Intuitively this says that, when doing dynamic programming on the treewidth, the equivalence “carries through”, that is, if we have some representatives of the equivalence classes of the failure set below y_1 and y_2 rather than the actual failure set, we can use these to construct a representative of the whole failure set below x .

► **Definition 25** (DP effective). *We say that our subset problem Π is DP effective if, for every graph G , and nice tree decomposition (T, β) of G , the following holds: For every $x, y_1, y_2 \in V(T)$ such that x is a common ancestor of y_1 and y_2 , and for every $F_x \subseteq \chi(x)$, $F_1, F_1^* \subseteq \chi^\downarrow(y_1)$, and $F_2, F_2^* \subseteq \chi^\downarrow(y_2)$, it holds that if $F_1^* \equiv_{G_{y_1}^\downarrow} F_1$ and $F_2^* \equiv_{G_{y_2}^\downarrow} F_2$, then $(F_x \cup F_1 \cup F_2) \equiv_{G_x^\downarrow} (F_x \cup F_1^* \cup F_2^*)$.*

From now onwards we will work on a given graph G , and failure set size ℓ , and assume we know a nice tree decomposition (T, β) of G of width at most k and size $\mathcal{O}(kn) = \mathcal{O}(n^2)$. The following lemma shows that we can calculate a type representative family for each node quickly. Recall that $f_R(k, \ell)$ is a bound on the size of any such family and that $f_{gs}(k, \ell)$ is the superpolynomial component of the runtime of the gap signature computation algorithm. The proof can be found in the appended full version.

► **Lemma 26.** *If Π admits a gap signature computation algorithm and is DP effective, then we can compute a type representative family $\mathfrak{R}(x)$ for all $x \in V(T)$ in time $2^{k^2} \cdot f_R(k, \ell)^3 \cdot f_{gs}(k, \ell) \cdot n^{\mathcal{O}(1)}$.*

We now have, at each node $x \in V(T)$, a small set of possible failure sets that together cover every equivalence class. The only remaining obstacle is that we cannot calculate which of these edge sets is equivalent to our actual failure set at query time since the gap signature computation algorithm is too slow. To this end we define the following tables. Together with DP effectiveness they will allow us to calculate a representative of the true failure set from the bottom up.

► **Definition 27.** Suppose $\mathfrak{R}(x)$ is a type representative family for all $x \in V(T)$. Let $x, y_1, y_2 \in V(T)$ such that x is an ancestor of y_1 and y_2 and $x \neq y_1$. Then, for every set $F_x \subseteq \chi(x)$, $F_1 \in \mathfrak{R}(y_1)$, and $F_2 \in \mathfrak{R}(y_2)$ we define the following.

1. $H_1[x, y_1, F_x, F_1] = Q_x$, where $F_x \cup F_1 \equiv_{G_x^\perp} Q_x$ and $Q_x \in \mathfrak{R}(x)$.
2. If $y_1 \neq y_2$ and x is the lowest common ancestor of y_1 and y_2 , then define $H_2[x, y_1, y_2, F_x, F_1, F_2] = Q_x$, where $F_x \cup F_1 \cup F_2 \equiv_{G_x^\perp} Q_x$ and $Q_x \in \mathfrak{R}(x)$.

► **Lemma 28.** If Π admits a gap signature computation function and is DP effective, then the tables H_1 and H_2 described in Definition 27 can be filled in time $2^{k^2} \cdot f_R(k, \ell)^3 \cdot f_{gs}(k, \ell) \cdot n^{\mathcal{O}(1)}$. Moreover, H_1 and H_2 both have size at most $2^{k^2} \cdot f_R(k, \ell)^2 \cdot n^{\mathcal{O}(1)}$.

Finally we will need to use an exact algorithm for the problem on the original graph as a black box. This is a very weak assumption since an FSO is itself an exact algorithm (simply call it with $F = \emptyset$) so we expect an exact algorithm to be easier to obtain than an FSO.

► **Definition 29.** An FPT exact algorithm takes as input a graph G and a nice tree decomposition of G , with width k , runs in time $f_{opt}(k)n^{\mathcal{O}(1)}$ and outputs $\eta_G^\xi(\varepsilon)$.

4 Concluding remarks

Parameterized sensitivity oracles provide a fertile middle ground of study between static FPT algorithms (where many problems are well-understood) and dynamic FPT algorithms (where many problems turn out to be hard) and deserve a thorough exploration. Along with the work of Bilò et al. [6], Alman and Hirsch [3] and Pilipczuk et al. [44], this paper furthers our understanding of the capabilities of state-of-the-art algorithm design techniques used in parameterized complexity. Indeed, Alman and Hirsch [3] in their paper ask whether there examples of techniques other than extensor coding, that are used to solve static versions of parameterized problems and which can be used to design faster dynamic algorithms or sensitivity oracles. Our three main results (Theorem 4, Theorem 1 and Theorem 17) provide useful classification tools to study other problems in this framework and importantly, gives a road map for obtaining explicit bounds. The possibility of obtaining similar classification results in the *fully dynamic* setting is a natural direction for future work.

References

- 1 Karl Abrahamson and Michael Fellows. Finite automata, bounded treewidth and well-quasiordering. In *Graph structure theory (Seattle, WA, 1991)*, volume 147 of *Contemp. Math.*, pages 539–563, Providence, RI, 1993. Amer. Math. Soc. doi:10.1090/conm/147/01199.
- 2 Isolde Adler, Martin Grohe, and Stephan Kreutzer. Computing excluded minors. In *Proceedings of the 19th annual ACM-SIAM symposium on Discrete algorithms (SODA 2008)*, pages 641–650. SIAM, 2008. URL: <http://portal.acm.org/citation.cfm?id=1347082.1347153>.
- 3 Josh Alman and Dean Hirsch. Parameterized sensitivity oracles and dynamic algorithms using exterior algebras. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 9:1–9:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.9.
- 4 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. *ACM Trans. Algorithms*, 16(4):45:1–45:46, 2020. doi:10.1145/3395037.
- 5 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

- 6 Davide Bilò, Katrin Casel, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, J. A. Gregor Lagodzinski, Martin Schirneck, and Simon Wietheger. Fixed-parameter sensitivity oracles. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICs*, pages 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ITCS.2022.23.
- 7 Václav Blazej, Pratibha Choudhary, Dusan Knop, Jan Matyás Kristan, Ondrej Suchý, and Tomás Valla. Constant factor approximation for tracking paths and fault tolerant feedback vertex set. In Jochen Könemann and Britta Peis, editors, *Approximation and Online Algorithms - 19th International Workshop, WAOA 2021, Lisbon, Portugal, September 6-10, 2021, Revised Selected Papers*, volume 12982 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 2021. doi:10.1007/978-3-030-92702-8_2.
- 8 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
- 9 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. URL: <http://dl.acm.org/citation.cfm?id=2973749>, doi:10.1145/2973749.
- 10 Mikolaj Bojanczyk. Separator logic and star-free expressions for graphs. *CoRR*, abs/2107.13953, 2021. arXiv:2107.13953.
- 11 Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.
- 12 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 151–164. ACM, 2018. doi:10.1145/3188745.3188902.
- 13 Jiehua Chen, Wojciech Czerwinski, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michal Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, and Anna Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.
- 14 B. Courcelle. The monadic second-order logic of graphs. III. Tree-decompositions, minors and complexity issues. *RAIRO Inform. Théor. Appl.*, 26(3):257–286, 1992.
- 15 B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of graph grammars and computing by graph transformation, Vol. 1*, pages 313–400. World Sci. Publ, River Edge, NJ, 1997. doi:10.1142/9789812384720_0005.
- 16 Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85:12–75, 1990.
- 17 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, 2012.
- 18 Bruno Courcelle and R. Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discret. Appl. Math.*, 131(1):129–150, 2003. doi:10.1016/S0166-218X(02)00421-3.
- 19 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 20 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 21 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.
- 22 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

- 23 Zdenek Dvorák, Martin Kupec, and Vojtech Tuma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2014. doi:10.1007/978-3-662-44777-2_28.
- 24 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, M. S. Ramanujan, and Saket Saurabh. Solving d -sat via backdoors to small treewidth. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 630–641. SIAM, 2015. doi:10.1137/1.9781611973730.43.
- 25 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Hitting topological minors is FPT. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1317–1326. ACM, 2020. doi:10.1145/3357713.3384318.
- 26 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear kernels via dynamic programming. *SIAM J. Discret. Math.*, 29(4):1864–1894, 2015. doi:10.1137/140968975.
- 27 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 479–488, 2011. doi:10.1145/1993636.1993700.
- 28 Martin Grohe, Ken-ichi Kawarabayashi, and Bruce A. Reed. A simple algorithm for the graph minor decomposition - logic meets structural graph theory. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 414–431. SIAM, 2013. doi:10.1137/1.9781611973105.30.
- 29 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.
- 30 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811, 2014.
- 31 Naonori Kakimura and Ken-ichi Kawarabayashi. Fixed-parameter tractability for subset feedback set problems with parity constraints. *Theor. Comput. Sci.*, 576:61–76, 2015. doi:10.1016/j.tcs.2015.02.004.
- 32 Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 639–648. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.45.
- 33 Ken-ichi Kawarabayashi and Yusuke Kobayashi. Fixed-parameter tractability for the subset feedback set problem and the s -cycle packing problem. *J. Comb. Theory, Ser. B*, 102(4):1020–1034, 2012. doi:10.1016/j.jctb.2011.12.001.
- 34 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory, Ser. B*, 102(2):424–435, 2012. doi:10.1016/j.jctb.2011.07.004.
- 35 Ken-ichi Kawarabayashi and Bruce A. Reed. An (almost) linear time algorithm for odd cycles transversal. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 365–378. SIAM, 2010. doi:10.1137/1.9781611973075.31.
- 36 Ken-ichi Kawarabayashi, Bruce A. Reed, and Paul Wollan. The graph minor algorithm with parity conditions. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 27–36. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.52.

- 37 Wojciech Kazana. *Query evaluation with constant delay. (L'évaluation de requêtes avec un délai constant)*. PhD thesis, École normale supérieure de Cachan, Paris, France, 2013. URL: <https://tel.archives-ouvertes.fr/tel-00919786>.
- 38 Jens Lagergren. Upper bounds on the size of obstructions and intertwiners. *J. Comb. Theory, Ser. B*, 73(1):7–40, 1998. doi:10.1006/jctb.1997.1788.
- 39 Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC '98*, pages 186–195, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/276698.276734.
- 40 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.
- 41 William Lochet, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Fault tolerant subgraphs with applications in kernelization. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPICs*, pages 47:1–47:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ITCS.2020.47.
- 42 Konrad Majewski, Michal Pilipczuk, and Marek Sokolowski. Maintaining cmso properties on dynamic structures with bounded feedback vertex number. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPICs*, pages 46:1–46:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.STACS.2023.46.
- 43 Pranabendu Misra. On fault tolerant feedback vertex set. *CoRR*, abs/2009.06063, 2020. arXiv:2009.06063.
- 44 Michal Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Torunczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 102:1–102:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.102.
- 45 Neil Robertson and Paul D. Seymour. Graph minors .XIII. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.
- 46 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 424–435. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00034.
- 47 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. *CoRR*, abs/1907.07982, 2019. arXiv:1907.07982.

Circuit Equivalence in 2-Nilpotent Algebras

Piotr Kawalek  

Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Austria
Institute of Computer Science, University of Maria Curie-Skłodowska, Lublin, Poland

Michael Kompatscher   

Department of Algebra, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Jacek Krzaczkowski  

Institute of Computer Science, University of Maria Curie-Skłodowska, Lublin, Poland

Abstract

The circuit equivalence problem $\text{CEQV}(\mathbf{A})$ of a finite algebra \mathbf{A} is the problem of deciding whether two circuits over \mathbf{A} compute the same function or not. This problem not only generalises the equivalence problem for Boolean circuits, but is also of interest in universal algebra, as it models the problem of checking identities in \mathbf{A} . In this paper we prove that $\text{CEQV}(\mathbf{A}) \in \text{P}$, if \mathbf{A} is a finite 2-nilpotent algebra from a congruence modular variety.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms; Theory of computation \rightarrow Complexity classes

Keywords and phrases circuit equivalence, identity checking, nilpotent algebra

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.45

Funding *Piotr Kawalek*: Funded by the European Union (ERC, POCOCOP, 101071674). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

This research was funded in whole or in part by National Science Centre, Poland #2021/41/N/ST6/03907. For the purpose of Open Access, the author has applied a CC-BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

Michael Kompatscher: This work has been supported by projects PRIMUS/24/SCI/008 and UN-CE/SCI/022 of Charles University.

Jacek Krzaczkowski: This research was funded in whole or in part by National Science Centre, Poland #2022/45/B/ST6/02229.

For the purpose of Open Access, the author has applied a CC-BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

Acknowledgements We would like to thank Prof. Paweł Idziak for bringing us together in the spring of 2018 (by inviting the second author to Kraków), which ultimately lead to this publication.

1 Introduction

It is a common problem in mathematics to decide whether two formal expressions are equivalent. Some well-known examples are the word problem for groups and semigroups, checking whether a Boolean formula is a Tautology, or whether two polynomials over a given ring define the same operation. In this paper we study a class of problems that generalize the latter two examples.

In the *polynomial equivalence problem* $\text{POLEQV}(\mathbf{A})$ the input consists of two polynomials p and q of the same arity over a finite algebra \mathbf{A} , and the task is to decide whether the (universally quantified) identity $p(x_1, \dots, x_n) \approx q(x_1, \dots, x_n)$ holds in \mathbf{A} . For every fixed finite algebra $\text{POLEQV}(\mathbf{A})$ is clearly in co-NP , since we can verify in polynomial time, whether



© Piotr Kawalek, Michael Kompatscher, and Jacek Krzaczkowski;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 45; pp. 45:1–45:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



the identity fails at a given tuple $(a_1, \dots, a_n) \in A^n$. So the chief question is to distinguish, for which algebras the problem is hard (i.e. co-NP-complete), tractable (in P), or possibly of some intermediate complexity. There are numerous papers investigating this question for algebras from concrete varieties, such as groups [8, 16, 10, 33, 20], rings [7, 19, 15], and semigroups [27, 4].

However, not much is known in general. One of the major obstacles in studying $\text{POLEQV}(\mathbf{A})$ systematically for all finite algebras is that the complexity strongly depends on the language of \mathbf{A} . For example, the alternating group (A_4, \cdot) has a polynomial equivalence problem that is in P; but after adding the commutator $[x, y] = x^{-1}y^{-1}xy$ to the signature we obtain the problem $\text{POLEQV}((A_4, \cdot, [x, y]))$, which is co-NP-complete (see [18]). Roughly speaking, this follows from the fact that some polynomials in the extended language are exponentially inflated in length, when expressed by only the group operations.

To resolve this problem, in [25] it was proposed to encode an input equation by circuits instead of polynomials. This approach prevents an artificial “inflation” of the input. As a consequence, the complexity then only depends on the clone of polynomial operations of the algebra, which allows for the use of universal algebraic methods. Formally we define the *circuit equivalence problem* $\text{CEQV}(\mathbf{A})$ as follows:

$\text{CEQV}(\mathbf{A})$

Input: Two circuits g_1, g_2 over \mathbf{A} with input gates x_1, \dots, x_n

Question: Is $g_1(a_1, \dots, a_n) = g_2(a_1, \dots, a_n)$ for all $(a_1, \dots, a_n) \in A^n$?

In [25], Idziak et al. set the goal to classify the computational complexity of $\text{CEQV}(\mathbf{A})$ for all algebras from congruence modular varieties. On one hand, such a classification would subsume many of the previously known results (e.g. for groups [17], rings [19] and lattices [31, 12]). On the other hand, congruence modular varieties offer a structural advantage, since tools from tame congruence theory and commutator theory work particularly well in them. As it turns out, the complexity of CEQV in the congruence modular case is strongly linked to commutator theoretical properties. By results contained in [25] and [23] it is known that CEQV for non-nilpotent algebras from congruence modular variety is co-NP-complete. On the other hand, it was shown in [3] that CEQV for supernilpotent algebras from congruence modular varieties is in P.

Since in congruence modular varieties supernilpotence implies nilpotence (see e.g. [30]), results mentioned above leave only a gap for nilpotent, but not supernilpotent algebras (Problem 2 in [25]). It was shown in [22], [28], that, under the assumption of the Exponential Time Hypothesis, the complexity of $\text{CEQV}(\mathbf{A})$ has quasipolynomial lower bounds $\Omega(2^{c(\log n)^{k-1}})$, if \mathbf{A} is nilpotent and of supernilpotent rank k (where the supernilpotent rank, introduced in [22], is one of the generalizations of group-theoretical Fitting length notion). On the other hand, under the assumption of an open conjecture in circuit complexity theory, for every nilpotent but not supernilpotent algebra there actually is an algorithm solving CEQV that has quasipolynomial running time [29]. These two conditional results indicate that nilpotent algebras of supernilpotent rank greater than 2 have coNP-intermediate complexities. Interestingly, this mirrors the situation for the polynomial equivalence problem $\text{POLEQV}(\mathbf{G})$ for solvable groups $\mathbf{G} = (G, \cdot, e, {}^{-1})$ of Fitting length k [33, 20].

As one can observe, many of recent results connected with complexity of POLEQV and CEQV are obtained under assumptions of some known hypotheses. For example, hardness results from [22], [20], [28] and [33] are proved under the assumption of Exponential Time Hypothesis (or its randomized version). On the other hand upper bounds for algorithms complexity are often obtained under the assumption of some conjectures (e.g. Strong

Exponential Size Hypothesis, Constant Degree Hypothesis), which assume lower bounds for the size of circuits computing AND. Such results can be found e.g. in [22], [24], [20] and [29]. In contrast, this paper provides unconditional results.

Our paper is structured as follows: In Section 2, we introduce some standard notation and definitions. Section 3 contains basic structural results about 2-nilpotent algebras. In Section 4 we prove that all operations $f: U^n \rightarrow L$ between a cyclic group $(U, +) = \mathbb{Z}_{p^k}$ of prime power order and a coprime $(L, +)$ are already generated by all unary functions $g: U \rightarrow L$ (and the addition on U and L). It provides us with a useful normal form for all the functions of type $U^n \rightarrow L$. In Section 5 we prove that we can check in polynomial time, whether such a normal form induces a constant function. Results of Sections 4 and 5 might be of independent interest in the study of linearly closed clonoids.

Section 6 then contains the proof that $\text{CEQV}(\mathbf{A}) \in \text{P}$ for 2-nilpotent \mathbf{A} from congruence modular varieties. Our algorithm mixes the two prevalent approaches for checking equivalence, as it first partially restricts the domain of a given circuit (as in [3]), while then doing a syntactic manipulation on the resulting circuits (as in [21]). The latter part is based on the algorithm from Section 5.

2 Preliminaries

In this paper, small bold letters always denote tuples. For instance, tuples of constants are denoted $\mathbf{a} = (a_1, \dots, a_n) \in A^n$ and tuples of variables are denoted $\mathbf{x} = (x_1, \dots, x_n)$. We are going to use standard notation and definitions from universal algebra (see e.g. [6]). We define *algebra type* (or *algebra signature*) to be a sequence of function symbols together with a corresponding arity for each symbol. For a signature F , an algebra \mathbf{A} over F is a pair $(A, F^{\mathbf{A}})$, where A is a set (the *universe* of \mathbf{A}) and $F^{\mathbf{A}} = (f^{\mathbf{A}})_{f \in F}$ is a family of finitary operations $f^{\mathbf{A}}: A^{\text{ar}(f)} \rightarrow A$. Each $f^{\mathbf{A}}$ is called a *basic operation* of \mathbf{A} . Sometimes we are not going to distinguish between the basic operation $f^{\mathbf{A}}$ and the corresponding function symbol f , but this should never cause confusion. We say \mathbf{A} is a *finite algebra* if it has a finite universe A and finitely many basic operations. By $\text{ar}(\mathbf{A})$ we denote the maximal arity of the basic operations in \mathbf{A} .

An operation that can be constructed by composing basic operations is called a *term operation* of \mathbf{A} . If also constants from A are allowed in its construction we call it a *polynomial operation* of \mathbf{A} . If for example $\mathbf{A} = (A, +, 0, -, \cdot)$ is a ring, its polynomial operations are exactly the polynomial operations over the ring in the traditional sense. The clone of all polynomial operations of \mathbf{A} is denoted by $\text{Pol}(\mathbf{A})$. We say that \mathbf{B} and \mathbf{A} are *polynomially equivalent* iff there exists an algebra \mathbf{B}' isomorphic to \mathbf{B} with $\text{Pol}(\mathbf{A}) = \text{Pol}(\mathbf{B}')$.

A properly formed string defining a polynomial operation is called a *polynomial* over \mathbf{A} (e.g. if $\mathbf{A} = (A, f^{\mathbf{A}}, g^{\mathbf{A}})$ such that f is ternary and g is binary, the expression $p(x, y, z) = g(g(x, a), f(x, x, y))$ for $a \in A$ is a polynomial over \mathbf{A}). It might seem that polynomials are the most natural way of encoding polynomial operations, however circuits offer some advantages. A *circuit* $p(x_1, \dots, x_n)$ over \mathbf{A} is a finite directed acyclic graph, such that

- all the vertices of in-degree 0 (*fan-in* 0) are labeled by a variable x_i (*input gates*), or a constant from A (*constant gates*),
- all other vertices (*gates*) are labeled by a basic operation f of \mathbf{A} , and an enumeration of the $\text{ar}(f)$ -many incoming edges (thus fan-in must be $\text{ar}(f)$).

The vertices with no outgoing edge are called *output-gates*. In this paper we will only consider circuits over \mathbf{A} with one output gate; such circuits also naturally encode the polynomial operations of \mathbf{A} . Two circuits (or polynomials) $p(x_1, \dots, x_n), q(x_1, \dots, x_n)$ over

\mathbf{A} are equivalent if they compute the same polynomial operation over \mathbf{A} . For short, let us then write $p(x_1, \dots, x_n) \approx q(x_1, \dots, x_n)$. Note that polynomials can be considered as those circuits, whose underlying digraph is a tree. Thus $\text{POLEQV}(\mathbf{A})$ reduces to $\text{CEQV}(\mathbf{A})$.

As pointed out in [25] circuits are well suited to discuss computational problems in universal algebra, by the following folklore result:

► **Lemma 2.1.** *Let \mathbf{A} and \mathbf{B} be two finite algebras with the same universe. If $\text{Pol}(\mathbf{A}) \subseteq \text{Pol}(\mathbf{B})$, then every circuit c_1 over \mathbf{A} can be rewritten in logspace into an equivalent circuit c_2 over \mathbf{B} , so that c_1 and c_2 compute the same function.*

In particular, this implies that whenever $\text{Pol}(\mathbf{A}) \subseteq \text{Pol}(\mathbf{B})$, the problem $\text{CEQV}(\mathbf{A})$ reduces to $\text{CEQV}(\mathbf{B})$ in polynomial time.

3 The structure of 2-nilpotent algebras

In this section we discuss 2-nilpotent algebras from congruence modular varieties and the structure of their polynomial clones. In general, nilpotent algebras can be defined by having a finite central series of congruences, where centrality is defined via the so-called term condition:

► **Definition 3.1.** Let \mathbf{A} be an algebra. For congruences $\alpha, \beta, \gamma \in \text{Con}(\mathbf{A})$ we say that α *centralizes β modulo γ* (and write $C(\alpha, \beta; \gamma)$) if and only if for all polynomials $p(\mathbf{x}, \mathbf{y}) \in \text{Pol}(\mathbf{A})$, and all tuples $\mathbf{a}, \mathbf{b} \in A^n$, $\mathbf{c}, \mathbf{d} \in A^m$, such that $a_i \sim_\alpha b_i$ for $i = 1, \dots, n$ and $c_j \sim_\beta d_j$ for $j = 1, \dots, m$, the implication

$$\begin{aligned} p(\mathbf{a}, \mathbf{c}) &\sim_\gamma p(\mathbf{a}, \mathbf{d}) \\ \Rightarrow p(\mathbf{b}, \mathbf{c}) &\sim_\gamma p(\mathbf{b}, \mathbf{d}) \end{aligned}$$

holds.

An algebra \mathbf{A} is called *n-nilpotent* if there is a central series of length n , i.e. a series of congruences $0_A = \alpha_0 \leq \alpha_1 \leq \dots \leq \alpha_n = 1_A$, such that $C(\alpha_{i+1}, 1_A; \alpha_i)$ for $i = 0, \dots, n-1$. An algebra \mathbf{A} is called *Abelian*, if it is 1-nilpotent, and is called *nilpotent* if it is n -nilpotent, for some natural number $n > 0$.

We refer to [11] for further background on commutator theory. For our purposes we however do not need this original definition of nilpotence, since in congruence modular varieties we have equivalent characterizations by properties of the polynomial clone. For Abelian algebras, a classical result of Herrmann states the following:

► **Theorem 3.2** ([14]). *Let \mathbf{A} be an algebra from a congruence modular variety. Then \mathbf{A} is Abelian if and only if it is polynomially equivalent to a module.*

Here R -modules are considered as algebras $(A, +, 0, -, (r)_{r \in R})$, where every scalar $r \in R$ is identified with the unary operation $r(x) = r \cdot x$. Abelian algebras from congruence modular varieties are also called *affine*, since their polynomial operations are exactly the affine operations of some module.

2-nilpotent algebras from congruence modular varieties can be characterized as a special kind of wreath product (in the sense of [32]) of two affine algebras, which is defined as follows:

► **Definition 3.3.** Let \mathbf{U} and \mathbf{L} be two affine algebras of the same type F , and let $\widehat{F} = (\widehat{f})_{f \in F}$ be a family of operations $\widehat{f}: U^k \rightarrow L$ such that the arity k of \widehat{f} is the arity of the corresponding operation symbol $f \in F$. We then define $\mathbf{L} \otimes^{\widehat{F}} \mathbf{U}$ as the algebra of type F with universe $L \times U$ and basic operations

$$f^{\mathbf{L} \otimes^{\widehat{F}} \mathbf{U}}((l_1, u_1), \dots, (l_k, u_k)) = (f^{\mathbf{L}}(l_1, \dots, l_k) + \widehat{f}(u_1, \dots, u_k), f^{\mathbf{U}}(u_1, \dots, u_k)).$$

If \widehat{F} is clear from the context, we also write $\mathbf{L} \otimes \mathbf{U}$.

By a result of Freese and McKenzie the following holds:

► **Theorem 3.4** (Corollary 7.2. in [11]). *An algebra $\mathbf{A} = (A, F^{\mathbf{A}})$ from a congruence modular variety is 2-nilpotent if and only if there are two affine algebras \mathbf{U}, \mathbf{L} of type F , and a set \widehat{F} such that $\mathbf{A} \cong \mathbf{L} \otimes^{\widehat{F}} \mathbf{U}$.*

In the following we are often going to identify 2-nilpotent \mathbf{A} with such a wreath product, and write $\mathbf{A} = \mathbf{L} \otimes \mathbf{U}$ for short. We remark however, that this representation of \mathbf{A} is in general not unique. Given a wreath product representation, we can use it to construct its polynomial expansion with some additional nice properties:

► **Lemma 3.5.** *For every finite 2-nilpotent algebra \mathbf{A}' from a congruence modular variety there exists a finite 2-nilpotent $\mathbf{A} = \mathbf{L} \otimes^{\widehat{F}} \mathbf{U}$ such that*

1. $\text{Pol}(\mathbf{A}') \subseteq \text{Pol}(\mathbf{A})$;
2. \mathbf{A} contains Abelian group operations $+, 0, -$;
3. all other basic operations $f^{\mathbf{A}}$ of \mathbf{A} are either
 - “scalar multiplications” $f^{\mathbf{A}}((l, u)) = (\lambda \cdot l, 0)$ or $f^{\mathbf{A}}((l, u)) = (0, \rho \cdot u)$, with respect to the modules equivalent to \mathbf{L} and \mathbf{U} ,
 - or of “hat type” $f^{\mathbf{A}}((l_1, u_1), \dots, (l_k, u_k)) = (\widehat{f}(u_1, \dots, u_k), 0)$.

Proof. By Theorem 3.4 we know that \mathbf{A}' is equal to a wreath product $\mathbf{L}' \otimes^{\widehat{F}'} \mathbf{U}'$ such that \mathbf{L}' and \mathbf{U}' are polynomially equivalent to two modules $(L, +, 0, -, (\lambda)_{\lambda \in R_L})$ and $(U, +, 0, -, (\rho)_{\rho \in R_U})$.

We define \mathbf{A} also to have the universe $L \times U$. The group operations of \mathbf{A} are defined by $(l_1, u_1) + (l_2, u_2) = (l_1 + l_2, u_1 + u_2)$, $0 = (0, 0)$ and $-(l, u) = (-l, -u)$. We further define the “scalar multiplications” on \mathbf{A} by $f^{\mathbf{A}}((l, u)) = (\lambda \cdot l, 0)$ for all $\lambda \in R_L$ and $f^{\mathbf{A}}((l, u)) = (0, \rho \cdot u)$ for all $\rho \in R_U$. Finally, for every $\widehat{f} \in \widehat{F}'$ we introduce a basic operation of “hat type” $(\widehat{f}(u_1, \dots, u_k), 0)$ in \mathbf{A} .

Note that \mathbf{A} is polynomially richer than \mathbf{A}' , since every basic operation $f^{\mathbf{A}'}((l_1, u_1), \dots, (l_k, u_k)) = (c + \sum_{i=1}^k \lambda_i l_i + \widehat{f}(u_1, \dots, u_k), d + \sum_{i=1}^k \rho_i u_i)$ of \mathbf{A}' is also a polynomial operation of \mathbf{A} . Moreover, \mathbf{A}' is finite, and 2-nilpotent by Theorem 3.4. ◀

For short, let us call the algebra \mathbf{A} given by Lemma 3.5 a *group coordinatization* of \mathbf{A}' . A similar construction for arbitrary nilpotent algebras (from congruence modular varieties) was discussed in [1, Theorem 4.2].

By Lemma 2.1 and Lemma 3.5 it is enough to prove that the circuit equivalence problem of every 2-nilpotent group coordinatization is in P in order to prove it for all finite 2-nilpotent algebras from congruence modular varieties. The main advantage of working in a group coordinatization $\mathbf{A} = \mathbf{L} \otimes \mathbf{U}$ is, that every circuits/polynomials can be rewritten easily, by simplifying linear combinations over the modules \mathbf{U}, \mathbf{L} , and observing that the composition of two or more operations of “hat type” is always trivial:

► **Observation 3.6.** Let $\mathbf{A} = \mathbf{L} \otimes \mathbf{U}$ be a two nilpotent group coordinatisation. If, for a circuit $p(x_1, \dots, x_n)$ over \mathbf{A} we identify every variable with $x_i = (l_i, u_i)$, then p can be rewritten in polynomial time to an expression

$$p^{\mathbf{A}}((l_1, u_1), \dots, (l_k, u_k)) = (p^{\mathbf{L}}(l_1, \dots, l_k) + \widehat{p}(u_1, \dots, u_k), p^{\mathbf{U}}(u_1, \dots, u_k)),$$

where $p^{\mathbf{L}}$ and $p^{\mathbf{U}}$ are affine combinations over the modules \mathbf{L} or \mathbf{U} respectively, and $\widehat{p}(u_1, \dots, u_n)$ is a sum of expressions of the form $\lambda \widehat{f}(\sum_{i=1}^k \rho_{1,i} u_i + c_1, \dots, \sum_{i=1}^k \rho_{1,m} u_i + c_m)$, such that $\widehat{f} \in \widehat{F}$, all $\rho_{i,j}$ are scalars of \mathbf{U} , $c_i \in U$, and λ is a scalar of \mathbf{L} .

Note that the expressions \widehat{p} in Observation 3.6 are formed by closing the basic operations $\widehat{f} \in \widehat{F}$ under affine combinations in \mathbf{U} (from the inside) and \mathbf{L} (from the outside). In the language of [9] the induced functions $\widehat{p}: U^n \rightarrow L$ form the (\mathbf{L}, \mathbf{U}) -linearly closed clonoid, which is generated by the operations \widehat{F} (and their translations by constants).

Now clearly $p^{\mathbf{A}}$ is constant, if and only if the operations given by $p^{\mathbf{L}}$, $p^{\mathbf{U}}$ and \widehat{p} are all constant. Since this task is easy to decide for the affine combinations $p^{\mathbf{L}}$ and $p^{\mathbf{U}}$, in this paper we focus mainly on the analysis of the functions $\widehat{p}: U^n \rightarrow L$.

In the special case that a finite nilpotent algebra \mathbf{A} from a modular variety is additionally of prime power size, it has several nice additional properties. In particular any such algebra is *supernilpotent*, see e.g. [3, 30] for background. We are going to use the following result for such prime power size algebras:

► **Theorem 3.7** ([3, 1]). *Assume that \mathbf{A} is a nilpotent algebra from a congruence modular variety that is finite and of prime power size. Then, there is a constant $C \leq \text{ar}(\mathbf{A})(|A| - 1)^{\log_2 |A| - 1}$ such that, for every polynomial $p(x_1, \dots, x_n)$ and any constant $0 \in A$:*

$$p(\mathbf{x}) \approx 0 \Leftrightarrow p(\mathbf{a}) = 0 \text{ for all } \mathbf{a} \in A^n(C, 0),$$

where $A^n(C, 0) := \{(a_1, \dots, a_n) \in A^n : |\{i : a_i \neq 0\}| \leq C\}$.

Theorem 3.7 was used in [3] to prove that POLEQV \mathbf{A} is in \mathbf{P} for supernilpotent algebras. This result implies also the existence of the polynomial time algorithm solving CEQV(\mathbf{A}).

4 A result on linearly closed clonoids

In this section we analyse functions $\widehat{p}: U^n \rightarrow L$ between Abelian groups $(U, +)$ and $(L, +)$ of coprime orders. We show that, in some cases, the unary functions between U and L already generate all such functions. In order to state our results, let us introduce the following notation:

► **Notation 4.1.** Let p be a prime and k be a natural number. Then, for two tuples $\mathbf{b} = (b_1, \dots, b_n)$, $\mathbf{u} = (u_1, \dots, u_n) \in (\mathbb{Z}_{p^k})^n$ we are going to use the notation $\mathbf{b} \odot \mathbf{u} = \sum_{i=1}^n b_i \cdot u_i$ for the “inner product” of the two tuples in the ring \mathbb{Z}_{p^k} .

For $\mathbf{U} = \mathbb{Z}_{p^k}$, let us call a tuple $\mathbf{b} \in U^n$ *non-degenerate*, if one of its entries is a multiplicative invertible element of U . Furthermore, let us call a non-degenerate tuple *normalized*, if the first invertible element in \mathbf{b} is equal to 1 and let us write $(U^n)^*$ for the set of all normalized tuples.

Note that, for a fixed tuple $\mathbf{b} \in U^n$ the map $\mathbf{u} \mapsto \mathbf{b} \odot \mathbf{u}$ is an affine operation and equal to a polynomial of $(U, +)$. In other words, the group $(U, +)$ can be regarded as a module over the ring $(U, +, \cdot)$. This explains the slight abuse of notation in the following, in which we use the inner product $\mathbf{b} \odot \mathbf{u}$, although talking about operations of the Abelian group $(U, +)$.

► **Theorem 4.2.** *Let $(U, +) = \mathbb{Z}_{p^k}$ for a prime power p^k and let $(L, +)$ be an Abelian group of order coprime to $|U|$. Then, for every function $f: U^n \rightarrow L$ there are unary functions $m_{\mathbf{b}}: U \rightarrow L$ for all $\mathbf{b} \in (U^n)^*$ such that*

$$f(\mathbf{u}) = \sum_{\mathbf{b} \in (U^n)^*} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}). \tag{1}$$

Let us call (1) a normal form of f .

Using the terminology from [9], Theorem 4.2 says that the set of all operations $f: U^n \rightarrow L$ is the $((U, +), (L, +))$ -linearly closed clonoid generated by all unary functions from U to L . Before we prove Theorem 4.2, note that the existence of a normal form for f is equivalent to the existence of a representation as sum $f(\mathbf{u}) = \sum_{\mathbf{b} \in U^n} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u})$, in which the coefficients range over arbitrary $\mathbf{b} \in U^n$. This follows directly from the fact that every $\mathbf{a} \in U^n$ can be uniquely written as $\mathbf{a} = c \cdot \mathbf{b}$, for $\mathbf{b} \in (U^n)^*$ and $c \in U$. Thus, if we define $m'_{\mathbf{b}}(u) = \sum_{c \in U} m_{c\mathbf{b}}(c \cdot u)$, for every $\mathbf{b} \in (U^n)^*$ we obtain a normal form $f(\mathbf{u}) = \sum_{\mathbf{b} \in (U^n)^*} m'_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u})$.

Proof of Theorem 4.2. We first show by induction on $n = 2, 3, \dots$ that the existence of normal forms for all binary functions $f: U^2 \rightarrow L$ implies that also all n -ary function $f: U^n \rightarrow L$ have a normal form.

For $n = 2$ this is trivial. For an induction step $n \rightarrow n + 1$, let $f: U^{n+1} \rightarrow L$ be an $n + 1$ -ary function. Then, for every $a \in U$, by induction hypothesis, there exist unary functions $m_{a,\mathbf{b}}: U \rightarrow L$ such that $f(\mathbf{u}, a) = \sum_{\mathbf{b} \in U^n} m_{\mathbf{b},a}(\mathbf{b} \odot \mathbf{u})$. For every $\mathbf{b} \in U^n$, we can then define the binary function $s_{\mathbf{b}}(u, v) = m_{\mathbf{b},v}(u)$. By our assumption, every $s_{\mathbf{b}}$ has a normal form. Thus also

$$f(\mathbf{u}, u_{n+1}) = \sum_{\mathbf{b} \in U^n} s_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}, u_{n+1})$$

has a normal form, which can be computed by substituting every binary $s_{\mathbf{b}}$ by its normal form and simplifying the resulting sum. This finishes the proof of our claim.

By the above, it is enough to prove the lemma for arity $n = 2$. Without loss of generality we can assume that $(L, +) = \mathbb{Z}_m$ is also a cyclic group (otherwise we take a direct decomposition $(L, +) \cong \prod_{i=1}^k (L_i, +)$ into cyclic groups. Then clearly f has a normal form, if all projections $\pi_i f$ have a normal form).

Note that it is further enough to prove that the function

$$w(u_1, u_2) := w_{(0,0)}(u_1, u_2) = \begin{cases} 1 & \text{if } (u_1, u_2) = (0, 0) \\ 0 & \text{else,} \end{cases}$$

has a normal form. If this is the case, then all other binary functions $f: U^2 \rightarrow L$ also have a normal form by the equation $f(u_1, u_2) = \sum_{a_1, a_2 \in U} f(a_1, a_2) \cdot w(u_1 - a_1, u_2 - a_2)$.

We prove that w has a normal form by induction on the exponent k of the prime power p^k . If $k = 1$, then note that

$$p \cdot w(u_1, u_2) = \sum_{i=0}^{p-1} w_0(u_1 + iu_2) - \sum_{j=1}^{p-1} w_0(j + u_2).$$

where

$$w_0(u) = \begin{cases} 1 & \text{if } u = 0 \\ 0 & \text{else.} \end{cases}$$

This was shown before in [2, Lemma 5.3], and can easily be verified by a case distinction. Since L is coprime to U , p has a multiplicative inverse p^{-1} in L , and thus $w(u_1, u_2) = p^{-1}(\sum_{i=0}^{p-1} w_0(u_1 + iu_2) - \sum_{j=1}^{p-1} w_0(j + u_2))$, which can be rewritten to a normal form.

For an induction step $k - 1 \rightarrow k$, let us first define the auxiliary function

$$t(u_1, u_2) = \sum_{i=0}^{p^k-1} w_0(u_1 + iu_2) + \sum_{i=0}^{p^{k-1}-1} w_0(pi u_1 + u_2).$$

We claim that $t(u_1, u_2) = p^k w(u_1, u_2) + \min(\frac{p^{k-1}}{|pu_1|}, \frac{p^{k-1}}{|pu_2|})$, where $|u|$ denotes the order of the group element $u \in U$. To prove this claim, note that for the two sums defining t we have:

$$\sum_{i=0}^{p^k-1} w_0(u_1 + iu_2) = \begin{cases} \frac{p^k}{|u_2|} & \text{if } |u_2| \geq |u_1| \\ 0 & \text{else,} \end{cases} \quad \text{and} \quad \sum_{i=0}^{p^{k-1}-1} w_0(piu_1 + u_2) = \begin{cases} \frac{p^{k-1}}{|pu_1|} & \text{if } |pu_1| \geq |u_2| \\ 0 & \text{else.} \end{cases}$$

Observe further that $u \neq 0$ is equivalent to $1 < |u| = p \cdot |pu|$. Thus, if $u_1 \neq 0$, then $t(u_1, u_2) = \frac{p^{k-1}}{|pu_2|}$ for $|u_2| \geq |u_1|$ and $t(u_1, u_2) = \frac{p^{k-1}}{|pu_1|}$ for $|u_2| < |u_1|$; in other words $t(u_1, u_2) = \min(\frac{p^{k-1}}{|pu_1|}, \frac{p^{k-1}}{|pu_2|})$. If $u_1 = 0$ and $u_2 \neq 0$, then $t(u_1, u_2) = \frac{p^k}{|u_2|} = \min(p^{k-1}, \frac{p^{k-1}}{|pu_2|})$. Finally if $u_1 = 0$ and $u_2 = 0$, then $t(u_1, u_2) = p^k + p^{k-1} = p^k + \min(p^{k-1}, p^{k-1})$.

Thus we have verified that $t(u_1, u_2) = p^k w(u_1, u_2) + \min(\frac{p^{k-1}}{|pu_2|}, \frac{p^{k-1}}{|pu_1|})$. If we define $r: (pU)^2 \rightarrow L$ as the function $r(pu_1, pu_2) = \min(\frac{p^{k-1}}{|pu_2|}, \frac{p^{k-1}}{|pu_1|})$, then, by the induction assumption (and $pU \cong \mathbb{Z}_{p^{k-1}}$) r has a normal form. Since also t has (by definition) a normal form, it follows that $w(u_1, u_2) = p^{-k} \cdot (t(u_1, u_2) - r(pu_1, pu_2))$ has a normal form. This finishes the proof. ◀

As a direct consequence of Theorem 4.2 we obtain the following version of it for direct products:

► **Corollary 4.3.** *Let $(U, +) = \mathbb{Z}_{p^k}$ for a prime power p^k , and $(L, +)$ be an Abelian group of coprime order. Then, for any set V and any function $f: U^n \times V \rightarrow L$ there are functions $m_{\mathbf{b}}: U \times V \rightarrow L$ for all $\mathbf{b} \in U^n$ such that $f(\mathbf{u}, v) = \sum_{\mathbf{b} \in (U^n)^*} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}, v)$.*

Proof. For any fixed value $a \in V$, there is a normal form $f(\mathbf{u}, a) = \sum_{\mathbf{b} \in (U^n)^*} m_{\mathbf{b},a}(\mathbf{b} \odot \mathbf{u})$ by Theorem 4.2. Thus the functions $m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}, v) = m_{\mathbf{b},v}(\mathbf{b} \odot \mathbf{u})$ give us the above normal form. ◀

This corollary is in particular of interest, if we consider the direct products of cyclic groups \mathbb{Z}_{p^k} . Let us then use the following notation:

► **Notation 4.4.** For a list of prime powers $\mathbf{p} = (p_1^{k_1}, p_2^{k_2}, \dots, p_m^{k_m})$, let us define the ring $\mathbb{Z}_{\mathbf{p}} = \prod_{i=1}^m (\mathbb{Z}_{p_i^{k_i}})$. Moreover, for a list of positive integers $\mathbf{n} = (n_1, n_2, \dots, n_m)$, let us define the \mathbf{n} -th power $\mathbb{Z}_{\mathbf{p}}$ as $(\mathbb{Z}_{\mathbf{p}})^{\mathbf{n}} = \prod_{i=1}^m (\mathbb{Z}_{p_i^{k_i}})^{n_i}$. For short, let us also write $\mathbb{Z}_{\mathbf{p}}$ for $\mathbb{Z}_{\mathbf{p}}^{(1,1,\dots,1)}$ and $\mathbb{Z}_{\mathbf{p}}^{\mathbf{n}}$ for $(\mathbb{Z}_{\mathbf{p}})^{\mathbf{n}}$. For every index $i = 1, \dots, m$, let $\mathbf{u}^{(i)}$ denote the projection of $\mathbf{u} \in \mathbb{Z}_{\mathbf{p}}^{\mathbf{n}}$ to $(\mathbb{Z}_{p_i^{k_i}})^{n_i}$.

For two tuples $\mathbf{b}, \mathbf{u} \in \mathbb{Z}_{\mathbf{p}}^{\mathbf{n}}$ we define their “inner product”

$$\mathbf{b} \odot \mathbf{u} = (\mathbf{b}^{(1)} \odot \mathbf{u}^{(1)}, \mathbf{b}^{(2)} \odot \mathbf{u}^{(2)}, \dots, \mathbf{b}^{(m)} \odot \mathbf{u}^{(m)}) \in \mathbb{Z}_{\mathbf{p}}.$$

Note that, for a fixed $\mathbf{b} \in \mathbb{Z}_{\mathbf{p}}^{\mathbf{n}}$ the map $\mathbf{u} \mapsto \mathbf{b} \odot \mathbf{u}$ is a linear map from $\mathbb{Z}_{\mathbf{p}}^{\mathbf{n}}$ to $\mathbb{Z}_{\mathbf{p}}$. In particular, for $\mathbb{Z}_{\mathbf{p}}^{(n,n,\dots,n)} \cong (\mathbb{Z}_{\mathbf{p}})^n$ it can be considered as an n -ary polynomials of the affine algebra $(\mathbb{Z}_{\mathbf{p}}, +, \pi_1, \dots, \pi_m)$, where $\pi_i((u^{(1)}, \dots, u^{(n)})) = (0, \dots, 0, u^{(i)}, 0, \dots, 0)$.

Let us call a tuple $\mathbf{b} \in \mathbb{Z}_{\mathbf{p}}^{\mathbf{n}}$ *non-degenerate/normalized*, if $\mathbf{b}^{(i)}$ is non-degenerate/normalized for every component $i = 1, \dots, m$, and let us write $(\mathbb{Z}_{\mathbf{p}}^{\mathbf{n}})^*$ for the set of normalized tuples.

► **Corollary 4.5.** *Let $(U, +) = \mathbb{Z}_{\mathbf{p}}$ for a list of prime powers $\mathbf{p} = (p_1^{k_1}, p_2^{k_2}, \dots, p_m^{k_m})$ and let $(L, +)$ be an Abelian group of order coprime to $|U|$. Then, for any $\mathbf{n} \in \mathbb{N}^m$ and any $f: U^{\mathbf{n}} \rightarrow L$ there are functions $m_{\mathbf{b}}: U \rightarrow L$ for all $\mathbf{b} \in (U^{\mathbf{n}})^*$ such that*

$$f(\mathbf{u}) = \sum_{\mathbf{b} \in (U^{\mathbf{n}})^*} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}) = \sum_{\mathbf{b} \in (U^{\mathbf{n}})^*} m_{\mathbf{b}}(\mathbf{b}^{(1)} \odot \mathbf{u}^{(1)}, \dots, \mathbf{b}^{(m)} \odot \mathbf{u}^{(m)}).$$

We call this representation a normal form of f .

Proof. This follows directly from iteratively applying Corollary 4.3 to all the components of the direct product $\mathbb{Z}_{\mathbf{p}} = \prod_{i \in 1}^m \mathbb{Z}_{p_i^{k_i}}$. ◀

At last, we show that for coprime modules \mathbf{U} and \mathbf{L} , all functions in a finitely generated (\mathbf{L}, \mathbf{U}) -clonoid can be rewritten into such a normal form in polynomial time. Note here, that whenever some $m_{\mathbf{b}}$ is equal to the constant $\mathbf{0}$ function, we can just skip it in the representation of the normal form. In this way we can avoid summing over the entire $(U^n)^*$ (which has exponential size).

► **Lemma 4.6.** *Let \mathbf{U} and \mathbf{L} be two finite modules over coprime domains, let \mathbf{p} be list of prime powers $\mathbf{p} = (p_1^{k_1}, p_2^{k_2}, \dots, p_m^{k_m})$ such that $(U, +) = \mathbb{Z}_{\mathbf{p}}$ is the group reduct of \mathbf{U} . Let \widehat{F} be a finite set of operations from U to L . Then any n -ary function \widehat{p} in the (\mathbf{L}, \mathbf{U}) -clonoid generated by \widehat{F} can be rewritten in polynomial time into a normal form*

$$\widehat{p}(\mathbf{u}) = \sum_{\mathbf{b} \in X} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u})$$

where $\mathbf{n} = (n, n, \dots, n) \in \mathbb{N}^m$ and $X \subseteq (U^n)^*$.

Proof. Recall that any $\widehat{p}(u_1, \dots, u_n)$ in an (\mathbf{L}, \mathbf{U}) -clonoid is a sum of expressions $\lambda \widehat{f}(\sum_{i=1}^k \rho_{1,i} u_i + c_1, \dots, \sum_{i=1}^k \rho_{m,i} u_i + c_m)$, with $\widehat{f} \in \widehat{F}$. It is thus enough to prove, that every such summand can be rewritten into a normal form.

The naive way to do this, would be to substitute every $\widehat{f} \in \widehat{F}$ by its normal form, and simplify the resulting sum. There is however a catch: The ring R of the module $\mathbf{U} = (U, +, 0, -, (\rho)_{\rho \in R})$ is possibly different from $(\mathbb{Z}_{\mathbf{p}}, +, 0, -, \cdot, 1)$. This problem can be resolved by computing normal forms for all functions

$$\widehat{f}(\sum_{\rho \in R} \rho u_{1,\rho}, \sum_{\rho \in R} \rho u_{2,\rho}, \dots, \sum_{\rho \in R} \rho u_{n,\rho}), \tag{2}$$

for $\widehat{f} \in \widehat{F}$ and distinct variables $u_{i,\rho}$ for all indices i and coefficients $\rho \in R$.

To rewrite an expression $\lambda \widehat{f}(\sum_{i=1}^k \rho_{1,i} u_i + c_1, \dots, \sum_{i=1}^k \rho_{1,m} u_i + c_m)$, we then collect in every argument of \widehat{f} all variables according to their coefficients from R , and then substitute the normal form for the expression (2). ◀

5 A recursive principle

Let $(U, +) = \mathbb{Z}_{\mathbf{p}}$ and $(L, +)$ be two finite Abelian groups of coprime order. By Corollary 4.5 we know that every function $f: U^n \rightarrow L$ is equal to the sum of operations $m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u})$. In this section we prove that we can check in polynomial time whether an f given by such a normal form is constant. Our algorithm is based on the fact that a normal form is constant, if and only if it can be partitioned in certain constant subsums, where the partition is formed with respect to the following equivalence relation \sim :

► **Definition 5.1.** Let $(U, +) = \mathbb{Z}_{p^k}$. Then, for two tuples $\mathbf{a}, \mathbf{b} \in (U^n)^*$, let us write $\mathbf{a} \sim \mathbf{b}$, if $\mathbf{a} - \mathbf{b} \in (pU)^n$.

Note that, if $\mathbf{a} \sim \mathbf{b}$, then an entry a_i is invertible (i.e. not a multiple of p) if and only if b_i is invertible.

45:10 Circuit Equivalence in 2-Nilpotent Algebras

► **Proposition 5.2.** *Let $(U, +) = \mathbb{Z}_{p^k}$ and let $(L, +)$ be of order coprime to U . Let $f: U^n \rightarrow L$ be an operation given by the normal form*

$$f(\mathbf{u}) = \sum_{\mathbf{b} \in (U^n)^*} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}).$$

Then f is constant if and only if for every $\mathbf{a} \in (U^n)^*$ the sum

$$f_{\mathbf{a}}(\mathbf{u}) = \sum_{\mathbf{b} \in [\mathbf{a}]_{\sim}} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u})$$

is constant.

Proof. If $f_{\mathbf{a}}$ is constant for all $\mathbf{a} \in (U^n)^*$, then obviously f is also constant, since we can pick the transversal $\mathbf{a}_1, \dots, \mathbf{a}_s$ of \sim and the statement can be inferred from $f(\mathbf{u}) = \sum_{i=1}^s f_{\mathbf{a}_i}(\mathbf{u})$.

For the other direction, we first assume that $\mathbf{a} = (1, 0, \dots, 0)$. Now in the case where $f(\mathbf{u}) = \sum_{\mathbf{b} \in (U^n)^*} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u})$ is constant, we are going to prove an even stronger statement, namely that for every $i = 0, \dots, k$ we have

$$\sum_{\mathbf{b} \in [\mathbf{a}]_{\sim}} \sum_{c \in p^i U} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u} + c) \text{ is constant.} \quad (3)$$

Note that the case $i = k$ in (3) says, that the expression $\sum_{\mathbf{b} \in [\mathbf{a}]_{\sim}} \sum_{c \in p^i U} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}) = f_{\mathbf{a}}(\mathbf{u})$ represents a constant function.

We prove (3) by induction on $i = 0, 1, \dots, k$. For $i = 0$, the statement is true, since then the inner sum $\sum_{c \in U} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u} + c) = \sum_{c \in U} m_{\mathbf{b}}(c)$ is constant for every \mathbf{b} .

For an induction step $i \rightarrow i + 1$, let us define $C = p^{i+1}U \times (p^i U)^{n-1}$. Then, the sum

$$\sum_{\mathbf{c} \in C} f(\mathbf{u} + \mathbf{c}) = \sum_{\mathbf{b} \in (U^n)^*} \sum_{\mathbf{c} \in C} m_{\mathbf{b}}(\mathbf{b} \odot (\mathbf{u} + \mathbf{c})), \quad (4)$$

is constant, since f is constant.

Note that for every $\mathbf{g} \in (U^n)^*$, which is not equivalent to $\mathbf{a} = (1, 0, \dots, 0)$, there is an index $j \neq 1$, such that g_j is invertible. Therefore, if we restrict the sum (4) to only summands from the equivalence class of such a \mathbf{g} , we obtain

$$\sum_{\mathbf{b} \in [\mathbf{g}]_{\sim}} \sum_{\mathbf{c} \in C} m_{\mathbf{b}}(\mathbf{b} \odot (\mathbf{u} + \mathbf{c})) = \frac{|C|}{p^{k-i}} \sum_{\mathbf{b} \in [\mathbf{g}]_{\sim}} \sum_{c_j \in p^i U} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u} + c_j),$$

which is constant by induction assumption.

This, together with (4) being constant implies that also

$$\sum_{\mathbf{b} \in [\mathbf{a}]_{\sim}} \sum_{\mathbf{c} \in C} m_{\mathbf{b}}(\mathbf{b} \odot (\mathbf{u} + \mathbf{c})) = \frac{|C|}{p^{k-i-1}} \sum_{\mathbf{b} \in [\mathbf{a}]_{\sim}} \sum_{c_1 \in p^{i+1}U} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u} + c_1)$$

is constant. Since $\frac{|C|}{p^{k-i-1}}$ is a power of p , it has an inverse in L . Thus also $\sum_{\mathbf{b} \in [\mathbf{a}]_{\sim}} \sum_{c_1 \in p^{i+1}U} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u} + c_1)$ is constant. This finishes the proof of (3), and therefore also the proof of the proposition in case $\mathbf{a} = (1, 0, \dots, 0)$.

Now, to make this proof work also for $\mathbf{a} \neq (1, 0, \dots, 0)$, we need the following.

▷ **Claim 1.** For a prime p and a natural number k , let $U = \mathbb{Z}_{p^k}$ and let $\mathbf{a} \in (U^n)^*$. There exist two linear maps $T'_{\mathbf{a}}(\mathbf{u})$ and $T_{\mathbf{a}}(\mathbf{u})$ of type $U^n \mapsto U^n$ such that

1. $T_{\mathbf{a}}$ maps $(a_1, \dots, a_n) \rightarrow (1, 0, \dots, 0)$,
2. both $T'_{\mathbf{a}}$ and $T_{\mathbf{a}}$ are linear bijections from $U^n \rightarrow U^n$,
3. function $T_{\mathbf{a}}$ preserves the equivalence relation \sim ,
4. $\mathbf{d} \odot T'_{\mathbf{a}}(\mathbf{u}) = T_{\mathbf{a}}(\mathbf{d}) \odot \mathbf{u}$

Let j denote the first coordinate such that $a_j = 1$ and let $S_j = \{1, \dots, n\} \setminus \{1, j\}$. One can check that the following definitions of

$$T'_{\mathbf{a}}(\mathbf{u}) = (u_j, u_2, u_3, \dots, u_{j-1}, u_1 - a_1 u_j - \sum_{i \in S_j} a_i u_i, u_{j+1}, \dots, u_n)$$

and

$$T_{\mathbf{a}}(\mathbf{d}) = (d_j, d_2 - d_j \cdot a_2, d_3 - d_j \cdot a_3, \dots, d_{j-1} - d_j \cdot a_{j-1}, d_1 - (d_j) \cdot a_1, d_{j+1} - d_j \cdot a_{j+1}, \dots, d_n - d_j \cdot a_n)$$

satisfy these four conditions (for $j = 1$ formulas should be interpreted as $T'_{\mathbf{a}}(\mathbf{u}) = (u_1 - \sum_{i=2}^n a_i u_i, u_2, \dots, u_n)$ and $T_{\mathbf{a}}(\mathbf{d}) = (d_1, d_2 - d_1 \cdot a_2, d_3 - d_1 \cdot a_3, \dots, d_n - d_1 \cdot a_n)$).

Hence, if we now define $f'(\mathbf{u}) := f(T'_{\mathbf{a}}(\mathbf{u}))$ we can see that f' is a function with a normal form defined by $m'_{\mathbf{b}} = m_{(T_{\mathbf{a}})^{-1}(\mathbf{b})}$. This shows, that the Proposition is true for the pair $(f, [\mathbf{a}]_{\sim})$ iff it is true for the pair $(f', [(1, 0, \dots, 0)]_{\sim})$. This finishes the proof, as we already considered the case when $\mathbf{a} = (1, 0, \dots, 0)$. \blacktriangleleft

As we work not only with $(U, +) = \mathbb{Z}_{p^k}$, but also with more general groups $(U, +) = \mathbb{Z}_{\mathbf{p}}$, we need to adjust our definitions accordingly.

► **Definition 5.3.** For a list of prime powers $\mathbf{p} = (p_1^{k_1}, \dots, p_m^{k_m})$, let $(U, +) = \mathbb{Z}_{\mathbf{p}}$ and let $\mathbf{n} \in \mathbb{N}^m$. For an $i \in \{1, \dots, m\}$ let us say that two elements the $\mathbf{a}, \mathbf{b} \in (U^{\mathbf{n}})^*$ are in equivalence relation $\mathbf{a} \sim_i \mathbf{b}$ if and only if they satisfy $\mathbf{a}^{(i)} \sim \mathbf{b}^{(i)}$ in $\mathbb{Z}_{p_i^{k_i}}$.

Here is a very important corollary which is a direct consequence of Proposition 5.2 applied to the direct component $\mathbb{Z}_{p_i^{k_i}}$ of $\mathbb{Z}_{\mathbf{p}}$.

► **Corollary 5.4.** For a list of prime powers $\mathbf{p} = (p_1^{k_1}, \dots, p_m^{k_m})$, let $(U, +) = \mathbb{Z}_{\mathbf{p}}$ and $(L, +)$ be a finite Abelian group of coprime order. For $\mathbf{n} \in \mathbb{N}^m$ let $f: U^{\mathbf{n}} \rightarrow L$ be an operation given by a normal form

$$f(\mathbf{u}) = \sum_{\mathbf{b} \in (U^{\mathbf{n}})^*} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}).$$

Then for every $i \in \{1, \dots, m\}$: function f is constant if and only if for every $\mathbf{a} \in (U^{\mathbf{n}})^*$ we have that

$$f_{i,\mathbf{a}}(\mathbf{u}) = \sum_{\mathbf{b} \in [\mathbf{a}]_{\sim_i}} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u})$$

is constant.

Now, we are going to use Corollary 5.4 to check in polynomial time, if a function in a normal form is constant.

► **Lemma 5.5.** For a fixed tuple of prime powers $\mathbf{p} = (p_1^{k_1}, \dots, p_m^{k_m})$, let $(U, +) = \mathbb{Z}_{\mathbf{p}}$, and $(L, +)$ be a finite Abelian group of coprime orders. Then, for $\mathbf{n} \in \mathbb{N}^m$ and any function $f: U^{\mathbf{n}} \rightarrow L$ that is given by a normal form

$$f(\mathbf{u}) = \sum_{\mathbf{b} \in X} m_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}),$$

for some $X \subseteq (U^{\mathbf{n}})^*$, we can decide in time $O(|f|^C)$ whether f is constant or not (with C depending only on \mathbf{p}).

45:12 Circuit Equivalence in 2-Nilpotent Algebras

Proof. The algorithm, that we are going to present, is based on a recursion. In each recursive call, both the total number of variables on all coordinates (i.e. $n_1 + n_2 + \dots + n_m$), as well as the sum of all exponents (i.e. $k_1 + \dots + k_m$) are decreased by at least one.

Note that, in the case when for all $i = 1, \dots, m$ we have either $n_i = 0$ or $k_i = 0$, the function f is constant. So first, we pick an appropriate coordinate i with $n_i \neq 0$ and $k_i \neq 0$. Now we take a transversal $\mathbf{a}_1, \dots, \mathbf{a}_l$ of $\sim_i \cap (X \times X)$. From Corollary 5.4 we know that the function f is constant iff. all the $f_{i,\mathbf{a}}$'s are constant, for all $\mathbf{a} \in \{\mathbf{a}_1, \dots, \mathbf{a}_l\}$.

Now, we will slightly transform those $f_{i,\mathbf{a}}$'s by applying linear maps to their arguments. We will use linear maps from Claim 1 applied to the i -th component of U , that is: $T'_{\mathbf{a}^{(i)}}(u_1^{(i)}, \dots, u_{n_i}^{(i)})$, $T_{\mathbf{a}^{(i)}}(d_1^{(i)}, \dots, d_{n_i}^{(i)})$ in order to define linear maps on the entire U as follows:

$$(T'_{\mathbf{a}}(\mathbf{u}))^{(k)} = \begin{cases} T'_{\mathbf{a}^{(i)}}(u_1^{(i)}, \dots, u_{n_i}^{(i)}) & \text{if } k = i \\ (u_1^{(k)}, \dots, u_{n_k}^{(k)}) & \text{otherwise} \end{cases}$$

as well as

$$(T_{\mathbf{a}}(\mathbf{d}))^{(k)} = \begin{cases} T_{\mathbf{a}^{(i)}}(d_1^{(i)}, \dots, d_{n_i}^{(i)}) & \text{if } k = i \\ (d_1^{(k)}, \dots, d_{n_k}^{(k)}) & \text{otherwise} \end{cases}$$

Note, that those maps only act on the variables from i -th component of U and keep other variables untouched. Since $T'_{\mathbf{a}}(\mathbf{u})$ is just a permutation of U^n , instead of checking that $f_{i,\mathbf{a}}(\mathbf{u})$ is constant we can check that $f'_{i,\mathbf{a}}(\mathbf{u}) = f_{i,\mathbf{a}}(T'_{\mathbf{a}}(\mathbf{u}))$ is constant. Moreover, we can see that $\mathbf{d} \odot T'_{\mathbf{a}}(\mathbf{u}) = T_{\mathbf{a}}(\mathbf{d}) \odot \mathbf{u}$ (like in the Claim 1), so we can actually compute the normal form of each such $f'_{i,\mathbf{a}}$, as it is given by the formula:

$$f'_{i,\mathbf{a}}(\mathbf{u}) = \sum_{\mathbf{b} \in X'} m'_{\mathbf{b}}(\mathbf{b} \odot \mathbf{u}),$$

where $m'_{\mathbf{b}} = m_{(T_{\mathbf{a}})^{-1}(\mathbf{b})}$ and $X' = T_{\mathbf{a}}([\mathbf{a}]_{\sim_i} \cap X)$.

In order to check that such created $f'_{i,\mathbf{a}}$'s are constant we will substitute constants $c \in \mathbb{Z}_{p_i^{k_i}}$ for the variable $u_1^{(i)}$ in $f'_{i,\mathbf{a}}$ and recursively check that such created $f'_{i,\mathbf{a}}[u_1^{(i)} = c]$ are constant. Additionally, we have to also make sure, that the returned constants are equal for all different $c \in \mathbb{Z}_{p_i^{k_i}}$. For this purpose, it is enough to assign to all variables the value 0 and check that the set $\{f'_{i,\mathbf{a}}[u_1^{(i)} = c](0, \dots, 0) : c \in \mathbb{Z}_{p_i^{k_i}}\}$ has size one.

Before the recursive call, we made a substitution for the variable and thus reduced the number of variables (on i -th coordinate) by one. But the effort taken to compute this $f'_{i,\mathbf{a}}$, instead of applying substitutions directly to f , will now provide us with an additional benefit. It turns out, that as a side effect of this substitution, we have also implicitly reduced the size of the domain U . To see it, recall that $T_{\mathbf{a}}(\mathbf{a})^{(i)} = (1, 0, \dots, 0)$ and $T_{\mathbf{a}}$ preserves the \sim_i relation (by Claim 1). It means that all the $\mathbf{b} \odot \mathbf{u}$ that occur in the normal form of $f'_{i,\mathbf{a}}$ on the i -th coordinate have a very special form: $\mathbf{b}^{(i)} \odot \mathbf{u}^{(i)} = u_1^{(i)} + p_i \cdot (\mathbf{d}_{\mathbf{b}} \odot (u_2^{(i)}, \dots, u_{n_i}^{(i)}))$, for some $\mathbf{d}_{\mathbf{b}} \in (\mathbb{Z}_{p_i^{k_i}})^{n_i-1}$. So now, when we substitute constant c for $u_1^{(i)}$, the normal form of $f'_{i,\mathbf{a}}$ transforms into the expression:

$$\sum_{\mathbf{b} \in X'} m'_{\mathbf{b}}(\mathbf{b}^{(1)} \odot \mathbf{u}^{(1)}, \dots, c + p_i \cdot (\mathbf{d}_{\mathbf{b}} \odot (u_2^{(i)}, \dots, u_{n_i}^{(i)})), \dots, \mathbf{b}^{(m)} \odot \mathbf{u}^{(m)}),$$

Here, linear combinations of variables $u_j^{(i)}$ are in fact computable in $\mathbb{Z}_{p_i^{k_i-1}}$, since $p_i \cdot \mathbb{Z}_{p_i^{k_i}} \cong \mathbb{Z}_{p_i^{k_i-1}}$ (where \cong denotes an additive group isomorphism). So we can just reinterpret the variables to the new domain, and normalize the obtained form, so that now we can

recursively check if $f'_{i,\mathbf{a}}[u_1^{(i)} = c]$ is constant, when treated as a function over the domain $U' = (\mathbb{Z}_{p_1^{k_1}}) \times \dots \times (\mathbb{Z}_{p_i^{k_i-1}}) \times \dots \times (\mathbb{Z}_{p_m^{k_m}})$. For completeness, notice, that while going from f to the normalized form of $f'_{i,\mathbf{a}}[u_1^{(i)} = c]$, some variable other than $u_1^{(i)}$ can disappear. To handle it, we can just decrease the number of variables appropriately before the recursive call. The described procedure can be summarized as follows.

■ **Algorithm 1** For a fixed Abelian $(L, +)$, this algorithm takes as input a list of prime powers $\mathbf{p} = (p_1^{k_1}, \dots, p_m^{k_m})$ coprime to $|L|$, a list of arities $\mathbf{n} = (n_1, \dots, n_m)$, and checks whether a function $f: (\mathbb{Z}_{\mathbf{p}})^{\mathbf{n}} \rightarrow L$ given by a normal form $f(\mathbf{u}) = \sum_{\mathbf{b} \in X} (\mathbf{b} \odot \mathbf{u})$ is constant.

```

1: procedure ISCONSTANT( $\mathbf{p}, \mathbf{n}, f: (\mathbb{Z}_{\mathbf{p}})^{\mathbf{n}} \rightarrow L$ )
2:   if for all  $i = 1, \dots, m$ :  $n_i = 0$  or  $p_i^{k_i} = 1$  then return True
3:   else
4:     Let  $i$  be the minimal value such that  $n_i, k_i \neq 0$ 
5:     Let  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_s$  be a transversal of  $\sim_i$  inside  $X \times X$ 
6:     for all  $\mathbf{a} \in \{\mathbf{a}_1, \dots, \mathbf{a}_s\}$  do
7:       for all  $c \in \mathbb{Z}_{p_i^{k_i}}$  do
8:         Compute a normal form of  $f'_{i,\mathbf{a}}[u_1^{(i)} = c]$ 
9:         Compute new domain  $\mathbf{p}'$  and new arities  $\mathbf{n}'$ 
10:        if  $\neg$  ISCONSTANT( $\mathbf{p}', \mathbf{n}', f'_{i,\mathbf{a}}[u_1^{(i)} = c]$ ) then return False
11:        if  $|\{f'_{i,\mathbf{a}}[u_1^{(i)} = c](0, \dots, 0) : c \in \mathbb{Z}_{p_i^{k_i}}\}| \neq 1$  then return False
12:   return True

```

Now we analyse the running time of the above algorithm. Procedure ISCONSTANT($\mathbf{p}, \mathbf{n}, f$) first computes at most $p_i^{k_i} |f|$ -many functions $f'_{i,\mathbf{a}}[u_1^{(i)} = c]$, whose normal forms have sizes bounded by $|f|$. To obtain them we need to regroup the normal form of f into \sim_i classes and apply linear map T' , which can be done with a naive quadratic algorithm. For the obtained functions, it compares values $f'_{i,\mathbf{a}}[u_1^{(i)} = c](0, \dots, 0)$, which takes linear time in $|f|$. Moreover, all the normalizations can be done in a linear time. The recursion depth of ISCONSTANT is at most $\sum_{i=1}^m k_i$, thus we obtain a running time of $O(|f|^2 \cdot |f|^{(k_1 + \dots + k_m)})$.

A careful reader can see, that actually the sum of lengths of expressions that are computed during the runtime of the above algorithm is bounded by a linear function in $|f|$. Using this observation one can prove an even more accurate result, namely that the presented algorithm is in fact quadratic in the worst case. ◀

6 Proof of the main theorem

We are now ready to prove the main theorem:

▶ **Theorem 6.1.** *Let \mathbf{A} be a finite 2-nilpotent algebra from a congruence modular variety. Then we can decide in time $\mathcal{O}(n^C)$ whether an n -ary circuit $p(x_1, \dots, x_n)$ over \mathbf{A} represents a constant function, where C depends only on \mathbf{A} . In particular, this implies that $\text{CEQV}(\mathbf{A}) \in \mathbf{P}$.*

Proof. By Lemma 2.1 and Lemma 3.5, we can without loss of generality assume that $\mathbf{A} = \mathbf{L} \otimes^{\widehat{F}} \mathbf{U}$ is a group extension. If we identify every variable x_i of the circuit $p(x_1, \dots, x_n)$ with a pair $x_i = (l_i, u_i)$ of variables over L and U , then, by Observation 3.6, we can rewrite it in polynomial time to an expression

$$p^{\mathbf{A}}((l_1, u_1), \dots, (l_k, u_k)) = (p^{\mathbf{L}}(l_1, \dots, l_k) + \widehat{p}(u_1, \dots, u_k), p^{\mathbf{U}}(u_1, \dots, u_k)),$$

where $p^{\mathbf{L}}(l_1, \dots, l_k) = c + \sum_{i=1}^k \lambda_i l_i$ and $p^{\mathbf{U}}(u_1, \dots, u_k) = d + \sum_{i=1}^k \rho_i u_i$ are affine combinations in the modules \mathbf{L} and \mathbf{U} respectively, and $\widehat{p}(u_1, \dots, u_n)$ is a sum of expressions $\lambda \widehat{f}(\sum_{i=1}^k \rho_{1,i} u_i + c_1, \dots, \sum_{i=1}^k \rho_{1,m} u_i + c_m)$, for $\widehat{f} \in \widehat{F}$.

Clearly $p^{\mathbf{A}}$ is constant if and only if $p^{\mathbf{L}}$, $p^{\mathbf{U}}$ and \widehat{p} are constant. For the affine operations $p^{\mathbf{L}}$, $p^{\mathbf{U}}$ we can check this, by simply checking whether all coefficients λ_i and ρ_i are equal to 0. Thus the problem reduces to checking, whether the expression $\widehat{p}(u_1, \dots, u_n)$ defines a constant function.

Since \mathbf{L} is a module, it has a direct decomposition $\mathbf{L} = \prod_{i=1}^r \mathbf{L}_i$ into factors of prime power size. If π_i denotes the projection of L to L_i , then \widehat{p} is constant if and only if $\pi_i \circ \widehat{p}$ is constant for every $i = 1, \dots, m$. Thus, without loss of generality, we can assume that the size of \mathbf{L} is a power of some prime q . Also \mathbf{U} can be directly decomposed into $\mathbf{U} = \mathbf{U}_1 \times \mathbf{U}_2$ such that $|\mathbf{U}_1|$ is a power of q , and $|\mathbf{U}_2|$ is coprime to q . Let us then identify every variable u over U with its direct decomposition $(u^{(1)}, u^{(2)})$ with respect to $\mathbf{U}_1 \times \mathbf{U}_2$.

Now we want to check that $\widehat{p}(\mathbf{u}^{(1)}, \mathbf{u}^{(2)})$ is constant. Note, that $\widehat{p}(\mathbf{u}^{(1)}, \mathbf{u}^{(2)})$ is an expression of $(\mathbf{L}, (\mathbf{U}_1 \times \mathbf{U}_2))$ -clonoid. However, by fixing $\mathbf{u}^{(2)}$ to some constant $\mathbf{a}^{(2)} \in (\mathbf{U}_2)^n$ we create $\widehat{p}(\mathbf{u}^{(1)}, \mathbf{a}^{(2)})$, which is an expression of $(\mathbf{L}, \mathbf{U}_1)$ -clonoid. This clonoid is generated by all the functions $\widehat{f}_{\mathbf{b}^{(2)}}(\mathbf{u}^{(1)}) = \widehat{f}(\mathbf{u}^{(1)}, \mathbf{b}^{(2)})$ and is of prime power order. Hence, we can associate this $(\mathbf{L}, \mathbf{U}_1)$ -clonoid with a 2-nilpotent algebra of prime power order (q is the prime here). By Theorem 3.7 there is a set S of polynomial size $O(n^C)$ (and independent of $\mathbf{a}^{(2)}$), such that $\widehat{p}(\mathbf{u}^{(1)}, \mathbf{a}^{(2)})$ is constant iff it is constant on the set S . So now, going back to $(\mathbf{L}, (\mathbf{U}_1 \times \mathbf{U}_2))$ -clonoid, our expression $\widehat{p}(\mathbf{u}^{(1)}, \mathbf{u}^{(2)})$ represents a constant function iff all the $\widehat{p}(\mathbf{a}^{(1)}, \mathbf{u}^{(2)})$ represent the same constant function c for all $\mathbf{a}^{(1)} \in S$. So, in order to check that $\widehat{p}(\mathbf{u}^{(1)}, \mathbf{u}^{(2)})$ is constant, it is enough to pick arbitrary tuple $\mathbf{0} \in \mathbf{U}_2$, check that $\{\mathbf{p}(\mathbf{a}^{(1)}, \mathbf{0}) : \mathbf{a}^{(1)} \in S\}$ is one element set, and then check that each $\widehat{p}(\mathbf{a}^{(1)}, \mathbf{u}^{(2)})$ is constant.

Since the set S is of polynomial size, this is a polynomial-time Turing reduction from the problem over the $(\mathbf{L}, (\mathbf{U}_1 \times \mathbf{U}_2))$ -clonoid to the problem over the $(\mathbf{L}, \mathbf{U}_2)$ -clonoid, where this $(\mathbf{L}, \mathbf{U}_2)$ -clonoid is generated by all operations $\widehat{f}_{\mathbf{b}^{(1)}}(\mathbf{u}^{(2)}) = \widehat{f}(\mathbf{b}^{(1)}, \mathbf{u}^{(2)})$, for $\widehat{f} \in \widehat{F}$ and $\mathbf{b}^{(1)} \in (\mathbf{U}_1)^{\text{ar}(f)}$. Since $|\mathbf{L}|$ and $|\mathbf{U}_2|$ are coprime, by Lemma 4.6 we know that $\widehat{p}(\mathbf{a}^{(1)}, \mathbf{u}^{(2)})$ can be rewritten into a normal form $\sum_{\mathbf{b}^{(2)} \in X} m_{\mathbf{b}^{(2)}}(\mathbf{b}^{(2)} \odot \mathbf{u}^{(2)})$ in polynomial time (where $X \subseteq ((\mathbf{U}_2)^n)^*$, with $\mathbf{n} = (\text{ar}(f), \dots, \text{ar}(f))$). By Lemma 5.5 we can check in polynomial time, whether this normal form represents a constant function. Thus we can check in polynomial time whether $\widehat{p}(\mathbf{u}^{(1)}, \mathbf{u}^{(2)})$ is constant.

In order to obtain an algorithm for $\text{CEQV}(\mathbf{A})$, note that any identity $p \approx q$ is equivalent to $p + (-q) \approx 0$ over \mathbf{A} (recall that \mathbf{A} is a group extension), so $\text{CEQV}(\mathbf{A})$ can be solved by checking whether $p + (-q)$ is constant and evaluates to 0 at some tuple. ◀

7 Conclusions and open problems

As it was mentioned in the introduction there is a characterization (under assumptions of ETH and CDH) of algebras from a congruence modular variety for which CEQV can be solved in randomized polynomial time. Moreover, we are not far from obtaining a similar characterization of algebras for which CEQV can be solved in deterministic polynomial time. The only case we have to consider to obtain such a characterization is algebras of supernilpotent rank 2, i.e for every algebra \mathbf{A} having supernilpotent congruence α such that \mathbf{A}/α is also supernilpotent. Note that in this paper we show a deterministic polynomial time algorithm for every algebra \mathbf{A} having an abelian congruence α such that \mathbf{A}/α is also abelian. The interesting question is if we can extend our recursive principle to all algebras with supernilpotent rank 2. Note that there are many structural similarities between 2-nilpotent algebras and algebras with supernilpotent rank 2.

This leads us to the following question.

► **Problem 1.** *Let \mathbf{A} be a finite algebra from congruence modular variety with supernilpotent rank 2.*

Is there a deterministic polynomial time algorithm solving $\text{CEQV } \mathbf{A}$?

Note that the probabilistic algorithm solving CSAT for nilpotent algebras of supernilpotent rank 2 relies on Constant Degree Hypothesis (introduced in [5]), i.e. the conjecture that there are no subexponential size $\text{AND}_d \circ \text{MOD}_m \circ \text{MOD}_p$ -circuits computing AND_n function of arbitrarily large arity, where d and m are some constant integers and p is a prime number. Despite the fact that proving CDH will not automatically give us a deterministic polynomial time algorithm solving CEQV for algebras of supernilpotent rank 2, it is hard to believe that such an algorithm can exist in case CDH fails. It leads us to a natural question.

► **Problem 2.** *Does Constant Degree Hypothesis hold?*

Although CDH is a quite a long-standing hypothesis, we strongly believe that it holds. It is already proven in some restricted settings, for instance when the number of connections between AND_d gates and MOD_m gates is restricted [13]. Recently, Kawałek and Weiss [26] have shown that if there exist circuits witnessing that CDH fails they have to be non-symmetric.

The natural next step after characterizing algebras from congruence modular varieties for which CEQV can be solved in polynomial time is to study the computational complexity of CEQV for algebras outside congruence modular variety. The most notable example of such algebras are semigroups.

► **Problem 3.** *For which semigroups can CEQV be solved in (deterministic) polynomial time?*

References

- 1 Erhard Aichinger. Bounding the free spectrum of nilpotent algebras of prime power order. *Israel Journal of Mathematics*, 230(2):919–947, 2019. doi:10.1007/s11856-019-1846-x.
- 2 Erhard Aichinger and Peter Mayr. Polynomial clones on groups of order pq . *Acta Mathematica Hungarica*, 114(3):267–285, 2006. doi:10.1007/s10474-006-0530-x.
- 3 Erhard Aichinger and Nebojša Mudrinski. Some applications of higher commutators in Mal’cev algebras. *Algebra universalis*, 63(4):367–403, 2010. doi:10.1007/s00012-010-0084-1.
- 4 Jorge Almeida, Mikhail V. Volkov, and Svetlana V. Goldberg. Complexity of the identity checking problem for finite semigroups. *Journal of Mathematical Sciences*, 158(5):605–614, 2009. doi:10.1007/s10958-009-9397-z.
- 5 David Mix Barrington, Howard Straubing, and Denis Thérien. Non-uniform automata over groups. *Information and Computation*, 89(2):109–132, 1990. doi:10.1016/0890-5401(90)90007-5.
- 6 Clifford Bergman. *Universal Algebra: Fundamentals and selected topics*. CRC Press, 2011.
- 7 Stanley Burris and John Lawrence. The equivalence problem for finite rings. *Journal of Symbolic Computation*, 15(1):67–71, 1993. doi:10.1006/jsc.1993.1004.
- 8 Stanley Burris and John Lawrence. Results on the equivalence problem for finite groups. *Algebra Universalis*, 52(4):495–500, 2005. doi:10.1007/s00012-004-1895-8.
- 9 Stefano Fioravanti. Closed sets of finitary functions between finite fields of coprime order. *Algebra universalis*, 81:52, 2020. published online. doi:10.1007/s00012-020-00683-5.
- 10 Attila Földvári and Gábor Horváth. The complexity of the equation solvability and equivalence problems over finite groups. *International Journal of Algebra and Computation*, 30(03):607–623, 2020. doi:10.1142/S0218196720500137.

- 11 Ralph Freese and Ralph McKenzie. *Commutator theory for congruence modular varieties*, volume 125. CUP Archive, 1987.
- 12 Tomasz Gorazd and Jacek Krzaczkowski. The complexity of problems connected with two-element algebras. *Reports on Mathematical Logic*, 2011(46):91–108, 2011.
- 13 Vince Grolmusz and Gábor Tardos. Lower bounds for $(\text{MOD}_p\text{-MOD}_m)$ circuits. *SIAM Journal on Computing*, 29(4):1209–1222, 2000. doi:10.1137/S0097539798340850.
- 14 Christian Herrmann. Affine algebras in congruence modular varieties. *Acta Universitatis Szegediensis*, 41:119–125, 1979.
- 15 Gábor Horváth. The complexity of the equivalence problem over finite rings. *Glasgow Mathematical Journal*, 54(1):193–199, 2012. doi:10.1017/S001708951100053X.
- 16 Gábor Horváth and Csaba Szabó. The complexity of checking identities over finite groups. *International Journal of Algebra and Computation*, 16(5):931–939, 2006. doi:10.1142/S0218196706003256.
- 17 Gábor Horváth and Csaba Szabó. The extended equivalence and equation solvability problems for groups. *Discrete Mathematics & Theoretical Computer Science*, 13(4):23–32, 2011. doi:10.46298/dmtcs.536.
- 18 Gábor Horváth and Csaba Szabó. Equivalence and equation solvability problems for the alternating group A_4 . *Journal of Pure and Applied Algebra*, 216(10):2170–2176, 2012. doi:10.1016/j.jpaa.2012.02.007.
- 19 Harry B. Hunt III and Richard Edwin Stearns. The complexity of equivalence for commutative rings. *Journal of Symbolic Computation*, 10(5):411–436, 1990. doi:10.1016/S0747-7171(08)80053-3.
- 20 Paweł Idziak, Piotr Kawałek, Jacek Krzaczkowski, and Armin Weiß. Equation satisfiability in solvable groups. *Theory of Computing Systems*, 2022. doi:10.1007/s00224-022-10082-z.
- 21 Paweł M. Idziak, Piotr Kawałek, and Jacek Krzaczkowski. Expressive power, satisfiability and equivalence of circuits over nilpotent algebras. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:15. EATCS, 2018. doi:10.4230/LIPIcs.MFCS.2018.17.
- 22 Paweł M. Idziak, Piotr Kawałek, and Jacek Krzaczkowski. Intermediate problems in modular circuits satisfiability. In *Proceedings of the 35th Annual Symposium on Logic in Computer Science (LICS)*, pages 578–590, 2020. doi:10.1145/3373718.3394780.
- 23 Paweł M. Idziak, Piotr Kawałek, and Jacek Krzaczkowski. Satisfiability of Circuits and Equations over Finite Malcev Algebras. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2022.37.
- 24 Paweł M. Idziak, Piotr Kawałek, and Jacek Krzaczkowski. Nonuniform deterministic finite automata over finite algebraic structures, 2024. Manuscript.
- 25 Paweł M Idziak and Jacek Krzaczkowski. Satisfiability in multivalued circuits. *SIAM Journal on Computing*, 51(3):337–378, 2022. doi:10.1137/18M122019.
- 26 Piotr Kawałek and Armin Weiß. Violating constant degree hypothesis requires breaking symmetry, 2023. arXiv:2311.17440.
- 27 Ondřej Klíma. Complexity issues of checking identities in finite monoids. *Semigroup Forum*, 79(3):435–444, 2009. doi:10.1007/s00233-009-9180-y.
- 28 Michael Kompatscher. CSAT and CEQV for nilpotent Maltsev algebras of Fitting length > 2 , 2021. arXiv:2105.00689.
- 29 Michael Kompatscher. CC-circuits and the expressive power of nilpotent algebras. *Logical Methods in Computer Science*, 18(2), 2022. doi:10.46298/lmcs-18(2:12)2022.
- 30 Andrew Moorhead. Higher commutator theory for congruence modular varieties. *Journal of Algebra*, 513:133–158, 2018. doi:10.1016/j.jalgebra.2018.07.026.

- 31 Bernhard Schwarz. The complexity of satisfiability problems over finite lattices. In *Proceedings of the 21st Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 31–43, 2004. doi:10.1007/978-3-540-24749-4_4.
- 32 Joel VanderWerf. Wreath products of algebras: generalizing the Krohn-Rhodes theorem to arbitrary algebras. *Semigroup Forum*, 52(1):93–100, 1996. doi:10.1007/BF02574084.
- 33 Armin Weiß. Hardness of Equations over Finite Solvable Groups Under the Exponential Time Hypothesis. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 102:1–102:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.102.

The Subpower Membership Problem of 2-Nilpotent Algebras

Michael Kompatscher   

Department of Algebra, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Abstract

The subpower membership problem $\text{SMP}(\mathbf{A})$ of a finite algebraic structure \mathbf{A} asks whether a given partial function from A^n to A can be interpolated by a term operation of \mathbf{A} , or not. While this problem can be EXPTIME-complete in general, Willard asked whether it is always solvable in polynomial time if \mathbf{A} is a Mal'tsev algebra. In particular, this includes many important structures studied in abstract algebra, such as groups, quasigroups, rings, Boolean algebras. In this paper we give an affirmative answer to Willard's question for a big class of 2-nilpotent Mal'tsev algebras. We furthermore develop tools that might be essential in answering the question for general nilpotent Mal'tsev algebras in the future.

2012 ACM Subject Classification Theory of computation \rightarrow Algebraic complexity theory

Keywords and phrases subpower membership problem, Mal'tsev algebra, compact representation, nilpotence, clonoids

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.46

Funding This paper was supported by projects PRIMUS/24/SCI/008 and UNCE/SCI/022 of Charles University.

Acknowledgements I would like to thank Peter Mayr for introducing me to difference clonoids and giving several helpful comments on earlier versions of this paper. Furthermore I would like to thank the anonymous referees for their many helpful remarks.

1 Introduction

It is a recurring and well-studied problem in algebra to describe the closure of a given list of elements under some algebraic operations (let us only mention the affine and linear closure of a list of vectors, or the ideal generated by a list of polynomials). But also in a computational context, this problem has a rich history, appearing in many areas of computer science. In its formulation as *subalgebra membership problem*, the task is to decide whether a given finite list of elements of an algebraic structure generates another element or not.

Depending on the algebraic structures studied, a variety of different problems emerges. One of the most well-known examples is the *subgroup membership problem*, in which the task is to decide, if for a given set of permutations $\alpha_1, \dots, \alpha_n$ on a finite set X , another permutation β belongs to the subgroup generated by $\alpha_1, \dots, \alpha_n$ in S_X . This problem can be solved in polynomial-time by the famous Schreier-Sims algorithm [30], whose runtime was analysed in [15] and [19]. The existence of such efficient algorithms is however not always guaranteed: if the symmetric group S_X is for instance replaced by the full transformation semigroup on X , the corresponding membership problem is PSPACE-complete [22].

A common feature of many algorithms for the subalgebra membership problem is to generate canonical generating sets of some sorts (such as computing the basis of a vector space via Gaussian elimination, or computing a Gröbner basis via Buchberger's algorithm to solve the ideal membership problem [6]). But, in general, this is where the similarities



© Michael Kompatscher;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 46; pp. 46:1–46:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



end - depending on the algebraic structure, and the encoding of the input, the problem can range over a wide range of complexities, and have applications in vastly different areas such as cryptography [28, 29], computer algebra [6, 24], or proof complexity [22, 21].

In this paper, we study a version of the subalgebra membership problem that is called the *subpower membership problem*. For a fixed, finite algebraic structure \mathbf{A} (henceforth also just called an *algebra*) its subpower membership problem $\text{SMP}(\mathbf{A})$ is the problem of deciding if a given tuple $\mathbf{b} \in \mathbf{A}^k$ is in the subalgebra of \mathbf{A}^k generated by some other input tuples $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbf{A}^k$ (here n and k are not fixed, but part of the input). This is equivalent to checking, whether the n -ary partial function that maps $\mathbf{a}_1, \dots, \mathbf{a}_n$ component-wise to \mathbf{b} can be interpolated by a term function of \mathbf{A} . For example, if p is a prime, $\text{SMP}(\mathbb{Z}_p)$ is the problem of checking whether some vector $\mathbf{b} \in \mathbb{Z}_p^k$ is in the linear closure of $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbb{Z}_p^k$; this can easily be solved by Gaussian elimination. More general, for any finite group \mathbf{G} , $\text{SMP}(\mathbf{G})$ can be solved in polynomial time by a version of the Schreier-Sims algorithm [32].

Besides being a natural problem in algebra, the subpower membership problem found some applications in some learning algorithms [7, 12, 17]. Moreover, an efficient algorithm for $\text{SMP}(\mathbf{A})$ implies that it is also feasible to represent the relations invariant by some generating set of tuples. It was in particular remarked (see e.g. [9]), that a polynomial-time algorithm for $\text{SMP}(\mathbf{A})$ would allow to define infinitary constraint satisfaction problems, in which the constraint relations are given by some generating tuples (with respect to \mathbf{A}). This infinitary version of CSPs has the benefit that most of the algebraic machinery to CSPs (see e.g. [3]) still applies.

Exhaustively generating the whole subalgebra generated by $\mathbf{a}_1, \dots, \mathbf{a}_n$ in \mathbf{A}^k gives an exponential time algorithm for $\text{SMP}(\mathbf{A})$. And, in general, we cannot expect to do better: In [23] Kozik constructed a finite algebra \mathbf{A} for which $\text{SMP}(\mathbf{A})$ is EXP-complete. Even semigroups can have PSPACE-complete subpower membership problem [8].

However, for so called *Mal'tsev algebras*, better upper bounds are known. Mal'tsev algebras are algebras defined by having a *Mal'tsev term* m , i.e. a term satisfying the identities $y = m(x, x, y) = m(y, x, x)$ for all x, y . Mal'tsev algebras lie at the intersection of many areas of mathematics: they include algebraic structures of ubiquitous importance (groups, fields, vector spaces), but also appear in logic (Boolean algebras, Heyting algebras), commutative algebra (rings, modules, K -algebras), and non-associative mathematics (quasigroups, loops). Mayr showed in [25] that the subpower membership problem of every Mal'tsev algebra is in NP. His proof is based on the fact that every subalgebra $\mathbf{R} \leq \mathbf{A}^n$ has a small generating set, which generates every element of \mathbf{R} in a canonical way (a so-called *compact representation*). Thus, to solve the subpower membership problem, one can "guess" a compact representation of the subalgebra generated by $\mathbf{a}_1, \dots, \mathbf{a}_k$, and then check in polynomial time if it generates \mathbf{b} . If such a compact representation can be moreover found in *deterministic* polynomial time, then $\text{SMP}(\mathbf{A})$ is in P; this is, in fact, the dominant strategy to prove tractability.

So far, the existence of such polynomial time algorithms was verified for groups and rings [32, 15], supernilpotent algebras [25], and algebras that generate residually finite varieties [9]. On the other hand, no examples of NP-hard or intermediate complexity are known. This leads to the question whether $\text{SMP}(\mathbf{A}) \in \text{P}$ for *all* finite Mal'tsev algebras \mathbf{A} [32]. On a broader scale, this question was also posed for algebras with *few subpowers* [17, Question 8].

An elementary class of Mal'tsev algebras, for which the question still remains open, are *nilpotent* algebras. In fact, they can also be seen as an important stepping stone in answering [17, Question 8], as nilpotent Mal'tsev algebras coincide with nilpotent algebras with few subpowers. Generalizing the concept of nilpotent groups, nilpotent algebras are defined by

having a central series of congruences. While they have several nice structural properties, in general nilpotent algebras do not satisfy the two finiteness conditions mentioned above (supernilpotence, residual finiteness), thus they are a natural starting point when trying to generalize known tractability results. But even for 2-nilpotent algebras not much is known: all polynomial-time algorithms were only constructed by ad-hoc arguments for concrete examples (such as Vaughan-Lee's 12-element loop [26]).

The first contribution of this paper is to prove that all 2-nilpotent algebras of size $p \cdot q$ for two primes $p \neq q$ have a tractable subpower membership problem. In fact, we prove an even stronger result in Theorem 20: $\text{SMP}(\mathbf{A})$ is in \mathbf{P} , whenever \mathbf{A} has a central series $0_{\mathbf{A}} < \rho < 1_{\mathbf{A}}$ such that $|\mathbf{A}/\rho| = p$ is a prime, and the blocks of ρ have size coprime to p .

While this is still a relatively restricted class of nilpotent algebras, our methods have the potential to generalize to all 2-nilpotent Mal'tsev algebras and beyond. Thus, our newly developed tools to analyze SMP can be regarded as the second main contribution. More specifically, in Theorem 11 we show that whenever $\mathbf{L} \otimes \mathbf{U}$ is a *wreath product* (see Section 3), such that \mathbf{U} is supernilpotent, then $\text{SMP}(\mathbf{L} \otimes \mathbf{U})$ reduces to $\text{SMP}(\mathbf{L} \times \mathbf{U})$ (which is polynomial-time solvable by [25]) and a version of the subpower membership problem for a multi-sorted algebraic object called a *clonoid* from \mathbf{U} to \mathbf{L} . This reduction in particular applies to all 2-nilpotent algebras; an analysis of clonoids between affine algebras then leads to Theorem 20. If, in future research, we could get rid of the condition of \mathbf{U} being supernilpotent, this would provide a strong tool in studying general Mal'tsev algebras, as every Mal'tsev algebra with non-trivial center can be decomposed into a wreath product.

Our paper is structured as follows: Section 2 contains preliminaries and some background on universal algebra. In Section 3 we discuss how Mal'tsev algebras with non-trivial center can be represented by a wreath product and we introduce the concept of *difference clonoid* of such a representation. In Section 4 we discuss some situations, in which the subpower membership problem of a wreath product can be reduced to the membership problem of the corresponding difference clonoid. In particular, we prove Theorem 11. Section 5 contains an analysis of clonoids between \mathbb{Z}_p and coprime Abelian groups, which then leads to the proof of our main result, Theorem 20. In Section 6 we discuss some possible directions for future research.

2 Preliminaries

In the following, we are going to discuss some necessary notions from universal algebra. For more general background we refer to the textbooks [4, 11]. For background on commutator theory we refer to [14] and [2]. For an introduction to Mal'tsev algebras and compact representations we refer to [5, Chapters 1.7-1.9].

In this paper, we are going to denote tuples by lower case bold letters, e.g. $\mathbf{a} \in A^k$. In order to avoid double indexing in some situations, we are going to use the notation $\mathbf{a}(i)$ to denote the i -th entry of \mathbf{a} , i.e. $\mathbf{a} = (\mathbf{a}(1), \mathbf{a}(2), \dots, \mathbf{a}(k))$. However, otherwise we are going to follow standard notation as used e.g. in [4].

2.1 Basic notions for general algebras

An *algebra* $\mathbf{A} = (A; (f_i^{\mathbf{A}})_{i \in I})$ is a first-order structure in a purely functional language $(f_i)_{i \in I}$ (where each symbol f_i has an associated *arity*). We say \mathbf{A} is finite if its domain A is finite. A *subalgebra* $\mathbf{B} = (B; (f_i^{\mathbf{B}})_{i \in I})$ of an algebra $\mathbf{A} = (A; (f_i^{\mathbf{A}})_{i \in I})$ (denoted $\mathbf{B} \leq \mathbf{A}$) is an algebra obtained by restricting all *basic operations* $f_i^{\mathbf{A}}$ to a subset $B \subseteq A$ that is invariant under all $f_i^{\mathbf{A}}$'s. The subalgebra generated by a list of elements a_1, \dots, a_n , denoted by $\text{Sg}_{\mathbf{A}}(a_1, \dots, a_n)$

is the smallest subalgebra of \mathbf{A} that contains a_1, \dots, a_n . The *product* $\prod_{i \in I} \mathbf{A}_i$ of a family of algebras $(\mathbf{A}_i)_{i \in I}$ in the same language is defined as the algebra with domain $\prod_{i \in I} A_i$, whose basic operations are defined coordinate-wise. The power \mathbf{A}^n is the product of n -many copies of \mathbf{A} . Subalgebras of (finite) powers of \mathbf{A} are sometimes also called *subpowers* of \mathbf{A} , which motivates the name “subpower membership problem”. So, formally the subpower membership problem of \mathbf{A} can be stated as follows:

SMP(\mathbf{A})

INPUT: $\mathbf{b}, \mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbf{A}^k$ for some $n, k \in \mathbb{N}$

QUESTION: Is $\mathbf{b} \in \text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$?

Note that the subpowers of \mathbf{A} are exactly the relations on A that are invariant under \mathbf{A} . A *congruence* α of \mathbf{A} is an equivalence relation on A that is invariant under \mathbf{A} . We write $\text{Con}(\mathbf{A})$ for the lattice of all congruence of \mathbf{A} . We denote the minimal and maximal element of this lattice by $0_{\mathbf{A}} = \{(x, x) \mid x \in A\}$ and $1_{\mathbf{A}} = \{(x, y) \mid x, y \in A\}$. For every congruence $\alpha \in \text{Con}(\mathbf{A})$, one can form a quotient algebra \mathbf{A}/α in the natural way.

The *term operations* of an algebra \mathbf{A} are all finitary operations that can be defined by a composition of basic operations of \mathbf{A} . Two standard ways to represent them is by terms or circuits in the language of \mathbf{A} . For a term or circuit $t(x_1, \dots, x_n)$ in the language of \mathbf{A} , we write $t^{\mathbf{A}}(x_1, \dots, x_n)$ for the induced term operation on A . Occasionally, if it is clear from the context, we are not going to distinguish between a term/circuit and the corresponding term operation. The term operations of an algebra \mathbf{A} are closed under composition and contain all projections, therefore they form an algebraic object called a *clone*. For short, we denote this *term clone* of an algebra \mathbf{A} by $\text{Clo}(\mathbf{A})$. Note that $\text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n) = \{t(\mathbf{a}_1, \dots, \mathbf{a}_n) \mid t \in \text{Clo}(\mathbf{A})\}$.

We call a ternary operation $m^{\mathbf{A}}(x, y, z) \in \text{Clo}(\mathbf{A})$ a *Mal'tsev term* if it satisfies the identities $m^{\mathbf{A}}(y, x, x) = m^{\mathbf{A}}(x, x, y) = y$ for all $x, y \in A$, and call \mathbf{A} a *Mal'tsev algebra* if it has a Mal'tsev term. For instance, every group is a Mal'tsev algebra with Mal'tsev term $m(x, y, z) = xy^{-1}z$. Mal'tsev terms are a classic topic of study in universal algebra (see e.g. [4, Chapter 7]), and are in particular known to characterize congruence permutable varieties.

2.2 Clonoids

We are also going to rely on a multi-sorted generalisation of clones, so-called *clonoids* that were first introduced in [1] (in a slightly less general way). For a set of operations between two sets $\mathcal{C} \subseteq \{f: A^n \rightarrow B \mid n \in \mathbb{N}\}$, and $k \in \mathbb{N}$ let us write $\mathcal{C}^{(k)} = \{f: A^k \rightarrow B \mid f \in \mathcal{C}\}$ for the subset of k -ary functions. Then, for two algebras $\mathbf{A} = (A, (f_i)_{i \in I})$, $\mathbf{B} = (B, (g_j)_{j \in J})$ (in possibly different domains and languages), a set $\mathcal{C} \subseteq \{f: A^n \rightarrow B \mid n \in \mathbb{N}\}$ is called a *clonoid from \mathbf{A} to \mathbf{B}* , or (\mathbf{A}, \mathbf{B}) -*clonoid*, if it is closed under composition with term operations of \mathbf{A} from the inside, and \mathbf{B} from the outside, i.e.: $\forall n, k \in \mathbb{N}$

- (1) $f \in \mathcal{C}^{(n)}, t_1, \dots, t_n \in \text{Clo}(\mathbf{A})^{(k)} \Rightarrow f \circ (t_1, \dots, t_n) \in \mathcal{C}^{(k)}$
- (2) $s \in \text{Clo}(\mathbf{B})^{(n)}, f_1, \dots, f_n \in \mathcal{C}^{(k)} \Rightarrow s \circ (f_1, \dots, f_n) \in \mathcal{C}^{(k)}$.

2.3 Commutator theory

Commutator theory is the subfield of universal algebra that tries to generalise notions such as central subgroups, nilpotence, or solvability from group theory to general algebras. The most commonly used framework is based on so-called term-conditions, which we outline in the following.

Let \mathbf{A} be an algebra. For congruences $\alpha, \beta, \gamma \in \text{Con}(\mathbf{A})$ we say that α *centralizes* β modulo γ (and write $C(\alpha, \beta; \gamma)$) if and only if for all $p(\mathbf{x}, \mathbf{y}) \in \text{Clo}(\mathbf{A})$, and all tuples $\mathbf{a}, \mathbf{b} \in A^n$, $\mathbf{c}, \mathbf{d} \in A^m$, such that $a_i \sim_\alpha b_i$ for $i = 1, \dots, n$ and $c_j \sim_\beta d_j$ for $j = 1, \dots, m$, the implication

$$p(\mathbf{a}, \mathbf{c}) \sim_\gamma p(\mathbf{a}, \mathbf{d}) \Rightarrow p(\mathbf{b}, \mathbf{c}) \sim_\gamma p(\mathbf{b}, \mathbf{d})$$

holds. A congruence α is called *central* if $C(\alpha, 1_{\mathbf{A}}; 0_{\mathbf{A}})$ holds. The *center* is the biggest central congruence. An algebra \mathbf{A} is called *n-nilpotent* if there is a *central series of length n*, i.e. a series of congruences $0_{\mathbf{A}} = \alpha_0 \leq \alpha_1 \leq \dots \leq \alpha_n = 1_{\mathbf{A}}$, such that $C(\alpha_{i+1}, 1_{\mathbf{A}}; \alpha_i)$ for $i = 0, \dots, n-1$. An algebra \mathbf{A} is called *Abelian*, if it is 1-nilpotent, i.e. $C(1_{\mathbf{A}}, 1_{\mathbf{A}}; 0_{\mathbf{A}})$ holds.

We are, however, not going to work directly with these definitions. There is a rich structural theory in the special case of Mal'tsev algebras (and, more general, in congruence modular varieties [14]) that gives us very useful characterizations of many commutator theoretical properties.

By a result of Herrmann [16], a Mal'tsev algebra \mathbf{A} is Abelian if and only if it is *affine*, i.e. all of its term operations are affine combination $\sum_{i=1}^n \alpha_i x_i + c$ over some module; in particular the Mal'tsev term is then equal to $x - y + z$. More generally, we are going to use a result of Freese and McKenzie [14] that states that a Mal'tsev algebra \mathbf{A} with a central congruence ρ can always be written as a *wreath product* $\mathbf{L} \otimes \mathbf{U}$, such that \mathbf{L} is affine and $\mathbf{U} = \mathbf{A}/\rho$. We are going to discuss such wreath product representations in Section 3.

Lastly, we want to mention that the definition of the relation C naturally generalizes to higher arities $C(\alpha_1, \dots, \alpha_n, \beta; \gamma)$. This notion was first introduced by Bulatov; we refer to [14] and [2] to more background on *higher commutators*. In particular, an algebra is called *k-supernilpotent* if $C(1_{\mathbf{A}}, \dots, 1_{\mathbf{A}}; 0_{\mathbf{A}})$, where $1_{\mathbf{A}}$ appears $k+1$ times. There are several known characterizations of supernilpotent Mal'tsev algebras. We are mainly going to use the following:

► **Theorem 1** (Proposition 7.7. in [2]). *Let \mathbf{A} be a k-supernilpotent Mal'tsev algebra, $0 \in A$ a constant and t, s two n-ary terms in the language of \mathbf{A} . Then $t^{\mathbf{A}} = s^{\mathbf{A}}$ if and only if they are equal on all tuples from the set $S = \{\mathbf{a} \in A^n \mid |\{i : \mathbf{a}(i) \neq 0\}| \leq k\}$. (In fact, \mathbf{A} is k-supernilpotent iff that equivalence holds for all terms t, s).*

2.4 Compact representations and SMP

For any subset $R \subseteq A^n$, we define its *signature* $\text{Sig}(R)$ to be the set of all triples $(i, a, b) \in \{1, \dots, n\} \times A^2$, such that there are $\mathbf{t}_a, \mathbf{t}_b \in R$ that agree on the first $i-1$ coordinates, and $t_a(i) = a$ and $t_b(i) = b$; we then also say that $\mathbf{t}_a, \mathbf{t}_b$ are *witnesses* for $(i, a, b) \in \text{Sig}(R)$.

If \mathbf{A} is a Mal'tsev algebra, and $\mathbf{R} \leq \mathbf{A}^n$, then it is known that \mathbf{R} is already generated by every subset $S \subseteq \mathbf{R}$ with $\text{Sig}(S) = \text{Sig}(\mathbf{R})$ [5, Theorem 1.8.2.]. In fact, \mathbf{R} is then equal to the closure of S under the Mal'tsev operation m alone, and a tuple \mathbf{a} is in \mathbf{R} if \mathbf{a} can be written as $m(\dots m(\mathbf{a}_1, \mathbf{b}_2, \mathbf{a}_2), \dots, \mathbf{b}_n, \mathbf{a}_n)$, for some $\mathbf{a}_i, \mathbf{b}_i \in S$. For given $\mathbf{a} \in \mathbf{R}$ such elements $\mathbf{a}_i, \mathbf{b}_i \in S$ can be found polynomial time in $|S|$, by picking \mathbf{a}_1 such that $\mathbf{a}_1(1) = \mathbf{a}(1)$, and $\mathbf{a}_i, \mathbf{b}_i \in S$ are witnesses for $a(i)$ and $m(\dots m(\mathbf{a}_1, \mathbf{b}_2, \mathbf{a}_2), \dots, \mathbf{b}_{i-1}, \mathbf{a}_{i-1})(i)$ at position i .

A *compact representation* of $\mathbf{R} \leq \mathbf{A}^n$ is a subset $S \subset \mathbf{R}$ with $\text{Sig}(S) = \text{Sig}(\mathbf{R})$ and $|S| \leq 2|\text{Sig}(\mathbf{R})| \leq 2n|A|^2$. So, informally speaking, compact representations are small generating sets of \mathbf{R} with the same signature. It is not hard to see that compact representations always exist. Generalizations of compact representations exist also for relations on different domains ($\mathbf{R} \leq \mathbf{A}_1 \times \dots \times \mathbf{A}_n$), and relations invariant under algebras with few subpowers, we refer to [5, Chapter 2] for more background.

By the above, $\text{SMP}(\mathbf{A})$ reduces in polynomial time to the problem of finding a compact representation of $\text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$ for some input tuples $\mathbf{a}_1, \dots, \mathbf{a}_n \in A^k$. We are going to denote this problem by $\text{CompRep}(\mathbf{A})$. Conversely, it was shown in [9] that finding a compact representations has a polynomial Turing reduction to $\text{SMP}(\mathbf{A})$. Note further that, to solve $\text{CompRep}(\mathbf{A})$ it is already enough to find a subset $S \subseteq R$ with $\text{Sig}(S) = \text{Sig}(R)$ of polynomial size, since such a set S can then be thinned out to a compact representation.

Let us call a set of pairs $\{(c, p_c) \mid c \in S\}$ an *enumerated compact representation* of $\text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$, if S is a compact representation of $\text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$, and every p_c is a circuit in the language of \mathbf{A} of polynomial size (in n and k), such that $p_c(\mathbf{a}_1, \dots, \mathbf{a}_n) = c$. Enumerated compact representations were already (implicitly) used in several proofs. In [9, Theorem 4.13.] it was shown that, for algebras with few subpowers, enumerated compact representations always exist; this was used to prove that $\text{SMP}(\mathbf{A}) \in \text{NP}$. Moreover, all of the known polynomial time algorithms for $\text{CompRep}(\mathbf{A})$, in fact, compute enumerated compact representations. We are in particular going to need the following result that follows from [25]:

► **Theorem 2** ([25]). *Let \mathbf{A} be a finite supernilpotent Mal'tsev algebra. Then, there is a polynomial time algorithm that computes an enumerated compact representations of $\text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$, for given $\mathbf{a}_1, \dots, \mathbf{a}_n \in A^k$.*

Theorem 2 can be seen as a generalization of Gaussian elimination from affine to supernilpotent algebras. We remark that Theorem 2, although not explicitly stated as such in [25], follows directly from Algorithm 6 in [25], which computes so-called *group representations* (T_1, T_2, \dots, T_k) of $\text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$ and the fact that for such a group representation, there is a constant q such that $T = (T_1 + q \cdot T_2 + \dots + q \cdot T_k)$ has the same signature as $\text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$ (see Lemma 3.1. in [25]). Thus, T together with its defining circuits forms an enumerated compact representation of $\text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$.

We are furthermore going to use that there is an algorithm that allows us to fix some values of a relation given by enumerated compact representation:

► **Lemma 3.** *Let \mathbf{A} be a Mal'tsev algebra. Then, there is a polynomial-time algorithm $\text{Fix-values}(R, a_1, \dots, a_m)$ that, for a given compact representation R of $\mathbf{R} = \text{Sg}_{\mathbf{A}^k}(X)$, and constants $a_1, \dots, a_m \in A$, returns a compact representation R' of $\{\mathbf{x} \in \mathbf{R} \mid \mathbf{x}(1) = a_1, \dots, \mathbf{x}(m) = a_m\}$ (or \emptyset if the relation is empty). If R is moreover enumerated then Fix-values also computes polynomial size circuits defining the elements of R' from X .*

The existence of such a Fix-values algorithm for compact representation is a well-known result ([7], see also [5, Algorithm 5]); the additional statement about *enumerated* compact representation follows easily from bookkeeping the defining circuits. We prove Lemma 3 in Appendix A.

3 Wreath products and difference clonoids

In this section, we discuss how to represent Mal'tsev algebras with non-trivial center by a so-called *wreath product* $\mathbf{L} \otimes \mathbf{U}$, and associate to it its *difference clonoid*, which gives us a measure on how far it is from being the direct product $\mathbf{L} \times \mathbf{U}$.

► **Definition 4.** *Let $\mathbf{U} = (U, (f^{\mathbf{U}})_{f \in F})$ and $\mathbf{L} = (L, (f^{\mathbf{L}})_{f \in F})$ be two algebras in the same language F , such that \mathbf{L} is affine. Furthermore, let $0 \in L$ and $T = (\hat{f})_{f \in F}$ be a family of operations $\hat{f}: U^n \rightarrow L$, for each $f \in F$ of arity n . Then we define the wreath product $\mathbf{L} \otimes^{T,0} \mathbf{U}$ as the algebra $(L \times U, (f^{\mathbf{L} \otimes^{T,0} \mathbf{U}})_{f \in F})$ with basic operations*

$$f^{\mathbf{L} \otimes^{T,0} \mathbf{U}}((l_1, u_1), \dots, (l_n, u_n)) = (f^{\mathbf{L}}(l_1, \dots, l_n) + \hat{f}(u_1, \dots, u_n), f^{\mathbf{U}}(u_1, \dots, u_n)),$$

(where $+$ is the addition on \mathbf{L} with respect to neutral element 0). For simplicity, we are going to write $\mathbf{L} \otimes \mathbf{U}$, if T and 0 are clear from the context.

The name *wreath product* refers to the fact that this is a special case of VanderWerf's wreath products [31]. We remark that recently also alternative names for $\mathbf{L} \otimes \mathbf{U}$ were suggested, such as *central extension* (by Mayr) and *semidirect product* (by Zhuk). By a result of Freese and McKenzie we can represent Mal'tsev algebras with non-trivial centers as wreath products:

► **Theorem 5** (Proposition 7.1. in [14]). *Let \mathbf{A} be a Mal'tsev algebra with a central congruence α , and let $\mathbf{U} = \mathbf{A}/\alpha$. Then there is an affine algebra \mathbf{L} , an element $0 \in L$ and a set of operations T , such that $\mathbf{A} \cong \mathbf{L} \otimes^{T,0} \mathbf{U}$.*

Note that, for a fixed quotient $\mathbf{U} = \mathbf{A}/\alpha$, there is still some freedom in how to choose the operations $f^{\mathbf{L}}$ of \mathbf{L} , and the operations $\hat{f}: U^n \rightarrow L$ in T (by adding/subtracting constants). To get rid of this problem, we are from now on always going to assume that \mathbf{L} preserves 0 , i.e. $f^{\mathbf{L}}(0, 0, \dots, 0) = 0$ for all $f \in F$. It is then easy to observe that wreath products $\mathbf{L} \otimes^{T,0} \mathbf{U}$ behaves nicely with respect to the direct product $\mathbf{L} \times \mathbf{U}$ in the same language:

► **Observation 6.** *Let \mathbf{A} be a Mal'tsev algebra with wreath product representation $\mathbf{A} = \mathbf{L} \otimes^{T,0} \mathbf{U}$. Then $t^{\mathbf{A}} = s^{\mathbf{A}} \Rightarrow t^{\mathbf{L} \times \mathbf{U}} = s^{\mathbf{L} \times \mathbf{U}}$.*

Proof. Note that, for every term t in the language of \mathbf{A} :

$$t^{\mathbf{A}}((l_1, u_1), \dots, (l_n, u_n)) = (t^{\mathbf{L}}(l_1, \dots, l_n) + \hat{t}(u_1, \dots, u_n), t^{\mathbf{U}}(u_1, \dots, u_n)),$$

for some $\hat{t}: U^n \rightarrow L$ (this can be shown by induction over the height of the term tree). Clearly $t^{\mathbf{A}} = s^{\mathbf{A}}$ implies $t^{\mathbf{U}} = s^{\mathbf{U}}$, and $t^{\mathbf{L}} - s^{\mathbf{L}} = c$, $\hat{t} - \hat{s} = -c$ for some constant $c \in L$. Since, by our assumptions, the operations of \mathbf{L} preserve 0 , we get $t^{\mathbf{L}} = s^{\mathbf{L}}$ and $\hat{t} = \hat{s}$. Thus $t^{\mathbf{L} \times \mathbf{U}} = s^{\mathbf{L} \times \mathbf{U}}$. ◀

In other terminology, the map $t^{\mathbf{A}} \mapsto t^{\mathbf{L} \times \mathbf{U}}$ is a surjective *clone homomorphism* from $\text{Clo}(\mathbf{A})$ to $\text{Clo}(\mathbf{L} \times \mathbf{U})$, i.e. a map that preserves arities, projections and compositions. The converse of Observation 6 does however not hold, since this map is usually not injective. We define the *difference clonoid* $\text{Diff}_0(\mathbf{A})$ as the kernel of the clone homomorphisms in the following sense:

► **Definition 7.** *Let $\mathbf{A} = \mathbf{L} \otimes^{T,0} \mathbf{U}$ be a Mal'tsev algebra given as a wreath product.*

(1) *We define the equivalence relation \sim on $\text{Clo}(\mathbf{A})$ by*

$$t^{\mathbf{A}} \sim s^{\mathbf{A}} :\Leftrightarrow t^{\mathbf{L} \times \mathbf{U}} = s^{\mathbf{L} \times \mathbf{U}}$$

(2) *the difference clonoid $\text{Diff}_0(\mathbf{A})$ is defined as the set of all operation $\hat{r}: U^n \rightarrow L$, such that there are $t^{\mathbf{A}} \sim s^{\mathbf{A}} \in \text{Clo}(\mathbf{A})$ with:*

$$t^{\mathbf{A}}((l_1, u_1), \dots, (l_n, u_n)) = (t^{\mathbf{L}}(\mathbf{l}) + \hat{t}(\mathbf{u}), t^{\mathbf{U}}(\mathbf{u})) \quad (1)$$

$$s^{\mathbf{A}}((l_1, u_1), \dots, (l_n, u_n)) = (t^{\mathbf{L}}(\mathbf{l}) + \hat{t}(\mathbf{u}) + \hat{r}(\mathbf{u}), t^{\mathbf{U}}(\mathbf{u})) \quad (2)$$

► **Notation 8.** In the following, we will stick to the following convention: Function symbols with a hat will always denote operations from some power of U to L . For operations $t, s: A^n \rightarrow A$, and $\hat{r}: U^n \rightarrow L$ such as in (1) and (2) we are slightly going to abuse notation, and write $s = t + \hat{r}$ and $\hat{r} = (s - t)$.

We next show that $\text{Diff}_0(\mathbf{A})$ is indeed a clonoid from \mathbf{U} to \mathbf{L} (extended by the constant 0).

► **Lemma 9.** *Let $\mathbf{A} = \mathbf{L} \otimes^{0,T} \mathbf{U}$ be a Mal'tsev algebra given as wreath product. Then:*

- (1) *For all $t \in \text{Clo}(\mathbf{A})$, $\hat{r} \in \text{Diff}_0(\mathbf{A})$ also $t + \hat{r} \in \text{Clo}(\mathbf{A})$,*
- (2) *$\text{Diff}_0(\mathbf{A})$ is a $(\mathbf{U}, (\mathbf{L}, 0))$ -clonoid.*

Here $(\mathbf{L}, 0)$ denotes the extension of \mathbf{L} by the basic operation 0.

Proof. To prove (1), let $t \in \text{Clo}(\mathbf{A})$ and $\hat{r} \in \text{Diff}_0(\mathbf{A})$. By definition of the difference clonoid, $\hat{r} = s_1 - s_2$ for two terms $s_1, s_2 \in \text{Clo}(\mathbf{A})$, with $s_1 \sim s_2$. In particular, $s_1^{\mathbf{U}} = s_2^{\mathbf{U}}$. For any Mal'tsev term $m \in \text{Clo}(\mathbf{A})$, necessarily $\hat{m}(u, u, v) = \hat{m}(v, u, u) = 0$ holds. This implies that

$$t + \hat{r} = m(t, s_2, s_1) \in \text{Clo}(\mathbf{A}).$$

We next prove (2). So we only need to verify that $\text{Diff}_0(\mathbf{A})$ is closed under composition with $\text{Clo}(\mathbf{U})$ (from the inside), respectively $\text{Clo}((\mathbf{L}, 0))$ (from the outside).

To see that $\text{Diff}_0(\mathbf{A})$ is closed under $(\mathbf{L}, 0)$, note that $0 \in \text{Diff}_0(\mathbf{A})$, as $t - t = 0$, for every term $t \in \text{Clo}(\mathbf{A})$. Further $\text{Diff}_0(\mathbf{A})$ is closed under $+$; for this, let $\hat{r}_1, \hat{r}_2 \in \text{Diff}_0(\mathbf{A})$. By (1), we know that $t + \hat{r}_1 \in \text{Clo}(\mathbf{A})$, for some term $t \in \text{Clo}(\mathbf{A})$. Again, by (1) also $(t + \hat{r}_1) + \hat{r}_2 \in \text{Clo}(\mathbf{A})$, which shows that $\hat{r}_1 + \hat{r}_2 \in \text{Diff}_0(\mathbf{A})$. For all unary $e^{\mathbf{L}} \in \text{Clo}(\mathbf{L})$, and $t \sim s$ with $\hat{r} = t - s$, note that $e^{\mathbf{A}}t - e^{\mathbf{A}}s = e^{\mathbf{L}} \circ \hat{r} \in \text{Diff}_0(\mathbf{A})$. Since \mathbf{L} is affine, $\text{Clo}(\mathbf{L}, 0)$ is generated by $+$ and its unary terms, thus $\text{Diff}_0(\mathbf{A})$ is closed under $(\mathbf{L}, 0)$.

To see that $\text{Diff}_0(\mathbf{A})$ is closed under \mathbf{U} from the inside, simply notice that $t(x_1, \dots, x_n) \sim s(x_1, \dots, x_n)$ implies $t(f_1(\mathbf{x}), \dots, f_n(\mathbf{x})) \sim s(f_1(\mathbf{x}), \dots, f_n(\mathbf{x}))$, for all terms f_1, \dots, f_n . If $\hat{r} = t^{\mathbf{A}} - s^{\mathbf{A}}$, then $\hat{r} \circ (f_1^{\mathbf{U}}, \dots, f_n^{\mathbf{U}}) = t \circ (f_1^{\mathbf{U}}, \dots, f_n^{\mathbf{U}}) - s \circ (f_1^{\mathbf{U}}, \dots, f_n^{\mathbf{U}}) \in \text{Diff}_0(\mathbf{A})$. ◀

We remark that the choice of the constant $0 \in L$ is not relevant in this construction: since for every $c \in L$ the map $\hat{r} \mapsto \hat{r} + c$ is an isomorphism between the $(\mathbf{U}, (\mathbf{L}, 0))$ -clonoid $\text{Diff}_0(\mathbf{A})$ and the $(\mathbf{U}, (\mathbf{L}', c))$ -clonoid $\text{Diff}_c(\mathbf{A})$ (where $f^{\mathbf{L}'}(\mathbf{1}) = f^{\mathbf{L}}(\mathbf{1} - (c, c, \dots, c)) + c$).

Our goal in the next section is to reduce the subpower membership problem to a version of the subpower membership problem for the difference clonoid in which we ask for membership of a tuple $\mathbf{l} \in L^k$ in the subalgebra of \mathbf{L} given by the image of $\mathbf{u}_1, \dots, \mathbf{u}_n \in U^k$ under the clonoid. In fact, it will be more convenient for us to ask for a compact representation, that's why we define the following problem, for a clonoid \mathcal{C} from \mathbf{U} to \mathbf{L} .

CompRep(\mathcal{C}):

INPUT: A list of tuples $\mathbf{u}_1, \dots, \mathbf{u}_n \in U^k$.

OUTPUT: A compact representation of $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n) = \{f(\mathbf{u}_1, \dots, \mathbf{u}_n) \mid f \in \mathcal{C}\} \leq \mathbf{L}^k$

In the case of the difference clonoid $\mathcal{C} = \text{Diff}_0(\mathbf{A})$ the image algebra \mathbf{L} is affine and contains a constant 0. Thus then this problem is then equivalent to finding generating set of $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$ as a subgroup of $(L, +, 0, -)^k$ of polynomial size. By then running Gaussian elimination (generalized to finite Abelian groups), or simply applying Theorem 2 one can then compute a compact representation of $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$.

4 The subpower membership problem of wreath products

In this section we discuss our main methodological results. We show that, in some cases the subpower membership problem $\text{SMP}(\mathbf{L} \otimes \mathbf{U})$ of a wreath product can be reduced to $\text{CompRep}(\mathbf{L} \times \mathbf{U})$ and $\text{CompRep}(\mathcal{C})$. We first show how such a reduction can be achieved relatively easily in the case where $\text{Clo}(\mathbf{L} \otimes \mathbf{U})$ contains $\text{Clo}(\mathbf{L} \times \mathbf{U})$ (i.e. the identity map is a retraction of the clone homomorphism from Observation 6):

► **Theorem 10.** *Let $\mathbf{A} = \mathbf{L} \otimes^{T,0} \mathbf{U}$ be a finite Mal'tsev algebra, and let $\mathcal{C} = \text{Diff}_0(\mathbf{A})$. Further assume that $\text{Clo}(\mathbf{L} \times \mathbf{U}) \subseteq \text{Clo}(\mathbf{A})$. Then $\text{CompRep}(\mathbf{A})$ (and hence also $\text{SMP}(\mathbf{A})$) reduces in polynomial time to $\text{CompRep}(\mathbf{L} \times \mathbf{U})$ and $\text{CompRep}(\mathcal{C})$.*

Proof. Let $\mathbf{a}_1, \dots, \mathbf{a}_n \in A^k$ an instance of $\text{CompRep}(\mathbf{A})$; our goal is to find a compact representation of $\mathbf{B} = \text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$. Let us write \mathbf{l}_i and \mathbf{u}_i for the projection of \mathbf{a}_i to L^k and U^k respectively. Let us further define $\mathbf{B}^+ = \text{Sg}_{(\mathbf{L} \times \mathbf{U})^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$. Then

$$\begin{aligned} \mathbf{B} &= \{(t^{\mathbf{L}}(\mathbf{l}_1, \dots, \mathbf{l}_n) + \hat{t}(\mathbf{u}_1, \dots, \mathbf{u}_n), t^{\mathbf{U}}(\mathbf{u}_1, \dots, \mathbf{u}_n) \mid t \text{ is } F\text{-term}\}, \text{ and} \\ \mathbf{B}^+ &= \{(t^{\mathbf{L}}(\mathbf{l}_1, \dots, \mathbf{l}_n), t^{\mathbf{U}}(\mathbf{u}_1, \dots, \mathbf{u}_n) \mid t \text{ is } F\text{-term}\}. \end{aligned}$$

Since $\text{Clo}(\mathbf{L} \times \mathbf{U}) \subseteq \text{Clo}(\mathbf{A})$, we can pick a Mal'tsev term of \mathbf{A} that is of the form $m^{\mathbf{A}}((l_1, u_1), (l_2, u_2), (l_3, u_3)) = (l_1 - l_2 + l_3, m^{\mathbf{U}}(u_1, u_2, u_3))$. Moreover, by Lemma 9, every term $t^{\mathbf{A}} \in \text{Clo}(\mathbf{A})$ can be uniquely written as the sum of $t^{\mathbf{L} \times \mathbf{U}}$ (which by assumption is also in $\text{Clo}(\mathbf{A})$) and some $\hat{t} \in \mathcal{C}$. Thus, every element of \mathbf{B} is equal to the sum of an element of \mathbf{B}^+ and an expression $\hat{t}(\mathbf{u}_1, \dots, \mathbf{u}_n)$.

Let C^+ be a compact representation of \mathbf{B}^+ , and \hat{C} a compact representation of $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$. Then, it follows that every tuple in \mathbf{B} can be written as

$$m(\dots, m(\mathbf{c}_1, \mathbf{d}_2, \mathbf{c}_2), \dots, \mathbf{d}_n, \mathbf{c}_n) + \hat{\mathbf{r}}_1 - \hat{\mathbf{s}}_2 + \hat{\mathbf{r}}_2 - \dots - \hat{\mathbf{s}}_n + \hat{\mathbf{r}}_n, \quad (3)$$

for $\mathbf{c}_i, \mathbf{d}_i \in C^+$ and $\hat{\mathbf{r}}_i, \hat{\mathbf{s}}_i \in \hat{C}$. (We are aware that tuples in C^+ and \hat{C} have different domains; here we follow the same convention as in Notation 8). Moreover, in formula (3), any pair $\mathbf{c}_i, \mathbf{d}_i$ (respectively $\hat{\mathbf{r}}_i, \hat{\mathbf{s}}_i$) witnesses a fork in the i -th coordinate. By our choice of m it is easy to see that formula (3) can be rewritten to

$$m(\dots, m(\mathbf{c}_1 + \hat{\mathbf{r}}_1, \mathbf{d}_2 + \hat{\mathbf{s}}_2, \mathbf{c}_2 + \hat{\mathbf{r}}_2), \dots, \mathbf{d}_n + \hat{\mathbf{s}}_n, \mathbf{c}_n + \hat{\mathbf{r}}_n),$$

Thus the elements $\mathbf{c}_i + \hat{\mathbf{r}}_i, \mathbf{d}_i + \hat{\mathbf{s}}_i$ witness forks of \mathbf{B} in the i -th coordinate. If we define $D = \{\mathbf{c} + \hat{\mathbf{r}} \mid \mathbf{c} \in C, \hat{\mathbf{r}} \in \hat{C}\}$, then it follows that $\text{Sig}(D) = \text{Sig}(\mathbf{B})$. Moreover $D \subset \mathbf{B}$, and it is of polynomial size in n and k , as $|D| \leq |C| \cdot |\hat{C}|$. Thus D can be thinned out in polynomial time to a compact representation of \mathbf{B} , which finishes our proof. ◀

We remark that, by following the proof of Theorem 10, also finding *enumerated* compact representations in \mathbf{A} can be reduced to finding *enumerated* compact representations in $\mathbf{L} \times \mathbf{U}$ and \mathcal{C} (if \mathcal{C} is given by some finite set of operations that generate it as a clonoid).

Unfortunately, the conditions of Theorem 10 are not met for general wreath-products, not even if both \mathbf{U} and \mathbf{L} are affine (the dihedral group D_4 can be shown to be a counterexample). But, if \mathbf{U} is supernilpotent, then we are able to prove the following reduction, independent of the conditions of Theorem 10:

► **Theorem 11.** *Let $\mathbf{A} = \mathbf{L} \otimes \mathbf{U}$ be a finite Mal'tsev algebra, and let $\mathcal{C} = \text{Diff}_0(\mathbf{A})$ for some $0 \in A$. Further, assume that \mathbf{U} is supernilpotent. Then $\text{SMP}(\mathbf{A})$ reduces in polynomial time to $\text{CompRep}(\mathcal{C})$.*

Proof. Let $\mathbf{a}_1, \dots, \mathbf{a}_n, \mathbf{b} \in A^k$ an instance of $\text{SMP}(\mathbf{A})$; our goal is to check whether $\mathbf{b} \in \mathbf{B} = \text{Sg}_{\mathbf{A}^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$. Let us write \mathbf{l}_i and \mathbf{u}_i for the projection of \mathbf{a}_i to L^k and U^k respectively, and \mathbf{l}_b and \mathbf{u}_b for the projections of \mathbf{b} to L^k and U^k . Let F be the signature of \mathbf{A} and $\mathbf{L} \times \mathbf{U}$, and let $\mathbf{B}^+ = \text{Sg}_{(\mathbf{L} \times \mathbf{U})^k}(\mathbf{a}_1, \dots, \mathbf{a}_n)$. Then

$$\begin{aligned} \mathbf{B} &= \{(t^{\mathbf{L}}(\mathbf{l}_1, \dots, \mathbf{l}_n) + \hat{t}(\mathbf{u}_1, \dots, \mathbf{u}_n), t^{\mathbf{U}}(\mathbf{u}_1, \dots, \mathbf{u}_n) \mid t \text{ is } F\text{-term}\}, \text{ and} \\ \mathbf{B}^+ &= \{(t^{\mathbf{L}}(\mathbf{l}_1, \dots, \mathbf{l}_n), t^{\mathbf{U}}(\mathbf{u}_1, \dots, \mathbf{u}_n) \mid t \text{ is } F\text{-term}\}. \end{aligned}$$

Recall the definition of $t^{\mathbf{A}} \sim s^{\mathbf{A}}$ from Definition 7. If T is a \sim -transversal set of $\{t^{\mathbf{A}} \in \text{Clo}(\mathbf{A}) \mid t^{\mathbf{U}}(\mathbf{u}_1, \dots, \mathbf{u}_n) = \mathbf{u}_b\}$, then clearly $\mathbf{b} \in B$ iff $\exists t \in T$ and $\mathbf{d} \in \mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$, with $\mathbf{b} = t(\mathbf{a}_1, \dots, \mathbf{a}_n) + \mathbf{d}$. So, intuitively speaking, the goal of this proof is to first compute such a transversal set, by computing an enumerated compact representation of $\{(\mathbf{l}, \mathbf{u}) \in \mathbf{B}^+ \mid \mathbf{u} = \mathbf{u}_b\}$ and then use it together with a compact representation of $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$ to check membership of \mathbf{b} in \mathbf{B} .

In practice we need however to consider a relation of higher arity than \mathbf{B}^+ , since term operations of $\mathbf{L} \times \mathbf{U}$ are not uniquely determined by their value on $\mathbf{a}_1, \dots, \mathbf{a}_n$. So let S be the degree of supernilpotence of \mathbf{U} (and hence also $\mathbf{L} \times \mathbf{U}$). If we think about $\mathbf{a}_1, \dots, \mathbf{a}_n$ as the columns of a matrix of dimension $k \times n$, then let $\tilde{\mathbf{a}}_1, \dots, \tilde{\mathbf{a}}_n \in A^l$ be its extension by rows that enumerate $H = \{(a_1, \dots, a_n) \in A^n \mid |\{i: a_i \neq 0\}| \leq S\}$ (hence $l \leq k + |A|^S \binom{n}{S}$).

It follows from Theorem 2 that we can compute an enumerated compact representation \tilde{C} of $\text{Sg}_{(\mathbf{L} \times \mathbf{U})^l}(\tilde{\mathbf{a}}_1, \dots, \tilde{\mathbf{a}}_n)$ in polynomial time in n and l . So, every element in $\tilde{B} = \text{Sg}_{(\mathbf{L} \times \mathbf{U})^l}(\tilde{\mathbf{a}}_1, \dots, \tilde{\mathbf{a}}_n)$ can be written as $m(\dots m(\tilde{\mathbf{c}}_1, \tilde{\mathbf{d}}_2, \tilde{\mathbf{c}}_2) \dots \tilde{\mathbf{d}}_l, \tilde{\mathbf{c}}_l)$, for $(\tilde{\mathbf{c}}_i, p_{\tilde{\mathbf{c}}_i}), (\tilde{\mathbf{d}}_i, p_{\tilde{\mathbf{d}}_i}) \in \tilde{C}$, where \tilde{C} is of size at most $2l|A|^2$, and every element of $\tilde{\mathbf{c}} \in \tilde{C}$ is equal to $p_{\tilde{\mathbf{c}}}(\tilde{\mathbf{a}}_1, \dots, \tilde{\mathbf{a}}_n) = \tilde{\mathbf{c}}$ for the given circuit $p_{\tilde{\mathbf{c}}}$ of polynomial size.

By Theorem 1, in an S -supernilpotent algebra, every term operation is already completely determined by its values on the subset H . It follows, that every n -ary term operation of $\mathbf{L} \times \mathbf{U}$ can be uniquely described by a circuit $m(\dots m(p_{\tilde{\mathbf{c}}_1}, p_{\tilde{\mathbf{d}}_2}, p_{\tilde{\mathbf{c}}_2}), \dots p_{\tilde{\mathbf{d}}_l}, p_{\tilde{\mathbf{c}}_l})$ for $\tilde{\mathbf{c}}_i, \tilde{\mathbf{d}}_i \in \tilde{C}$. By definition of \sim , it follows that also every n -ary term operation of \mathbf{A} is \sim -equivalent to the operation given by the circuit described by a circuit $m(\dots m(p_{\tilde{\mathbf{c}}_1}, p_{\tilde{\mathbf{d}}_2}, p_{\tilde{\mathbf{c}}_2}), \dots p_{\tilde{\mathbf{d}}_l}, p_{\tilde{\mathbf{c}}_l})$ for $\tilde{\mathbf{c}}_i, \tilde{\mathbf{d}}_i \in \tilde{C}$.

We are however only interested in terms t such that $t^{\mathbf{U}}$ maps $\mathbf{u}_1, \dots, \mathbf{u}_n$ to the value \mathbf{u}_b . By Lemma 3, we can also compute an enumerated compact representation \tilde{C}' of $\{(\tilde{\mathbf{l}}, \tilde{\mathbf{u}}) \in \text{Sg}_{\mathbf{L} \times \mathbf{U}}(\tilde{\mathbf{a}}_1, \dots, \tilde{\mathbf{a}}_n) \mid \tilde{\mathbf{u}}(i) = \mathbf{u}_b(i) \text{ for all } i = 1, \dots, k\}$ in polynomial time. Note that this is possible, as $\{(\tilde{\mathbf{l}}, \tilde{\mathbf{u}}) \in \text{Sg}_{\mathbf{L} \times \mathbf{U}}(\tilde{\mathbf{a}}_1, \dots, \tilde{\mathbf{a}}_n) \mid \tilde{\mathbf{u}}(i) = \mathbf{u}_b(i) \text{ for all } i = 1, \dots, k\}$ is closed under the Mal'tsev operation $m^{\mathbf{L} \times \mathbf{U}}$. Also, although we only prove Lemma 3 for fixing variables to constants, we remark that it can straightforwardly be generalized to fixing the value of the variables to domains $L \times \{u\}$ (alternatively, this can also be achieved by regarding $\text{Sg}_{(\mathbf{L} \times \mathbf{U})^l}(\tilde{\mathbf{a}}_1, \dots, \tilde{\mathbf{a}}_n)$ as a subalgebra of $\mathbf{U}^l \times \mathbf{L}^l$, which however would require us to work with relations on different domains).

If $\tilde{C}' = \emptyset$, then we output “False”, as then $\mathbf{u}_b \notin \text{Sg}_{\mathbf{U}^k}(\mathbf{u}_1, \dots, \mathbf{u}_n)$. Otherwise, let $C = \{p_{\tilde{\mathbf{c}}}^{\mathbf{A}}(\mathbf{a}_1, \dots, \mathbf{a}_n) \mid \tilde{\mathbf{c}} \in \tilde{C}'\}$. Also, let \hat{C} be a compact representation of $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$. By our proof, every element of $\{(\mathbf{l}, \mathbf{u}) \in \mathbf{B} \mid \mathbf{u} = \mathbf{u}_b\}$ is equal to the sum of an element $m^{\mathbf{A}}(\dots, m^{\mathbf{A}}(\mathbf{c}_1, \mathbf{d}_2, \mathbf{c}_2), \dots \mathbf{d}_n, \mathbf{c}_n)$ with $\mathbf{c}_i, \mathbf{d}_i \in C$ and an element of $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$. Since m is an affine Mal'tsev operation when restricted to $\{(\mathbf{l}, \mathbf{u}) \in \mathbf{B} \mid \mathbf{u} = \mathbf{u}_b\}$ this means that $\mathbf{b} \in \mathbf{B}$ iff \mathbf{l}_b is in the affine closure of all elements $\mathbf{c} + \hat{\mathbf{r}}$ with $\mathbf{c} \in C$ and $\hat{\mathbf{r}} \in \hat{C}$. But this can be checked in polynomial time (by generalized Gaussian elimination, or Theorem 2), which finishes the proof. \blacktriangleleft

5 Clonoids between affine algebras

We continue our paper with an analysis of clonoids between affine algebras to prove our main result, Theorem 20.

For a prime p , let us write \mathbb{Z}_p for the cyclic group of order p , i.e. $\mathbb{Z}_p = (\{0, 1, \dots, p-1\}, +, 0, -)$. Let us further define the idempotent reduct $\mathbb{Z}_p^{id} = (\{0, 1, \dots, p-1\}, x - y + z)$. Using the unary terms $ax = x + \dots + x$ (a -times), for $a \in \mathbb{Z}_p$, we can regard \mathbb{Z}_p as a vector space over the p -element field. More general, using this notation, we will also consider finite Abelian groups $(L, +, 0, -)$ as modules over $\mathbb{Z}_{|L|}$.

For short, we are going to denote constant 1-tuples by $\mathbf{1} = (1, 1, \dots, 1) \in \mathbb{Z}_p^n$. For two vectors $\mathbf{a}, \mathbf{x} \in \mathbb{Z}_p^n$, we further denote by $\mathbf{a} \cdot \mathbf{x} = \sum_{i=1}^n \mathbf{a}(i) \cdot \mathbf{x}(i)$ the standard inner product. Then $\text{Clo}(\mathbb{Z}_p) = \{\mathbf{x} \mapsto \mathbf{a} \cdot \mathbf{x} \mid \mathbf{a} \in \mathbb{Z}_p^n\}$ and $\text{Clo}(\mathbb{Z}_p^{\text{id}}) = \{\mathbf{x} \mapsto \mathbf{a} \cdot \mathbf{x} \mid \mathbf{a} \in \mathbb{Z}_p^n, \mathbf{a} \cdot \mathbf{1} = 1\}$.

In this section, we are going to study clonoids between affine algebras \mathbf{U} and \mathbf{L} , such that $|U| = p$ for some prime p , and $p \nmid |L|$. Since every such affine algebra \mathbf{U} has $x - y + z$ as a term operation, it makes sense to study the special case $\mathbf{U} = \mathbb{Z}_p^{\text{id}}$. As we are in particular interested in difference clonoids, we furthermore can assume that \mathbf{L} contains a constant operation 0 (see Lemma 9), and hence the operations of the Abelian group $(L, +, 0, -)$. We remark that our analysis is structurally similar to (but not covered by) Fioravanti's classification of $(\mathbb{Z}_p, \mathbb{Z}_q)$ -clonoids [13].

5.1 $(\mathbb{Z}_p^{\text{id}}, \mathbf{L})$ -clonoids satisfying $p \nmid |L|$ and $f(x, x, \dots, x) = 0$

Throughout this subsection, let p be a prime, and $\mathbf{L} = (L, +, 0, -)$ an Abelian group with $p \nmid |L|$, and \mathcal{C} be a $(\mathbb{Z}_p^{\text{id}}, \mathbf{L})$ -clonoid satisfying $f(x, x, \dots, x) = 0$ for all $f \in \mathcal{C}$ and $x \in \mathbb{Z}_p$. In other words, for every $n \in \mathbb{N}$, \mathcal{C} maps all tuples from the diagonal $\Delta^n = \{(x, x, \dots, x) \in \mathbb{Z}_p^n\}$ to 0 . We are going to prove that \mathcal{C} is generated by its binary elements, and therefore by any set of generators B of $\mathcal{C}^{(2)} \leq \mathbf{L}^{\mathbb{Z}_p^2}$. Moreover, from B , we are going to construct a canonical generating set of the n -ary functions $\mathcal{C}^{(n)} \leq \mathbf{L}^{\mathbb{Z}_p^n}$. We are, in particular going to use the following set of coefficient vectors for every $n > 2$:

$$C_n = \{\mathbf{a} \in \mathbb{Z}_p^n \mid \exists i > 1: \mathbf{a}(1) = \mathbf{a}(2) = \dots = \mathbf{a}(i-1) = 0, \mathbf{a}(i) = 1\}.$$

► **Observation 12.** Every 2-dimensional subspace $V \leq \mathbb{Z}_p^n$ containing the diagonal Δ^n has a unique parameterization by the map

$$e_{\mathbf{c}}(x, y) = x(\mathbf{1} - \mathbf{c}) + y\mathbf{c} = (x, \mathbf{c}(2)x + (1 - \mathbf{c}(2))y, \dots, \mathbf{c}(n)x + (1 - \mathbf{c}(n))y),$$

for some $\mathbf{c} \in C_n$, i.e. it is equal to the range of a unique such map.

Proof. To see this, note that V contains $\mathbf{1}$, and can be therefore parameterized by $e_{\mathbf{d}}(x, y)$, for some $\mathbf{d} \notin \Delta^n$. So there is an index i with $\mathbf{d}(1) = \dots = \mathbf{d}(i-1) \neq \mathbf{d}(i)$. If $\mathbf{d} \notin C_n$, then we define $\mathbf{c} = (\mathbf{d}(i) - \mathbf{d}(1))^{-1}(\mathbf{d} - \mathbf{d}(1)\mathbf{1})$; clearly $\mathbf{c} \in C_n$, and \mathbf{c} and $\mathbf{1}$ still generate V . It is further not hard to see that different elements of C_n generate different planes together with $\mathbf{1}$, thus we obtain a unique parameterization of V by $e_{\mathbf{c}}(x, y)$. ◀

► **Lemma 13.** Let $f \in \mathcal{C}^{(2)}$. Then, there is a function $f_n \in \mathcal{C}^{(n)}$, such that

$$f_n(x_1, x_2, \dots, x_n) = \begin{cases} f(x_1, x_2) & \text{if } x_2 = x_3 = \dots = x_n \\ 0 & \text{else.} \end{cases}$$

Proof. We prove the lemma by induction on n . For $n = 2$, we simply set $f_2 = f$. For an induction step $n \rightarrow n + 1$, we first define $t_{n+1}(x_1, x_2, \dots, x_n, x_{n+1})$ as the sum

$$\begin{aligned} & \sum_{\mathbf{a} \in \mathbb{Z}_p^{n-1}} f_n(x_1, x_2 + \mathbf{a}(1)(x_{n+1} - x_n), \dots, x_n + \mathbf{a}(n-1)(x_{n+1} - x_n)) \\ & - \sum_{\mathbf{a} \in \mathbb{Z}_p^{n-1}} f_n(x_1, x_1 + \mathbf{a}(1)(x_{n+1} - x_n), \dots, x_1 + \mathbf{a}(n-1)(x_{n+1} - x_n)). \end{aligned}$$

Note that, if $x_{n+1} \neq x_n$, then t_{n+1} evaluates to $\sum_{\mathbf{a} \in \mathbb{Z}_p^{n-1}} f(x_1, \mathbf{a}) - \sum_{\mathbf{a} \in \mathbb{Z}_p^{n-1}} f(x_1, \mathbf{a}) = 0$. On the other hand, if $x_n = x_{n+1}$, then the second sum is equal to 0, while the first one is equal to $p^{n-1}f_n(x_1, x_2, \dots, x_n)$. By the induction hypothesis, the function $f_{n+1} = p^{-(n-1)}t_{n+1}$ satisfies the statement of the lemma (note that $p^{-(n-1)}$ exist modulo $|L|$, since $p \nmid |L|$). ◀

46:12 The Subpower Membership Problem of 2-Nilpotent Algebras

We can prove an analogue statement for all 2-dimensional subspaces of \mathbb{Z}_p^n containing Δ^n :

► **Lemma 14.** *Let $f \in \mathcal{C}^{(2)}$, and $\mathbf{c} \in C_n$. Then there is a function $f^{\mathbf{c}} \in \mathcal{C}^{(n)}$, such that*

$$f^{\mathbf{c}}(x_1, x_2, \dots, x_n) = \begin{cases} f(x, y) & \text{if } (x_1, x_2, \dots, x_n) = e_{\mathbf{c}}(x, y) \\ 0 & \text{else.} \end{cases}$$

Proof. Let $\mathbf{c} \in C^{(n)}$. It is easy to see that there is an invertible matrix $\mathbf{T} \in \mathbb{Z}_p^{n \times n}$, such that $\mathbf{T} \cdot \mathbf{1} = \mathbf{1}$ and $\mathbf{T} \cdot (\mathbf{1} - \mathbf{c}) = \mathbf{e}_1$. Let $T: \mathbb{Z}_p^n \rightarrow \mathbb{Z}_p^n$ be the corresponding linear map $T(\mathbf{x}) = \mathbf{T} \cdot \mathbf{x}$. Let f_n as in Lemma 13 and $f^{\mathbf{c}}(\mathbf{x}) := f_n \circ T$. Note that by the first condition, all rows of \mathbf{T} sum up to 1, hence T can be expressed by terms of \mathbb{Z}_p^{id} . Then $f^{\mathbf{c}}(e_{\mathbf{c}}(x, y)) = f_n(T(x(\mathbf{1} - \mathbf{c}) + y\mathbf{c})) = f_n(x\mathbf{e}_1 + y(\mathbf{1} - \mathbf{e}_1)) = f(x, y)$, and $f^{\mathbf{c}}(\mathbf{x}) = 0$ for $\mathbf{x} \notin e_{\mathbf{c}}(\mathbb{Z}_p^2)$. ◀

We are now ready to prove the main result of this section:

► **Lemma 15.** *Let \mathcal{C} be a $(\mathbb{Z}_p^{id}, \mathbf{L})$ -clonoid satisfying $\forall f \in \mathcal{C}, x \in \mathbb{Z}_p: f(x, \dots, x) = 0$, and let B be a generating set of $\mathcal{C}^{(2)} \leq \mathbf{L}^{\mathbb{Z}_p^2}$. Then*

- (1) \mathcal{C} is the $(\mathbb{Z}_p^{id}, \mathbf{L})$ -clonoid generated by B , and
- (2) $B_n := \{f^{\mathbf{c}} \mid f \in B, \mathbf{c} \in C_n\}$ is a generating set of $\mathcal{C}^{(n)}$ in $\mathbf{L}^{\mathbb{Z}_p^n}$,

Proof. For any $g \in \mathcal{C}^{(n)}$ and $\mathbf{c} \in C^{(n)}$, let us define the binary operation $g_{\mathbf{c}} = f(e_{\mathbf{c}}(x, y)) \in \mathcal{C}^{(2)}$. By Lemma 14, $g_{\mathbf{c}}$ generates a function $g_{\mathbf{c}}^{\mathbf{c}} \in \mathcal{C}^{(n)}$, that agrees with $f(x, y)$ on all tuples of the form $e_{\mathbf{c}}(x, y)$, and that is 0 else. Since every point of $\mathbb{Z}_p^n \setminus \Delta^n$ is in the image of a unique map $e_{\mathbf{c}}$, we get $g = \sum_{\mathbf{c} \in C_n} g_{\mathbf{c}}^{\mathbf{c}}$. Every element of the form $g_{\mathbf{c}}^{\mathbf{c}}$ can be clearly written as a linear combination of elements $f^{\mathbf{c}}$, where $f \in B$. It follows that B_n generates $\mathcal{C}^{(n)}$ in $\mathbf{L}^{\mathbb{Z}_p^n}$, and that the clonoid generated by B is \mathcal{C} . ◀

We remark that if $\mathbf{L} = \mathbb{Z}_q$ for a prime $q \neq p$, and B is a basis of the vector space $\mathcal{C}^{(2)} \leq \mathbf{L}^{\mathbb{Z}_p^2}$, then B_n is a basis of $\mathcal{C}^{(n)}$. The generating set B_n can be used to decide efficiently the following version of the subpower membership problem for \mathcal{C} :

► **Lemma 16.** *Let \mathcal{C} be a $(\mathbb{Z}_p^{id}, \mathbf{L})$ -clonoid satisfying $\forall f \in \mathcal{C}, x \in \mathbb{Z}_p: f(x, \dots, x) = 0$. Then we can solve $\text{CompRep}(\mathcal{C})$ in polynomial time.*

Proof. By Lemma 15, $\mathcal{C}^{(n)}$ is the linear closure of B_n . Thus $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$ is equal to the linear closure of $B_n(\mathbf{u}_1, \dots, \mathbf{u}_n) := \{f^{\mathbf{c}}(\mathbf{u}_1, \dots, \mathbf{u}_n) \mid f \in B, \mathbf{c} \in C_n\}$.

Note that the i -th entry $f^{\mathbf{c}}(\mathbf{u}_1, \dots, \mathbf{u}_n)(i)$ of such a generating element can only be different from 0 if $(\mathbf{u}_1, \dots, \mathbf{u}_n)(i)$ lies in the 2-dimensional subspace generated by the diagonal Δ^n and \mathbf{c} . Thus, there are at most k many vectors $\mathbf{c} \in C_n$ such that $f^{\mathbf{c}}(\mathbf{u}_1, \dots, \mathbf{u}_n) \neq \mathbf{0}$, let $\mathbf{c}_1, \dots, \mathbf{c}_l$ be an enumeration of them. Clearly $D = \{f^{\mathbf{c}}(\mathbf{u}_1, \dots, \mathbf{u}_n) \mid f \in B, \mathbf{c} \in \{\mathbf{c}_1, \dots, \mathbf{c}_l\}\}$ generates $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$; note that we can compute it in linear time $O(kn)$. From the generating set D we can compute a compact representation of $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$ in polynomial time (by generalized Gaussian elimination, or Theorem 2). ◀

5.2 General $(\mathbb{Z}_p^{id}, \mathbf{L})$ -clonoids satisfying $p \nmid |L|$

For an arbitrary $(\mathbb{Z}_p^{id}, \mathbf{L})$ -clonoid \mathcal{C} , let us define the subclonoid $\mathcal{C}_{\Delta} = \{f \in \mathcal{C}: f(x, \dots, x) = 0\}$. We then show, that every $f \in \mathcal{C}$ can be written in a unique way as the sum of an element of \mathcal{C}_{Δ} , and a function that is generated by $\mathcal{C}^{(1)}$. For this, we need the following lemma:

► **Lemma 17.** *For any $f \in \mathcal{C}^{(n)}$, let us define $f'(\mathbf{x}) = f(x_1, x_1, \dots, x_1)$. Then $f - f' \in \mathcal{C}_{\Delta}$, and f' is generated by $\mathcal{C}^{(1)}$.*

Proof. trivial. ◀

It follows in particular from Lemma 17 and Lemma 15 that every $(\mathbb{Z}_p^{id}, \mathbf{L})$ -clonoid \mathcal{C} is generated by any set $A \cup B$, such that A generates $\mathcal{C}^{(1)}$ in $\mathbf{L}^{\mathbb{Z}_p}$ and B generates $\mathcal{C}_\Delta^{(2)}$ in $\mathbf{L}^{\mathbb{Z}_p^2}$. Note that the clonoid generated by A does not need to be disjoint from \mathcal{C}_Δ . We can, however, still prove results analogous to the previous section.

► **Lemma 18.** *Let \mathcal{C} be a $(\mathbb{Z}_p^{id}, \mathbf{L})$ -clonoid, let A be a generating set of $\mathcal{C}^{(1)} \leq \mathbf{L}^{\mathbb{Z}_p}$ and B a generating set of $\mathcal{C}_\Delta^{(2)} \leq \mathbf{L}^{\mathbb{Z}_p^2}$. For every n , let us define $A_n = \{\sum_{\mathbf{a} \in \mathbb{Z}_p^n, \mathbf{a} \cdot \mathbf{1} = 1} f(\mathbf{a} \cdot \mathbf{x}) \mid f \in A\}$ and let B_n be defined as in Lemma 15. Then $A_n \cup B_n$ is a generating set of $\mathcal{C}^{(n)}$ in $\mathbf{L}^{\mathbb{Z}_p^n}$.*

Proof. We already know from Lemma 15 that B_n generates $\mathcal{C}_\Delta^{(n)} \leq \mathbf{L}^{\mathbb{Z}_p^n}$.

By Lemma 17, every element $f \in \mathcal{C}^{(n)}$ can be uniquely written as the sum f' and $f - f'$. Furthermore f' , by definition, is generated by A_n , and $f - f'$ is in $\mathcal{C}_\Delta^{(n)}$, which finishes our proof. ◀

Lemma 18 allows us to straightforwardly generalize Lemma 16 to arbitrary $(\mathbb{Z}_p^{id}, \mathbf{L})$ -clonoids:

► **Lemma 19.** *Let \mathcal{C} be a $(\mathbb{Z}_p^{id}, \mathbf{L})$ -clonoid. Then $\text{CompRep}(\mathcal{C}) \in \mathbf{P}$.*

Proof. Let A_n and B_n be defined as in Lemma 18. Our goal is to compute a compact representation of $\mathcal{C}(\mathbf{u}_1, \dots, \mathbf{u}_n)$ for some given $\mathbf{u}_1, \dots, \mathbf{u}_n \in \mathbb{Z}_p^k$. By Lemma 18, every $g \in \mathcal{C}$ decomposes into the sum of g' and $g - g'$, where g' is generated by A_n and $g - g'$ is generated by B_n . Thus any image $g(\mathbf{u}_1, \dots, \mathbf{u}_n)$ is in the linear closure of all tuples $f(\mathbf{u}_1, \dots, \mathbf{u}_n)$, for $f \in A_n$ and $B_n(\mathbf{u}_1, \dots, \mathbf{u}_n) = \{f(\mathbf{u}_1, \dots, \mathbf{u}_n) \mid f \in B, \in \mathcal{C}_n\}$ in \mathbf{L}^k . There are at most $|A|$ -many tuples of the first form. Furthermore, as in the proof of Lemma 16 we can compute a generating set of $B_n(\mathbf{u}_1, \dots, \mathbf{u}_n)$ in polynomial time. By generalized Gaussian elimination (or Theorem 2), we can obtain a compact representation from these generators in polynomial time. ◀

Lemma 19 allows us to finish the proof of our main result:

► **Theorem 20.** *Let \mathbf{A} be a finite Mal'tsev algebra, with a central series $0_{\mathbf{A}} < \rho < 1_{\mathbf{A}}$ such that $|\mathbf{A}/\rho| = p$ is a prime, and the blocks of ρ are of size coprime to p . Then $\text{SMP}(\mathbf{A}) \in \mathbf{P}$.*

Proof. By Theorem 5, \mathbf{A} is isomorphic to a wreath product $\mathbf{L} \otimes \mathbf{U}$, such that \mathbf{U}, \mathbf{L} are affine with $|\mathbf{U}| = p$ and $|\mathbf{L}|$ coprime to p (as $|\mathbf{L}|$ is the size of every block of ρ). By Theorem 11, $\text{SMP}(\mathbf{A})$ reduces to $\text{CompRep}(\text{Diff}_0(\mathbf{A}))$ in polynomial time. The difference clonoid is a clonoid from \mathbf{U} to $(\mathbf{L}, 0)$. Since both \mathbf{L} and \mathbf{U} are affine, and therefore have term operations describing $x - y + z$, $\text{Diff}_0(\mathbf{A})$ is also a clonoid from \mathbb{Z}_p^{id} to $(L, +, 0, -)$. By Lemma 19, $\text{CompRep}(\text{Diff}_0(\mathbf{A}))$ is solvable in polynomial time, which finishes the proof. ◀

► **Corollary 21.** *For every nilpotent Mal'tsev algebra \mathbf{A} with $|A| = pq$ for distinct primes $p \neq q$, we have $\text{SMP}(\mathbf{A}) \in \mathbf{P}$.*

Proof. If \mathbf{A} is affine, then the result holds by (generalized) Gaussian elimination. So assume that \mathbf{A} is 2-nilpotent, but not affine. So \mathbf{A} is isomorphic to $\mathbf{L} \otimes \mathbf{U}$, and wlog. $|\mathbf{L}| = q$ and $|\mathbf{U}| = p$. Then the result follows directly from Theorem 20. ◀

6 Discussion

In Theorem 20 we proved that every Mal'tsev algebra, which can be written as a wreath product $\mathbf{L} \otimes \mathbf{U}$ with $|U| = p$ and $p \nmid |L|$ has a tractable subpower membership problem. But, since the reduction discussed in Theorem 11 extends beyond this case, it is natural to ask, whether we can also extend the tractability also extends to all those cases:

► **Question 22.** *Is $\text{SMP}(\mathbf{L} \otimes \mathbf{U}) \in \text{P}$ for every supernilpotent Mal'tsev algebra \mathbf{U} ?*

In particular, if \mathbf{U} is affine, Question 22 asks, whether the subpower membership problem of all finite 2-nilpotent Mal'tsev algebras can be solved in polynomial time. By Theorem 11, this reduces to computing compact representations with respect the clonoids between affine algebras. Thus answering the question requires a better understanding of such clonoids.

The recent preprint [27] studies such clonoids in the case where \mathbf{U} has a distributive congruence lattice, and \mathbf{L} is coprime to \mathbf{U} . Such clonoids are always generated by functions of bounded arity (as in Lemma 14), thus we expect a similar argument as in Lemma 19 to work in solving $\text{CompRep}(\mathcal{C})$. We remark that, in the case of the clonoid of *all* operations from \mathbf{U} and \mathbf{L} this was already implicitly shown in [18] to obtain a polynomial time algorithm for checking whether two circuits over a 2-nilpotent algebra are equivalent. However [27] does not cover all clonoids between affine algebras; e.g. for the case $\mathbf{U} = \mathbb{Z}_p \times \mathbb{Z}_p$ and coprime \mathbf{L} nothing is known so far.

A reason why much emphasis is placed on coprime \mathbf{U} and \mathbf{L} is, that their wreath products $\mathbf{L} \otimes^{T,0} \mathbf{U}$ are not supernilpotent (for non-trivial operations T), and therefore not covered by Theorem 2. In fact, finite Mal'tsev algebras in finite language are supernilpotent if and only if they decompose into the direct product of nilpotent algebras of prime power size (see e.g. [2, Lemma 7.6.]). It is further still consistent with our current knowledge that the conditions of Theorem 10 are always met, for coprime \mathbf{L} and \mathbf{U} . This naturally leads to the question:

► **Question 23.** *Is $\text{Clo}(\mathbf{L} \times \mathbf{U}) \subseteq \text{Clo}(\mathbf{L} \otimes \mathbf{U})$, for all finite nilpotent Mal'tsev algebras $\mathbf{L} \otimes \mathbf{U}$ where \mathbf{L} and \mathbf{U} have coprime size?*

In fact, in an unpublished proof [20], a positive answer to Question 23 is given in the case that $\text{Clo}(\mathbf{L} \otimes \mathbf{U})$ contains a constant operation. A more general version of Question 23 would ask, whether every finite nilpotent Mal'tsev algebra \mathbf{A} has a Mal'tsev term m , such that (A, m) is supernilpotent.

Lastly we would like to mention that recently the property of *short pp-definitions* was suggested as a witnesses for $\text{SMP}(\mathbf{A}) \in \text{coNP}$. While Mal'tsev algebras that generate residually finite varieties have short pp-definitions [10], it is not know whether this is true in the nilpotent case. Thus we ask:

► **Question 24.** *Does every finite nilpotent Mal'tsev algebras \mathbf{A} have short pp-definitions (and hence $\text{SMP}(\mathbf{A}) \in \text{NP} \cap \text{coNP}$)?*

Studying Question 24 might especially be a useful approach to discuss the complexity for algebras of high nilpotent degree, if studying the corresponding difference clonoids turns out to be too difficult or technical.

References

- 1 Erhard Aichinger and Peter Mayr. Finitely generated equational classes. *Journal of Pure and Applied Algebra*, 220(8):2816–2827, 2016. doi:10.1016/j.jpaa.2016.01.001.

- 2 Erhard Aichinger and Nebojša Mudrinski. Some applications of higher commutators in Mal'cev algebras. *Algebra universalis*, 63(4):367–403, 2010. doi:10.1007/s00012-010-0084-1.
- 3 Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and how to use them. In *Dagstuhl Follow-Ups*, volume 7. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/DFU.Vol17.15301.1.
- 4 Clifford Bergman. *Universal algebra: Fundamentals and selected topics*. CRC Press, 2011.
- 5 Zarathustra Brady. Notes on CSPs and Polymorphisms. arXiv preprint arXiv:2210.07383v1, 2022.
- 6 Bruno Buchberger. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of symbolic computation*, 41(3-4):475–511, 2006. doi:10.1016/j.jsc.2005.09.007.
- 7 Andrei Bulatov and Víctor Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM Journal on Computing*, 36(1):16–27, 2006. doi:10.1137/050628957.
- 8 Andrei Bulatov, Marcin Kozik, Peter Mayr, and Markus Steindl. The subpower membership problem for semigroups. *International Journal of Algebra and Computation*, 26(07):1435–1451, 2016. doi:10.1142/S0218196716500612.
- 9 Andrei Bulatov, Peter Mayr, and Ágnes Szendrei. The subpower membership problem for finite algebras with cube terms. *Logical Methods in Computer Science*, 15(1), 2019. doi:10.23638/LMCS-15(1:11)2019.
- 10 Jakub Bulín and Michael Kompatscher. Short definitions in constraint languages. In *Proceedings of the 48th International Symposium on Mathematical Foundations of Computer Science, (MFCS 2023)*, volume 272 of *LIPICs*, pages 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.MFCS.2023.28.
- 11 Stanley Burris and Hanamantagouda Sankappanavar. *A course in universal algebra*, volume 78. Springer, 1981.
- 12 Victor Dalmau and Peter Jeavons. Learnability of quantified formulas. *Theoretical Computer Science*, 306(1-3):485–511, 2003. doi:10.1016/S0304-3975(03)00342-6.
- 13 Stefano Fioravanti. Closed sets of finitary functions between finite fields of coprime order. *Algebra universalis*, 81(4):Art. No 52, 2020. doi:10.1007/s00012-020-00683-5.
- 14 Ralph Freese and Ralph McKenzie. *Commutator theory for congruence modular varieties*, volume 125. CUP Archive, 1987.
- 15 Merrick Furst, John Hopcroft, and Eugene Luks. Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 36–41. IEEE, 1980.
- 16 Christian Herrmann. Affine algebras in congruence modular varieties. *Acta Universitatis Szegediensis*, 41:119–11, 1979.
- 17 Paweł Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM Journal on Computing*, 39(7):3023–3037, 2010.
- 18 Piotr Kawałek, Michael Kompatscher, and Jacek Krzaczkowski. Circuit equivalence in 2-nilpotent algebras. arXiv preprint arXiv:1909.12256, accepted for publication in STACS 2024.
- 19 Donald E Knuth. Efficient representation of perm groups. *Combinatorica*, 11(1):33–43, 1991. doi:10.1007/BF01375471.
- 20 Michael Kompatscher, Peter Mayr, and Patrick Wynne. Personal communication. presented at AAA102, Szeged, 2022. URL: <https://www.math.u-szeged.hu/aaa102/Slides/Mayr,%20Peter2.pdf>.
- 21 Dexter Kozen. Complexity of finitely presented algebras. In *Proceedings of the 9th annual ACM Symposium on Theory of Computing (STOC)*, pages 164–177, 1977. doi:10.1145/800105.803406.

- 22 Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 254–266, 1977. doi:10.1109/SFCS.1977.16.
- 23 Marcin Kozik. A finite set of functions with an EXPTIME-complete composition problem. *Theoretical Computer Science*, 407(1-3):330–341, 2008.
- 24 Ernst W Mayr and Albert R Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in mathematics*, 46(3):305–329, 1982. doi:10.1016/0001-8708(82)90048-2.
- 25 Peter Mayr. The subpower membership problem for Mal’cev algebras. *International Journal of Algebra and Computation*, 22(07):1250075, 2012. doi:10.1142/S0218196712500750.
- 26 Peter Mayr. Vaughan–Lee’s nilpotent loop of size 12 is finitely based. *Algebra universalis*, 85(1):1–12, 2024. doi:10.1007/s00012-023-00832-6.
- 27 Peter Mayr and Patrick Wynne. Clonoids between modules. arXiv preprint arXiv:2307.00076, 2023.
- 28 Vladimir Shpilrain and Alexander Ushakov. Thompson’s group and public key cryptography. In *Applied Cryptography and Network Security: Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005. Proceedings 3*, pages 151–163. Springer, 2005. doi:10.1007/11496137_11.
- 29 Vladimir Shpilrain and Gabriel Zapata. Using the subgroup membership search problem in public key cryptography. *Contemporary Mathematics*, 418:169, 2006. doi:10.1090/conm/418/07955.
- 30 Charles C Sims. Computational methods in the study of permutation groups. In *Computational problems in abstract algebra*, pages 169–183. Elsevier, 1970. doi:10.1016/B978-0-08-012975-4.50020-5.
- 31 Joel VanderWerf. Wreath products of algebras: generalizing the Krohn-Rhodes theorem to arbitrary algebras. *Semigroup Forum*, 52(1):93–100, 1996. doi:10.1007/BF02574084.
- 32 Ross Willard. Four unsolved problems in congruence permutable varieties, 2007. Talk at the Conference on Order, Algebra, and Logics, Nashville. URL: https://www.math.uwaterloo.ca/~rdwillar/documents/Slides/willard_nashville_2007_slides.pdf.

A Proof of Lemma 3

In this appendix we prove the second statement of Lemma 3, i.e. we show that for a given *enumerated* compact representation R of a subpower $\mathbf{R} = \text{Sg}_{\mathbf{A}^k}(X)$ of some Mal’cev algebra, we can obtain an *enumerated* compact representation R' of $\mathbf{R} \cap \{\mathbf{x} \in \mathbf{A}^k \mid \mathbf{x}(1) = a_1, \dots, \mathbf{x}(k) = a_k\}$ for a given list of constants a_1, \dots, a_k . In Algorithm 1 we describe the algorithm $\text{Fix-value}(R, a)$ that fixes the first coordinate of \mathbf{R} to a ; iterating this algorithm m times results in the statement of the Lemma.

We remark that $\text{Fix-value}(R, a)$ is based on the Fix-values algorithm in [5, Algorithm 5]); although, for simplicity, we only fix the value of one coordinate. Line 7 and 8 corresponds to the call of the subroutine Nonempty in [5, Algorithm 5]), with the difference that we compute all the elements of the set $T_j = \{(x, y) \in \text{pr}_{1,j} \mathbf{R} \mid x = a\}$, instead of computing a witness for $(a, y) \in T_j$ once at a time.

We claim that the running time of $\text{Fix-Value}(R, a)$ is $O(|A|^2 \cdot n)$. For this note that the exhaustive search in line 7 and 8 will simply recursively apply m to elements from $\text{pr}_{1,j}(R)$ until the set is closed under m , and then select all values with $x_1 = a$. Since $|\text{pr}_{1,j} \mathbf{R}| \leq A^2$ this takes at most $|A|^2$ steps. For this reason also the size of the defining circuits C_j (when e.g. stored all together as a circuit with multiple out-gates) is bounded by $|R| + |A|^2$. Since the for-loop of line 6 has at most n iterations, it follows that both the running time of the algorithm and the size of the defining circuits in R' are bounded by $O(|A|^2 \cdot n)$.

If we then repeatedly call **Fix-value** to fix the value of the first m -many values of \mathbf{R} , this results in an algorithm that runs in time $O(|A|^2 \cdot nm)$.

Thus, the only thing that remains to prove is that the algorithm **Fix-Value** is correct. i.e. it indeed outputs an enumerated R' with $\text{Sig}(R') = \text{Sig}(\mathbf{R} \cap \{\mathbf{x} \in A^k \mid \mathbf{x}(1) = a\})$ (if the output is not empty). So assume that $(i, b, c) \in \text{Sig}(\mathbf{R} \cap \{\mathbf{x} \in A^k \mid \mathbf{x}(1) = a\})$. If $i = 1$, then clearly $(i, b, c) = (1, a, a)$, which is in $\text{Sig}(R')$. So let us assume wlog. that $i > 1$. Since R is a compact representation of \mathbf{R} , there exist tuples $\mathbf{r}_b, \mathbf{r}_c \in R$ (and defining circuits $p_{\mathbf{r}_b}$ and $p_{\mathbf{r}_c}$), witnessing that $(i, b, c) \in \text{Sig}(R)$. Then R' contains the tuples \mathbf{t} and $\mathbf{s} = m(\mathbf{t}, \mathbf{r}_b, \mathbf{r}_c)$, as constructed in line 12 and 13 of Algorithm 1. Since \mathbf{r}_b and \mathbf{r}_c agree on the first $i - 1$ coordinates also \mathbf{t} and \mathbf{s} do. Moreover $\mathbf{t}(1) = a$, $\mathbf{t}(i) = b$, and $\mathbf{s}(i) = m(b, b, c) = c$, thus \mathbf{t} and \mathbf{s} witness $(i, b, c) \in \text{Sig}(\mathbf{R} \cap \{\mathbf{x} \in A^k \mid \mathbf{x}(1) = a\})$. It follows that $\text{Sig}(R') = \text{Sig}(\mathbf{R} \cap \{\mathbf{x} \in A^k \mid \mathbf{x}(1) = a\})$, which is what we wanted to prove.

■ **Algorithm 1** An algorithm that, for a given enumerated compact representations R of $\mathbf{R} = \text{Sg}_{A^k}(X)$ outputs an enumerated compact representation R' of the relation that fixes $x_1 = a$, (where the defining circuits of R' are evaluated on X).

```

1: procedure FIX-VALUE( $a \in A$ ,  $R$  (enum. c.r. of  $\mathbf{R} = \text{Sg}_{A^k}(X)$ ), Mal'tsev term  $m$ )
2:   if  $(1, a, a) \notin \text{Sig}(R)$  then return  $\emptyset$ 
3:   else
4:     Let  $(\mathbf{t}, p_{\mathbf{t}}) \in R$  be such that  $(\mathbf{t}, \mathbf{t})$  is a witness of  $(1, a, a) \in \text{Sig}(R)$ .
5:      $R' = \{(\mathbf{t}, p_{\mathbf{t}})\}$ 
6:     for  $j > 1$  do
7:       Recursively apply  $m$  to  $\text{pr}_{1,j}(R)$  to compute  $T_j = \{(x, y) \in \text{pr}_{1,j}(\mathbf{R}) \mid x = a\}$ ,
8:       and circuits  $C_j = \{p_{(x,y)} \mid (x, y) \in T_j\}$  such that  $\text{pr}_{1,j}(p_{(x,y)}(X)) = (x, y)$ .
9:       for  $(j, b, c) \in \text{Sig}(R)$  do
10:        Let  $(\mathbf{r}_b, p_{\mathbf{r}_b}), (\mathbf{r}_c, p_{\mathbf{r}_c}) \in R$  be witnesses of  $(j, b, c) \in \text{Sig}(R)$ 
11:        if  $(a, b) \in T_j$  then
12:          Let  $\mathbf{t} = p_{(a,b)}(X)$ 
13:           $\mathbf{s} = m(\mathbf{t}, \mathbf{r}_b, \mathbf{r}_c)$  and  $p_{\mathbf{s}} = m(p_{(a,b)}, p_{\mathbf{r}_b}, p_{\mathbf{r}_c})$ 
14:           $R' = R' \cup \{(\mathbf{t}, p_{(a,b)}), (\mathbf{s}, p_{\mathbf{s}})\}$ 
15:   return  $R'$ 

```

Parameterized and Approximation Algorithms for Coverings Points with Segments in the Plane

Katarzyna Kowalska ✉

Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Poland

Michał Pilipczuk ✉

Institute of Informatics, University of Warsaw, Poland

Abstract

We study parameterized and approximation algorithms for a variant of SET COVER, where the universe of elements to be covered consists of points in the plane and the sets with which the points should be covered are segments. We call this problem SEGMENT SET COVER. We also consider a relaxation of the problem called δ -extension, where we need to cover the points by segments that are extended by a tiny fraction, but we compare the solution's quality to the optimum without extension.

For the unparameterized variant, we prove that SEGMENT SET COVER does not admit a PTAS unless $P=NP$, even if we restrict segments to be axis-parallel and allow $\frac{1}{2}$ -extension. On the other hand, we show that parameterization helps for the tractability of SEGMENT SET COVER: we give an FPT algorithm for unweighted SEGMENT SET COVER parameterized by the solution size k , a parameterized approximation scheme for WEIGHTED SEGMENT SET COVER with k being the parameter, and an FPT algorithm for WEIGHTED SEGMENT SET COVER with δ -extension parameterized by k and δ . In the last two results, relaxing the problem is probably necessary: we prove that WEIGHTED SEGMENT SET COVER without any relaxation is $W[1]$ -hard and, assuming ETH, there does not exist an algorithm running in time $f(k) \cdot n^{o(k/\log k)}$. This holds even if one restricts attention to axis-parallel segments.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases Geometric Set Cover, fixed-parameter tractability, weighted parameterized problems, parameterized approximation scheme

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.47

Related Version Full Version: <https://arxiv.org/abs/2402.16466> [9]

Funding *Michał Pilipczuk*: This work is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 948057).



Acknowledgements The authors would like to thank Krzysztof Maziarczyk for help with proofreading the manuscript.

1 Introduction

In the classic SET COVER problem, we are given a set of elements (universe) \mathcal{U} and a family of sets \mathcal{F} that are subsets of \mathcal{U} and sum up to the whole \mathcal{U} . The task is to find a subfamily $\mathcal{S} \subseteq \mathcal{F}$ such that $\bigcup \mathcal{S} = \mathcal{U}$ and the size of \mathcal{S} is minimum possible.

In the most general form, SET COVER is NP-complete, inapproximable within factor $(1 - \delta) \ln |\mathcal{U}|$ for any $\delta > 0$ assuming $P \neq NP$ [5], and $W[2]$ -complete with the natural parameterization by the size of the solution [4, Theorem 13.21]. However, restricting the problem to various specialized settings can lead to more tractable special cases. Particularly well-studied setting is that of GEOMETRIC SET COVER, where \mathcal{U} consists of points in some Euclidean space V (most often the plane \mathbb{R}^2), while \mathcal{F} consists of various geometric objects



© Katarzyna Kowalska and Michał Pilipczuk;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 47; pp. 47:1–47:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



in V . In this paper we take a closer look at the SEGMENT SET COVER problem, where we assume that \mathcal{U} is a finite set of points in the plane and \mathcal{F} consists of segments in the plane (not necessarily axis-parallel). Each of these problems has also a natural weighted variant, where each set $A \in \mathcal{F}$ comes with a nonnegative real weight $\mathbf{w}(A)$ and the task is to find a solution with the minimum possible total weight.

Approximation. Over the years there has been a lot of work related to approximation algorithms for GEOMETRIC SET COVER. Notably, GEOMETRIC SET COVER with unweighted unit disks or weighted unit squares admits a PTAS [6,17]. When we consider the same problem with weighted disks or squares (not necessarily unit), the problem admits a QPTAS [16], see also [19]. On the other hand, Chan and Grant proved that unweighted GEOMETRIC SET COVER with axis-parallel fat rectangles is APX-hard [3]. They also showed similar hardness for GEOMETRIC SET COVER with many other standard geometric objects. See the introductory section of [3] for a wider discussion of approximation algorithms for GEOMETRIC SET COVER with various kinds of geometric objects.

Parameterization. We consider GEOMETRIC SET COVER parameterized by the size of solution: Given an instance $(\mathcal{U}, \mathcal{F})$ and a parameter k , the task is to decide whether there is a solution of cardinality at most k . In the weighted setting, we look for a minimum-weight solution among those of cardinality at most k , and k remains a parameter.

(Unweighted) GEOMETRIC SET COVER where \mathcal{F} consists of lines in the plane is called POINT LINE COVER, and it is a textbook example of a problem that admits a quadratic kernel and a $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ -time fixed-parameter algorithm (cf. [4, Exercise 2.4]). See also the work of Kratsch et al. [10] for a matching lower bound on the kernel size and a discussion of the relevant literature. The simple branching and kernelization ideas behind the parameterized algorithms for POINT LINE COVER were generalized by Langerman and Morin [11] to an abstract variant of GEOMETRIC SET COVER where the sets of \mathcal{F} can be assigned a suitable notion of dimension. This framework in particular applies to the problem of covering points with hyperspaces in \mathbb{R}^d .

As proved by Marx, unweighted GEOMETRIC SET COVER with unit squares in the plane is already W[1]-hard [12, Theorem 5]. Later, Marx and Pilipczuk showed that there is an algorithm running in time $n^{\mathcal{O}(\sqrt{k})}$ that solves weighted GEOMETRIC SET COVER with squares or with disks, and that this running time is tight under the Exponential-Time Hypothesis (ETH) [15]. However, they also showed that any small deviations from the setting of squares or disks – for instance considering thin rectangles or rectangles with sidelengths in the interval $[1, 1 + \delta]$ for any $\delta > 0$ – lead to problems for which there are lower bounds refuting running times of the form $f(k) \cdot n^{o(k)}$ or $f(k) \cdot n^{o(k/\log k)}$, for any computable f . See [15] for a broader exposition of these results and for more literature pointers.

We are not aware of any previous work that concretely considered the SEGMENT SET COVER problem. In particular, it seems that the framework of Langerman and Morin [11] does *not* apply to this problem, since no suitable notion of dimension can be assigned to segments in the plane (more concretely, the fundamental [11, Lemma 1] fails, which renders further arguments not applicable). In [13] Marx considered the related DOMINATING SET problem in intersection graphs of axis-parallel segments, and proved it to be W[1]-hard. The parameterized complexity of the INDEPENDENT SET problem for segments in the plane was studied in the same work of Marx, and independently by Kára and Kratochvíl [8].

δ -extension. We also consider the δ -extension relaxation of the SEGMENT SET COVER problem. Formally, for a center-symmetric object $L \subseteq \mathbb{R}^2$ with center of symmetry $S = (x_s, y_s)$, the δ -extension of L is the set:

$$L^{+\delta} = \{(1 + \epsilon) \cdot (x - x_s, y - y_s) + (x_s, y_s) : (x, y) \in L, 0 \leq \epsilon < \delta\}.$$

That is, $L^{+\delta}$ is the image of L under homothety centred at S with scale $(1 + \delta)$ but with the extreme points excluded. In particular, δ -extension turns a closed segment into a segment without endpoints and a rectangle into the interior of a rectangle; this is a technical detail that will turn out to be useful in presentation.

In GEOMETRIC SET COVER with δ -extension, we assume that in the given instance $(\mathcal{U}, \mathcal{F})$, \mathcal{F} consists of center-symmetric objects, and we are additionally given the accuracy parameter $\delta > 0$. The task is to find $\mathcal{S} \subseteq \mathcal{F}$ such that $\mathcal{S}^{+\delta} := \{L^{+\delta} : L \in \mathcal{S}\}$ covers all points in \mathcal{U} , but the quality of the solution – be it the cardinality or the weight of \mathcal{S} – is compared to the optimum without assuming extension. Thus, requirements on the the output solution are relaxed: the points of \mathcal{U} have to be covered only after expanding every object of \mathcal{S} a tiny bit. The parameterized variants of GEOMETRIC SET COVER with δ -extension are defined naturally: the task is to either find a solution of size at most k that covers all of \mathcal{U} after δ -extension, or conclude that there is no solution of size k that covers \mathcal{U} without extension.

The study of the δ -extension relaxation is motivated by the δ -shrinking relaxation considered in the context of the GEOMETRIC INDEPENDENT SET problem: given a family \mathcal{F} of objects in the plane, find the maximum size subfamily of pairwise disjoint objects. In the δ -shrinking model, the output solution is required to be disjoint only after shrinking every object by a $1 - \delta$ multiplicative factor. GEOMETRIC INDEPENDENT SET remains W[1]-hard for as simple objects as unit disks or unit squares [13] and admits a QPTAS for polygons [2], but the existence of a PTAS for the problem is widely open. However, as first observed by Adamaszek et al. [1], and then confirmed by subsequent works of Wiese [20] and of Pilipczuk et al. [18], adopting the δ -shrinking relaxation leads to a robust set of FPT algorithms and (efficient or parameterized) approximation schemes. The motivation of this work is to explore if the analogous δ -extension relaxation of GEOMETRIC SET COVER also leads to more positive results.

In fact, we are not the first to consider the δ -extension relaxation of GEOMETRIC SET COVER. In [7], Har-Peled and Lee considered the WEIGHTED GEOMETRIC SET COVER problem with δ -extension¹ for fat polygons, and proved that the problem admits a PTAS with running time $|\mathcal{F}|^{\mathcal{O}(\epsilon^{-2}\delta^{-2})} \cdot |\mathcal{U}|$. Given this result, our goal is to understand the complexity in the setting of ultimately non-fat polygons: segments.

Our contribution. First, we show that SEGMENT SET COVER does not have a polynomial-time approximation scheme (PTAS) assuming $P \neq NP$, even if segments are axis-parallel and we relax the problem with $\frac{1}{2}$ -extension. Thus, there is no hope for the analog of the result of Har-Peled and Lee [7] in the setting of segments.

► **Theorem 1.** *There exists a constant $\gamma > 0$ such that, unless $P=NP$, there is no polynomial-time algorithm that given an instance $(\mathcal{U}, \mathcal{F})$ of (unweighted) SEGMENT SET COVER in which all segments are axis-parallel, returns a set $\mathcal{S} \subseteq \mathcal{F}$ such that $\mathcal{S}^{+\frac{1}{2}}$ covers \mathcal{U} and $|\mathcal{S}| \leq (1 + \gamma) \cdot \text{opt}$, where opt denotes the minimum size of a subset of \mathcal{F} that covers \mathcal{U} .*

¹ We note that Har-Peled and Lee considered a different definition of δ -extension, where every object L is extended by all points at distance at most $\delta \cdot \text{rad}(L)$, where $\text{rad}(L)$ is the radius of the largest circle inscribed in L . This definition works well for fat polygons, but not so for segments, hence we adopt the homothetical definition of δ -extension discussed above.

Theorem 1 justifies also considering parameterization by the solution size k . For this parameterization, we provide three parameterized algorithms:

- an FPT algorithm for (unweighted) SEGMENT SET COVER with k being the parameter;
- a *parameterized approximation scheme* (PAS) for WEIGHTED SEGMENT SET COVER: a $(1 + \epsilon)$ -approximation algorithm with running time of the form $f(k, \epsilon) \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)}$; and
- an FPT algorithm for WEIGHTED SEGMENT SET COVER with δ -extension, where both k and $\delta > 0$ are the parameters.

Formal statements of these results follow below.

► **Theorem 2.** *There is an algorithm that given a family \mathcal{F} of segments in the plane, a set \mathcal{U} of points in the plane, and a parameter k , runs in time $k^{\mathcal{O}(k)} \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)}$, and either outputs a set $\mathcal{S} \subseteq \mathcal{F}$ such that $|\mathcal{S}| \leq k$ and \mathcal{S} covers all points in \mathcal{U} , or determines that such a set \mathcal{S} does not exist.*

► **Theorem 3.** *There is an algorithm that given a family \mathcal{F} of weighted segments in the plane, a set \mathcal{U} of points in the plane, and parameters k and $\epsilon > 0$, runs in time $(k/\epsilon)^{\mathcal{O}(k)} \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)}$ and outputs a set \mathcal{S} such that:*

- $\mathcal{S} \subseteq \mathcal{F}$, $|\mathcal{S}| \leq k$, and \mathcal{S} covers all points in \mathcal{U} , and
 - the weight of \mathcal{S} is not greater than $1 + \epsilon$ times the minimum weight of a subset of \mathcal{F} of size at most k that covers \mathcal{U} ,
- or determines that there is no set $\mathcal{S} \subseteq \mathcal{F}$ with $|\mathcal{S}| \leq k$ such that \mathcal{S} covers all points in \mathcal{U} .

► **Theorem 4.** *There is an algorithm that given a family \mathcal{F} of weighted segments in the plane, a set \mathcal{U} of points in the plane, and parameters k and $\delta > 0$, runs in time $f(k, \delta) \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)}$ for some computable function f and outputs a set \mathcal{S} such that:*

- $\mathcal{S} \subseteq \mathcal{F}$, $|\mathcal{S}| \leq k$, $\mathcal{S}^{+\delta}$ covers all points in \mathcal{U} , and
 - the weight of \mathcal{S} is not greater than the minimum weight of a subset of \mathcal{F} that covers \mathcal{U} without δ -extension,
- or determines that there is no set $\mathcal{S} \subseteq \mathcal{F}$ with $|\mathcal{S}| \leq k$ such that \mathcal{S} covers all points in \mathcal{U} .

It is natural to ask whether relying on relaxations – $(1 + \epsilon)$ -approximation or δ -extension – is really necessary for WEIGHTED SEGMENT SET COVER, as Theorem 2 shows that it is not in the unweighted setting. Somewhat surprisingly, we show that this is the case by proving the following result. Recall that here we consider WEIGHTED SEGMENT SET COVER as a parameterized problem where we seek a solution of the minimum total weight among those of cardinality at most k .

► **Theorem 5.** *The WEIGHTED SEGMENT SET COVER problem is $W[1]$ -hard when parameterized by k and assuming ETH, there is no algorithm for this problem with running time $f(k) \cdot (|\mathcal{U}| + |\mathcal{F}|)^{o(k/\log k)}$ for any computable function f . Moreover, this holds even if all segments in \mathcal{F} are axis-parallel.*

Thus, the uncovered parameterized complexity of SEGMENT SET COVER is an interesting one: the problem is FPT when parameterized by the solution size k in the unweighted setting, but this tractability ceases to hold when moving to the weighted setting. However, fixed-parameter tractability in the weighted setting can be restored if one considers any of the following relaxations: $(1 + \epsilon)$ -approximation or δ -extension.

Organization. In Section 2 we prove Theorems 2, 3 and 4, while in Section 3 we prove Theorem 5. Due to space constraints, the proof of Theorem 1 is presented only in the full version of this article [9].

2 Algorithms

In this section we give our positive results – Theorems 2, 3, and 4. We start with a shared definition. For a set of collinear points C in the plane, *extreme points* of C are the endpoints of the smallest segment that covers all points from set C . In particular, if C consists of one point or is empty, then there are 1 or 0 extreme points, respectively.

2.1 Unweighted segments and a parameterized approximation scheme

We first give an FPT algorithm for WEIGHTED SEGMENT SET COVER where we additionally consider the number of different weights to be the parameter.

► **Theorem 6.** *There is an algorithm that given a family \mathcal{F} of weighted segments in the plane, a set \mathcal{U} of points in the plane, and a parameter k , runs in time $(qk)^{\mathcal{O}(k)} \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)}$, where q is the number of different weights used by the weight function, and either outputs a solution $\mathcal{S} \subseteq \mathcal{F}$ such that $|\mathcal{S}| \leq k$ and \mathcal{S} covers all points in \mathcal{U} , or determines that such a set \mathcal{S} does not exist.*

Clearly, Theorem 2 follows from applying Theorem 6 for $q = 1$. However, later we use Theorem 6 for larger values of q to obtain our parameterized approximation scheme: Theorem 3.

We remark that the proof of Theorem 6 relies on branching and kernelization arguments that are standard in parameterized algorithms. Even though the statement does not formally follow from the work of Langerman and Morin [11], the basic technique is very similar.

Towards the proof of Theorem 6, we may assume that the given instance $(\mathcal{U}, \mathcal{F}, \mathbf{w})$, where $\mathbf{w}: \mathcal{F} \rightarrow \mathbb{R}_{\geq 0}$ denotes the weight function on \mathcal{F} , is *reasonable* in the following sense: there do not exist distinct $A, B \in \mathcal{F}$ with the same weight such that $A \cap \mathcal{U} \subseteq B \cap \mathcal{U}$. Indeed, then A could be safely removed from \mathcal{F} , since in any solution, taking B instead of A does not increase the weight and may only result in covering more points in \mathcal{U} . In the next lemma we show that in reasonable instances we can find a small subset of \mathcal{F} that is guaranteed to intersect every small solution.

► **Lemma 7.** *Suppose $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ is a reasonable instance of WEIGHTED SEGMENT SET COVER where the weight function \mathbf{w} uses at most q different values. Suppose further that there exists a line L in the plane with at least $k + 1$ points of \mathcal{U} on it. Then there exists a subset $\mathcal{R} \subseteq \mathcal{F}$ of size at most qk such that every subset $\mathcal{S} \subseteq \mathcal{F}$ with $|\mathcal{S}| \leq k$ that covers \mathcal{U} satisfies $|\mathcal{R} \cap \mathcal{S}| \geq 1$. Moreover, such a subset \mathcal{R} can be found in polynomial time.*

Proof. Let us enumerate the points of \mathcal{U} that lie on L as x_1, x_2, \dots, x_i in the order in which they appear on L . By reasonability of $(\mathcal{U}, \mathcal{F})$, for every $i \in \{1, \dots, k\}$ there exist at most q different segments in \mathcal{F} that are collinear with L and cover x_i , but do not cover x_{i-1} (or just cover x_1 , in case $i = 1$). Indeed, if $A \in \mathcal{F}$ is collinear with L , covers x_i and does not cover x_{i-1} , then $A \cap \mathcal{U} = \{x_i, \dots, x_j\}$ for some $j \geq i$; so if there was another $B \in \mathcal{F}$ with the same property and the same weight as A , then the reasonability of $(\mathcal{U}, \mathcal{F})$ would imply that $A = B$. Let \mathcal{R}_i be the set of segments with the property discussed above; then $|\mathcal{R}_i| \leq q$. Our proposed set is defined as:

$$\mathcal{R} := \bigcup_{i=1}^k \mathcal{R}_i.$$

Clearly, \mathcal{R} can be found in polynomial time and $|\mathcal{R}| \leq qk$. It remains to prove that \mathcal{R} has the desired property. Consider any set $\mathcal{S} \subseteq \mathcal{F}$ of size at most k that covers \mathcal{U} .

Let \mathcal{S}_L be the set of segments from \mathcal{S} that are collinear with L . Every segment that is not collinear with L can cover at most one of the points that lie on this line. Hence, if \mathcal{S}_L was empty, then \mathcal{S} would cover at most k points on line L , but L had at least $k + 1$ different points from \mathcal{U} on it.

Therefore, we know that \mathcal{S}_L is not empty and hence $|\mathcal{S} - \mathcal{S}_L| \leq k - 1$. Segments from $\mathcal{S} - \mathcal{S}_L$ can cover at most $k - 1$ points among $\{x_1, x_2, \dots, x_k\}$, therefore at least one of these points must be covered by segments from \mathcal{S}_L . Let $i \in \{1, \dots, k\}$ be the smallest index such that x_i is covered by a segment in \mathcal{S}_L . Then, by minimality, this segment cannot cover x_{i-1} (if existent), so it must belong to \mathcal{R}_i . We conclude that $\mathcal{R} \cap \mathcal{S}$ is nonempty, as desired. ◀

With Lemma 7 in hand, we prove Theorem 6 using a straightforward branching strategy.

Proof of Theorem 6. Let $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ be the given instance and k be the given parameter: the target size of a solution. We present a recursive algorithm that proceeds as follows:

- (1) As long as there are distinct sets $A, B \in \mathcal{F}$ with $A \cap \mathcal{U} \subseteq B \cap \mathcal{U}$ and $\mathbf{w}(A) = \mathbf{w}(B)$, remove A from \mathcal{F} . Once this step is applied exhaustively, the instance $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ is reasonable.
- (2) If there is a line with at least $k + 1$ points from \mathcal{U} , we branch over the choice of adding to the solution one of the at most qk possible segments from the set \mathcal{R} provided by Lemma 7. That is, for every $s \in \mathcal{R}$, we recurse on the instance $(\mathcal{U} - s, \mathcal{F} - \{s\}, \mathbf{w})$, and parameter $k - 1$. If any such recursive call returned a solution \mathcal{S}' , then return the lightest among solutions $\mathcal{S}' \cup \{s\}$ obtained in this way. Otherwise, return that there is no solution.
- (3) If every line has at most k points on it and $|\mathcal{U}| > k^2$, then return that there is no solution.
- (4) If $|\mathcal{U}| \leq k^2$, solve the problem by brute force: check all subsets of \mathcal{F} of size at most k .

That the algorithm is correct is clear: the correctness of step (2) follows from Lemma 7, and to see the correctness of step (3) note that if no line contains more than k points, than no segment of \mathcal{F} can cover more than k points in \mathcal{U} , hence having more than k^2 points in \mathcal{U} implies that there is no solution of size at most k .

For the time complexity, observe that in the leaves of the recursion we have $|\mathcal{U}| \leq k^2$, so $|\mathcal{F}| \leq qk^4$, because every segment can be uniquely identified by its weight and the two extreme points it covers (this follows from reasonability). Therefore, there are $\binom{qk^4}{\leq k} \leq (qk)^{\mathcal{O}(k)}$ possible solutions to check, each can be checked in polynomial time. Thus, step (4) takes time $(qk)^{\mathcal{O}(k)}$ whenever applied in the leaf of the recursion.

During the recursion, the parameter k is decreased with every recursive call, so the recursion tree has depth at most k and at each node we branch over at most qk possibilities. Thus, there are at most $\mathcal{O}((qk)^k)$ nodes in the recursion tree, and local computation in each of them can be done in time $(|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)} \cdot (qk)^{\mathcal{O}(k)}$ (the second factor is due to possibly applying step (4) in the leaves). Thus, the time complexity of the algorithm is $(qk)^{\mathcal{O}(k)} \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)}$. ◀

Finally, we use Theorem 6 to prove Theorem 3, recalled below for convenience. The idea is to multiplicatively round the weights so that we obtain an instance with only few different weight values, on which the algorithm of Theorem 6 can be employed.

► **Theorem 3.** *There is an algorithm that given a family \mathcal{F} of weighted segments in the plane, a set \mathcal{U} of points in the plane, and parameters k and $\epsilon > 0$, runs in time $(k/\epsilon)^{\mathcal{O}(k)} \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)}$ and outputs a set \mathcal{S} such that:*

- $\mathcal{S} \subseteq \mathcal{F}$, $|\mathcal{S}| \leq k$, and \mathcal{S} covers all points in \mathcal{U} , and
 - the weight of \mathcal{S} is not greater than $1 + \epsilon$ times the minimum weight of a subset of \mathcal{F} of size at most k that covers \mathcal{U} ,
- or determines that there is no set $\mathcal{S} \subseteq \mathcal{F}$ with $|\mathcal{S}| \leq k$ such that \mathcal{S} covers all points in \mathcal{U} .

Proof. Let \mathcal{S}^* be an optimum solution: a minimum-weight set at most k segments in \mathcal{F} that covers \mathcal{U} . The algorithm does not know \mathcal{S}^* , but by branching into at most $|\mathcal{F}|$ choices we may assume that it knows the weight of the heaviest segment in \mathcal{S}^* ; call this weight W . Thus, we have $W \leq \mathbf{w}(\mathcal{S}^*) \leq kW$. We may dispose of all segments in \mathcal{F} whose weight is larger than W , as they will for sure not participate in the solution.

We define a new weight function $\mathbf{w}' : \mathcal{F} \rightarrow \mathbb{R}_{\geq 0}$ as follows. Consider any segment $A \in \mathcal{F}$. If $\mathbf{w}(A) \leq \frac{\epsilon}{2k} \cdot W$, then set $\mathbf{w}'(A) := \frac{\epsilon}{2k} \cdot W$. Otherwise, set $\mathbf{w}'(A) := \frac{W}{(1+\epsilon/2)^i}$, where i is the unique integer such that

$$\frac{W}{(1+\epsilon/2)^{i+1}} < \mathbf{w}(A) \leq \frac{W}{(1+\epsilon/2)^i}.$$

Note that the assumption $\mathbf{w}(A) > \frac{\epsilon}{2k} \cdot W$ implies that we always have $i \leq \log_{1+\epsilon/2}(2k/\epsilon) = \mathcal{O}(1/\epsilon \log(k/\epsilon))$. As we also have $i \geq 0$ due to removing segments of weight larger than W , we conclude that the weight function \mathbf{w}' uses at most $\mathcal{O}(1/\epsilon \log(k/\epsilon))$ different weight values.

Next, observe that for every segment $A \in \mathcal{F}$, we have

$$\mathbf{w}'(A) \leq (1+\epsilon/2) \cdot \mathbf{w}(A) + \frac{\epsilon}{2k} \cdot W.$$

Summing this inequality through all segments of \mathcal{S}^* yields

$$\mathbf{w}'(\mathcal{S}^*) \leq (1+\epsilon/2) \cdot \mathbf{w}(\mathcal{S}^*) + k \cdot \frac{\epsilon}{2k} \cdot W \leq (1+\epsilon/2) \cdot \mathbf{w}(\mathcal{S}^*) + \epsilon/2 \cdot \mathbf{w}(\mathcal{S}^*) = (1+\epsilon) \cdot \mathbf{w}(\mathcal{S}^*).$$

As \mathcal{S}^* is an optimum solution, we conclude that the optimum solution in the instance $(\mathcal{U}, \mathcal{F}, \mathbf{w}')$ for parameter k is at most $(1+\epsilon)$ times heavier than the optimum solution in the instance $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ for parameter k . Hence, it suffices to apply the algorithm of Theorem 6 to the instance $(\mathcal{U}, \mathcal{F}, \mathbf{w}')$ and parameter k and return the obtained solution. The running time is $(1/\epsilon \cdot k \log(k/\epsilon))^{\mathcal{O}(k)} \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)} = (k/\epsilon)^{\mathcal{O}(k)} \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)}$, as promised. \blacktriangleleft

2.2 Weighted segments with δ -extension

In this section we prove Theorem 4, restated below for convenience.

► Theorem 4. *There is an algorithm that given a family \mathcal{F} of weighted segments in the plane, a set \mathcal{U} of points in the plane, and parameters k and $\delta > 0$, runs in time $f(k, \delta) \cdot (|\mathcal{U}||\mathcal{F}|)^{\mathcal{O}(1)}$ for some computable function f and outputs a set \mathcal{S} such that:*

- $\mathcal{S} \subseteq \mathcal{F}$, $|\mathcal{S}| \leq k$, $\mathcal{S}^{+\delta}$ covers all points in \mathcal{U} , and
- the weight of \mathcal{S} is not greater than the minimum weight of a subset of \mathcal{F} that covers \mathcal{U} without δ -extension,

or determines that there is no set $\mathcal{S} \subseteq \mathcal{F}$ with $|\mathcal{S}| \leq k$ such that \mathcal{S} covers all points in \mathcal{U} .

Roughly speaking, our approach to prove Theorem 4 is to find a small kernel for the problem; but we need to be careful with the definition of kernelization, because we work in the δ -extension model. The key technical tool will be the notion of a *dense subset*.

Dense subsets. Intuitively speaking, for a set of collinear points C , a subset $A \subseteq C$ is dense if any small cover of A becomes a cover of C after a tiny extension. This is formalized in the following definition.

► Definition 8. *For a set of collinear points C , a subset $A \subseteq C$ is (k, δ) -dense in C if for any set of segments \mathcal{R} that covers A and such that $|\mathcal{R}| \leq k$, it holds that $\mathcal{R}^{+\delta}$ covers C .*

The key combinatorial observation in our approach is expressed in the following Lemma 9: in every collinear set C one can always find a (k, δ) -dense subset of size bounded by a function of k and δ . Later, this lemma will allow us to find a kernel for our original problem.

► **Lemma 9.** *For every set C of collinear points in the plane, $\delta > 0$ and $k \geq 1$, there exists a (k, δ) -dense set $A \subseteq C$ of size at most $(2 + \frac{4}{\delta})^k$. Moreover, such a (k, δ) -dense set can be computed in time $\mathcal{O}(|C| \cdot (2 + \frac{4}{\delta})^k)$.*

Proof. We give a proof of the existence of such a dense subset A , and at the end we will argue that the proof naturally gives rise to an algorithm with the promised complexity. We fix δ and proceed by induction on k . Formally, we shall prove the following stronger statement: For any set of collinear points C , there exists a subset $A \subseteq C$ such that:

- A is (k, δ) -dense in C ,
- $|A| \leq (2 + \frac{4}{\delta})^k$, and
- the extreme points of C are in A .

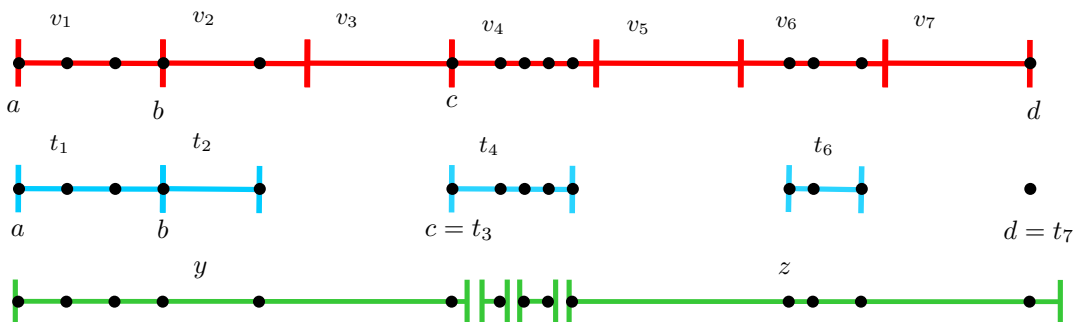
Consider first the base case when $k = 1$. Then it is sufficient to just take A that consists of the (at most 2) extreme points of C . Indeed, if the extreme points of C are covered with one segment, then this segment must cover the whole set C (even without extension). Thus, the set A has size at most $2 < (2 + \frac{4}{\delta})^1$, as required.

We now proceed to the inductive step. Assuming inductive hypothesis for any set of collinear points C and for parameter k , we will prove it for $k + 1$.

Let s be the minimal segment that includes all points from C . That is, s is the segment whose endpoints are the extreme points of C .

Split s into $M := \lceil 1 + \frac{4}{\delta} \rceil$ subsegments of equal length. We name these segments v_1, v_2, \dots, v_M in order, and we consider them closed. Note that $|v_i| = \frac{|s|}{M}$ for each $1 \leq i \leq M$, where $|\cdot|$ denotes the length of a segment.

Let C_i be the subset of C consisting of points belonging to v_i . Further, let t_i be the segment with endpoints being the extreme points of C_i . Note that t_i might be a degenerate single-point segment if C_i consists of one point, or even t_i might be empty if C_i is empty. Figure 1 presents an example of the construction.



■ **Figure 1** Example of the construction in the proof of Lemma 9 for $M = 7$ and some set of points C (marked with black circles). The top panel shows segments v_i . The middle panel shows segments t_i . Note that t_5 is an empty segment, because there are no points in C that belong to v_5 , while each of the segments t_3 and t_7 is degenerated to a single point: c and d , respectively. Segments t_1 and t_2 share one point b . The bottom panel shows an example of the second case in the correctness proof: a solution \mathcal{R} of size 4 whose all segments intersect t_4 . Then one of y and z will cover the whole of C_4 after extension.

We use the inductive hypothesis to choose a (k, δ) -dense subset A_i of C_i , for each $i \in \{1, \dots, M\}$. Note that if $|C_i| \leq 1$, then $A_i = C_i$, so A_i is (k, δ) -dense set for C_i . Also, by assumption, A_i contains the extreme points of C_i .

Next, we define $A := \bigcup_{i=1}^M A_i$. Thus A includes the extreme points of C , because they are included in the sets A_1 and A_M .

By induction, the size of each A_i is at most $(2 + \frac{4}{\delta})^k$. Therefore,

$$|A| \leq M \left(2 + \frac{4}{\delta}\right)^k = \left[1 + \frac{4}{\delta}\right] \cdot \left(2 + \frac{4}{\delta}\right)^k \leq \left(2 + \frac{4}{\delta}\right)^{k+1}.$$

We are left with verifying that A is $(k+1, \delta)$ -dense in C . For this, consider any cover of A with $k+1$ segments and call it \mathcal{R} .

Consider any segment t_i . If there exists a segment $x \in \mathcal{R}$ that is disjoint with t_i , then $\mathcal{R} - \{x\}$ constitutes a cover of A_i with at most k segments. Since A_i is (k, δ) -dense in C_i , $(\mathcal{R} - \{x\})^{+\delta}$ covers C_i . So $\mathcal{R}^{+\delta}$ covers C_i as well.

On the other hand, if for any fixed t_i a segment $x \in \mathcal{R}$ as above does not exist, then all the $k+1$ segments of \mathcal{R} intersect t_i . An example of such a situation is depicted in the bottom panel of Figure 1. Let us consider any such t_i . By induction, the endpoints of s are in A_1 and A_M respectively, so \mathcal{R} must cover them. So for each endpoint of s , there exists a segment in \mathcal{R} that contains this endpoint and intersects t_i . Let us call these two segments y and z . It follows that $|y| + |z| + |t_i| \geq |s|$. Since $|t_i| \leq |v_i| = \frac{|s|}{M} \leq \frac{|s|}{1 + \frac{4}{\delta}} = \frac{\delta|s|}{\delta+4}$, we have

$$\max(|y|, |z|) \geq |s| \left(1 - \frac{\delta}{\delta+4}\right) / 2 = \frac{2|s|}{\delta+4}.$$

After δ -extension, the longer of the segments y and z will expand at both ends by at least:

$$\delta/2 \cdot \max(|y|, |z|) \geq \frac{\delta|s|}{\delta+4} = \frac{|s|}{1 + \frac{4}{\delta}} \geq \frac{|s|}{M} = |v_i| \geq |t_i|.$$

Therefore, the longer of segments y and z will cover the whole segment t_i after δ -extension. We conclude that $\mathcal{R}^{+\delta}$ covers C_i as well.

Since $C = \bigcup_{i=1}^M C_i$, we conclude that $\mathcal{R}^{+\delta}$ covers C . So indeed, A is $(k+1, \delta)$ -dense in C . This concludes the proof of the existence of such a dense set A . To compute A in time $\mathcal{O}\left(|C| \cdot \left(2 + \frac{4}{\delta}\right)^k\right)$ observe that the inductive proof explained above can be easily turned into a recursive procedure that for a given C and k , outputs a (k, δ) -dense subset of C . The recursion tree of this procedure has size $\mathcal{O}\left(\left(2 + \frac{4}{\delta}\right)^k\right)$ in total, while every recursive calls uses $\mathcal{O}(|C|)$ time for internal computation, so the total running time is $\mathcal{O}\left(|C| \cdot \left(2 + \frac{4}{\delta}\right)^k\right)$. ◀

Long lines. We need a few additional observations in the spirit of the algorithm of Theorem 6. For a finite set of points \mathcal{U} in the plane, call a line L k -long with respect to \mathcal{U} if L contains more than k points from \mathcal{U} . We have the following observations.

► **Lemma 10.** *Let \mathcal{U} be a finite set of points in the plane such that there are more than k lines that are k -long with respect to \mathcal{U} . Then \mathcal{U} cannot be covered with k segments.*

Proof. We proceed by contradiction. Assume there are at least $k+1$ different k -long lines and there is a set of segments \mathcal{R} of size at most k covering all points in \mathcal{U} .

Consider any k -long line L . Note that every segment \mathcal{R} which is not collinear with L , covers at most one point that lies on L . Since L is long, there are at least $k+1$ points from \mathcal{U} that lie on L . This implies that there must be a segment in \mathcal{R} that is collinear with L .

Since we have at least $k+1$ different long lines, there are at least $k+1$ segments in \mathcal{R} collinear with different lines. This contradicts the assumption that $|\mathcal{R}| \leq k$. ◀

► **Lemma 11.** *Let \mathcal{U} be a finite set of points in the plane such that there are more than k^2 points from \mathcal{U} that do not lie on any line that is k -long with respect to \mathcal{U} . Then \mathcal{U} cannot be covered with k segments.*

Proof. We proceed by contradiction. Assume that we have more than k^2 points in \mathcal{U} that do not lie on any k -long line. Call this set A . Suppose there is a set of segments \mathcal{R} of size at most k that covers all points in \mathcal{U} .

Since any line in the plane can cover only at most k points in A , the same is also true for every segment in \mathcal{R} . Therefore, the segments from \mathcal{R} can cover at most k^2 points in A in total. As $|A| > k^2$, \mathcal{R} cannot cover the whole A , which is a subset of \mathcal{U} ; a contradiction. ◀

We are now ready to give a proof of Theorem 4.

Proof of Theorem 4. Let $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ be the input instance of WEIGHTED SEGMENT SET COVER, where $\mathbf{w}: \mathcal{F} \rightarrow \mathbb{R}_{\geq 0}$ is the weight function. Further, let k and $\delta > 0$ be the input parameters. Our goal is to either conclude that $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ has no solution of cardinality at most k , or to find an instance $(\mathcal{U}', \mathcal{F}', \mathbf{w})$ of size bounded by $f(k, \delta)$ for some computable function f and satisfying $\mathcal{U}' \subseteq \mathcal{U}$ and $\mathcal{F}' \subseteq \mathcal{F}$, such that the following two properties hold:

- (Property 1) For every set $\mathcal{S} \subseteq \mathcal{F}$ such that $|\mathcal{S}| \leq k$ and \mathcal{S} covers \mathcal{U} , there is a set $\mathcal{S}_1 \subseteq \mathcal{F}'$ such that $|\mathcal{S}_1| \leq k$, the weight of \mathcal{S}_1 is not greater than the weight of \mathcal{S} , and \mathcal{S}_1 covers \mathcal{U}' .
- (Property 2) For every set $\mathcal{S} \subseteq \mathcal{F}'$ such that $|\mathcal{S}| \leq k$ and \mathcal{S} covers all points in \mathcal{U}' , $\mathcal{S}^{+\delta}$ covers all points in the original set \mathcal{U} .

Suppose we constructed such an instance $(\mathcal{U}', \mathcal{F}', \mathbf{w})$. Then using *Property 1* we know that an optimum solution of size at most k to $(\mathcal{U}', \mathcal{F}', \mathbf{w})$ has no greater weight than an optimum solution of size at most k to $(\mathcal{U}, \mathcal{F}, \mathbf{w})$. On the other hand, using *Property 2* we know that any solution to $(\mathcal{U}', \mathcal{F}', \mathbf{w})$ after δ -extension covers \mathcal{U} . So it will remain to find an optimum solution to the instance $(\mathcal{U}', \mathcal{F}', \mathbf{w})$. This can be done by brute-force in time $|\mathcal{F}'|^{k+\mathcal{O}(1)} \cdot |\mathcal{U}'|^{\mathcal{O}(1)}$, which is bounded by a computable function of k and δ .

It remains to construct the instance $(\mathcal{U}', \mathcal{F}', \mathbf{w})$. Let ℓ be the number of different lines that are k -long with respect to \mathcal{U} . By Lemmas 10 and 11, if we had more than k different k -long lines or more than k^2 points from \mathcal{U} that do not lie on any k -long line, then we can safely conclude that $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ has no solution of cardinality at most k , and terminate the algorithm. So assume otherwise, in particular $\ell \leq k$.

Next, we cover \mathcal{U} with at most $k + 1$ sets:

- D consists of all points in \mathcal{U} that do not lie on any k -long line. Then we have $|D| \leq k^2$.
 - For $1 \leq i \leq \ell$, C_i consists of all points in \mathcal{U} that lie on the i -th long line. Then $|C_i| > k$.
- Note that sets C_i do not need to be disjoint.

For every set C_i , we apply Lemma 9 to obtain a subset $A_i \subseteq C_i$ that is (k, δ) -dense and satisfies $|A_i| \leq (2 + \frac{4}{\delta})^k$. We define $\mathcal{U}' := D \cup \bigcup_{i=1}^{\ell} A_i$. Thus, \mathcal{U}' has size at most $k^2 + k(2 + \frac{4}{\delta})^k$. Further, we define \mathcal{F}' as follows: for every pair of points in \mathcal{U}' , if there are segments in \mathcal{F} that cover this pair of points, we choose one such segment with the lowest weight and include it in \mathcal{F}' . Thus \mathcal{F}' has size at most $|\mathcal{U}'|^2$, which means that both \mathcal{F}' and \mathcal{U}' have sizes bounded by $\mathcal{O}((k^2 + k(2 + \frac{4}{\delta})^k)^2)$. We are left with verifying Properties 1 and 2.

For Property 2, consider any set $\mathcal{S} \subseteq \mathcal{F}'$ such that $|\mathcal{S}| \leq k$ and \mathcal{S} covers all points in \mathcal{U}' . Then in particular, for every $i \in \{1, \dots, \ell\}$, \mathcal{S} covers all points in A_i . As A_i is (k, δ) -dense in C_i , we conclude that $\mathcal{S}^{+\delta}$ covers C_i . Hence $\mathcal{S}^{+\delta}$ covers $D \cup \bigcup_{i=1}^{\ell} C_i = \mathcal{U}$, as required.

For Property 1, consider any solution \mathcal{S} to $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ of size at most k . For every segment $s \in \mathcal{S}$, let B_s be the set of points in \mathcal{U}' that are covered by s . B_s is of course a set of collinear points, hence B_s can be covered by any segment that covers the extreme points of B_s . Therefore, we can replace s with a segment $s' \in \mathcal{F}$ that has the lowest weight among the

segments that cover the extreme points of B_s . Such a segment belongs to \mathcal{F}' by construction, and s' has weight no greater than the weight of s , because s also covers the extreme points of B_s . Therefore, if $\mathcal{S}_1 \subseteq \mathcal{F}'$ is the set obtained by performing such replacement for every $s \in \mathcal{S}$, then \mathcal{S}_1 has both size and weight not greater than \mathcal{S} , and \mathcal{S}_1 covers \mathcal{U}' . \blacktriangleleft

3 W[1]-hardness of WEIGHTED SEGMENT SET COVER

In this section we prove Theorem 5, recalled below for convenience.

► **Theorem 5.** *The WEIGHTED SEGMENT SET COVER problem is W[1]-hard when parameterized by k and assuming ETH, there is no algorithm for this problem with running time $f(k) \cdot (|\mathcal{U}| + |\mathcal{F}|)^{o(k/\log k)}$ for any computable function f . Moreover, this holds even if all segments in \mathcal{F} are axis-parallel.*

To prove Theorem 5, we give a reduction from a W[1]-hard problem: PARTITIONED SUBGRAPH ISOMORPHISM, defined as follows. An instance of PARTITIONED SUBGRAPH ISOMORPHISM consists of a *pattern graph* H , a *host graph* G , and a function $\lambda: V(G) \rightarrow V(H)$ that colors every vertex of G with a vertex of H . The task is to decide whether there exists a subgraph embedding $\phi: V(H) \rightarrow V(G)$ that respects the coloring λ . That is, the following conditions have to be satisfied.

- $\lambda(\phi(a)) = a$ for each $a \in V(H)$, and
- $\phi(a)$ and $\phi(b)$ are adjacent in G for every edge $ab \in E(H)$.

The following complexity lower bound for PARTITIONED SUBGRAPH ISOMORPHISM was proved by Marx in [14].

► **Theorem 12** ([14]). *Consider the PARTITIONED SUBGRAPH ISOMORPHISM problem where the pattern graph H is assumed to be 3-regular. Then this problem is W[1]-hard when parameterized by k , the number of vertices of H , and assuming the ETH there is no algorithm solving this problem in time $f(k) \cdot |V(G)|^{o(k/\log k)}$, where f is any computable function.*

In the remainder of this section we prove Theorem 5 by providing a parameterized reduction from PARTITIONED SUBGRAPH ISOMORPHISM to WEIGHTED SEGMENT SET COVER. The technical statement of the reduction is encapsulated in the following lemma.

► **Lemma 13.** *Given an instance (H, G, λ) of PARTITIONED SUBGRAPH ISOMORPHISM where H is 3-regular and has k vertices, one can in polynomial time construct an instance $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ of WEIGHTED SEGMENT SET COVER and a positive real W such that:*

- (1) *all segments in \mathcal{F} are axis-parallel;*
- (2) *if the instance (H, G, λ) has a solution, then there exists a solution to $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ of cardinality $\frac{11}{2}k$ and weight at most W ; and*
- (3) *if there exists a solution to $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ of weight at most W , then the instance (H, G, λ) has a solution.*

Note that in (3) we in fact do not require any bound on the cardinality of the solution, just on its weight.

It is easy to see that Lemma 13 implies Theorem 12. First, Lemma 13 gives a parameterized reduction from the W[1]-hard PARTITIONED SUBGRAPH ISOMORPHISM problem with 3-regular pattern graphs to WEIGHTED SEGMENT SET COVER parameterized by the cardinality of the sought solution, which shows that the latter problem is also W[1]-hard. Second, combining the reduction with an algorithm for WEIGHTED SEGMENT SET COVER with running time as postulated in Theorem 5 would give an algorithm for PARTITIONED SUBGRAPH ISOMORPHISM

47:12 Parameterized and Approximation Algorithms for SEGMENT SET COVER

with running time $f(k) \cdot |V(G)|^{o(k/\log k)}$ for a computable function f , which would contradict ETH by Theorem 12. So we are left with giving a proof of Lemma 13, which spans the remainder of this section.

The key element of the proof will be a construction of a *choice gadget* that works within a single line; this construction is presented in the lemma below. Here, a *chain* is a sequence $(A_1, A_2, \dots, A_\ell)$ of subsets of \mathbb{R} such that for each $i \in \{1, \dots, \ell - 1\}$, all numbers in A_i are strictly smaller than all numbers in A_{i+1} .

► **Lemma 14.** *Suppose we are given an integer $N > 100$ and p chains $\{(A_{j,1}, \dots, A_{j,\ell}) : j \in \{1, \dots, p\}\}$ of length ℓ each such that the sets $\{A_{j,t} : j \in \{1, \dots, p\}, t \in \{1, \dots, \ell\}\}$ are all pairwise disjoint and contained in $\{1, \dots, N\}$. Then one can in polynomial time construct a set of points $\mathcal{U} \subseteq \mathbb{R}$, $\mathcal{U} \supseteq \{1, \dots, N\}$, as well as a set of segments \mathcal{F} contained in \mathbb{R} such that the following holds:*

- For every $j \in \{1, \dots, p\}$ and every set B that contains exactly one point from each element of the chain $(A_{j,1}, \dots, A_{j,\ell})$, there exists $\mathcal{R}_B \subseteq \mathcal{F}$ such that $|\mathcal{R}_B| = \ell + 1$, \mathcal{R}_B covers all points of \mathcal{U} except for B , and the total length of the segments in \mathcal{R}_B is equal to $N + 1 - 2\ell/N^2$.
- For every subset of segments $\mathcal{R} \subseteq \mathcal{F}$, if \mathcal{R} covers all points in $\mathcal{U} - \{1, \dots, N\}$, then the total length of segments in \mathcal{R} is at least $N + 1 - 2/N$.
- For every subset of segments $\mathcal{R} \subseteq \mathcal{F}$, if the total length of segments of \mathcal{R} is not larger than $N + \frac{3}{2}$ and \mathcal{R} covers all points in $\mathcal{U} - \{1, \dots, N\}$, then the total length of segments of \mathcal{R} is equal to $N + 1 - 2\ell/N^2$ and there exists $j \in \{1, \dots, p\}$ such that for every $t \in \{1, \dots, \ell\}$, \mathcal{R} does not cover the whole set $A_{j,t}$.

Proof. Denote $I := \{1, \dots, N\}$ and $\epsilon := 1/N^2$ for convenience. For every $i \in I$, let

$$i^- := i - \epsilon \quad \text{and} \quad i^+ := i + \epsilon.$$

Define $I^- := \{i^- : i \in I\}$, $I^+ := \{i^+ : i \in I\}$, and

$$\mathcal{U} := \{0\} \cup I^- \cup I \cup I^+.$$

Next, for every $j \in \{1, \dots, p\}$, define the following set of segments:

$$\mathcal{R}_j := \{[0, a^-] : a \in A_{j,1}\} \cup \bigcup_{t=1}^{\ell-1} \{[a^+, b^-] : (a, b) \in A_{j,t} \times A_{j,t+1}\} \cup \{[a^+, N + 1] : a \in A_{j,\ell}\}.$$

We set

$$\mathcal{F} := \bigcup_{j=1}^p \mathcal{R}_j.$$

See Figure 2 for a visualization of the construction. We are left with verifying the three postulated properties of \mathcal{U} and \mathcal{F} .



■ **Figure 2** Construction of Lemma 14 for $N = 8$. Elements of $I \cup \{0\}$ are depicted with circles and elements of $I^+ \cup I^-$ are depicted with squares. Blue segments represent the set \mathcal{R}_B for $B = \{3, 7\}$.

For the first property, let b_t be the unique element of $B \cap A_{j,t}$, for $t \in \{1, \dots, \ell\}$, and let

$$\mathcal{R}_B := \{[0, b_1^-], [b_1^+, b_2^-], \dots, [b_{\ell-1}^+, b_\ell^-], [b_\ell^+, N + 1]\}.$$

It is straightforward to see that \mathcal{R}_B covers all the points of \mathcal{U} except for B , and that the total sum of lengths of segments in \mathcal{R}_B is $N + 1 - 2\ell\epsilon = N + 1 - 2\ell/N^2$.

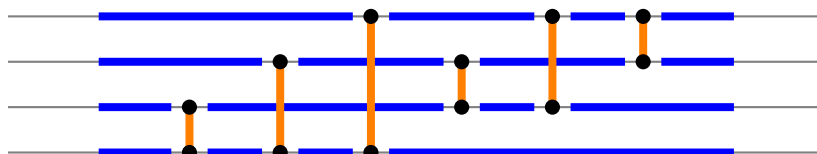
For the second postulated property, observe that each segment of \mathcal{F} that covers any point $i^+ \in I^+$, in fact covers the whole interval $[i^+, (i + 1)^-]$ (where $(N + 1)^- = N + 1$). Similarly, each segment of \mathcal{F} that covers any point $i^- \in I^-$, in fact covers the whole interval $[(i - 1)^+, i^-]$ (where $0^+ = 0$). Hence, if $\mathcal{R} \subseteq \mathcal{F}$ covers all points of $\mathcal{U} - I$, in particular \mathcal{R} covers all points in $I^+ \cup I^-$, hence also all intervals of the form $[i^+, (i + 1)^-]$ for $i \in \{0, 1, \dots, N\}$. The sum of the lengths of those intervals is equal to $N + 1 - 2\epsilon N = N + 1 - 2/N$. Hence, the sum of length of intervals in \mathcal{R} must be at least $N + 1 - 2/N$.

For the third postulated property, observe that if two segments of \mathcal{F} intersect, then their intersection is a segment of length at least $1 - 2\epsilon$. Since \mathcal{R} covers all points of $\mathcal{U} - I$, by the second property the sum of lengths of the segments in \mathcal{R} is at least $N + 1 - 2/N$. Now if any of those segments intersected, then the total sum of lengths of the segments in \mathcal{R} would be at least $N + 1 - 2/N + (1 - 2\epsilon)$, which is larger than $N + \frac{3}{2}$. We conclude that the segments of \mathcal{R} are pairwise disjoint.

Since $0 \in \mathcal{U} - I$, there is a segment $s_1 \in \mathcal{R}$ that covers 0. By construction, there exists $j \in \{1, \dots, p\}$ such that $s_1 = [0, b_{j,1}^-]$ for some $b_{j,1} \in A_{j,1}$. As the segments of \mathcal{R} are pairwise disjoint and cover all points in I^+ , the next (in the natural order on \mathbb{R}) segment in \mathcal{R} must start at $b_{j,1}^+$, and in particular $b_{j,1}$ is not covered by \mathcal{R} . Since all sets in all chains on input are pairwise disjoint, the segment in \mathcal{R} starting at $b_{j,1}^+$ must be of the form $s_2 = [b_{j,1}^+, b_{j,2}^-]$ for some $b_{j,2} \in A_{j,2}$. Continuing this reasoning, we find that in fact $\mathcal{R} = \mathcal{R}_B$ for some set $B = \{b_{j,1}, b_{j,2}, \dots, b_{j,\ell}\}$ such that $b_{j,t} \in A_{j,t}$ for each $t \in \{1, \dots, \ell\}$. In particular, the total length of segments in \mathcal{R} is equal to $N + 1 - 2\epsilon\ell$ and \mathcal{R} does not cover any point in B ; the latter implies that for each $t \in \{1, \dots, \ell\}$, \mathcal{R} does not cover $A_{j,t}$ entirely. ◀

With Lemma 14 established, we proceed to the proof of Lemma 13.

Let (H, G, λ) be the given instance of PARTITIONED SUBGRAPH ISOMORPHISM where H is a 3-regular graph. Let $k := |V(H)|$ and $\ell := |E(H)|$; note that $\ell = \frac{3}{2}k$. We may assume that $V(H) = \{1, \dots, k\}$, and that whenever uv is an edge in G , we have that $\lambda(u)\lambda(v)$ is an edge of H (other edges in G play no role in the problem and can be discarded). We construct an instance $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ of WEIGHTED SEGMENT SET COVER as follows; see Figure 3 for a visualization.



■ **Figure 3** Example solution in the instance $(\mathcal{U}, \mathcal{F})$ constructed in the proof of Lemma 13 for $H = K_4$. Blue segments belong to the sets \mathcal{S}_i for $i \in \{1, 2, 3, 4\}$ and orange segments belong to \mathcal{D} .

For each edge $ab \in E(H)$, let E_{ab} be the subset of those edges uv of G for which $\lambda(u) = a$ and $\lambda(v) = b$. Thus, $\{E_{ab} : ab \in E(H)\}$ is a partition of $E(G)$. Let $N := |E(G)|$ and $\xi : E(G) \rightarrow \{1, \dots, N\}$ be any bijection such that for each $ab \in E(H)$, $\xi(E_{ab})$ is a contiguous interval of integers. By copying some vertices of G if necessary, we may assume that $N > 100k$.

47:14 Parameterized and Approximation Algorithms for SEGMENT SET COVER

Consider any $a \in \{1, \dots, k\}$ and let b_1, b_2, b_3 be the three neighbors of a in H , ordered so that $(\xi(E_{ab_1}), \xi(E_{ab_2}), \xi(E_{ab_3}))$ is a chain. For each $u \in \lambda^{-1}(a)$, let E_u be the set of edges of G incident to u , and let us construct the chain

$$C_u := (\xi(E_u \cap E_{ab_1}), \xi(E_u \cap E_{ab_2}), \xi(E_u \cap E_{ab_3})).$$

Note that all sets featured in all the chains C_u , for $u \in \lambda^{-1}(a)$, are pairwise disjoint. We now apply Lemma 14 for the integer N and the chains $\{C_u : u \in \lambda^{-1}(a)\}$. This way, we construct a suitable point set $\mathcal{U}_a \subseteq \mathbb{R}$ and a set of segments \mathcal{F}_a contained in \mathbb{R} . We put all those points and segments on the line $\{(x, a) : x \in \mathbb{R}\}$; that is, every point $x \in \mathcal{U}_a$ is replaced with the point (x, a) , and similarly for the segments of \mathcal{F}_a . By somehow abusing the notation, we let \mathcal{U}_a and \mathcal{F}_a be the point set and the segment set after the replacement.

Next, for every edge uv of G , we define s_{uv} to be the segment with endpoints $(\xi(uv), a)$ and $(\xi(uv), b)$, where $a = \lambda(u)$ and $b = \lambda(v)$.

We set

$$\mathcal{U} := \bigcup_{a=1}^k \mathcal{U}_a \quad \text{and} \quad \mathcal{F} := \{s_{uv} : uv \in E(G)\} \cup \bigcup_{a=1}^k \mathcal{F}_a.$$

Note that all segments in sets \mathcal{F}_a are horizontal and each segment s_{uv} is vertical, thus \mathcal{F} consists of axis-parallel segments. Each segment $s \in \bigcup_{a=1}^k \mathcal{F}_a$ is assigned weight $\mathbf{w}(s)$ equal to the length of s , and each segment s_{uv} for $uv \in E(G)$ is assigned weight $\mathbf{w}(s_{uv}) = \delta$, where $\delta := 1/N^4$. Finally, we set

$$W := k \cdot (N + 1 - 6/N^2) + \delta \ell.$$

This concludes the construction of the instance $(\mathcal{U}, \mathcal{F}, \mathbf{w})$. We are left with verifying the correctness of the reduction, which is done in the following two claims.

▷ **Claim 15.** Suppose the input instance (H, G, λ) of PARTITIONED SUBGRAPH ISOMORPHISM has a solution. Then the output instance $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ of WEIGHTED SEGMENT SET COVER has a solution of cardinality $4k + \ell = \frac{11}{2}k$ and weight at most W .

Proof. Let ϕ be the supposed solution to (H, G, λ) . By the first property of Lemma 14, for every $a \in \{1, \dots, k\}$ there is a set $\mathcal{R}_{\phi, a}$ of size 4 and total weight $N + 1 - 6/N^2$ that covers all points from \mathcal{U}_a except for the points

$$(\xi(\phi(a)\phi(b_1)), a), (\xi(\phi(a)\phi(b_2)), a), (\xi(\phi(a)\phi(b_3)), a),$$

where b_1, b_2, b_3 are the neighbors of a in H . Define

$$\mathcal{S} := \{s_{\phi(a)\phi(b)} : ab \in E(H)\} \cup \bigcup_{a=1}^k \mathcal{R}_{\phi, a}.$$

Thus, for each $a \in \{1, \dots, k\}$, the aforementioned points of \mathcal{U}_a not covered by $\mathcal{R}_{\phi, a}$ are actually covered by the segments $s_{\phi(a)\phi(b_1)}, s_{\phi(a)\phi(b_2)}, s_{\phi(a)\phi(b_3)}$. We conclude that \mathcal{S} covers all the points in \mathcal{U} and has cardinality $4k + \ell = \frac{11}{2}k$ and total weight W , as promised. ◁

▷ **Claim 16.** Suppose the output instance $(\mathcal{U}, \mathcal{F}, \mathbf{w})$ of WEIGHTED SEGMENT SET COVER has a solution of weight at most W . Then the input instance (H, G, λ) of PARTITIONED SUBGRAPH ISOMORPHISM has a solution.

Proof. Let \mathcal{S} be the supposed solution to $(\mathcal{U}, \mathcal{F}, \mathbf{w})$. Denote

$$\mathcal{D} := \mathcal{S} \cap \{s_{uv} : uv \in E(G)\}$$

and

$$\mathcal{S}_a := \mathcal{S} \cap \mathcal{F}_a \quad \text{for } a \in \{1, \dots, k\}.$$

Fix $a \in \{1, \dots, k\}$ for a moment. Observe that the segments from \mathcal{D} can only cover points with positive integer coordinates within the set \mathcal{U}_a , hence the whole point set $\mathcal{U}_a - (\{1, \dots, N\} \times \{a\})$ has to be covered by \mathcal{S}_a . By the second property of Lemma 14 we infer that the total weight of \mathcal{S}_a must be at least $N + 1 - 2/N$.

Observe now that

$$W - k \cdot (N + 1 - 2/N) = \delta\ell + 2k/N - 6k/N^2 < \frac{1}{2}.$$

It follows that the total weight of each set \mathcal{S}_a must be smaller than $N + \frac{3}{2}$, for otherwise the sum of weights of sets \mathcal{S}_a would be larger than W . By the third property of Lemma 14, we infer that for every $a \in \{1, \dots, k\}$, the total weight of \mathcal{S}_a is equal to $N + 1 - 6/N^2$ and there exists $\phi(a) \in \lambda^{-1}(a)$ such that \mathcal{S}_a does not entirely cover any of the sets $\xi(E_{\phi(a)} \cap E_{ab_1}), \xi(E_{\phi(a)} \cap E_{ab_2}), \xi(E_{\phi(a)} \cap E_{ab_3})$, where b_1, b_2, b_3 are the three neighbors of a in H . In particular, there are edges $e_{a,b_1} \in E_{ab_1}, e_{a,b_2} \in E_{ab_2}, e_{a,b_3} \in E_{ab_3}$, all sharing the endpoint $\phi(a)$, such that \mathcal{S}_a does not cover the points $(\xi(e_{a,b_1}), a), (\xi(e_{a,b_2}), a), (\xi(e_{a,b_3}), a)$. Call these points X_a and let $X := \bigcup_{a=1}^k X_a$. Note that

$$|X| = 3k = 2\ell$$

and that X must be entirely covered by \mathcal{D} .

Since the weight of \mathcal{S}_a is equal to $N + 1 - 6/N^2$ for each $a \in \{1, \dots, k\}$, the weight of \mathcal{D} is upper bounded by

$$W - k \cdot (N + 1 - 6/N^2) = \delta\ell.$$

As every member of \mathcal{D} has weight δ , we conclude that $|\mathcal{D}| \leq \ell$. Now, one can readily verify that every segment $s_{uv} \in \mathcal{D}$ can cover at most two points in X , as X cannot contain more than two points with the same horizontal coordinate (recall that this coordinate is the index of an edge of G). Moreover, s_{uv} can cover two points in X only if $u = \phi(a)$ and $v = \phi(b)$, where $a = \lambda^{-1}(u)$ and $b = \lambda^{-1}(v)$. As $|X| = 2\ell$ and $|\mathcal{D}| \leq \ell$, this must be the case for every segment in \mathcal{D} . In particular, $\phi(a)\phi(b)$ must be an edge in G for every edge $ab \in E(H)$, so ϕ is a solution to the instance (H, G, λ) of PARTITIONED SUBGRAPH ISOMORPHISM. \triangleleft

Claims 15 and 16 finish the proof of Lemma 13. So the proof of Theorem 5 is also done.

References

- 1 Anna Adamaszek, Parinya Chalermsook, and Andreas Wiese. How to tame rectangles: Solving Independent Set and Coloring of rectangles via shrinking. In *18th International Conference on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2015*, volume 40 of *LIPICs*, pages 43–60. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.APPROX-RANDOM.2015.43.
- 2 Anna Adamaszek, Sariel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *J. ACM*, 66(4):29:1–29:40, 2019. doi:10.1145/3326122.



- 3 Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Comput. Geom.*, 47(2):112–124, 2014. doi:10.1016/j.comgeo.2012.04.001.
- 4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 5 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *46th ACM Symposium on Theory of Computing, STOC 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.
- 6 Thomas Erlebach and Erik Jan van Leeuwen. PTAS for Weighted Set Cover on unit squares. In *13th International Workshop on Approximation and Combinatorial Optimization, APPROX 2010*, volume 6302 of *Lecture Notes in Computer Science*, pages 166–177. Springer, 2010. doi:10.1007/978-3-642-15369-3_13.
- 7 Sarel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *J. Comput. Geom.*, 3(1):65–85, 2012. doi:10.20382/jocg.v3i1a4.
- 8 Jan Kára and Jan Kratochvíl. Fixed parameter tractability of independent set in segment intersection graphs. In *Second International Workshop on Parameterized and Exact Computation, IWPEC 2006*, volume 4169 of *Lecture Notes in Computer Science*, pages 166–174. Springer, 2006. doi:10.1007/11847250_15.
- 9 Katarzyna Kowalska and Michał Pilipczuk. Parameterized and approximation algorithms for coverings points with segments in the plane. *CoRR*, abs/2402.16466, 2024. arXiv:2402.16466.
- 10 Stefan Kratsch, Geevarghese Philip, and Saurabh Ray. Point Line Cover: The easy kernel is essentially tight. *ACM Trans. Algorithms*, 12(3):40:1–40:16, 2016. doi:10.1145/2832912.
- 11 Stefan Langerman and Pat Morin. Covering things with things. *Discret. Comput. Geom.*, 33(4):717–729, 2005. doi:10.1007/s00454-004-1108-4.
- 12 Dániel Marx. Efficient approximation schemes for geometric problems? In *13th European Symposium on Algorithms, ESA 2005*, pages 448–459. Springer, 2005.
- 13 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Second International Workshop on Parameterized and Exact Computation, IWPEC 2006*, volume 4169 of *Lecture Notes in Computer Science*, pages 154–165. Springer, 2006. doi:10.1007/11847250_14.
- 14 Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.
- 15 Dániel Marx and Michał Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. *ACM Trans. Algorithms*, 18(2):13:1–13:64, 2022. doi:10.1145/3483425.
- 16 Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Settling the apx-hardness status for geometric set cover. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 541–550. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.64.
- 17 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discret. Comput. Geom.*, 44(4):883–895, 2010. doi:10.1007/s00454-010-9285-9.
- 18 Michał Pilipczuk, Erik Jan van Leeuwen, and Andreas Wiese. Approximation and parameterized algorithms for geometric independent set with shrinking. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017*, volume 83 of *LIPICs*, pages 42:1–42:13. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.MFCS.2017.42.
- 19 Michał Pilipczuk, Erik Jan van Leeuwen, and Andreas Wiese. Quasi-polynomial time approximation schemes for packing and covering problems in planar graphs. *Algorithmica*, 82(6):1703–1739, 2020. doi:10.1007/s00453-019-00670-w.
- 20 Andreas Wiese. Independent Set of convex polygons: From n^ϵ to $1 + \epsilon$ via shrinking. *Algorithmica*, 80(3):918–934, 2018. doi:10.1007/s00453-017-0347-8.

FPT Approximation of Generalised Hypertree Width for Bounded Intersection Hypergraphs

Matthias Lanzinger  

TU Wien, Austria

University of Oxford, UK

Igor Razgon  

Birkbeck, University of London, UK

Abstract

Generalised hypertree width (ghw) is a hypergraph parameter that is central to the tractability of many prominent problems with natural hypergraph structure. Computing ghw of a hypergraph is notoriously hard. The decision version of the problem, checking whether $ghw(H) \leq k$, is PARANP-hard when parameterised by k . Furthermore, approximation of ghw is at least as hard as approximation of SET-COVER, which is known to not admit any FPT approximation algorithms.

Research in the computation of ghw so far has focused on identifying structural restrictions to hypergraphs – such as bounds on the size of edge intersections – that permit XP algorithms for ghw . Yet, even under these restrictions that problem has so far evaded any kind of FPT algorithm. In this paper we make the first step towards FPT algorithms for ghw by showing that the parameter can be approximated in FPT time for graphs of bounded edge intersection size. In concrete terms we show that there exists an FPT algorithm, parameterised by k and d , that for input hypergraph H with maximal cardinality of edge intersections d and integer k either outputs a tree decomposition with $ghw(H) \leq 4k(k+d+1)(2k-1)$, or rejects, in which case it is guaranteed that $ghw(H) > k$. Thus, in the special case of hypergraphs of bounded edge intersection, we obtain an FPT $O(k^3)$ -approximation algorithm for ghw .

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Approximation algorithms analysis; Mathematics of computing → Hypergraphs

Keywords and phrases generalized hypertree width, hypergraphs, parameterized algorithms, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.48

Related Version *Full Version*: <https://arxiv.org/abs/2309.17049> [17]

Funding *Matthias Lanzinger*: Matthias Lanzinger acknowledges support by the Royal Society “RAISON DATA” project (Reference No. RP\R1\201074). Lanzinger has been supported by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT2201].

1 Introduction

A tree decomposition of a hypergraph H is a pair (T, \mathbf{B}) where T is a tree and $\mathbf{B} : V(T) \rightarrow 2^{V(H)}$ assigns a *bag* to each node, that satisfies certain properties. The treewidth of a decomposition is $\max_{u \in V(T)} |\mathbf{B}(u)| - 1$ and the treewidth of H is the least treewidth taken over all decompositions. In many graph algorithms, treewidth is the key parameter that determines the complexity of the problem. However, for many problems whose underlying structure is naturally expressed in terms of hypergraphs the situation is different. The treewidth of a hypergraph is always at least as large as its *rank* (-1), i.e., the maximal size of an edge. Yet, many standard hypergraph problems can be tractable even with unbounded rank. To counteract this problem, *generalised hypertree width* (ghw) often takes the place of treewidth in these cases. The definition of ghw is also based on tree decompositions, with the



© Matthias Lanzinger and Igor Razgon;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 48; pp. 48:1–48:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



only difference being that the ghw of a decomposition is $\max_{u \in V(T)} \rho(\mathbf{B}(u))$, where $\rho(U)$ is the least number of edges required to cover $U \subseteq V(H)$. Parallel to treewidth in graphs, low ghw is a key criterion for tractability in the hypergraph setting. Prominent examples include the evaluation of conjunctive queries, database factorisation [19], winner determination in combinatorial auctions [7], and determining Nash Equilibria in strategic games [8].

Computation of ghw is computationally challenging. The problem is known to be **paraNP**-hard [13, 6] and $W[2]$ -hard [9] in the parameterised setting¹ (see the discussion of related work below for details). This is shown by reduction from **SET-COVER**, which together with recent breakthrough results on the approximability of **SET-COVER** [16], also implies that there can be no **FPT** approximation algorithms for ghw under standard assumptions. In this paper, we introduce the first **FPT** algorithm for approximation of ghw in unbounded rank hypergraphs. Notably, this comes over 20 years after the parameter was first introduced [12]. As there can be no such algorithm in the general case, we instead consider the restriction of the cardinality of the intersections of any two edges. Formally, a $(2, d)$ -hypergraph is a hypergraph where the intersection of any pair of edges has cardinality at most d . We will study the following problem f -APPROXGHW.

f -APPROXGHW

Input $(2, d)$ -hypergraph H , positive integer k
Parameters k and d
Output A tree decomposition of H with ghw at most $f(k, d)$,
 or **Reject**, in which case $ghw(H) > k$.

Let $\alpha(k, d)$ refer to the term $k(3k + d + 1)(2k - 1)$. Our main result is the following.

► **Theorem 1.** $4\alpha(k, d)$ -APPROXGHW is fixed-parameter tractable.

Our algorithm follows classic ideas for **FPT** algorithms for treewidth but requires significant new developments. Most critically, we propose an **FPT** algorithm for computing approximate (A, B) -separators in $(2, d)$ -hypergraph, i.e., set of vertices S such that sets of vertices A and B are not connected in H without S .

Related Work. Despite the close relationship between ghw and treewidth, there is a stark difference in the complexity of recognising the respective widths. While it is famously possible to decide $tw(\cdot) \leq k$ in fixed-parameter linear time [2], deciding $ghw(\cdot) \leq k$ is significantly harder. Intuitively, this is because techniques for efficiently deciding treewidth fundamentally rely on bounding the number of vertices in the bag, yet even α -acyclic hypergraphs (those with ghw 1) can require decompositions with arbitrarily large bags to achieve minimal ghw . In concrete terms, deciding $ghw(\cdot) \leq k$ has been shown to be **NP**-hard for all fixed $k > 1$ (or **paraNP**-hard in terms of parameterised complexity) [13, 6]. Additionally, deciding $ghw(\cdot) \leq k$ is known to be $W[2]$ -hard by a reduction from **SET COVER** [9].

In response, significant effort has been invested in identifying conditions under which deciding $ghw(\cdot) \leq k$ is tractable [13, 11] for fixed k (i.e., the problem is in **XP**). Of particular note here is the observation that the problem is in **XP** if we restrict the problem to so-called (c, d) -hypergraphs, i.e., hypergraphs where any intersection of at least c edges has cardinality at most d . Notably, this coincides with the most general condition known to allow kernelization for **SET-COVER** [20].

¹ When discussing the parameterised complexity of deciding whether a width parameter is at most k , we always refer to the parameterisation by k if not specified otherwise.

As mentioned above, negative results for FPT approximation of SET-COVER apply also to approximating ghw . Nonetheless, some approximation results are known when looking beyond FPT. Importantly, ghw is 3-approximable in XP via the notion of (not generalised) *hypertree width* (hw) [12, 1]. Despite this improvement in the case of fixed k , deciding hypertree width is also W[2]-hard [9] by the same reduction from SET-COVER as ghw . *Fractional hypertree width* fhw generalises ghw in the sense that the width is determined by the *fractional cover number* of the bags [15]. This width notion is strictly more general than ghw and allows for cubic approximation on XP [18]. Recently, Razgon [21] proposed an FPT algorithm for constant factor approximation of ghw for hypergraphs of bounded rank. While we consider this to be an important conceptual step towards our result, it should be noted that for any hypergraph H it holds that $ghw(H) \leq tw(H) \cdot rank(H)$, i.e., assuming bounded rank tightly couples the problem to deciding treewidth.

Structure of the paper. Our main result combines on multiple novel combinatorial observations and FPT algorithms and the main body of this paper is therefore focused on presenting the main ideas and how these parts interact. Full proof details are provided in the appendix of the full version [17]. After basic technical preliminaries in Section 2, we give a high-level overview of the individual parts that lead to the main result in Section 3. After the initial overview, Section 4 presents the key algorithms formally, together with sketches of their correctness and complexity. Similarly, the main combinatorial ideas are discussed in Section 5. We discuss potential avenues for future research in Section 6.

2 Preliminaries

We will frequently write $[n]$ for the set $\{1, 2, \dots, n\}$. We say that (possibly empty) pairwise disjoint sets X_1, X_2, \dots, X_n are a *weak partition* of set X if their union equals X . We assume familiarity with standard concepts of parameterised algorithms and refer to [3] for details. We use $poly(\cdot)$ to represent some polynomial function in the representation size of the argument.

A *hypergraph* H is a pair of sets $(V(H), E(H))$ where we call $V(H)$ the *vertices* of H and $E(H) \subseteq 2^{V(H)}$ the (*hyper*)*edges* of H . We assume throughout that H has no isolated vertices, i.e., vertices that are not in any edge. For $v \in V(H)$, define $I(v) := \{e \in E(H) \mid v \in e\}$, i.e., the set of edges incident to v . We say that H is a (c, d) -hypergraph if for any set $\{e_1, \dots, e_c\} \subseteq E(H)$ of c edges, it holds that $|\bigcap_i^c e_i| \leq d$. We will refer to the set of (maximal) connected components of a hypergraph H as $CComp(H)$. The *induced subhypergraph* of H induced by $U \subseteq V(H)$ is the hypergraph H' with $V(H') = U$ and $E(H') = \{e \cap U \mid e \in E(H)\} \setminus \emptyset$. We use the notation $H[U]$ to mean the induced subhypergraph of H induced by U . For tree T and $v \in V(T)$ we sometimes write $T - v$ to mean the subgraph of T obtained by deleting v and its incident edges. Let $A, B \subseteq V(H)$. An (A, B) -separator is a set $S \subseteq V(H)$ such that there is no path from an $a \in A \setminus S$ to a $b \in B \setminus S$ in $H[V(H) \setminus S]$. For $e \in E(H)$, $U_1, \dots, U_n \subseteq V(H)$ we say that e *touches* U_1, \dots, U_n if $e \cap U_i \neq \emptyset$ for all $i \in [n]$.

An *edge cover* μ for $U \subseteq V(H)$ is a subset of $E(H)$ such that $U \subseteq \bigcup \mu$. We sometimes refer to the cardinality of an edge cover as its *weight*. The *edge cover number* $\rho(U)$ for set $U \subseteq V(H)$ is the minimal weight over all edge covers for U . We sometimes say that μ is an edge cover of H to mean an edge cover of $V(H)$. Similarly, we use $\rho(H)$ instead of $\rho(V(H))$. A set of edges $E' \subseteq E(H)$ is ρ -*stable* if E' is a minimal weight cover for $\bigcup E'$.

A set of sets S_1, \dots can naturally be interpreted as a hypergraph, by considering each set S_i as an edge. In that light, it is clear that deciding $\rho(H) \leq k$ is precisely the same as deciding whether a set system admits a set cover of size k .

► **Proposition 2** ([4]). *There is an FPT algorithm parameterised by $k + c + d$ that decides for a given (c, d) -hypergraphs H and $k \geq 1$ whether $\rho(H) \leq k$.*

A *tree decomposition* (TD) of hypergraph H is a pair (T, \mathbf{B}) where T is a tree and $\mathbf{B} : V(T) \rightarrow 2^{V(H)}$ labels each node of T with its so-called *bag*, such that the following hold:

- (i) for each $e \in E(H)$, there is a $u \in V(T)$ such that $e \subseteq \mathbf{B}(u)$, and
- (ii) for each $v \in V(H)$, the set $\{u \in V(T) \mid v \in \mathbf{B}(u)\}$ induces a non-empty subtree of T .

We refer to the first property as the *containment property* and to the second as the *connectedness condition*. For a set $U \subseteq V(T)$ we use $\mathbf{B}(U)$ as a shorthand for $\bigcup_{u \in U} \mathbf{B}(u)$, i.e., all the vertices that occur in bags of nodes in U . Similarly, for subtree T' of T , we sometimes use $\mathbf{B}(T')$ instead of $\mathbf{B}(V(T'))$. The *generalised hypertree width* (ghw) of a tree decomposition is $\max_{u \in V(T)} \rho(\mathbf{B}(u))$, and the generalised hypertree width of H (we write $ghw(H)$) is the minimal ghw over all tree decompositions for H . It will be important to remember at various points that ghw is monotone under taking induced subhypergraphs, i.e., $ghw(H[U]) \leq ghw(H)$ for all $U \subseteq V(H)$.

3 High-level Overview of Algorithm

The algorithm for $4\alpha(k, d)$ -APPROXGHW is based on a standard approach for treewidth computation [22]. However, this approach is in fact applied for a tree decomposition *compression* rather than approximation from scratch. In other words, the actual problem being solved is the following one.

COMPRESS

<i>Input</i>	A $(2, d)$ -hypergraph H , integer k , a TD (T, \mathbf{B}) of H with $ghw \leq 4\alpha(k, d) + 1$, $W \subseteq V(H)$ with $\rho(W) \leq 3\alpha(k, d)$
<i>Parameters</i>	k and d
<i>Output</i>	A tree decomposition (T^*, \mathbf{B}^*) of H with ghw at most $4\alpha(k, d)$ such that $W \subseteq \mathbf{B}^*(u)$ for some $u \in V(T^*)$, or Reject , in which case $ghw(H) > k$.

► **Theorem 3.** *COMPRESS is fixed-parameter tractable.*

One natural question is how the COMPRESS being FPT implies $4\alpha(k, d)$ -APPROXGHW. This is done through the use of *iterative compression*, a well known methodology for the design of FPT algorithms. The resulting algorithm for Theorem 1 is presented in Algorithm 1. In particular, we let $V(H) = \{v_1, \dots, v_n\}$, set $V_i = \{v_1, \dots, v_i\}$ and $H_i = H[V_i]$ and solve the $4\alpha(k, d)$ -APPROXGHW for graphs H_1, \dots, H_n . If some intermediate H_i is rejected, the whole H can be rejected. Otherwise, the application to H_i results in a tree decomposition (T_i, \mathbf{B}_i) of ghw at most $4\alpha(k, d)$. Add v_{i+1} to each bag of (T_i, \mathbf{B}_i) . If the ghw of the resulting decomposition is still at most $4\alpha(k, d)$ simply move on to the next iteration. Otherwise, apply the algorithm for COMPRESS rejecting if the algorithm rejects and moving to the next iteration if a compressed tree decomposition is returned.

Let us turn our attention to the algorithm for COMPRESS. A central ingredient of the algorithm [22] considers a set S of size $O(k)$ goes through all partitions of S into two balanced subsets and for each such a partition checks existence of a small separator. However, in our setting the set S can contain arbitrarily many vertices, as long as it can be covered by a bounded number of hyperedges. The following statement provides us with an appropriate variant of the classic result for treewidth that is applicable to our setting.

■ **Algorithm 1** An FPT algorithm for $4\alpha(k, d)$ -APPROXGHW.

Input : $(2, d)$ -hypergraph H , positive integer k

- 1 $\{v_1, \dots, v_n\} \leftarrow V(H)$
- 2 Let T_1 be the tree with a single node r
- 3 Let \mathbf{B}_1 be the function $r \mapsto \{v_1\}$
- 4 **for** $1 < i \leq n$ **do**
- 5 $V_i \leftarrow \{v_1, \dots, v_i\}$
- 6 $H_i \leftarrow H[V_i]$
- 7 $T_i \leftarrow T_{i-1}$
- 8 $\mathbf{B}_i \leftarrow \{t \mapsto \mathbf{B}_{i-1}(t) \cup \{v_i\} \mid t \in T_i\}$
- 9 **if** $ghw((T_i, \mathbf{B}_i)) > 4\alpha(k, d)$ **then**
- 10 $X \leftarrow \text{Compress}(H_i, k, (T_i, \mathbf{B}_i), \emptyset)$
- 11 **if** X is **Reject** **then**
- 12 **return** **Reject**
- 13 $(T_i, \mathbf{B}_i) \leftarrow X$
- 14 **return** (T_n, \mathbf{B}_n)

► **Theorem 4.** Let H be a hypergraph with $ghw(H) \leq k$ and let $E' \subseteq E(H)$. Then there exists a weak partition of E' into three sets E'_0, E'_1, E'_2 such that

1. there is a $(\bigcup E'_1, \bigcup E'_2)$ -separator S such that $\bigcup E'_0 \subseteq S$ and $\rho(S) \leq k$,
2. $|E'_1| \leq \frac{2}{3}|E'|$, and $|E'_2| \leq \frac{2}{3}|E'|$.

Theorem 4 allows us to consider all partitions of a small set of hyperedges covering the given potentially large set of vertices thus guaranteeing an FPT upper bound for the number of such partitions.

The other obstacle in upgrading the result [22] is that a balanced separator is no longer required to be small but rather to have a small edge cover number. In order to compute such a separator we will need a witnessing tree decomposition of H of a small ghw . This is also the reason why we employ iterative compression rather than providing a direct algorithm for approximation. However, even in presence of the tree decomposition, we were still unable to design a 'neat' algorithm that would either produce an (approximately) small separator or reject, implying that a small separator does not exist. Instead, we propose an algorithm for the following problem with a *nuanced* reject that is still suitable for our purposes.

APPROXSEP

Input A $(2, d)$ -hypergraph H , sets $A_1, A_2 \subseteq V(H)$,
 TD (T, \mathbf{B}) of H with ghw p , integers $0 \leq k_0 \leq k \leq p$

Parameters p and d

Output An (A_1, A_2) -separator with edge cover number
 at most $(3k + d + 1)(2k - 1)k_0$,
 or **Reject**, in which case there either exists no (A_1, A_2) -separator with
 edge cover number at most k_0 or $ghw(H) > k$.

► **Theorem 5.** APPROXSEP is fixed-parameter tractable.

We postpone to the next section a more detailed consideration of the algorithm for APPROXSEP. In the rest of this section we discuss the criterion for large GHW used by the algorithm and the context in which the criterion is checked. For this purpose, we will require

some technical definitions. For hypergraph H , let us call $U \subseteq V(H)$ a *subedge* (of H) if there is $e \in E(H)$ such that $U \subseteq e$. We say that two subedges U_1, U_2 are *incompatible* if their union $U_1 \cup U_2$ is not a subedge.

► **Definition 6.** An (a, b) subedge hypergrid (or (a, b) -shyg) consists of pairwise incompatible subedges $U_1, \dots, U_a, S_1, \dots, S_b$ such that:

1. U_1, \dots, U_a are pairwise disjoint, and
2. for each $j \in [b]$, S_j touches U_1, \dots, U_a .

Throughout this paper we will be interested in a specific dimension of subedge hypergrid. Namely for deciding width k in $(2, d)$ -hypergraphs, we will be interested in the existence of $(3k + d + 1, \xi(k, d))$ subedge hypergrids, where ξ is a function in $O((kd)^d)$ (refer to Section 5 for details). We will denote the set of all subedge grids of H of this dimension by $\mathbf{S}_{k,d}(H)$. The reason we care about these subedge hypergrids in particular is that their existence is a sufficient condition for high *ghw*.

► **Theorem 7.** Let H be a $(2, d)$ -hypergraph such that $\mathbf{S}_{k,d}(H) \neq \emptyset$. Then $\text{ghw}(H) > k$.

In fact, the algorithm for the APPROXSEP problem either constructs a required separator or discovers that $\mathbf{S}_{k,d}(H) \neq \emptyset$. More precisely, the identification of an element of $\mathbf{S}_{k,d}(H)$ takes place within the procedure described in Theorem 9 below. The central part the algorithm for APPROXSEP is to pick a vertex $t \in T$ (recall that (T, \mathbf{B}) is the input tree decomposition for H) and then guess a set $W \subseteq \mathbf{B}(t)$ that shall be part of the separator being constructed. Technically, the guessing means a loop exploring a family of subsets of $\mathbf{B}(t)$. This family must be of an FPT size and the edge cover of each element of the family must not be too large compared to k_0 . This idea is formalised in the notion of *gap cover approximator* formally defined below.

► **Definition 8.** For hypergraph H and set $U \subseteq V(H)$ a (β, γ) -gap cover approximator (for U) is a set $\mathbf{X} \subseteq 2^{V(H)}$ such that

- (i) For each $X \in \mathbf{X}$, $\rho(X) \leq \beta$.
- (ii) For each $U' \subseteq U$ with $\rho(U') \leq \gamma$, there is a $X \in \mathbf{X}$ such that $U' \subseteq X$.

In order to produce the desired gap cover approximator, the algorithm solving the APPROXSEP problem runs a function *GapCoverApprox*. The function computes either a gap cover approximator or an element of $\mathbf{S}_{k,d}(H)$ and, in the latter case, rejects. In particular, when the algorithm rejects, we know (implicitly) that $\mathbf{S}_{k,d}(H) \neq \emptyset$, which in turn guarantees that *ghw* is greater than k in this case and the rejection can be propagated to the top-level. A formal description of the behaviour of *GapCoverApprox* is provided below.

► **Theorem 9.** There is an algorithm *GapCoverApprox* (H, U, p, k, k_0) whose input is a $(2, d)$ -hypergraph H , $U \subseteq V(H)$ with $\rho(U) \leq p$, and integers $k_0 \leq k \leq p$. The algorithm returns a $((3k + d + 1)k_0, k_0)$ -gap cover approximator of U or **Reject**, in which case it is guaranteed that $\text{ghw}(H) > k$. The algorithm is in FPT when parameterised in p and d .

4 Algorithmic Details

In this section we sketch proofs of Theorems 3 and 5. In particular, we provide pseudocodes of the corresponding algorithms and intuitive justification of their correctness and FPT membership. Algorithm 2 uses as a subroutine the algorithm *AppSep*, which is discussed afterwards in Section 4.2.

4.1 An FPT Algorithm for the Compression Step (Theorem 3)

To prove Theorem 3 we define Algorithm 2, prove that it is correct, and that the algorithm works FPT time. In general principle the algorithm follows similar ideas to previous algorithms for checking ghw (e.g., [5]) in that, at each stage, we separate the problem into subproblems for each connected component, and recurse. The set W provides an interface to how the subproblem connects to the rest of the decomposition. By guaranteeing that W is covered in the root of the decomposition for the subproblem (line 8), we guarantee that the decompositions for all the subproblems can be assembled into a decomposition for the parent call in lines 8 to 22. (see also [10] where a similar idea is formalised in terms of *extended hypergraphs* in the context of checking plain hypertree width).

We move on to giving an overview of the argument for the runtime and correctness of the algorithm. For the overall time complexity of the algorithm, we first observe that a single recursive application of the algorithm runs in FPT time. The search for E_W in line 1 is FPT by using Proposition 2 to find a cover for W (the procedure from the proposition is constructive). If the produced cover is smaller than $3\alpha(k, d) + 1$ we can incrementally increase the size of the cover by searching for covers for $W' \supseteq W$ created by adding a vertex outside of the cover to W . For line 2 we can naively iterate through all possible partitions of E_W into three sets and call *AppSep*, which itself is in FPT by Theorem 5. For line 7 we note again that testing ρ is FPT by Proposition 2. From line 8 onward, except for the recursion, the algorithm performs straightforward manipulations of sets and hypergraphs that are of no deeper interest to our time bound.

Next, we observe that the number of recursive applications is, in fact, polynomial in H . We naturally organise recursive applications into a successors (recursion) tree and upper bound the number of nodes of the tree by the product of the height of the tree and the number of leaves. We observe that the height of the tree is at most $|V(H)|$. For this we prove two auxiliary statements. The first is that for X , as computed in line 6, $H \setminus X$ has at least two connected components. The second, immediately following from the first one is that for each H_i , created in line 10, $|V(H_i)| < |V(H)|$. Thus it follows that the number of vertices of the input hypergraph decreases as we go down the recursion tree thus implying the upper bound on the height of the tree.

Additionally, we prove that the number of leaves is no larger than $\rho(H)^2$. The main part of this proof is an induction for the case where the number of sets U_1, \dots, U_q obtained at line 7 is at least 2. In particular, we notice that since $\rho(U_i) \geq 3\rho(X)$ for each $i \in [q]$ and since $\rho(\bigcup_{i=1}^q U_i) = \sum_{i=1}^q \rho(U_i)$, it holds that

$$\rho(U_i \cup X)^2 \leq (\rho(U_i) + \rho(X))^2 \leq \left(\sum_{i=1}^q \rho(U_i)\right)^2 = \rho\left(\bigcup_{i=1}^q U_i\right)^2 \leq \rho(H)^2.$$

To prove correctness of the **Reject** output, we observe that return of **Reject** by the whole algorithm is triggered by return of **Reject** on line 4, failure to find an appropriate weak partition, or by rejection in one of its recursive applications. By Theorem 4 and the ρ -stability of E' , the **Reject** on line 4 implies that either H , or one of its induced subgraphs have ghw greater than k . In the latter case, of course also $ghw(H) > k$.

Finally, the two main aspects of correctness of the non-rejection output are the upper bound on the ghw of the resulting tree decomposition and that the properties of the tree decomposition are not lost by the 'gluing' procedure as specified in lines 8-22 of the algorithm. The requirement that E_W must cover W is essential for ensuring that the properties of the tree decomposition are not destroyed by the gluing. Intuitively, the parameter W_i in the recursion on Line 13 represents the connection of the component H_i with the rest of the decomposition.

■ **Algorithm 2** The algorithm $Compress(H, k, (T, \mathbf{B}), W)$.

Input : $(2, d)$ -hypergraph H , a positive integer k , a TD (T, \mathbf{B}) of H with ghw
 $4\alpha(k, d) + 1$, $W \subseteq V(H)$ with $\rho(W) \leq 3\alpha(k, d)$

- 1 $E_W \leftarrow$ any ρ -stable subset of $E(H)$ covering W of cardinality $3\alpha(k, d) + 1$
- 2 Find a weak partition E_0, E_1, E_2 of E_W s.t. $\rho(E_0) \leq k$, $\rho(E_1), \rho(E_2) \leq 2\alpha(k, d)$, and
 $AppSep(H, \bigcup E_1, \bigcup E_2, (T, \mathbf{B}), k, k, p, V(T))$ does not return **Reject**
- 3 **if** no such weak partition exists **then**
- 4 | **return Reject**
- 5 **else**
- 6 | $X \leftarrow AppSep(H, \bigcup E_1, \bigcup E_2, (T, \mathbf{B}), k, k, p, V(T))$
- 7 $U_1, \dots, U_q \leftarrow \{C \in CComps(H \setminus X) \mid \rho(C \cup X) > 4\alpha(k, d)\}$
- 8 Let T^* be a tree with a new node r and $B^*(r) = W \cup X$
- 9 **for** $i \in [q]$ **do**
- 10 | $H_i \leftarrow H[U_i \cup X]$
- 11 | $W_i \leftarrow (\bigcup E_W \cap U_i) \cup X$
- 12 | $\mathbf{B}_i \leftarrow \{t \mapsto \mathbf{B}(t) \cap (U_i \cup X) \mid t \in V(T)\}$
- 13 | $O_i \leftarrow Compress(H_i, k, (T, \mathbf{B}_i), W_i)$
- 14 | **if** O_i is **Reject** **then**
- 15 | | **return Reject**
- 16 | **else**
- 17 | | $(T'_i, \mathbf{B}'_i) \leftarrow O_i$
- 18 | | $t_i \leftarrow$ a node u of T'_i s.t. $W_i \subseteq \mathbf{B}'_i(u)$
- 19 | | Add T'_i to T^* by making t_i a neighbour of r and let $\mathbf{B}^*(u) = \mathbf{B}'_i(u)$ for all
| | $u \in T'_i$.
- 20 **for** $U \in CComps(H \setminus X)$ where $\rho(U \cup X) \leq 4\alpha(k, d)$ **do**
- 21 | Add new node u as a neighbour of r to T^* .
- 22 | Set $\mathbf{B}^*(u) = U \cup X$.
- 23 **return** (T^*, \mathbf{B}^*)

4.2 Finding Approximate Separators in FPT (Theorem 5)

The proof of Theorem 5 requires significant extension of notation. First, for a tree decomposition (T, \mathbf{B}) of a hypergraph and $X \subseteq V(T)$, we denote by $ghw(T, \mathbf{B}, X)$ the maximum of $\rho(\mathbf{B}(t))$ among $t \in X$. We are looking for a separator subject to several constraints. Repeating these constraints every time we refer to a separator is somewhat distracting and we therefore define the set of separators that we need to consider for this overview. Define $sep(H, A, B, k_0, (T, \mathbf{B}), X)$ as the set of all (A, B) -separators W of H with $\rho(W) \leq k_0$ and $W \subseteq \mathbf{B}(X)$ where (T, \mathbf{B}) is a tree decomposition of H and $X \subseteq V(T)$.

The following theorem is a generalisation of Theorem 5.

► **Theorem 10.** *There is an algorithm $AppSep(H, A, B, (T, \mathbf{B}), k_0, k, p, X)$ whose input is a $(2, d)$ -hypergraph H , $A, B \subseteq V(H)$, three positive integers $k_0 \leq k \leq p$, a tree decomposition (T, \mathbf{B}) of H and $X \subseteq V(T)$ such that all the elements of $V(T) \setminus X$ are leaves and $ghw(T, \mathbf{B}, X) \leq p$. The algorithm either returns an element of $sep(H, A, B, (3k + d + 1)(2k - 1)k_0, (T, \mathbf{B}), X)$ or **Reject**. In the latter case, it is guaranteed that either $ghw(H) > k$ or $sep(H, A, B, k_0, (T, \mathbf{B}), X) = \emptyset$*

Clearly, Theorem 10 implies Theorem 5 by setting $X = V(T)$. The reason we need this extra parameter is that in the recursive applications of *AppSep* some bags may have the edge cover number larger than p , so we keep track of the set of nodes whose bags are 'small'.

To present the pseudocode, we define a specific choice of a subtree of the given tree. Let T be a tree, $t \in V(T)$, $Y \subset V(T)$. Then $T_{t,Y}$ is the subtree of T which is the union of all paths starting from t whose second vertex belongs to Y . The notion of $T_{t,Y}$ naturally extends to subsets of $V(T)$ and to tree decompositions where T is the underlying tree. In particular, for $X \subseteq V(T)$, we denote $X \cap V(T_{t,Y})$ by $X_{t,Y}$. Next, if (T, \mathbf{B}) is a tree decomposition of H then by denote by $\mathbf{B}_{t,Y}$ the restriction to \mathbf{B} to $V(T_{t,Y})$ and by $H_{t,Y}$ the graph $H[\mathbf{B}(V(T_{t,Y}))]$.

We also introduce a variant $T_{t,Y}^+$ of $T_{t,Y}$ which will be needed for recursive applications of *AppSep*. The tree $T_{t,Y}^+$ is obtained from $T_{t,Y}$ by introducing a new node r and making it adjacent to t . The function $\mathbf{B}_{t,Y}^+$ is obtained from $\mathbf{B}_{t,Y}$ by setting $\mathbf{B}_{t,Y}^+(r) = \bigcup_{t' \in V(T) \setminus V(T_{t,Y})} \mathbf{B}(t')$. One final notational convention concerns adjusting a tree decomposition (T, \mathbf{B}) of H in case a set $W \subseteq V(H)$ is removed from H . In this case we set $\mathbf{B}^{-W}(t') = \mathbf{B}(t') \setminus W$ for each $t' \in V(T)$.

The following statement is important for verifying that these recursive applications are well-formed.

► **Theorem 11.** *Let H be a hypergraph, (T, \mathbf{B}) a TD of H , $t \in V(T)$, $Y \subseteq N_T(t)$. Then $(T_{t,Y}, \mathbf{B}_{t,Y})$ is a TD of $H_{t,Y}$, $(T_{t,Y}^+, \mathbf{B}_{t,Y}^+)$ is a TD of H and (T, \mathbf{B}^{-W}) is a TD of $H \setminus W$. Moreover, let $X \subseteq V(T)$ such that all the vertices of $V(T) \setminus X$ are leaves of T . Then all the vertices of $V(T_{t,Y}) \setminus X_{t,Y}$ are leaves of $T_{t,Y}$.*

The algorithm (roughly speaking) chooses a vertex $t \in V(T)$, partitions $N(t)$ into Y_1 and Y_2 and applies recursively to H_{t,Y_1} and H_{t,Y_2} . However, the triple (t, Y_1, Y_2) is chosen not arbitrarily but in a way that both X_{t,Y_1} and X_{t,Y_2} are significantly smaller than X . The possibility of such a choice is guaranteed by the following theorem.

► **Theorem 12.** *Let T be a tree, $X \subseteq V(T)$ such that all $|X| \geq 3$ and all the vertices of $V(T) \setminus X$ are leaves. Then there is $t \in X$ with $\deg_{T[X]}(t) \geq 2$ and a partition Y_1, Y_2 on $N_T(t)$ so that for each $i \in \{1, 2\}$, $|X_{t,Y_i}| \leq 3/4|X|$. Moreover, the triple (t, Y_1, Y_2) can be computed in a polynomial time.*

Theorem 12 can be seen as a variant of a classical statement that a rooted tree has a descendant rooting a subtree with the number of leaves between one third to two third of the total number of leaves. The proof is based on a similar argument of picking a root and gradually descending towards a 'large' subtree until the desired triple is found.

For the validity of *AppSep*, it is important to note that each X_{t,Y_i} preserves for T_{t,Y_i} the invariant that all the $V(T_{t,Y_i}) \setminus X_{t,Y_i}$ are leaves of T_{t,Y_i} . We are almost ready to consider the pseudocode, it only remains to identify auxiliary functions. In particular *GetBalVert*(T, X) is a polynomial time algorithm as specified in Theorem 12. Also recall that *GapCoverApprox* is an FPT algorithm constructing a $((3k + d + 1)k_0, k_0)$ -gap cover approximator in the way specified by Theorem 9.

The pseudocode of *AppSep* is presented in Algorithm 1. For the sake of readability, we make two notational conventions. First, since parameters p and k do not change when passed through recursive calls, we consider them fixed and do not mention them as part of the input when recursing. Second, we move consideration of the case with $|X| \leq 2$ into a separate function *SmallSep* provided in Algorithm 3 and use it as an auxiliary function in Algorithm 4. Here the idea is straightforward, we naively test for all gap cover approximators whether they are (A, B) -separators.

■ **Algorithm 3** The algorithm $SmallSep(H, A, B, p, k, k_0, (T, \mathbf{B}), X)$.

Input : $(2, d)$ -hypergraph H , $A, B \subseteq V(H)$, positive integers $k_0 \leq k \leq p$, a TD (T, \mathbf{B}) of H , $X \subseteq V(T)$, $|X| \leq 2$ all the elements of $V(T) \setminus X$ are leaves

```

1 if  $|X| = 1$  then
2    $\{t\} \leftarrow X$ 
3    $Sets \leftarrow GapCoverApprox(H, \mathbf{B}(t), p, k, k_0)$ 
4 else
5    $\{t_1, t_2\} \leftarrow X$ 
6    $Sets \leftarrow GapCoverApprox(H, \mathbf{B}(t_1) \cup \mathbf{B}(t_2), p, k, k_0)$ 
7 if  $Sets$  is Reject then
8   return Reject
9 for  $W \in Sets$  do
10  if  $W$  is an  $(A, B)$ -separator then
11  return  $U$ 
12 return Reject

```

The general case in $AppSep$ is considerably more complex. Intuitively, the algorithm searches for separators in small local parts of the tree decomposition by searching through the output of $GapCoverApprox$ called on the component of that local part of the decomposition (Lines 1 and 2 of the algorithm). In general this is of course not sufficient to find (A, B) -separators. What we do instead is to try and find partial separators that ultimately combine into a single (A, B) -separator. To that end we split the tree decomposition into two decompositions in a balanced fashion (Line 3). The two cases handled from Lines 4 to 9 cover the special case where the search is propagated to one part of the tree decomposition, while the body of the loop at Line 13 considers (roughly speaking) all possibilities of splitting up the separator over both parts.

The first step of proving Theorem 10 is to prove the FPT runtime of $AppSep$. We first observe that a single application of $AppSep$ takes FPT time. The efficiency of $GetBalVert$ and $GapCoverApprox$ has been discussed above. $SmallSep$ is effectively a loop over the output of $GapCoverApprox$ with polynomial time spent per element. Note that since $GapCoverApprox$ is computed in FPT time we also obtain a corresponding bound on the size of the $((3k + d + 1)k_0, k_0)$ -gap cover approximator to iterate over. Finally, in the loop of Line 13, we only need to observe that the number of connected components of $\mathbf{B}(t) \setminus W$ is at most p as otherwise the edge cover number of $\mathbf{B}(t)$ is greater than p . Hence, the number of partitions C_1, C_2 considered in the loop is $O(2^p)$.

Next, we need to demonstrate that the number of recursive applications of $AppSep$ is FPT. We present the number of applications as a recursive function $F(k_0, m)$ where $m = |X|$. If $m \leq 2$ then there is only a single application through running $SmallSep$. Otherwise, there is one recursive application at Line 4 and one at Line 7 where the first parameter remains the same and the second parameter is at most $3/4m$ (by selection of t, Y_1, Y_2). Additionally, there are also the recursive applications in Lines 14 and 15 where the first parameter is at most $k_0 - 1$ and the second parameter is at most $3/4m$. As a result, we obtain a recursive formula $F(k_0, m) \leq 2F(k_0, 3/4m) + g(p)F(k_0 - 1, 3/4m)$. We note that this function can be bounded above by a fixed-parameter cubic function (see Lemma 43 in the full version appendix for details) thus establishing the FPT runtime of $AppSep$.

■ **Algorithm 4** The algorithm $AppSep(H, A, B, (T, \mathbf{B}), k_0, k, p, X)$.

Input : $(2, d)$ -hypergraph H , $A, B \subseteq V(H)$, a TD (T, \mathbf{B}) of H , positive integers $0 \leq k_0 \leq k \leq p$, $X \subseteq V(T)$ s.t. all the elements of $V(T) \setminus X$ are leaves

- 1 **if** $|X| \leq 2$ **then**
- 2 **return** $SmallSep(H, A, B, p, k, k_0, (T, \mathbf{B}), X)$
- 3 $(t, Y_1, Y_2) \leftarrow GetBalVert(T, X)$
- 4 $Out \leftarrow AppSep(H, A, B, (T_{t, Y_1}^+, \mathbf{B}_{t, Y_1}^+), k_0, X_{t, Y_1})$
- 5 **if** Out is not **Reject** **then**
- 6 **return** Out
- 7 $Out \leftarrow AppSep(H, A, B, (T_{t, Y_2}^+, \mathbf{B}_{t, Y_2}^+), k_0, X_{t, Y_2})$
- 8 **if** Out is not **Reject** **then**
- 9 **return** Out
- 10 $Sets \leftarrow GapCoverApprox(H, \mathbf{B}(t), p, k, k_0)$
- 11 **if** $Sets$ is **Reject** **then**
- 12 **return** **Reject**
- 13 **for** each $W \in Sets$, each $k_1, k_2 > 0$ s.t. $k_1 + k_2 \leq k_0$, and each weak partition C_1, C_2 of $\mathbf{B}(t) \setminus W$ into unions of connected components of $H[\mathbf{B}(t) \setminus W]$ **do**
- 14 $Out_1 \leftarrow AppSep(H_{t, Y_1} \setminus W, (A_{t, Y_1} \cup C_1) \setminus W, (B_{t, Y_1} \cup C_1) \setminus W, (T_{t, Y_1}, \mathbf{B}_{t, Y_1}^{-W}), k_1, X_{t, Y_1})$
- 15 $Out_2 \leftarrow AppSep(H_{t, Y_2} \setminus W, (A_{t, Y_2} \cup C_2) \setminus W, (B_{t, Y_2} \cup C_2) \setminus W, (T_{t, Y_2}, \mathbf{B}_{t, Y_2}^{-W}), k_2, X_{t, Y_2})$
- 16 **if** neither of Out_1, Out_2 is **Reject** **then**
- 17 **return** $Out_1 \cup Out_2 \cup W$
- 18 **return** **Reject**

Next, we need to demonstrate correctness of the non-rejection output. It is straightforward to see by construction that the returned set S is a subset of $\mathbf{B}(X)$: ultimately, the set S is comprised of unions of outputs of $GapCoverApprox(H, U, \dots)$, either directly in Line 10, or indirectly via $SmallSep$. In the first case, the returned sets are a subset of $\mathbf{B}(t)$, where $t \in X$ by definition of $GetBalVert$. In the second case, $U \subseteq \mathbf{B}(X_{t, Y_i})$ for $i = 1$ or $i = 2$ by definition of $SmallSep$. By definition, both X_{t, Y_i} are subsets of X and thus $\mathbf{B}(X_{t, Y_i})$ is a subset of $\mathbf{B}(X)$. Since the recursion always restricts parameter X to either X_{t, Y_1} or X_{t, Y_2} the inductive application of this observation is immediate.

We need to show S is an (A, B) -separator and that its edge cover number is within a specified upper bound. Both claims are established by induction. The main part of proving that S is an (A, B) -separator is showing that if S as returned on Line 17 then it is an (A, B) -separator. This follows from the induction assumption applied to Out_1 and Out_2 and the following statement.

► **Lemma 13.** *Let H be a hypergraph, $V_1, V_2 \subseteq V(H)$ be such that $V_1 \cup V_2 = V(H)$ and $Y = V_1 \cap V_2$ is a (V_1, V_2) -separator. Let $W \subseteq Y$ and let C_1, C_2 be a weak partition of $Y \setminus W$. Let $H_1 = H[V_1 \setminus W]$ and $H_2 = H[V_2 \setminus W]$. Let $A, B \subseteq V(H)$. For each $i \in \{1, 2\}$ let $A_i = (A \cap V(H_i)) \cup C_1$, let $B_i = (B \cap V(H_i)) \cup C_2$, and let W_i be an (A_i, B_i) -separator of H_i . Then $W_1 \cup W_2 \cup W$ is an (A, B) -separator of H .*

To prove that the size of the output S of *AppSep* matches the required upper bound, we observe that the output is the union of several sets S_1, \dots, S_q each of which is in a family returned by an application of *GapCoverApprox*. This guarantees that $\rho(S_i) \leq (3k + d + 1)k_0$. It only remains to show that $q \leq 2k_0 - 1$ (q is the number of sets S_1, \dots, S_q whose union make up S). To this end we observe that the recursive applications invoking *GapCoverApprox* can be naturally organised into a recursion tree where each node has two children, accounting for the recursive applications in Lines 14 and 15 (the applications on Lines 4 and 7 are not relevant since there is no invocation of *GapCoverApprox* associated with them). Then q is simply the number of nodes of the tree. By a simple induction we observe that if k_1 and k_2 are the numbers as obtained in Line 13 then $k_1 + k_2 \leq k_0$ and the number of nodes rooted by children of the tree is at most $2k_1 - 1$ and $2k_2 - 1$, respectively. Hence, the total number of nodes, accounting for the root, is at most $(2k_1 - 1) + (2k_2 - 1) + 1 \leq 2(k_1 + k_2) - 1 \leq 2k_0 - 1$ as required. For the correctness of the **Reject** output we first recursively define the **Reject** triggered by *GapCoverApprox*. This happens when *AppSep* runs *SmallSep* and the latter returns **Reject** in Lines 5 or 10 of Algorithm 3, or **Reject** is returned in Line 12 of Algorithm 4, or when **Reject** is returned in Line 18 of Algorithm 4 and one of recursive applications leading to this output returns **Reject** triggered by *GapCoverApprox*. By inductive application of Theorem 9 we observe that **Reject** triggered by *GapCoverApprox* implies that the *ghw* of some induced subgraph of H (and hence of H itself) is greater than k .

Finally, we demonstrate that if the **Reject** output is not triggered by *GapCoverApprox* then $\text{sep}(H, A, B, k_0, (T, \mathbf{B}), X) = \emptyset$. First, we assume that $|X| \leq 2$ and demonstrate this for Algorithm 3. It follows from the description that, in the considered case, no element of *Sets* is an (A, B) separator. As each $W' \subseteq \mathbf{B}(X)$ with $\rho(W') \leq k_0$ is a subset of some element of *Sets*, it follows that no such W' is an (A, B) -separator of H . In the case where $|X| \geq 3$, a **Reject** not inherited from *GapCoverApprox* can only be returned in Line 18 of Algorithm 1. This, in particular, requires **Reject** to be returned by recursive applications in Lines 4 or 7. By the induction assumption, $\text{sep}(H, A, B, k_0, (T_{t,Y_i}^+, \mathbf{B}_{t,Y_i}^+), X_{t,Y_i}) = \emptyset$ for each $i \in \{1, 2\}$. This means that if there is $W^* \in \text{sep}(H, A, B, k_0, (T, \mathbf{B}), X)$ then W^* is *not* a subset of $\mathbf{B}(X_{t,Y_1})$ nor of $\mathbf{B}(X_{t,Y_2})$. We conclude that by the induction assumption, such a W^* would cause a non-rejection output in one of iterations of the loop in Line 13. Since the algorithm passes through to Line 18, such an iteration does not happen so we conclude that such a W^* does not exist.

5 Combinatorial Statements

In this section we prove Theorem 4 and sketch the proofs for Theorems 7 and 9. While Theorem 9 also refers to the existence of an algorithm, we consider the nature of the theorem to be purely combinatorial. The resulting algorithm is simply a naive enumeration of all possibilities of combining certain sets.

5.1 Theorem 4

For Theorem 4 we can make use of a result from the literature and prove the statement in full here. We first recall key terminology from Adler et al. [1], who proved the result that we will use. For a set $E' \subseteq E(H)$, and $C \subseteq V(H)$ define $\text{ext}(C, E') := \{e \in E' \mid e \cap C \neq \emptyset\}$. We say that C is E' -big if $|\text{ext}(C, E')| > \frac{|E'|}{2}$. A set $E' \subseteq E(H)$ is k -hyperlinked if for every set $S \subseteq E(H)$ with $|S| < k$, $H \setminus \bigcup S$ has an E' -big connected component. The hyperlinkedness $\text{hlink}(H)$ of H is the maximal k such that H contains a k -hyperlinked set. From another perspective, if $\text{hlink}(H) \leq k$, then for any set $E' \subseteq E(H)$, there is an $S \subseteq E(H)$ with $|S| \leq k$ such that no connected component of $H \setminus \bigcup S$ is E' -big. Adler et al. [1] showed the following.

► **Proposition 14** ([1]). *For every hypergraph H , $hlink(H) \leq ghw(H)$.*

Proof of Theorem 4. By assumption and Proposition 14, we have $hlink(H) \leq k$. Then for E' , there is a set of k edges $S \subseteq E(H)$ such that no connected component of $H \setminus S$ is E' -big. Let C_1, \dots, C_ℓ be the connected components of $H \setminus \bigcup S$. First, observe that $ext(C_i, E') \cap ext(C_j, E') = \emptyset$ for any distinct $i, j \in [\ell]$. Suppose, w.l.o.g., that $|ext(C_i, E')| \geq |ext(C_{i+1}, E')|$ for $i \in [\ell - 1]$. Let m be the highest integer such that $\sum_{i=1}^m |ext(C_i, E')| < \frac{2}{3}|E'|$. Such an $m \geq 1$ always exists because no component is E' -big. We claim that $E'_0 = S \cap E'$, $E'_1 = \bigcup_{i=1}^m ext(C_i, E')$, and $E'_2 = \bigcup_{i=m+1}^\ell ext(C_i, E')$ are as desired by the statement.

It is clear that E'_0 satisfies the condition of the lemma (for separator $\bigcup S$). Furthermore, $\bigcup S$ is an $(\bigcup E'_1, \bigcup E'_2)$ -separator as the two sets touch unions of different $H \setminus \bigcup S$ components. The size bound $|E'_1| \leq \frac{2}{3}|E'|$ holds by construction.

What is left to show is that the size bound also holds for E'_2 . To that end we first observe that $|E'_1| \geq \frac{1}{3}|E'|$. Indeed, by the ordering of components by the size of ext , we have that $ext(C_{m+1}, E') \leq |E'_1|$. Thus, $|E'_1| < \frac{1}{3}|E'|$ would contradict the choice of m . Since E'_1 and E'_2 are disjoint, this leaves at most $|E'| - |E'_1| \leq \frac{2}{3}|E'|$ edges for E'_2 . ◀

Proposition 14 already plays an important role in the state of the art of ghw computation. The implication of a so-called *balanced separator* of size k has been a key ingredient for various practical implementations for computing ghw and related parameters [6, 14, 10]. Our application of this idea is somewhat different from this prior work. There, balanced separators are used to reduce the search space of separators that need to be checked, and to split up the problem into small subproblems. Our application of Theorem 4 is different: in Algorithm 2 for COMPRESS we use it to find a way to separate the interface to the parent node in the decomposition. Notably, this search requires the split into only a constant number of sets (rather than the possibly linear number of connected components) which is part of why we require our variation of the previous hyperlinkedness result.

5.2 Large Subedge Hypergrids (Theorem 7)

Recall from Definition 6 that an (a, b) *subedge hypergrid* (or (a, b) -shyg) consists of pairwise incompatible subedges $U_1, \dots, U_a, S_1, \dots, S_b$ such that: U_1, \dots, U_a are pairwise disjoint, and for each $j \in [b]$, S_j touches U_1, \dots, U_a . Our proof of Theorem 7 first relates (a, b) -shygs to more restricted structures that we call *strong* (a, b) -shygs.

► **Definition 15.** *An (a, b) -shyg $U_1, \dots, U_a, S_1, \dots, S_b$ is strong if $S_j \cap S_{j'} \cap \bigcup_{i \in [a]} U_i = \emptyset$ for each $j \neq j' \in [b]$.*

► **Theorem 16.** *Let H be a $(2, d)$ hypergraph having a strong $(3k + 1, (3k + 1)d + 1)$ -shyg. Then $ghw(H) > k$.*

To prove Theorem 7, we first prove Theorem 16 and then demonstrate that non-emptiness of $\mathbf{S}_{k,d}$ implies existence of a strong $(3k + 1, (3k + 1)d + 1)$ -shyg. We continue with an overview of our proof of Theorem 16.

An important observation for subedges in $(2, d)$ -hypergraphs is that if a subedge U is large enough, and in particular if $|U| > d$, then this will uniquely determine the edge e such that $U \subseteq e$. In this section we will refer to this uniquely determined e as $e(U)$. Using this we state the following auxiliary lemma that gives us a lower bound for separating two subedges that are part of a shyg.

► **Lemma 17.** *Let $U_1, U_2, S_1, \dots, S_b$ be a strong $(2, b)$ -shyg of a $(2, d)$ -hypergraph H . Let $\{e_1, \dots, e_q\} \subseteq E(H)$ be such that $\{e(U_1), e(U_2)\} \cap \{e_1, \dots, e_q\} = \emptyset$ and $W = \bigcup_{i \in [q]} e_i$ is a U_1, U_2 -separator. Then $q \geq b/2d$.*

Back to the proof of Theorem 16, we observe that $e(U_1), \dots, e(U_{3k+1})$ is ρ -stable. This is because each $e(U_i)$, to 'incorporate' all S_j must be of size at least $(3k+1)d+1$ and, in a $(2, d)$ -hypergraph, this is too large to be covered by $(3k+1)$ other hyperedges. We will then use Theorem 4 to prove the desired lower bound on $ghw(H)$, by showing that there is the set $\{e(U_1), \dots, e(U_{3k+1})\}$ cannot be separated in the way specified by Theorem 4. Towards a contradiction, we assume existence of a weak partition E'_0, E'_1, E'_2 of $\{e(U_1), \dots, e(U_{3k+1})\}$, $|E'_i| \leq 2k$ for each $i \in \{1, 2\}$ and $W \subseteq V(H)$ such that $\rho(W) \leq k$, $\bigcup E'_0 \subseteq W$ and W is a $\bigcup E'_1, \bigcup E'_2$ -separator. W.l.o.g. we assume existence of $\{e_1, \dots, e_r\} \subseteq E(H)$, $r \leq k$ such that $W = \bigcup_{i \in [r]} e_i$. Let E_0^* be the set of all elements of $e(U_1), \dots, e(U_{3k+1})$ that are subsets of W . We note that $E'_0 \subseteq E_0^*$ and, since no $e(U_i)$ can be covered by k other hyperedges, $E_0^* \subseteq \{e_1, \dots, e_r\}$. We also note that both $E'_1 \setminus E_0^*$ and $E'_2 \setminus E_0^*$ are nonempty. Indeed, if say $E'_2 \setminus E_0^* = \emptyset$ then $3k+1 = |\{e(U_1), \dots, e(U_{3k+1})\}| = |E'_1 \cup E_0^*| \leq 2k+k$, or in other words, in such a situation E'_0, E'_1, E'_2 could not form a weak partition of $3k+1$ edges. Let $e(U_{i_1}) \in E'_1 \setminus E_0^*$ and $e(U_{i_2}) \in E'_2 \setminus E_0^*$. Then W is a (U_{i_1}, U_{i_2}) -separator. By Lemma 17, $r \geq (3k+1)d/2d > k$, and we arrive at a contradiction. This completes the sketch of the proof for Theorem 16.

As a next step, we show that a large enough shyg will imply the existence of a strong $(3k+1, (3k+1)d+1)$ -shyg. We will show this inductively via a graded version of strong shygs that we will call *c-strong shygs*. The only difference from Definition 15 is that $|S_j \cap S_{j'} \cap \bigcup_{i \in [a]} U_i| \leq c$ for each $j \neq j' \in [b]$. Thus a strong shyg is 0-strong one and any ordinary shyg is d -strong by definition of a $(2, d)$ -hypergraph.

Let us recursively define a function $g(c) = g_{k,d}(c)$ as follows. Let $g(0) = (3k+1)d+1$. For $c > 0$, assuming that $g(c-1)$ has been defined, we let $g(c) = g(0)^2(g(c-1)-2) + 1$. Further on, we let $\xi(k, d) = g_{k,d}(d)$ and let $\mathbf{S}_{k,d}(H)$ to be the set of all $(3k+d+1, \xi(k, d))$ -shygs of H . The second part of the proof of Theorem 7 is the following statement.

► **Theorem 18.** *Let $c \geq 0$ and let H be a $(2, d)$ -hypergraph that has a c -strong $(3k+1+c, g(c))$ -shyg. Then H has a strong $(3k+1, (3k+1)d+1)$ -shyg.*

Theorem 7 is immediate from the combination of Theorem 18 with $c = d$ and Theorem 16. So, let us discuss the proof of Theorem 18.

Proof Sketch. The proof is by induction on c . The case $c = 0$ is immediate as the considered shyg is exactly the desired strong shyg. For $c > 0$, we demonstrate we can 'extract' from the considered shyg either a $c-1$ -strong $(3k+c, g(c-1))$ shyg (implying the theorem by the induction assumption) a strong $(3k+1, g(0))$ -shyg exactly as required by the theorem.

So, let $U_1, \dots, U_{3k+c+1}, S_1, \dots, S_{g(c)}$ be the considered c -strong shyg. Assume first that there is $u \in \bigcup_{i \in [3k+c+1]} U_i$ that touches $g(c-1)$ sets S_j . We assume w.l.o.g that $u \in U_{3k+c+1}$ and that the sets S_j touching u are precisely $S_1, \dots, S_{g(c)-1}$. Since one intersection point between these sets is spent on U_{3k+c+1} for any $j \neq j' \in [g(c)-1]$, $|S_j \cap S_{j'} \cap \bigcup_{i \in [3k+c]} U_i| \leq c-1$. In other words, $U_1, \dots, U_{3k+c}, S_1, \dots, S_{g(c)-1}$ is a $(c-1)$ -strong shyg implying the theorem by the induction assumption. It remains to assume that each $u \in \bigcup_{i \in [3k+c+1]} U_i$ touches at most $g(c-1) - 1$ sets S_j . We are going to identify $I \subseteq [g(c)]$ of size $g(0)$ so that for each $j \neq j' \in I$, $S_i \cap S_j \cap \bigcup_{i \in [3k+1]} U_i = \emptyset$. This means that U_1, \dots, U_{3k+1} along with S_j for each $j \in I$ will form a strong $(3k+1, g(0))$ -shyg as required by the theorem.

We use the same elementary argument as if we wanted to show that a graph with many vertices and a small max-degree contains a large independent set. The initial set CI of candidate indices is $[g(c)]$ and initially $I = \emptyset$. We choose $j \in CI$ into I and remove from CI the j and all the j' such that $S_j \cap S_{j'} \cap \bigcup_{i \in [3k+1]} U_i \neq \emptyset$. By definition of $g(c)$, it is enough to show that, apart from j itself, we remove at most $(g(0) - 1)(g(c - 1) - 2)$ other elements. Indeed, the size of $S_j^* = S_j \cap \bigcup_{i \in [3k+1]} U_i$ is at most $(3k + 1)d = g(0) - 1$ and each point of S_i^* , apart from S_j touches at most $g(c - 1) - 2$ elements simply by assumption. \blacktriangleleft

5.3 Constructing Gap Cover Approximators (Theorem 9)

Our overall plan for the proof of Theorem 9 is to show that we can produce the elements that make up the desired gap cover approximator, as a combination of four parts, each of which we can bound appropriately. The resulting algorithm is then primarily a matter of enumerating all combinations of elements from these parts. In the following we discuss the construction of these parts and why this yields an FPT algorithm.

The first part is what we will refer to as the set $\text{BE}_p(U)$ of p -big edges w.r.t. U , which are those edges $e \in E(H[U])$ for which $|e| > pd$. The intuition for the importance of big edges is simple, in a $(2, d)$ -hypergraph, they are necessary to obtain low weight covers. In particular, any edge cover of U with weight at most p must contain all edges of $\text{BE}_p(U)$: if $|e| > pd$ then the vertices in e cannot be covered by less than $p + 1$ other edges, since any other $e' \neq e$ will only intersect e in at most d vertices (see Appendix B in the full version for in-depth discussion of $\text{BE}_p(U)$ sets). For the rest of this section we will simply say that edges are big to mean p -big. Similarly, we will refer to all edges that are not p -big as small.

The more challenging part is to determine the structure of those vertices that are not covered by the big hyperedges. To this end we first define the boundary BE_p^* of the big edges:

$$\text{BE}_p^*(U) := \{e \setminus \bigcup (\text{BE}_p(U) \setminus \{e\}) \mid e \in \text{BE}_p(U)\}.$$

That is $\text{BE}_p^*(U)$ contains those subedges of big edges that are unique to a single big edge.

With respect to this set we will be particularly interested in those members that together cover the intersection of a small edge with the vertices in the boundary. We formalise this via the *spanning set* $sp(e)$ for $e \in E(H[U]) \setminus \text{BE}_p(U)$, which is the set $E' \subseteq \text{BE}_p^*(U)$ such that $e \cap \bigcup E' = e \cap \bigcup \text{BE}_p^*(U)$. We will refer to the cardinality of $sp(e)$ as the *span* of e . The final part we need is those vertices U_p^0 that are not part of any subedge in the boundary BE_p^* , formally $U_p^0 = U \setminus \bigcup \text{BE}_p^*(U)$. In addition to the three parts described above, may need to add subsets of U_p^0 to construct the elements of the gap cover approximator. In particular, to add those vertices that are not part of any big edge. The key observation here is that, under the assumption that $\rho(U) \leq p$, there cannot be too many such vertices and specifically $|U_p^0| = O(p^2d)$.

Ultimately, what we prove is that for any $U' \subseteq U$ with $\rho(U') \leq k_0$, there is a set X such that $X \supseteq U'$ and $\rho(X) \leq (3k + 1 + d)k_0$, where X is the union of big edges, short edges and $Y \subseteq U_p^0$. The short edges are actually split in two cases, depending on their span. The construction of X may require some number of short edges with span at most $3k + d$, as well some short edges with span greater than $3k + d$. The last set is the most challenging in terms of achieving an FPT algorithm. All other sets can be bounded in terms of p and d (a small edge with a small span is covered by the union of its span leading to the stated approximation factor). Such a bound seems to not be achievable for the set of small edges with large span. To get around this issue, we show that if there are many small edges with large span, then this implies the existence of a large subedge hypergrid. In more concrete

terms, if there is a set $E' \subseteq \text{BE}_p^*(U)$ with $|E'| > 3k + d$ and $sp^{-1}(E') > \xi(k, d)$ (with ξ as in the definition of $\mathbf{S}_{k,d}(H)$), then $\mathbf{S}_{k,d}(H) \neq \emptyset$. In consequence, we can detect the case for which we could not achieve an FPT bound, and we know that in this case we can safely reject as $ghw(H)$ is guaranteed to be greater than k .

6 Conclusion

We have presented a fixed-parameter tractable algorithm for approximating the generalised hypertree width of hypergraphs with bounded intersection size. In particular, we give an algorithm that either decides in $f(k, d) \text{poly}(H)$ time whether a $(2, d)$ -hypergraph H has $ghw(H) \leq 4k(k + d + 1)(2k - 1)$ or rejects, in the latter case it is guaranteed that $ghw(H) \geq k$.

Our main result represents a first step into the area of FPT algorithms for ghw . Our focus has been on developing the overarching framework, and we expect that with further refinement, better approximation factors are achievable and present a natural avenue for further research. The most immediate question is whether subcubic approximation can be achieved. Recall that the $(3k + d + 1)(2k - 1)k_0$ factor for APPROXSEP comes from two sources, $(3k + d + 1)k_0$ is a result of using $(3k + d + 1)k_0, k_0$ -gap cover approximators to find partial covers. The factor $(2k - 1)$ is a result of combining the partial separators. It is unclear whether either of these factors can be avoided.

For full proof details and an extensive discussion of possible future work we refer to reader to the full version of this paper [17].

References

- 1 Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007. doi:10.1016/j.ejc.2007.04.013.
- 2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 4 Grzegorz Fabianski, Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. Progressive algorithms for domination and independence. In Rolf Niedermeier and Christophe Paul, editors, *STACS 2019*, volume 126 of *LIPICs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.STACS.2019.27.
- 5 Wolfgang Fischl, Georg Gottlob, Davide Mario Longo, and Reinhard Pichler. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. *ACM J. Exp. Algorithmics*, 26:1.6:1–1.6:40, 2021. doi:10.1145/3440015.
- 6 Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In *Proceedings PODS*, pages 17–32. ACM, 2018. doi:10.1145/3196959.3196962.
- 7 Georg Gottlob and Gianluigi Greco. Decomposing combinatorial auctions and set packing problems. *J. ACM*, 60(4):24:1–24:39, 2013. doi:10.1145/2508028.2505987.
- 8 Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Pure nash equilibria: Hard and easy games. *J. Artif. Intell. Res.*, 24:357–406, 2005. doi:10.1613/jair.1683.
- 9 Georg Gottlob, Martin Grohe, Nysret Musliu, Marko Samer, and Francesco Scarcello. Hypertree decompositions: Structure, algorithms, and applications. In *WG*, volume 3787 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2005. doi:10.1007/11604686_1.
- 10 Georg Gottlob, Matthias Lanzinger, Cem Okulmus, and Reinhard Pichler. Fast parallel hypertree decompositions in logarithmic recursion depth. In *Proceedings PODS*, pages 325–336. ACM, 2022. doi:10.1145/3517804.3524153.

- 11 Georg Gottlob, Matthias Lanzinger, Reinhard Pichler, and Igor Razgon. Complexity analysis of generalized and fractional hypertree decompositions. *J. ACM*, 68(5):38:1–38:50, 2021. doi:10.1145/3457374.
- 12 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci.*, 66(4):775–808, 2003. doi:10.1016/S0022-0000(03)00030-8.
- 13 Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM*, 56(6):30:1–30:32, 2009. doi:10.1145/1568318.1568320.
- 14 Georg Gottlob, Cem Okulmus, and Reinhard Pichler. Fast and parallel decomposition of constraint satisfaction problems. *Constraints An Int. J.*, 27(3):284–326, 2022. doi:10.1007/s10601-022-09332-1.
- 15 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014. doi:10.1145/2636918.
- 16 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5):33:1–33:38, 2019. doi:10.1145/3325116.
- 17 Matthias Lanzinger and Igor Razgon. FPT approximation of generalised hypertree width for bounded intersection hypergraphs. *CoRR*, abs/2309.17049, 2023. doi:10.48550/ARXIV.2309.17049.
- 18 Dániel Marx. Approximating fractional hypertree width. *ACM Trans. Algorithms*, 6(2):29:1–29:17, 2010. doi:10.1145/1721837.1721845.
- 19 Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016. doi:10.1145/3003665.3003667.
- 20 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In *Proceedings ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 694–705. Springer, 2009. doi:10.1007/978-3-642-04128-0_62.
- 21 Igor Razgon. FPT algorithms providing constant ratio approximation of hypertree width parameters for hypergraphs of bounded rank. *CoRR*, abs/2212.13423, 2022. doi:10.48550/arXiv.2212.13423.
- 22 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

Sub-Exponential Time Lower Bounds for #VC and #Matching on 3-Regular Graphs

Ying Liu¹  

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

University of Chinese Academy of Sciences, Beijing, China

Shiteng Chen  

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

University of Chinese Academy of Sciences, Beijing, China

Abstract

This article focuses on the sub-exponential time lower bounds for two canonical #P-hard problems: counting the vertex covers of a given graph (#VC) and counting the matchings of a given graph (#Matching), under the well-known *counting exponential time hypothesis* (#ETH).

Interpolation is an essential method to build reductions in this article and in the literature. We use the idea of block interpolation to prove that both #VC and #Matching have no $2^{o(N)}$ time deterministic algorithm, even if the given graph with N vertices is a 3-regular graph. However, when it comes to proving the lower bounds for #VC and #Matching on planar graphs, both block interpolation and polynomial interpolation do not work. We prove that, for any integer $N > 0$, we can simulate N pairwise linearly independent unary functions by gadgets with only $O(\log N)$ size in the context of #VC and #Matching. Then we use log-size gadgets in the polynomial interpolation to prove that planar #VC and planar #Matching have no $2^{o(\sqrt{\frac{N}{\log N}})}$ time deterministic algorithm. The lower bounds hold even if the given graph with N vertices is a 3-regular graph.

Based on a stronger hypothesis, *randomized exponential time hypothesis* (rETH), we can avoid using interpolation. We prove that if rETH holds, both planar #VC and planar #Matching have no $2^{o(\sqrt{N})}$ time randomized algorithm, even that the given graph with N vertices is a planar 3-regular graph. The $2^{\Omega(\sqrt{N})}$ time lower bounds are tight, since there exist $2^{O(\sqrt{N})}$ time algorithms for planar #VC and planar #Matching.

We also develop a fine-grained dichotomy for a class of counting problems, symmetric Holant*.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases computational complexity, planar Holant, polynomial interpolation, rETH, sub-exponential, #ETH, #Matching, #VC

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.49

Funding Ying Liu : Supported by NSFC 61932002 and NSFC 62272448.

Shiteng Chen: Supported by National Key R&D Program of China (2023YFA1009500), NSFC 61932002 and NSFC 62272448

Acknowledgements The authors are very grateful to Prof. Mingji Xia for his beneficial guidance and advice.

¹ Corresponding author



© Ying Liu and Shiteng Chen;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 49; pp. 49:1–49:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

As an analog of NP, Valiant [19] defined the class #P of counting problems, which non-deterministic polynomial time Turing machines can compute with outputting the number of accepting computations. #P-hardness is similarly defined as NP-hardness. Two canonical counting problems, counting the vertex covers of a given graph (#VC) and counting the matchings of a given graph (#Matching), were proven to be #P-hard [18], even if the graph is sparse or planar. The two problems caused continuous attention in the past years [1, 6–8].

For the two problems, trivial $2^n \text{poly}(n)$ or $2^m \text{poly}(m)$ time algorithms exist by checking all possible solutions, where n or m denotes the number of vertices or edges in the input graph. A series of improved algorithms [9, 10, 16, 17, 22] for the two problems still need exponential time. According to the well-believed *exponential time hypothesis* (ETH) [12, 13], Dell et al. [8] put forward the counting version #ETH which states #3-SAT can not be solved in sub-exponential time. Under #ETH, many counting problems were proved that they have no $2^{o(n)}$ or $2^{o(m)}$ time deterministic algorithm. For example, counting the vertex covers of a graph with maximum degree 3 [1, 7, 15] and counting the matchings of a graph with maximum degree 4 [7] both have no $2^{o(n)}$ time algorithm. And there are also some fine-grained dichotomies [1, 15] driven from the lower bound of #VC. If the two problems are restricted on planar graphs, there are $O(2^{\sqrt{n}})$ time algorithms [16, 22]. However, the tight lower bounds for the two problems restricted on planar graphs are still open. Marx et al. [16] proved that counting the matchings of a graph, with a tree decomposition of width tw given, has no $2^{o(tw)}$ time algorithm under #ETH, even if the graph has maximum degree 8.

This article considers the lower bound results for #VC and #Matching on 3-regular graphs. We represent the two problems in the Holant framework [3], where each counting problem is defined by a set of functions. This helps us comprehend the difficulty of problems and build reductions between them more easily. One motivation behind this work is to enhance and complement the current lower bound results. Besides, the two problems on 3-regular graphs are starting problems that drive a series of dichotomy theorems [2, 4, 14]. Compared to the restriction that graphs are with maximum degree 3, the two problems on 3-regular graphs are defined by fewer functions in the Holant framework. For example, #Matching on 3-regular graphs is defined by only a ternary function in the Holant framework. This brings great convenience in building reductions. So another motivation is to prepare for developing fine-grained dichotomy theorems, which state that a problem in a class either is tractable in polynomial time or has no $2^{o(n)}$ time algorithm (or no $2^{o(\sqrt{n})}$ time algorithm on planar graphs).

We prove the $2^{\Omega(n)}$ time lower bound for #VC and #Matching on 3-regular graphs, presented in Section 3. In Section 4, we apply polynomial interpolation via log size gadgets to obtaining the nearly tight $2^{\Omega(\sqrt{\frac{n}{\log n}})}$ time lower bound for #VC and #Matching on planar 3-regular graphs under #ETH. In Section 5, we avoid the use of interpolation and prove the tight $2^{\Omega(\sqrt{n})}$ time lower bound for #VC and #Matching on planar 3-regular graphs, based on a stronger assumption rETH. In Section 6, we develop a simple fine-grained dichotomy for a class of counting problems.

2 Preliminaries

2.1 Notations and definitions

Let \mathbb{N} , \mathbb{Z} , and \mathbb{C} be the set of natural numbers, the set of integers, and the set of complex numbers, respectively. $[q]$ of some positive integer q denotes the finite domain $\{1, 2, \dots, q\}$. A domain of size 2 is called the Boolean domain, where any entry is assigned 0 or 1. A function

(or called signature) $F : \{0, 1\}^k \rightarrow \mathbb{C}$ of some non-negative integer k is a complex-valued Boolean function, where k is called the arity of F . F is a *symmetric* function if its value is invariant under the permutation of its variables. The value of a Boolean symmetric function F only depends on the Hamming weight of the input assignment, so F can be written as $[f_0, f_1, \dots, f_k]$ where f_i is the value of F accepting an assignment with Hamming weight $i \in [k]$. A function of arity 1 or 2 is called a *unary function* or a *binary function*, respectively. A Boolean unary function F is usually written as a vector $[F(0), F(1)]$, and a Boolean binary function H is usually written as a matrix $\begin{pmatrix} H(0,0) & H(0,1) \\ H(1,0) & H(1,1) \end{pmatrix}$. For example, the binary equality function $=_2$ is written as $[1, 0, 1]$, the binary dis-equality function \neq_2 is written as $[0, 1, 0]$, the equality function $=_k$ of some arity $k \geq 3$ is written as $[1, 0, \dots, 0, 1]$, the binary function OR_2 is written as $[0, 1, 1]$, and the ternary function OR_3 is written as $[0, 1, 1, 1]$.

An undirected graph G is denoted by a pair (V, E) , where V is the set of vertices and E is the set of edges. Multiple edges may exist between the same pair of vertices and self-loops in E . Let $N(v) \subseteq V$ and $E(v) \subseteq E$ denote the adjacent vertices and incident edges of v , respectively. The degree of a vertex v is denoted by $d_v = |E(v)|$ (a self-loop is counted twice, and an l -multiple edge of some integer l is counted l times). $\Delta = \max\{d_v \mid v \in V\}$ denotes the maximum degree of G . A bipartite graph G is also written as a tuple $(V_L \cup V_R, E)$, where V_L, V_R are two disjoint nonempty sets of vertices, and each edge $e \in E$ has one endpoint in V_L and another in V_R .

We introduce two individual counting problems on graphs in the following.

► **Definition 1 (#VC).** A *vertex cover* of a graph $G(V, E)$ is a set $S \subseteq V$, such that S contains at least one endpoint of e for every edge $e \in E$. The problem *#Vertex Cover (#VC)* is defined as

Input: a graph $G(V, E)$,

Output: the number of vertex covers of G .

An *independent set* of a graph $G(V, E)$ is a set $S \subseteq V$, such that S contains at most one endpoint of e for every edge $e \in E$. The problem *#Independent Set (#IS)* is defined as counting the independent sets of a given graph. $V - S$ for any vertex cover S must be an independent set, so $\#IS(G) = \#VC(G)$ for any graph G with n vertices, i.e., the two problems *#IS* and *#VC* are equivalent.

► **Definition 2 (#Matching).** A *matching* of a graph $G(V, E)$ is a set $M \subseteq E$, such that e_1 and e_2 do not intersect for any pair $e_1, e_2 \in M$. The problem *#Matching* is defined as

Input: a graph $G(V, E)$,

Output: the number of matchings of G .

We use the prefix 3R-, 3 Δ -, or pl- to denote the restriction that the input graphs are 3-regular, have max-degree no more than 3, or are planar, respectively. For example, #pl-3R-VC denotes counting the vertex covers for a given planar 3-regular graph.

To better analyze the complexity of the above two problems, we express them in the Boolean Holant [3] framework. A Boolean Holant problem, dubbed *Holant(\mathcal{F})*, is parameterized by a set \mathcal{F} of Boolean functions. A tuple $\Omega = (G, \pi)$ is called *signature grid* over \mathcal{F} , where $G(V, E)$ is a graph, and the mapping π assigns to every vertex $v \in V$ a function $F_v \in \mathcal{F}$ with a linear order to the edges in $E(v)$.

► **Definition 3** (Holant [3]). Let \mathcal{F} be a set of Boolean functions. The Boolean Holant problem $\text{Holant}(\mathcal{F})$ is defined as

Input: $\Omega = (G, \pi)$ over \mathcal{F}

$$\text{Output: } \text{Holant}(\Omega) = \sum_{\sigma: E \rightarrow \{0,1\}} \prod_{v \in V} F_v(\sigma|_{E(v)}),$$

where $\sigma|_{E(v)}$ is the restriction of σ to $E(v)$ and $F_v(\sigma|_{E(v)})$ depends on the ordered input tuple $\sigma|_{E(v)}$. If $\prod_{v \in V} F_v(\sigma|_{E(v)}) \neq 0$ for a fixed assignment σ , we called σ a satisfying assignment.

Given any function F and any non-zero constant λ , the two functions λF and F are equivalent in the context of Holant, since the factor λ only brings a constant multiplicative factor to the final result. The operator transforming any function F to λF for some constant $\lambda \in \mathbb{C} - \{0\}$ is called *normalization*. We usually hide the information of π into the graph G , so Ω is represented by G for simplification. If \mathcal{F} is finite, then the graph G must be bounded degree. If $\mathcal{F} = \{F\}$ consists of only one function F , we directly use $\text{Holant}(F)$ denotes $\text{Holant}(\mathcal{F})$. The problem $\text{Holant}^*(\mathcal{F})$ denotes the problem $\text{Holant}(\mathcal{F} \cup \mathcal{U})$ where \mathcal{U} is the set of all unary functions.

Let \mathcal{F}, \mathcal{H} be two sets of Boolean functions. A bipartite signature grid $\Omega(G, \pi)$ over $\mathcal{F}|\mathcal{H}$ consists of a bipartite graph $G(V_L \cup V_R, E)$ and a mapping π which maps each vertex $v \in V_L$ or $v \in V_R$ to a function $F \in \mathcal{F}$ or a function $H \in \mathcal{H}$, respectively. The problem $\text{Holant}(\mathcal{F}|\mathcal{H})$ is similarly defined, with a bipartite signature grid Ω over $\mathcal{F}|\mathcal{H}$ as an input. Trivially, $\text{Holant}(\mathcal{F})$ is equivalent to $\text{Holant}(\mathcal{F}|\ =_2)$.

We re-describe the problems #VC and #Matching as Boolean Holant problems. It is trivial to re-describe #Matching. Given a graph $G(V, E)$, we map the function $F_v = [1, 1, 0, \dots, 0]$ of arity d_v to every vertex $v \in V$. The function F_v takes value 1 if no more than one incident edge of v is assigned 1; otherwise, it takes 0. $\text{Holant}(G)$ is the number of matching of G . #Matching is exactly the problem $\text{Holant}(\{[1, 1], [1, 1, 0], [1, 1, 0, 0], \dots\})$, #3 Δ -Matching is the problem $\text{Holant}(\{[1, 1], [1, 1, 0], [1, 1, 0, 0]\})$, and #3R-Matching is the problem $\text{Holant}([1, 1, 0, 0])$.

Considering re-describing the problem #VC. Given the input graph $G(V, E)$, we map the equality function $=_{d_v}$ to each vertex $v \in V$. For every edge $e \in E$, we add a extra vertex u_e assigned the function OR_2 to divide it. These define a new bipartite signature grid $G'(V \cup V_e, E')$ where $V_e = \{u_e | e \in E\}$ denotes the set of new vertices. Given a satisfying assignment to E' , let the set $S \subseteq V$ consist of the vertices $v \in V$ whose incident edges all are assigned 1. S must be a vertex cover of G . Conversely, given a vertex cover S of G , the incident edges $E'(v) \in E'$ of each $v \in S$ are assigned 1, and the other edges are assigned 0. Such an assignment must satisfy $\prod_{v \in V \cup V_e} F_v = 1$. So $\text{Holant}(G')$ is the number of the vertex covers of G . #VC is exactly the problem $\text{Holant}(\{=1, =2, \dots\} | OR_2)$, #3 Δ -VC is the problem $\text{Holant}(\{=1, =2, =3\} | OR_2)$, and #3R-VC is the problem $\text{Holant}(=3 | OR_2)$.

2.2 Counting exponential time hypothesis

Impagliazzo et al. [12, 13] put forward the well-known *exponential time hypothesis* (ETH), which states that the satisfiability of a given 3-CNF formula (3-SAT) can not be decided in sub-exponential time. Dell et al. [8] put forward the relaxed counting version #ETH.

► **Conjecture 4** (#ETH [8]). *There exists a constant $\varepsilon > 0$ such that no deterministic algorithm can solve #3-SAT in $O(2^{\varepsilon n})$ time, where n denotes the number of variables of the input formula.*

The $2^{\Omega(n)}$ time lower bound can be strengthened to $2^{\Omega(m)}$ where m denotes the number of clauses in the input formula, according to the *Sparsification Lemma* [13].

A stronger hypothesis, *randomized exponential time hypothesis* (rETH) [8], states that the same lower bound also holds for the probabilistic algorithm.

► **Conjecture 5** (rETH [8]). *There is a constant $\varepsilon > 0$ such that no randomized algorithm can decide 3-SAT in $O(2^{\varepsilon n})$ time with error probability at most $1/3$.*

Polynomial-time Turing reductions, signed as \leq_{poly} or \leq_{p} , can not always preserve the sub-exponential time lower bound since the size of the generated instances may increase non-linearly. Impagliazzo et al. [13] introduced another particular class of Turing reductions: *sub-exponential time reduction families* (SERF), which preserves the sub-exponential time lower bound. We restrict the definition of SERF to problems on graphs.

► **Definition 6** ([7, 13]). *Let A and B be two problems on graphs. A sub-exponential time reduction family from A to B is an algorithm T with oracle access for B . T accepts a tuple (G, ε) as input, where G is an input of A and $\varepsilon > 0$ is a running time parameter of T , and*

- (1) *computes $A(G)$ in time $O(2^{\varepsilon|V(G)|})$ with*
- (2) *only invoking the oracle of B on graphs with $O(|V(G)|)$ vertices.*

If such an algorithm exists, We say that A is SERF-reducible to B , written as $A \leq_{\text{serf}} B$.

Suppose $A \leq_{\text{serf}} B$. If B has a $O(2^{\varepsilon n})$ time algorithm for some constant $\varepsilon > 0$, where n denotes the number of vertices in the input graph, then we can solve $A(G)$ in $O(2^{\varepsilon|V(G)|}) \cdot O(2^{\varepsilon \cdot O(|V(G)|)}) = O(2^{\varepsilon'|V(G)|})$ time for some constant $\varepsilon' > 0$. Conversely, if A has no sub-exponential time algorithm, so does B . SERF reductions are known to be transitive [13, Section 1.1.4].

Ying [15] built a series of SERF reductions and proved the sub-exponential time lower bound for $\#3\Delta\text{-VC}$ under $\#ETH$.

► **Lemma 7** ([15]). *If $\#ETH$ holds, then there exists a constant $\varepsilon > 0$ such that $\#3\Delta\text{-VC}$ has no $O(2^{\varepsilon n})$ time deterministic algorithm, where n denotes the number of vertices in the input graph.*

Calabro et al. [5] proved an isolation lemma and built a SERF reduction from 3-SAT to *Unique 3-SAT*, which is a sub-problem of 3-SAT with the restriction that the input 3-CNF formula has at most one satisfying assignment.

► **Lemma 8** ([5]). *If rETH holds, then there exists a constant $\varepsilon > 0$ such that Unique 3-SAT has no $O(2^{\varepsilon m})$ time randomized algorithm, where m denotes the number of clauses in the input formula.*

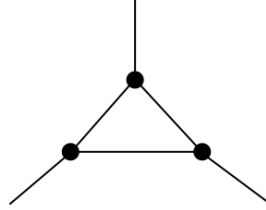
2.3 Gadget construction

To preserve the $2^{\Omega(\sqrt{N})}$ time lower bound, the SERF reduction should be strengthened to only $2^{o(\sqrt{N})}$ time cost.

A trivial but useful method to build such a reduction is called *gadget construction*. A *gadget* is also a signature grid (G, π) , where $G = (V, E \cup X)$ is a graph with some dangling edges X . A dangling edge has one endpoint in V and the other dangling. The gadget defines a function of arity $k = |X|$

$$\Gamma(x_1, x_2, \dots, x_k) = \sum_{\sigma: E \rightarrow \{0,1\}} \prod_{v \in V} F_v(\hat{\sigma}|_{E(v)}),$$

where $(x_1, x_2, \dots, x_k) \in \{0, 1\}^k$ is an assignment to X and $\hat{\sigma}$ is the extension of the assignment σ by (x_1, x_2, \dots, x_k) . If $F_v \in \mathcal{F}$ for every vertex $v \in V$, the gadget is called an \mathcal{F} -gate with signature Γ . For example, Figure 1 shows a $\{[1, 1, 0, 0]\}$ -gate with signature $[4, 2, 1, 1]$. Given



■ **Figure 1** A $\{[1, 1, 0, 0]\}$ -gate with signature $[4, 2, 1, 1]$.

an instance G of $\text{Holant}([4, 2, 1, 1])$, we can replace each occurrence of $[4, 2, 1, 1]$ by such a gadget. The value of the new instance equals the original value. So $\text{Holant}([4, 2, 1, 1])$ is reduced to $\text{Holant}([1, 1, 0, 0])$. Such a reduction built by *gadget construction* is a SERF reduction. Besides, the reduction only costs $\text{poly}(N)$ time, where N denotes the number of vertices in the input graph. So the reductions built by gadget constructions can preserve the $2^{\Omega(\sqrt{N})}$ time lower bound. Ying [23] used gadget constructions to prove the following lemma.

► **Lemma 9** ([23]). *If #ETH holds, then there exists a constant $\varepsilon > 0$ such that planar $\text{Holant}(\{=2, =3, \neq2, OR_3\})$ has no $O(2^{\varepsilon\sqrt{N}})$ time algorithm, where N denotes the number of vertices in the input graph.*

2.4 Holographic transformation

Another method is called *holographic transformation* [20,21], which also preserves the $2^{\Omega(\sqrt{N})}$ time lower bound.

The tensor product \otimes is the Kronecker product, that is, for two matrices $X = X_{a \times b}$ and $Y = Y_{c \times d}$, $X \otimes Y$ is an $ac \times bd$ matrix with entry $X_{i,j}Y_{k,l}$ at $(i, k) \in [a] \times [c]$ row and $(j, l) \in [b] \times [d]$ column, where a, b, c, d are some positive integers. Tensor power is defined recursively $X^{\otimes k} = X^{\otimes(k-1)} \otimes X$ for some integer $k > 0$.

Let T be a 2×2 invertible matrix. Given a Boolean function F of some arity k , written as a column vector in \mathbb{C}^{2^k} , we write $TF = T^{\otimes k}F$ as the transformed function. For a set \mathcal{F} of functions, $T\mathcal{F} = \{TF | F \in \mathcal{F}\}$. $FT = FT^{\otimes k}$ and $\mathcal{F}T = \{FT | F \in \mathcal{F}\}$ are similarly defined, where F is written as a row vector. Given an instance G of $\text{Holant}(\mathcal{F}|\mathcal{H})$, we generate a new bipartite graph G' from G , by reassigning the function FT or $T^{-1}H$ to each vertex which is assigned the function $F \in \mathcal{F}$ or $H \in \mathcal{H}$, respectively. Valiant's Holant Theorem [21] shows that $\text{Holant}(G) = \text{Holant}(G')$.

► **Lemma 10** ([21]). *Let \mathcal{F} and \mathcal{H} be two function sets. Given an invertible 2×2 matrix T ,*

$$\text{Holant}(\mathcal{F}|\mathcal{H}) \leq_p \text{Holant}(\mathcal{F}T|T^{-1}\mathcal{H}).$$

In addition, the generated instance has the same number of vertices and edges as the original.

3 Lower bounds for #3R-VC and #3R-Matching under #ETH

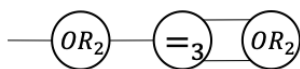
In this section, we build SERF-reductions from #3 Δ -VC to prove the $2^{\Omega(N)}$ time lower bounds for #3R-VC and #3R-Matching, where N denotes the number of vertices in the input graph.

3.1 Lower bound for #3R-VC

We build a SERF-reduction from #3 Δ -VC to #3R-VC. In fact, we prove the equivalent reduction $\text{Holant}(\{=_1, =_2, =_3\} | OR_2) \leq_{\text{serf}} \text{Holant}(=_3 | OR_2)$ by gadget constructions.

► **Theorem 11.** *If #ETH holds, then there exists a constant $\varepsilon > 0$ such that the number of vertex covers of a 3-regular graph with N vertices can not be computed in $O(2^{\varepsilon N})$ time.*

Proof. We first reduce $\text{Holant}(\{=_1, =_2, =_3\} | OR_2)$ to $\text{Holant}(=_3 | \{OR_2, [1, 1]\})$. A function $=_1$ can be realized by one function $=_3$ connected with two unary functions $[1, 1]$, and a function $=_2$ can be realized by one function $=_3$ connected with one unary function $[1, 1]$. We realize $[1, 1]$ by the $\{=_3\} | \{OR_2\}$ -gate shown in Figure 2.



■ **Figure 2** A $\{=_3\} | \{OR_2\}$ gate with signature $[1, 1]$.

By the above, $\text{Holant}(\{=_1, =_2, =_3\} | OR_2) \leq_{\text{serf}} \text{Holant}(=_3 | OR_2)$ and we can conclude this theorem, according to Lemma 7. ◀

3.2 Lower bound for #3R-Matching

We build a SERF-reduction from #3R-VC to #3R-Matching, i.e., we demonstrate that $\text{Holant}(OR_2 | =_3) \leq_{\text{serf}} \text{Holant}([1, 1, 0, 0])$ by the idea of *block interpolation* [7], which is actually multivariate polynomial interpolation.

We first introduce a lemma, which is essential during the interpolation process.

► **Lemma 12** ([11, 18]). *Let A, B, C, D be positive rational numbers, and x_0, y_0 be rational numbers. Define the sequences $\{x_l\}_{l \geq 0}$ and $\{y_l\}_{l \geq 0}$ recursively by $x_{l+1} = Ax_l + By_l$ and $y_{l+1} = Cx_l + Dy_l$. Then the sequence $\{\frac{x_l}{y_l}\}_{l \geq 0}$ is pairwise different as long as $AD - BC \neq 0$ and $By_0^2 - Cx_0^2 - (A - D)x_0y_0 \neq 0$.*

► **Theorem 13.** *If #ETH holds, then there exists some constant $\varepsilon > 0$ such that #3R-Matching can not be calculated in $O(2^{\varepsilon N})$ time, where N is the number of vertices in the input graph.*

Proof. We establish the following reduction chain.

$$\text{Holant}([0, 1, 1] | [1, 0, 0, 1]) \leq_{\text{serf}} \text{Holant}([-1, 2, 0] | [4, 2, 1, 1]) \tag{1}$$

$$\leq_{\text{serf}} \text{Holant}(\{[-1, 2, 0], [1, 1, 0, 0]\}) \tag{2}$$

$$\leq_{\text{serf}} \text{Holant}([1, 1, 0, 0]) \tag{3}$$

1. The reduction (1) is proved by the holographic transformation defined by $Q = \begin{pmatrix} 0 & \sqrt[3]{4} \\ \frac{1}{2}\sqrt[3]{4} & \frac{1}{2}\sqrt[3]{4} \end{pmatrix}$, since

$$\begin{aligned}
 [0, 1, 1](Q^{-1})^{\otimes 2} &= (0, 1, 1, 1) \left(\frac{1}{\sqrt[3]{2}} \begin{pmatrix} \frac{1}{2}\sqrt[3]{4} & -\sqrt[3]{4} \\ -\frac{1}{2}\sqrt[3]{4} & 0 \end{pmatrix} \otimes \frac{1}{\sqrt[3]{2}} \begin{pmatrix} \frac{1}{2}\sqrt[3]{4} & -\sqrt[3]{4} \\ -\frac{1}{2}\sqrt[3]{4} & 0 \end{pmatrix} \right) \\
 &= (0, 1, 1, 1) \frac{\sqrt[3]{4}}{4} \begin{pmatrix} 1 & -2 & -2 & 4 \\ -1 & 0 & 2 & 0 \\ -1 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \\
 &= \frac{\sqrt[3]{4}}{4} (-1, 2, 2, 0) = \frac{\sqrt[3]{4}}{4} [-1, 2, 0]
 \end{aligned}$$

and

$$Q^{\otimes 3}[1, 0, 0, 1] = \left(\frac{\sqrt[3]{4}}{2} \right)^3 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 4 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 \\ 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 \\ 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 8 \\ 4 \\ 4 \\ 2 \\ 4 \\ 2 \\ 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ 2 \\ 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

$(4, 2, 2, 1, 2, 1, 1, 1)^T$ has the indexes 000, 001, 010, 011, 100, 101, 110, and 111 in order, and it is the column vector of the function $[4, 2, 1, 1]$.

2. The reduction (2) is built by gadget constructions. We replace each occurrence of the function $[4, 2, 1, 1]$ by the $\{[1, 1, 0, 0]\}$ -gate showed in Figure-1.
3. The reduction (3) is built by the idea of block interpolation. Let G be an instance of $\text{Holant}(\{[-1, 2, 0], [1, 1, 0, 0]\})$ and $V' \subseteq V(G)$ denote the vertices assigned the function $[-1, 2, 0]$. Suppose $|V'| = n \leq |V(G)|$. We divide V' to $r = \frac{n}{d}$ disjoint blocks B_1, B_2, \dots, B_r for some positive integer d , such that each block $|B_i| = d$ for $i \in [r]$ (w.l.o.g, n is divisible by d).² We label each satisfying assignment to $E(G)$ a type $\vec{t} = (t_1, t_2, \dots, t_r) \in \{0, 1, \dots, d\}^r$, where t_i denotes the number of vertices $v \in B_i$ with $E(v)$ assigned $(0, 0)$. Let $\rho_{\vec{t}}$ denote the number of satisfying assignments with type \vec{t} . Then $\text{Holant}(G) = \sum_{\vec{t}} \rho_{\vec{t}} \prod_{i=1}^r (-1)^{t_i} (2)^{d-t_i}$. Define a multivariate polynomial

$$\mu(x_1, x_2, \dots, x_r) = \sum_{\vec{t}} \rho_{\vec{t}} \prod_{i=1}^r (x_i)^{t_i}$$

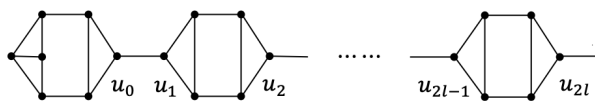
on variables $x_1, x_2, \dots, x_r \in \mathbb{C}$. $\text{Holant}(G) = 2^n \cdot \mu(-\frac{1}{2}, -\frac{1}{2}, \dots, -\frac{1}{2})$.

Given $\vec{l} = (l_1, l_2, \dots, l_r) \in \mathbb{N}^r$, we replace each vertex $v \in B_i$ by a gadget with a binary signature which is realized by a ternary function $[1, 1, 0, 0]$ connecting with a unary function showed in Figure 3.

The unary function realized by the gadget S_l can be written as $[s_l^0, s_l^1]$ where $\{s_l^0\}$ and $\{s_l^1\}$ satisfy the recurrences: $s_l^0 = 25s_{l-1}^0 + 13s_{l-1}^1$ and $s_l^1 = 13s_{l-1}^0 + 7s_{l-1}^1$ with initial conditions $s_0^0 = 50$ and $s_0^1 = 26$. Actually, s_l^0 represents the number of matchings of the underlying graph of S_l , which do not include the dangling edge. And s_l^1 represents the number of matchings that include the dangling edge. If we connect $[s_l^0, s_l^1]$ to a ternary function $[1, 1, 0, 0]$, then we realize a binary function $[s_l^0 + s_l^1, s_l^0, 0]$. The value of $G_{\vec{l}}$ is

$$\text{Holant}(G_{\vec{l}}) = \sum_{\vec{t}} \rho_{\vec{t}} \prod_{i=1}^r (s_{l_i}^0 + s_{l_i}^1)^{t_i} (s_{l_i}^0)^{d-t_i} = \prod_{i=1}^r (s_{l_i}^0)^d \cdot \mu\left(1 + \frac{s_{l_i}^1}{s_{l_i}^0}, 1 + \frac{s_{l_2}^1}{s_{l_2}^0}, \dots, 1 + \frac{s_{l_r}^1}{s_{l_r}^0}\right).$$

² We can add the gadget, an isolated 3-multiple edge whose two endpoints are assigned $[1, 1, 0, 0]$ and one edge is divided by an extra vertex assigned $[-1, 2, 0]$, to fit the assumption without affecting $\text{Holant}(G)$.



■ **Figure 3** A $\{[1, 1, 0, 0]\}$ -gate S_l for some integer l .

We construct a series of graphs $G_{\vec{l}}$ for $\vec{l} \in [d+1]^r$ to invoke the oracle of $\text{Holant}([1, 1, 0, 0])$. Then, we obtain the values of μ in $(d+1)^r$ distinct points, since $\{\frac{s_l^1}{s_l^0}\}_{l \geq 0}$ is pairwise different according to Lemma 12. So we can solve all coefficients $\rho_{\vec{l}}$ of μ by Lagrange interpolation, and compute $\text{Holant}(G)$ in $\text{poly}((d+1)^r)$ time. Each $G_{\vec{l}}$ is constructed in $\text{poly}(|V(G)|)$ time and it has $O(d|V(G)|)$ vertices.

Give a running time parameter ε , we choose a constant d such that $\frac{\log(d+1)}{d} \leq \varepsilon$. The total time of the above reduction is

$$(d+1)^r (\text{poly}(|V(G)|)) + \text{poly}((d+1)^r) = O(2^{\frac{\log(d+1)}{d}n}) = O(2^{\varepsilon|V(G)|}).$$

The above reduction is a SERF reduction.

SERF reductions are transitive. So $\text{Holant}(OR_2 | =_3) \leq_{\text{serf}} \text{Holant}([1, 1, 0, 0])$. This lemma is true by Theorem 11. ◀

4 Lower bounds for #pl-3R-VC and #pl-3R-Matching under #ETH

We build reductions from $\text{pl-Holant}(\{=_2, =_3, \neq_2, OR_3\})$ which has the $2^{\Omega(\sqrt{N})}$ time lower bound. Gadget constructions and holographic transformations preserve the $2^{\Omega(\sqrt{N})}$ time lower bound, but their ability is limited in building reductions. We need a more potent method: interpolation.

Unfortunately, both polynomial interpolation and block interpolation can not build a reduction preserving the $2^{\Omega(\sqrt{N})}$ time lower bound. The reduction built by polynomial interpolation costs polynomial time but generates new graphs with $O(N^2)$ size. The reduction built by block interpolation generates new graphs with $O(N)$ size but costs $2^{o(N)}$ time. In the process of block interpolation, the proof of Theorem 13 as an example, we can choose $d = O(\sqrt{N} \log N)$ such that the time costs is $2^{o(\sqrt{N})}$, but the generated graphs have $O(N\sqrt{N} \log N)$ size.

The type of interpolations that preserves the $2^{\Omega(\sqrt{N})}$ time lower bound has yet to be developed. We struggled for it but failed. We step back and consider building reductions that cost $2^{o(\sqrt{N})}$ time and generate graphs with $O(N \log N)$ vertices.

4.1 Polynomial interpolation via log size gadgets

In the traditional application of interpolation, for any integer $d > 0$, people build a series of $O(d)$ size gadgets to realize a sequence of d pairwise linearly independent functions and generate new instances with size $O(dN)$. Inspired by the proof of Theorem 1.3 in [8], we innovatively put up the way to construct a sequence of d pairwise linearly independent functions by gadgets only with size $O(\log d)$.

► **Lemma 14.** *Let $A, B \in \mathbb{C}^{2 \times 2}$ be two non-singular matrices.*

Given a nonzero column vector $s \in \mathbb{C}^2$, the sequence of column vectors $\{M_i \cdots M_1 s\}_{i \geq 1}$ with $M_i \in \{A, B\}$ is pairwise linearly independent if the following conditions are satisfied.

- (1) $\det([As \ Bs]) \neq 0$, and
- (2) *for any pair column vectors $v_1, v_2 \in \{M_i \cdots M_1 s\}_{i \geq 1}$ (v_1, v_2 are not necessarily distinct), $\det([Av_1 \ Bv_2]) \neq 0$.*

Proof. We prove by induction on the positive integer i .

1. $i = 1$. As and Bs are linearly independent according to the condition (1).
2. Inductively we assume the lemma is proven for $i = l - 1$ for any $l \geq 2$, and now assume $i = l$. Let v_1, v_2 be two distinct column vectors in $\{M_{l-1} \times \cdots \times M_1 s\}$. v_1, v_2 are linearly independent. Av_1, Av_2 are linearly independent, otherwise, $|A| = 0$ or v_1, v_2 are linearly dependent, which contradicts to the assumption. So the sequence $\{AM_{l-1} \cdots M_1 s\}$ is pairwise linearly independent. Similarly, the sequence $\{B \times M_{l-1} \cdots M_1 s\}$ is pairwise linearly independent.

Av_1, Bv_2 for any pair column vectors in $\{M_{l-1} \times \cdots \times M_1 s\}$ are linearly independent according to the condition (2). So The sequence $\{AM_{l-1} \cdots M_1 s, BM_{l-1} \cdots M_1 s\}$ is pairwise linearly independent.

This lemma is true. ◀

If we have three gadgets with signatures A, B, s which satisfy the conditions in Lemma 14, then, for any $d \in \mathbb{N}$, we can construct a sequence of d gadgets with unary signatures which are pairwise linearly independent. Besides, each gadget has $O(\log d)$ size.

► **Lemma 15.** *Let \mathcal{F} be a set of complex-valued Boolean functions. Suppose the following gadgets can be realized by the \mathcal{F} .*

- (1) *two non-singular recursive gadgets with binary signature $A, B \in \mathbb{C}^{2 \times 2}$ and*
 - (2) *a unary start gadget with signature s written as a column vector,*
- which satisfy $\det([As \ Bs]) \neq 0$ and $\det([Av_1 \ Bv_2]) \neq 0$ for any unary signatures $v_1, v_2 \in \{M_i \cdots M_1 s\}_{i \geq 0}$ with $M_i \in \{A, B\}$.*

Then for a finite sequence of unary functions $\mathcal{S} = \{[x_1, y_1], \dots, [x_m, y_m]\}$ with $x_j, y_j \in \mathbb{C}$ for any $j \in [m]$, $\text{Holant}(\mathcal{F} \cup \mathcal{S}) \leq_p \text{Holant}(\mathcal{F})$. Furthermore, $\text{Holant}(\mathcal{F})$ has no $2^{o(\sqrt{\frac{N}{\log N}})}$ time algorithm if $\text{Holant}(\mathcal{F} \cup \mathcal{S})$ has no $2^{o(\sqrt{n})}$ time algorithm, where N, n denote the number of vertices of the input.

The result also holds for planar $\text{Holant}(\mathcal{F})$ if the gadgets are planar.

Proof. Let $G(V \cup S, E)$ with n vertices be an instance of $\text{Holant}(\mathcal{F} \cup \mathcal{S})$, where S denotes the set of vertices each assigned a unary function in \mathcal{S} . Let $S_j \subseteq S$ denotes the set of vertices assigned $[x_j, y_j]$ where $j \in [m]$. We label each assignment to E a type $t = (t_1, t_2, \dots, t_m)$ where $t_j \in \{0, 1, 2, \dots, |S_j|\}$ denotes the number of vertices $v \in S_j$ whose incident edge is assigned 0. Then $\text{Holant}(G) = \sum_t \rho_t \prod_{j \in [m]} (x_j)^{t_j} (y_j)^{|S_j| - t_j}$ where ρ_t denotes the sum of the products of the signatures in V under the assignments with type t .

If we obtain the values of all coefficients ρ_t , then we can compute $\text{Holant}(G)$. According to Lemma 14, for any integer $n > 0$, we can simulate a sequence of n pairwise linearly independent unary gadgets $\{[w_1, z_1], [w_2, z_2], \dots, [w_n, z_n]\}$ by the signature set \mathcal{F} . Each gadget has $O(\log n)$ size.

Define $l = (l_1, l_2, \dots, l_m)$ where $l_j \in \mathbb{N}$ for $j \in [m]$. We construct a graph G_l by replacing each vertex in S_j with a vertex assigned the signature $[w_{l_j}, z_{l_j}]$. The value of G_l is

$$\text{Holant}(G_l) = \sum_t \rho_t \prod_{j \in [m]} (w_{l_j})^{t_j} (z_{l_j})^{|S_j| - t_j} \quad (1)$$

By taking $l \in [|S_1| + 1] \times [|S_2| + 1] \times \cdots \times [|S_m| + 1]$, we construct a system of $(|S_1| + 1) \times (|S_2| + 1) \times \cdots \times (|S_m| + 1) \leq (n + 1)^m$ equations of the form as (1):

$$\begin{pmatrix} \text{Holant}(G_{(1,1,\dots,1)}) \\ \text{Holant}(G_{(1,1,\dots,2)}) \\ \vdots \\ \text{Holant}(G_{(|S_1|+1,|S_2|+1,\dots,|S_m|+1)}) \end{pmatrix} = M \cdot \begin{pmatrix} \rho_{(0,0,\dots,0)} \\ \rho_{(0,0,\dots,1)} \\ \vdots \\ \rho_{(|S_1|,|S_2|,\dots,|S_m|)} \end{pmatrix}$$

where the matrix $M = M_1 \otimes M_2 \otimes \cdots \otimes M_m$ with

$$M_j = \begin{pmatrix} z_1^{|S_j|} & w_1 z_1^{|S_j|-1} & \cdots & w_1^{|S_j|} \\ z_2^{|S_j|} & w_2 z_2^{|S_j|-1} & \cdots & w_2^{|S_j|} \\ \vdots & \vdots & \ddots & \vdots \\ z_{|S_j|+1}^{|S_j|} & w_{|S_j|+1} z_{|S_j|+1}^{|S_j|-1} & \cdots & w_{|S_j|+1}^{|S_j|} \end{pmatrix}$$

for $j \in [m]$. Since $\{[w_1, z_1], \dots, [w_{|S_j|+1}, z_{|S_j|+1}]\}$ is pairwise linearly independent, $\det(M_j) \neq 0$ for any $j \in [m]$. So the matrix M is invertible.

We can compute all ρ_t by solving the system after obtaining the values of $\text{Holant}(G_l)$ by invoking the oracle of $\text{Holant}(\mathcal{F})$. Let $T(N)$ be the time cost of the oracle, where N is the vertices number of the input graph. In the above reduction, we build at most $(n + 1)^m$ new graphs in $(n + 1)^m \text{poly}(n)$ time, and each new graph has $O(n \log(n + 1))$ vertices. The reduction time is $(n + 1)^m \text{poly}(n) + (n + 1)^m T(c \cdot n \log(n + 1)) + \text{poly}((n + 1)^m)$ where c is some constant. Since m is also a constant, the reduction is a polynomial time reduction. So $\text{Holant}(\mathcal{F} \cup \mathcal{S}) \leq_p \text{Holant}(\mathcal{F})$.

Suppose $\text{Holant}(\mathcal{F})$ has $2^{o(\sqrt{\frac{N}{\log N}})}$ time algorithm, that is, $T(N) = 2^{\varepsilon(\sqrt{\frac{N}{\log N}})}$ for any $\varepsilon > 0$. Then we can solve $\text{Holant}(G)$ in

$$(n + 1)^m \text{poly}(n) + (n + 1)^m 2^{\varepsilon(\sqrt{\frac{cn \log(n+1)}{\log c + \log n + \log \log(n+1)}})} + \text{poly}((n + 1)^m) \leq 2^{\varepsilon' \sqrt{n}} \quad (2)$$

for any $\varepsilon' > 0$, by choose small enough ε . Then we obtain the $2^{o(\sqrt{n})}$ algorithm for the problem $\text{Holant}(\mathcal{F} \cup \mathcal{S})$. It is a contradiction to the assumption that $\text{Holant}(\mathcal{F} \cup \mathcal{S})$ has no $2^{o(\sqrt{n})}$ time algorithm. \blacktriangleleft

Using block interpolation via log size gadget does not improve the lower bound.

4.2 Lower bound for planar #3R-VC

We reduce $\text{pl-Holant}(\{=_2, =_3, \neq_2, OR_3\})$ to $\text{pl-Holant}(=_3 |OR_2)$. We use the problem $\text{pl-Holant}(=_3 | \{OR_2, [-1, 1]\})$ as an intermediate.

► **Lemma 16.** *If #ETH holds, then there exists some constant $\varepsilon > 0$ such that planar $\text{Holant}(=_3 | \{OR_2, [-1, 1]\})$ has no $O(2^{\varepsilon \sqrt{N}})$ time algorithm, where N denotes the number of vertices of the instance.*

Proof. We prove that $\text{pl-Holant}(\{=_2, =_3, \neq_2, OR_3\}) \leq_p \text{pl-Holant}(=_3 | \{OR_2, [-1, 1]\})$ by gadget constructions. For any instance G of $\text{pl-Holant}(\{=_2, =_3, \neq_2, OR_3\})$, we add a vertex assigned the function $=_2$ to divide each edge, and replace every occurrence of \neq_2 or OR_3 by the corresponding $\{=_3\} | \{OR_2, =_2, [-1, 1]\}$ -gates shown in Figure 4. We generate a new graph G' which is an instance of $\text{pl-Holant}(\{=_2, =_3\} | \{OR_2, [-1, 1], =_2\})$.

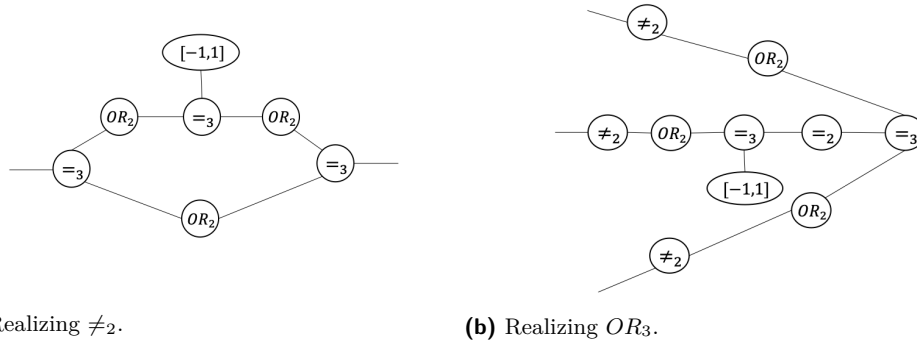


Figure 4 The constructions of some gadgets with signature \neq_2 and OR_3 .

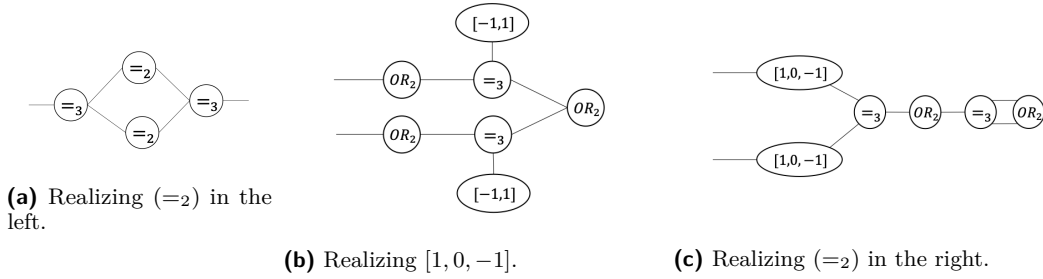


Figure 5 The constructions of gadgets with signature $(=)_2$.

Then we replace each the occurrence of $=_2$ in the left of G' by a $\{=₃\}|\{=₂\}$ -gate shown in Figure 5-(a) and each occurrence of $=_2$ in the right of G' by the $\{=₃\}|\{OR_2, [-1, 1]\}$ -gate shown in Figure 5-(c). The final graph G'' is an instance of $\text{Holant}(=₃ | \{OR_2, [-1, 1]\})$.

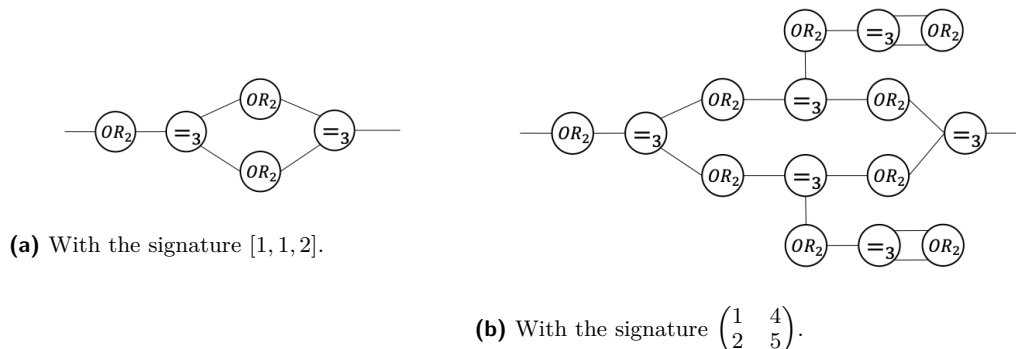
So $\text{Holant}(\{=₂, =₃, \neq_2, OR_3\}) \leq_p \text{Holant}(=₃ | \{OR_2, [-1, 1]\})$ with preserving planarity. The reductions cost polynomial time, and G'' has cN vertices for some constant c , where N denotes the number of vertices in G . Suppose $\text{Holant}(G'')$ can be solved in $O(2^{\varepsilon\sqrt{cN}})$ time for any $\varepsilon > 0$, then $\text{Holant}(G)$ can be solved in $\text{poly}(N) + O(2^{\varepsilon'\sqrt{cN}}) = O(2^{\varepsilon'\sqrt{cN}})$ time for any $\varepsilon' > 0$. It is a contradiction to Lemma 9. \blacktriangleleft

Next we interpolate the function $[-1, 1]$ in the context of planar $\text{Holant}(=₃ | OR_2)$.

► **Theorem 17.** *If #ETH holds, then there exists some constant $\varepsilon > 0$ such that planar $\text{Holant}(=₃ | OR_2)$ has no $O(2^{\varepsilon\sqrt{\frac{N}{\log N}}})$ time algorithm, where N denotes the number of vertices of the instance.*

Proof. In the context of planar $\text{Holant}(=₃ | OR_2)$, we build two recursive gadgets with signatures $A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$, $B = \begin{pmatrix} 1 & 4 \\ 2 & 5 \end{pmatrix}$ shown in Figure 6, and a start gadget with signature $s = [1, 1]^T$ shown in Figure 2. $\det(A), \det(B) \neq 0$ and $\det([As \ Bs]) \neq 0$.

Consider the sequence $\{M_i \cdots M_1 s\}_{i \geq 1}$ where $M_i \in \{A, B\}$. The elements in A, B, s are positive, so any unary signature in this sequence is written as $[z, w]$ with $z, w > 0$ and can be normalized as $[x, 1]$ with $0 < x \leq 1$. Let $v_1 = [x_1, 1]^T, v_2 = [x_2, 1]^T$ be two unary functions in the sequence. If $\det([Av_1 \ Bv_2]) = 0$ then $x_1 x_2 + x_1 = 3$, that is a contradiction since $0 < x_1, x_2 \leq 1$. So, the conditions in Lemma 15 are satisfied. According to Lemma 15 and Lemma 16, this theorem is true. \blacktriangleleft



■ **Figure 6** The recursive gadgets to simulate a sequence of unary functions of the form $\{[x_i, 1]\}_{i \geq 0}$ in the context of planar $\text{Holant}(=_3 | OR_2)$.

4.3 Lower bound for planar #3R-Matching

We reduce from planar $\text{Holant}(=_3 | \{OR_2, [-1, 1]\})$. We firstly do a holographic transformation defined by $Q = \begin{pmatrix} \frac{\sqrt[3]{4}}{2} & -\frac{\sqrt[3]{4}}{2} \\ -\sqrt[3]{4} & 0 \end{pmatrix}$. $Q^{\otimes 2}(0, 1, 1, 1)^T = \frac{(\sqrt[3]{4})^2}{4}(-1, 2, 2, 0)^T$, $Q(-1, 1)^T = \sqrt[3]{4}(-1, 1)^T$, and $(=_3)(Q^{-1})^{\otimes 3} = [4, 2, 1, 1]$. Then $\text{pl-Holant}(=_3 | \{OR_2, [-1, 1]\})$ is reduced to $\text{pl-Holant}([4, 2, 1, 1] | \{[-1, 2, 0], [-1, 1]\})$. The function $[4, 2, 1, 1]$ can be realized by a $\{[1, 1, 0, 0]\}$ -gate showed in Figure 3 and the function $[-1, 2, 0]$ can be realized by the function $[1, 1, 0, 0]$ connected with a unary function $[2, -3]$, so $\text{pl-Holant}([4, 2, 1, 1] | \{[-1, 2, 0], [-1, 1]\})$ can be reduce to $\text{pl-Holant}(\{[1, 1, 0, 0], [2, -3], [-1, 1]\})$ with preserving the $2^{\Omega(\sqrt{N})}$ time lower bound.

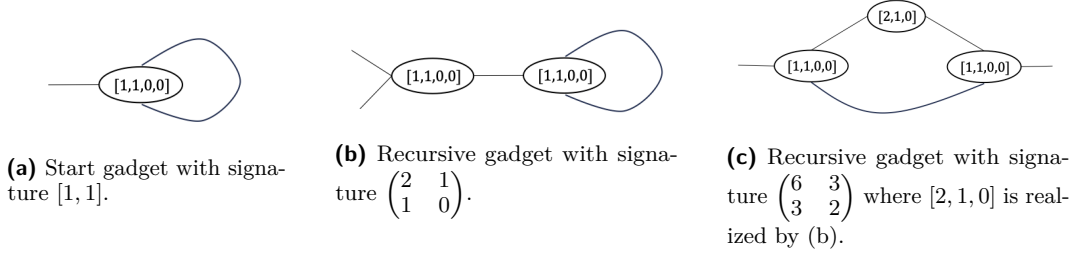
► **Lemma 18.** *If #ETH holds, then there exists some constant $\varepsilon > 0$ such that planar $\text{Holant}(\{[1, 1, 0, 0], [2, -3], [-1, 1]\})$ has no $O(2^{\varepsilon\sqrt{N}})$ time algorithm, where N is the number of vertices in the input.*

Then we interpolate the two unary functions in the context of planar $\text{Holant}(\{[1, 1, 0, 0]\})$.

► **Theorem 19.** *If #ETH holds, then there exists some constant $\varepsilon > 0$ such that planar $\text{Holant}([1, 1, 0, 0])$ has no $O(2^{\varepsilon\sqrt{\frac{N}{\log N}}})$ time algorithm, where N is the number of vertices in the input.*

Proof. In the context of planar $\text{Holant}([1, 1, 0, 0])$, we construct two recursive gadgets $A = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 6 & 3 \\ 3 & 2 \end{pmatrix}$ and a start gadget $s = [1, 1]^T$, showed in Figure 7. $\det(A), \det(B) \neq 0$ and $\det([As \ Bs]) \neq 0$.

Consider the sequence $\{M_i \cdots M_1 s\}_{i \geq 1}$ where $M_i \in \{A, B\}$. Each unary function in the sequence can be normalized to $[x, 1]$ with $x > 0$. Let $v_1 = [x_1, 1]^T, v_2 = [x_2, 1]^T$ be two unary signatures in the sequence. If $\det([Av_1 \ Bv_2]) = 0$ then $x_1 + 3x_2 + 2 = 0$, which is a contradiction. So this lemma is true by Lemma 15 and Lemma 18. ◀



■ **Figure 7** The gadgets to simulate a sequence of unary functions of the form $\{[x_i, 1]\}_{i \geq 0}$ in the context of planar $\text{Holant}(\{[1, 1, 0, 0]\})$.

5 Lower bounds for #pl-3R-VC and #pl-3R-Matching under rETH

It can be observed that the use of interpolation weakens the lower bounds with the $\sqrt{\log N}$ factor in the exponent. In this section, based on a stronger assumption rETH, we build polynomial reductions from Unique 3-SAT. We use the character that, for any instance G of Unique 3-SAT, $\text{Holant}(G)$ is either 0 or 1. In these reductions, we do not require the exact values of the generated instances; we only need to decide whether the values are 0 or not. Therefore, we can avoid the use of interpolation.

Unique 3-SAT can also be treated as a sub-problem of #3-SAT that is equivalent to $\text{Holant}(\{=_1, =_2, \dots\} \cup \{\neq_2, OR_1, OR_2, OR_3\})$, where $OR_1 = [0, 1]$. According to the reductions in [23], $\text{Holant}(\{=_1, =_2, \dots\} \cup \{\neq_2, OR_1, OR_2, OR_3\})$ can be reduced to $\text{pl-Holant}(\{=_2, =_3, \neq_2, OR_3\})$ in polynomial time by equivalent gadget constructions. That is, given a 3-CNF formula ϕ with N variables and M clauses, we can transform ϕ to a planar graph G_ϕ with $O((M + N)^2) = O(M^2)$ vertices in $\text{poly}((M + N)^2) = \text{poly}(M^2)$ time. And G_ϕ is an instance of $\text{pl-Holant}(\{=_2, =_3, \neq_2, OR_3\})$. Suppose ϕ has at most one satisfying assignment, that is, $\#\text{SAT}(\phi) = \text{Holant}(G_\phi)$ is either 0 or 1.

In the following, we consider transforming G_ϕ to some instance of #pl-3R-VC or #pl-3R-Matching.

5.1 Lower bound for planar #3R-VC

According to the proof of Lemma 16, we transfer G_ϕ to a graph G which is an instance of $\text{pl-Holant}(=_3 \mid \{OR_2, [-1, 1]\})$. G has $O(M^2)$ vertices and $\text{Holant}(G) = \text{Holant}(G_\phi)$ is either 0 or 1.

Let G' be a graph constructed from G by replacing each occurrence of $[1, -1]$ by a gadget with signature $[1, 1]$, showed in Figure 2. Since $1 \equiv -1 \pmod{2}$, $(\text{Holant}(G') \pmod{2}) = \text{Holant}(G)$. G' is an instance of planar $\text{Holant}(=_3 \mid OR_2)$. Suppose G has cM^2 vertices for some constant c . If we can compute $\text{Holant}(G')$ in $2^{o(\sqrt{cM^2})}$ time, i.e., in $2^{\varepsilon\sqrt{cM^2}}$ time for any $\varepsilon > 0$, then we can compute $\#\text{SAT}(\phi)$ by the above in $2^{\varepsilon\sqrt{cM^2}} + \text{poly}(M^2) \leq 2^{\varepsilon' M}$ time for any $\varepsilon' > 0$. It contradicts Lemma 8. So we can obtain the following lemma.

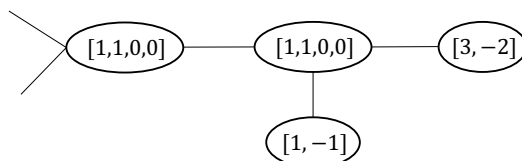
► **Theorem 20.** *If rETH holds, then there exists some constant $\varepsilon > 0$ such that planar $\text{Holant}(=_3 \mid OR_2)$, i.e., counting the vertex covers of a given planar 3-regular graph, has no $O(2^{\varepsilon\sqrt{N}})$ time randomized algorithm. N denotes the number of vertices of the input graph.*

5.2 Lower bound for planar #3R-Matching

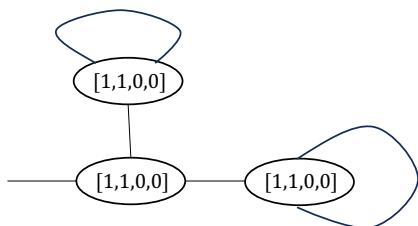
We obtain the graph G from G_ϕ as in Section 5.1. Then we do a holographic transformation defined by $Q = \begin{pmatrix} \frac{\sqrt[3]{4}}{2} & -\frac{\sqrt[3]{4}}{2} \\ -\sqrt[3]{4} & 0 \end{pmatrix}$. Because $(=_3)(Q^{-1})^{\otimes 3} = [4, 2, 1, 1]$, $Q^{\otimes 2}(0, 1, 1, 1)^T = (\sqrt[3]{4})^{-1}(-1, 2, 2, 0)^T$ or $Q(-1, 1)^T = \sqrt[3]{4}(-1, 1)^T$. The generated instance G_1 is an instance of $\text{Holant}([4, 2, 1, 1] | \{(\sqrt[3]{4})^{-1}[-1, 2, 0], \sqrt[3]{4}[-1, 1]\})$. $\text{Holant}(G_1) = \text{Holant}(G)$.

It is worth noting that the non-zero multiple factor to a function can not be ignored when considering the value of an individual instance.

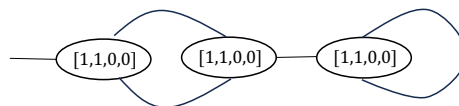
We replace each $[4, 2, 1, 1]$ by the $\{[1, 1, 0, 0]\}$ -gate showed in Figure 3, replace each $\sqrt[3]{4}[-1, 1]$ by $[1, -1]$, and replace each $(\sqrt[3]{4})^{-1}[-1, 2, 0]$ by $[1, -2, 0]$. The new graph, denoted by G_2 , is an instance of planar $\text{Holant}(\{[1, 1, 0, 0], [1, -1], [1, -2, 0]\})$. For some fixed integers c, d , $\text{Holant}(G_1) = (-1)^c (\sqrt[3]{4})^d \cdot \text{Holant}(G_2)$. We further replace each occurrence of $[1, -2, 0]$ by the gadget showed in Figure 8-(a). This defines G_3 with $\text{Holant}(G_2) = \text{Holant}(G_3)$. G_3 is an instance of planar $\text{Holant}(\{[1, 1, 0, 0], [1, -1], [3, -2]\})$.



(a) Realizing $[1, -2, 0]$.



(b) Realizing $[3, 1]$.



(c) Realizing $[4, 2]$.

■ **Figure 8** The constructions of some gadgets.

We replace each occurrence of $[1, -1]$ and each occurrence of $[3, -2]$ by the gadget with signature $[4, 2]$ and the gadget with signature $[3, 1]$, respectively. Such gadgets are showed in Figure 8. This defines a new instance G'' of planar $\text{Holant}([1, 1, 0, 0])$. Because $1 \equiv 4 \pmod 3$, $2 \equiv -1 \pmod 3$ and $1 \equiv -2 \pmod 3$, so $\text{Holant}(G'') = \text{Holant}(G_3) \pmod 3$.

Since $(-1)^c (\sqrt[3]{4})^d \cdot \text{Holant}(G_2) = \text{Holant}(G)$ is either 0 or 1, $\text{Holant}(G) = 0$ if $((-1)^c (\sqrt[3]{4})^d \cdot \text{Holant}(G_3) \pmod 3) = 0$, and $\text{Holant}(G) = 1$ if $((-1)^c (\sqrt[3]{4})^d \cdot \text{Holant}(G_3) \pmod 3) \neq 0$.

If $\text{Holant}(G'')$ can be computed in $O(2^{\varepsilon(\sqrt{cM^2})})$ time for any $\varepsilon > 0$, then $\#\text{SAT}(\phi)$ can be solved in $O(2^{\varepsilon' M})$ time for any constant $\varepsilon' > 0$. It is a contradiction to Lemma 8.

► **Theorem 21.** *If rETH holds, then there exists some constant $\varepsilon > 0$ such that planar $\text{Holant}([1, 1, 0, 0])$, i.e., counting all matchings of any given planar 3-regular graph, has no $O(2^{\varepsilon\sqrt{N}})$ time randomized algorithm. N denotes the number of vertices in the input graph.*

6 Fine-grained dichotomy for symmetric Holant*

We develop a fine-grained dichotomy theorem for a class of counting problems: symmetric Holant*, from #3R-VC and #3R-Matching. A function is *degenerate* if it can be written as the tensor power of some unary functions. A Holant problem defined by a set of degenerate functions is trivially computed in polynomial time, so we only consider non-degenerate functions.

► **Theorem 22.** *Let \mathcal{F} be a set of non-degenerate symmetric Boolean functions. $\text{Holant}^*(\mathcal{F})$ is computable in polynomial time if \mathcal{F} satisfies one of the following cases.*

1. *Every function in \mathcal{F} is of arity no more than two.*
2. *There exist two constants a and b , which are not both zero and depending only on \mathcal{F} , such that for all functions $[x_0, x_1, \dots, x_n] \in \mathcal{F}$ one of the two conditions is satisfied: (1) for every $k = 0, 1, \dots, n-2$, there is $ax_k + bx_{k+1} - ax_{k+2} = 0$; (2) $n = 2$ and the function $[x_0, x_1, x_2]$ is of the form $[2a\lambda, b\lambda, -2a\lambda]$ for some constant $\lambda \in \mathbb{C}$.*
3. *For every function $[x_0, x_1, \dots, x_n] \in \mathcal{F}$ one of the two conditions is satisfied: (1) for every $k = 0, 1, \dots, n-2$, there is $x_k + x_{k+2} = 0$; (2) $n = 2$ and the function $[x_0, x_1, x_2]$ is of the form $[\lambda, 0, \lambda]$ for some constant $\lambda \in \mathbb{C}$.*

Otherwise, there exists some constant $\varepsilon > 0$ such that it has no $2^{\varepsilon N}$ time deterministic algorithm under #ETH. N denotes the number of vertices in the input graph.

For planar $\text{Holant}^(\mathcal{F})$ which does not satisfy the tractable conditions, there also exists some constant $\varepsilon' > 0$ such that it has no $2^{\varepsilon' \sqrt{N}}$ time deterministic algorithm under #ETH. Besides, it has no $2^{\varepsilon' \sqrt{N}}$ time randomized algorithm under rETH.*

Proof. We follow the original proofs in [4] to develop a fine-grained dichotomy theorem for the class symmetric Holant*. Given the problem $\text{Holant}^*(\mathcal{F})$, the problem has a polynomial time algorithm [4] if \mathcal{F} satisfies the tractable conditions; otherwise, Cai et al. used gadget constructions and holographic transformations to build the polynomial reductions from $\text{Holant}^*(=_3 |OR_2)$ or $\text{Holant}^*([1, 0, 0, 1])$ to $\text{Holant}^*(\mathcal{F})$. Since gadget constructions and holographic transformations all preserve the $2^{\Omega(N)}$ or $2^{\Omega(\sqrt{N})}$ time lower bound. Besides, these reductions only use planar gadgets.

So the problem $\text{Holant}^*(\mathcal{F})$, with \mathcal{F} violating the tractable conditions, has the $2^{\Omega(N)}$ time lower bound under #ETH. Moreover, it has the $2^{\Omega(\sqrt{N})}$ time lower bound if the inputs are restricted to planar graphs. ◀

7 Conclusion

Based on #ETH, we prove the tight $2^{\Omega(N)}$ time lower bounds for $\text{Holant}(=_3 |OR_2)$ and $\text{Holant}([1, 0, 0, 1])$ by Theorem 11 and Theorem 13. And we prove the tight $2^{\Omega(\sqrt{N})}$ time lower bounds for pl-Holant*($=_3 |OR_2$) and pl-Holant*([1, 0, 0, 1]) under #ETH. We also present a fine-grained dichotomy theorem for a class of counting problems, symmetric Holant*.

One of the further works is the development of the fine-grained dichotomy theorem under #ETH. The development is challenged when the inputs of counting problems are restricted to planar graphs since we only prove the nearly tight $2^{\Omega(\sqrt{\frac{N}{\log N}})}$ time lower bound for pl-Holant*($=_3 |OR_2$) and pl-Holant*([1, 0, 0, 1]). The problem that whether pl-Holant*($=_3 |OR_2$) or pl-Holant*([1, 0, 0, 1]) has $2^{o(\sqrt{N})}$ time algorithm or not, under #ETH, is still open. However, this paper still presents a novelty application of polynomial interpolation, which can be popularized to prove the nearly tight $2^{\Omega(\sqrt{\frac{N}{\log N}})}$ time lower bound for more generalized planar counting problems.

Based on rETH, we prove the tight $2^{\Omega(\sqrt{N})}$ time lower bound for #pl-3R-VC and #pl-3R-Matching. The reductions under rETH need to pay close attention to the exact values of the generated instances, so their generality is limited. However, these reductions still provide a method that avoids interpolation for developing the tight lower bound for planar counting problems.

References

- 1 Cornelius Brand, Holger Dell, and Marc Roth. Fine-grained dichotomies for the tutte plane and boolean #csp. *Algorithmica*, 81(2):541–556, 2019.
- 2 Jin-Yi Cai, Sangxia Huang, and Pinyan Lu. From holant to #csp and back: Dichotomy for holant problems. *Algorithmica*, 64(3):511–533, November 2012. doi:10.1007/s00453-012-9626-6.
- 3 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Holographic algorithms by fibonacci gates and holographic reductions for hardness. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '08, pages 644–653, USA, 2008. IEEE Computer Society. doi:10.1109/FOCS.2008.34.
- 4 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Holant problems and counting csp. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pages 715–724, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1536414.1536511.
- 5 Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi. The complexity of unique k-sat: An isolation lemma for k-cnfs. *Journal of Computer and System Sciences*, 74(3):386–393, 2008. Computational Complexity 2003. doi:10.1016/j.jcss.2007.06.015.
- 6 Katrin Casel, Henning Fernau, Mehdi Ghadikoalei, Jérôme Monnot, and Florian Sikora. Extension of vertex cover and independent set in some classes of graphs. *International Conference on Algorithms and Complexity (ICAC 2019)*, 11485:124–136, April 2019. doi:10.1007/978-3-030-17402-6_11.
- 7 Radu Curticapean. Block interpolation: A framework for tight exponential-time counting complexity. *Information and Computation*, 261:265–280, 2018.
- 8 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the tutte polynomial. *ACM Transaction on Algorithms*, 10(4):21:1–21:32, 2014.
- 9 J. Flum and M. Grohe. The parameterized complexity of counting problems. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 538–547, 2002. doi:10.1109/SFCS.2002.1181978.
- 10 Mark K. Goldberg, Thomas H. Spencer, and Dave A. Berque. A low-exponential algorithm for counting vertex covers. *Journal of Graph Theory*, 1992.
- 11 Catherine Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *Comput. Complex.*, 9(1):52–72, January 2000. doi:10.1007/PL00001601.
- 12 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
- 13 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
- 14 Michael Kowalczyk and Jin-Yi Cai. Holant problems for 3-regular graphs with complex edge functions. *Theory of Computing Systems*, 59(1):133–158, July 2016. doi:10.1007/s00224-016-9671-7.
- 15 Ying Liu. Exponential time complexity of the complex weighted boolean #csp. In Weili Wu and Guangmo Tong, editors, *Computing and Combinatorics*, pages 83–96, Cham, 2024. Springer Nature Switzerland.

- 16 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and Gaps: Tight Complexity Results of General Factor Problems Parameterized by Treewidth and Cutwidth. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 95:1–95:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.95.
- 17 Dimitrios M. Thilikos. Compactors for parameterized counting problems. *Computer Science Review*, 39:100344, 2021. doi:10.1016/j.cosrev.2020.100344.
- 18 Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001. doi:10.1137/S0097539797321602.
- 19 Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.
- 20 Leslie G. Valiant. Accidental algorithms. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 509–517, USA, 2006. IEEE Computer Society. doi:10.1109/FOCS.2006.7.
- 21 Leslie G Valiant. Holographic algorithms. *SIAM Journal on Computing*, 37(5):1565–1594, 2008.
- 22 Yitong Yin and Chihao Zhang. Approximate counting via correlation decay on planar graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 47–66, USA, 2013. Society for Industrial and Applied Mathematics.
- 23 Liu Ying. The complexity of contracting planar tensor network. *ArXiv*, abs/2001.10204, 2020.

Arena-Independent Memory Bounds for Nash Equilibria in Reachability Games

James C. A. Main 

F.R.S.-FNRS & UMONS – Université de Mons, Belgium

Abstract

We study the memory requirements of Nash equilibria in turn-based multiplayer games on possibly *infinite graphs* with reachability, shortest path and Büchi objectives.

We present constructions for *finite-memory* Nash equilibria in these games that apply to arbitrary game graphs, bypassing the finite-arena requirement that is central in existing approaches. We show that, for these three types of games, from any Nash equilibrium, we can derive another Nash equilibrium where all strategies are finite-memory such that the same players accomplish their objective, without increasing their cost for shortest path games.

Furthermore, we provide memory bounds that are *independent of the size of the game graph* for reachability and shortest path games. These bounds depend only on the number of players.

To the best of our knowledge, we provide the first results pertaining to finite-memory constrained Nash equilibria in infinite arenas and the first arena-independent memory bounds for Nash equilibria.

2012 ACM Subject Classification Theory of computation → Solution concepts in game theory

Keywords and phrases multiplayer games on graphs, Nash equilibrium, finite-memory strategies

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.50

Related Version *Full Version*: <https://arxiv.org/abs/2310.02142> [20]

Funding This work has been supported by the Fonds de la Recherche Scientifique - FNRS under Grant n° T.0188.23 (PDR ControllerRS). The author is a Research Fellow of the Fonds de la Recherche Scientifique – FNRS and member of the TRAIL institute.

Acknowledgements I thank Thomas Brihaye, Aline Goeminne and Mickael Randour for fruitful discussions and their comments on a preliminary version of this paper.

1 Introduction

Games on graphs. *Games on graphs* are a prevalent framework to model reactive systems, i.e., systems that continuously interact with their environment. Typically, this interaction is modelled as an infinite-duration *two-player (turn-based) zero-sum game* played on an arena (i.e., a game graph) where a system player and an environment player are adversaries competing for opposing goals (e.g., [18, 1, 14]), which can be modelled, e.g., by numerical costs for the system player. Determining whether the system can enforce some specification boils down to computing how low of a cost the system player can guarantee. We then construct an *optimal strategy* for the system which can be seen as a *formal blueprint* for a controller of the system to be implemented [24, 1]. For implementation purposes, strategies should have a finite representation. We consider *finite-memory strategies* (e.g., [3]) which are strategies defined by Mealy machines, i.e., automata with outputs on their edges.

Nash equilibria. In some applications, this purely adversarial model may be too restrictive. This is the case in settings with several agents, each with their own objective, who are not necessarily opposed to one another. Such situations are modelled by *multiplayer non-zero-sum games* on graphs. The counterpart of optimal strategies in this setting is typically a notion



© James C. A. Main;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 50; pp. 50:1–50:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



of *equilibrium*. We focus on *Nash equilibria* [22] (NEs) in the following; an NE is a tuple of strategies, one per player, such that no player has an incentive to unilaterally deviate from their strategy.

Reachability games. We focus on variants of *reachability* games on possibly infinite arenas. In a reachability game, the goal of each player is given by a set of target vertices to be visited. We also study *shortest path* games, where players aim to visit their targets as soon as possible (where time is modelled by non-negative edge weights), and *Büchi* games, where players aim to visit their targets infinitely often. NEs are guaranteed to exist for these games: see [7, 12] for reachability games and for shortest path games in finite arenas, the full version of this work [20] for shortest path games in general and [25] for Büchi games.

Usually, finite-memory NEs for these games are given by strategies whose size depends on the arena (e.g., [7, 26, 6]). These constructions consequently do not generalise to infinite arenas. The main idea of these approaches is as follows. First, one shows that there exist plays resulting from NEs with a *finite representation*, e.g., a lasso. This play is then encoded in a Mealy machine. If some player is inconsistent with the play, the other players switch to a (finite-memory) *punishing strategy* to sabotage the deviating player; this enforces the stability of the equilibrium. This punishing mechanism is inspired by the proof of the folk theorem for NEs in repeated games [15, 23].

Contributions. Our contributions are twofold. First, we present constructions for *finite-memory* NEs for reachability, shortest path and Büchi games that apply to arbitrary arenas, bypassing the finite-arena requirement that is central in existing approaches. More precisely, for these three types of games, we show that from any NE, we can derive another NE where all strategies are finite-memory and such that the same players accomplish their objective, without increasing their cost for shortest path games. In other words, our constructions are general and can be used to match or improve any NE cost profile.

Second, for reachability and shortest-path games, we provide memory bounds that are *independent* of the size of the arena which are quadratic in the number of players.

Our key observation is that it is not necessary to fully implement the punishment mechanism: some deviations do not warrant switching to punishing strategies. This allows us to encode only part of the information in the memory instead of an entire play.

Related work. We refer to the survey [9] for an extensive bibliography on games played on finite graphs, to [8] for a survey centred around reachability games and to [14] as a general reference on games on graphs. We discuss three research directions related to this work.

The first direction is related to *computational problems* for NEs. In the settings we consider, NEs are guaranteed to exist. However, NEs where no player satisfies their objective can coexist with NEs where all players satisfy their objective [25, 26]. A classical problem is to decide if there exists a constrained NE, i.e., such that certain players satisfy their objective in the qualitative case or such that the cost incurred by players is bounded from above in a quantitative case (e.g., [10, 2]). Deciding the existence of a constrained NE is NP-complete for reachability and shortest path games [6] and is in P for Büchi objectives [26].

Second, the construction of our finite-memory NEs rely on *characterisations* of plays resulting from NEs. Their purpose is to ensure that the punishment mechanism described above can be used to guarantee the stability of an equilibrium. In general, these characterisations can be useful from an algorithmic perspective; deciding the existence of a constrained NE boils down to finding a play that satisfies the characterisation. Characterisations appear in the literature for NEs [26, 27, 2], but also for other types of equilibria, e.g., subgame perfect equilibria [5] and secure equilibria [10].

Finally, there exists a body of work dedicated to better understanding the complexity of optimal strategies in zero-sum games. We mention [17] for memoryless strategies, and [3] and [4] for finite-memory strategies in finite and infinite arenas respectively. In finite arenas, for the finite-memory case, a key notion is *arena-independent* finite-memory strategies, i.e., strategies based on a memory structure that is sufficient to win in all arenas whenever possible. In this work, the finite-memory strategies we propose actually depend on the arena; only their size does not. We also mention [19]: in games on finite arenas with objectives from a given class, finite-memory NEs exist if certain conditions on the corresponding zero-sum games hold.

Outline. Due to space constraints, we only provide an overview of our work: technical details can be found in the full paper [20]. This work is structured as follows. In Sect. 2, we summarise prerequisite definitions. We establish the existence of memoryless punishing strategies by studying zero-sum games in Sect. 3. Characterisations of NEs are provided in Sect. 4. We prove our main results on finite-memory NEs in reachability and shortest path games in Sect. 5. Finally, Sect. 6 is dedicated to the corresponding result for Büchi games.

2 Preliminaries

Notation. We write \mathbb{N} , \mathbb{R} for the sets of natural and real numbers respectively, and let $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty, -\infty\}$ and $\overline{\mathbb{N}} = \mathbb{N} \cup \{+\infty\}$. For any $n \in \mathbb{N}$, $n \geq 1$, we let $\llbracket n \rrbracket = \{1, \dots, n\}$.

Games. Let (V, E) be a directed graph where V is a (possibly infinite) set of vertices and $E \subseteq V \times V$ is an edge relation. For any $v \in V$, we write $\text{Succ}_E(v) = \{v' \in V \mid (v, v') \in E\}$ for the set of successor vertices of v . An n -player arena is a tuple $\mathcal{A} = ((V_i)_{i \in \llbracket n \rrbracket}, E)$, where $(V_i)_{i \in \llbracket n \rrbracket}$ is a partition of V . We assume that there are no deadlocks in the arenas we consider, i.e., for all $v \in V$, $\text{Succ}_E(v)$ is not empty. We write \mathcal{P}_i for player i .

A play starts in an initial vertex and proceeds as follows. At each round of the game, the player controlling the current vertex selects a successor of this vertex and the current vertex is updated accordingly. The play continues in this manner infinitely. Formally, a *play* of \mathcal{A} is an infinite sequence $v_0 v_1 \dots \in V^\omega$ such that $(v_\ell, v_{\ell+1}) \in E$ for all $\ell \in \mathbb{N}$. For a play $\pi = v_0 v_1 \dots$ and $\ell \in \mathbb{N}$, we let $\pi_{\geq \ell} = v_\ell v_{\ell+1} \dots$ denote the suffix of π from position ℓ and $\pi_{\leq \ell} = v_0 \dots v_\ell$ denote the prefix of π up to position ℓ . A *history* is any finite non-empty prefix of a play. We write $\text{Plays}(\mathcal{A})$ and $\text{Hist}(\mathcal{A})$ for the set of plays and histories of \mathcal{A} respectively. For $i \in \llbracket n \rrbracket$, we let $\text{Hist}_i(\mathcal{A}) = \text{Hist}(\mathcal{A}) \cap V^* V_i$. For any history $h = v_0 \dots v_r$, we let $\text{first}(h)$ and $\text{last}(h)$ respectively denote v_0 and v_r . For any play π , $\text{first}(\pi)$ is defined similarly.

We formalise the goal of a player in two ways. In the qualitative case, we describe the goal of a player by a set of plays, called an *objective*. We say that a play π satisfies an objective Ω if $\pi \in \Omega$. For quantitative specifications, we assign to each play a quantity using a *cost function* $\text{cost}_i: \text{Plays}(\mathcal{A}) \rightarrow \overline{\mathbb{R}}$ that \mathcal{P}_i intends to minimise. Any goal expressed by an objective Ω can be encoded using a cost function cost_i which assigns 0 to plays in Ω and 1 to others; aiming to minimise this cost is equivalent to aiming to satisfy the objective. For this reason, we present further definitions using cost functions, and explicitly mention when notions are specific to objectives.

A *game* is an arena augmented with the goals of each player. Formally, a game is a tuple $\mathcal{G} = (\mathcal{A}, (\text{cost}_i)_{i \in \llbracket n \rrbracket})$ where \mathcal{A} is an arena and, for all $i \in \llbracket n \rrbracket$, cost_i is the cost function of \mathcal{P}_i . The *cost profile* of a play π is $(\text{cost}_i(\pi))_{i \in \llbracket n \rrbracket}$. Given two plays π and π' , we say that the cost profile of π is preferable to that of π' if $\text{cost}_i(\pi) \leq \text{cost}_i(\pi')$ for all $i \in \llbracket n \rrbracket$.

Objectives and costs. We consider a qualitative and quantitative formulation for the goal of reaching a target, and the goal of infinitely often reaching a target. Let $T \subseteq V$ denote a set of target vertices. We often refer to the set T as a *target*.

We first consider the reachability objective, which expresses the goal of reaching T . Formally, the *reachability objective* (for T) $\text{Reach}(T)$ is defined by the set $\{v_0v_1v_2\dots \in \text{Plays}(\mathcal{A}) \mid \exists \ell \in \mathbb{N}, v_\ell \in T\}$. The complement of the reachability objective $\text{Safe}(T) = \text{Plays}(\mathcal{A}) \setminus \text{Reach}(T)$, which expresses the goal of avoiding T , is called the *safety* objective.

Second, we introduce a cost function formalising the goal of reaching a target as soon as possible. In this context, we assign (non-negative) weights to edges via a *weight function* $w: E \rightarrow \mathbb{N}$, which model, e.g., the time taken when traversing an edge. The weight function is extended to histories as follows; for $h = v_0 \dots v_r \in \text{Hist}(\mathcal{A})$, we let $w(h) = \sum_{\ell=0}^{r-1} w((v_\ell, v_{\ell+1}))$. We define the *truncated sum* cost function (for T and w), for all plays $\pi = v_0v_1\dots$, by $\text{TS}_w^T(\pi) = w(\pi_{\leq r})$ if $r = \min\{\ell \in \mathbb{N} \mid v_\ell \in T\}$ exists and $\text{TS}_w^T(\pi) = +\infty$ otherwise.

Finally, we define the Büchi objective, expressing the goal of reaching a target infinitely often. Formally, the *Büchi objective* (for T) $\text{Büchi}(T)$ is defined by the set $\{v_0v_1v_2\dots \in \text{Plays}(\mathcal{A}) \mid \forall \ell \in \mathbb{N}, \exists \ell' \geq \ell, v_{\ell'} \in T\}$. The complement of a Büchi objective is a co-Büchi objective: the *co-Büchi* objective (for T), which expresses the goal of visiting T finitely often, is defined as $\text{coBüchi}(T) = \text{Plays}(\mathcal{A}) \setminus \text{Büchi}(T)$.

We refer to games where all players have a reachability objective (resp. a truncated sum cost function, a Büchi objective) as *reachability* (resp. *shortest path*, *Büchi*) *games*.

Let $T_1, \dots, T_n \subseteq V$ be targets for each player and $\pi = v_0v_1\dots \in \text{Plays}(\mathcal{A})$. For reachability and shortest path games, we introduce the notation $\text{VisPl}_{T_1, \dots, T_n}(\pi) = \{i \in \llbracket n \rrbracket \mid \pi \in \text{Reach}(T_i)\}$ as the set of players whose targets are visited in π and $\text{VisPos}_{T_1, \dots, T_n}(\pi) = \{\min\{\ell \in \mathbb{N} \mid v_\ell \in T_i\} \mid i \in \text{VisPl}(\pi)\}$ as the set of earliest positions at which targets are visited along π . For Büchi games, we define $\text{InfPl}_{T_1, \dots, T_n}(\pi) = \{i \in \llbracket n \rrbracket \mid \pi \in \text{Büchi}(T_i)\}$ as the set of players whose target is visited infinitely often in π . When T_1, \dots, T_n are clear from the context, we omit them.

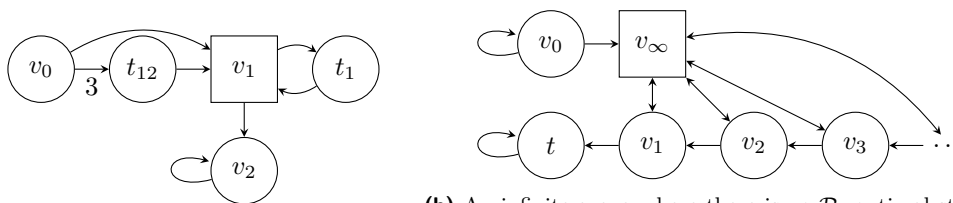
Strategies. Strategies describe the decisions of players during a play. These choices may depend on the past, and not only the current vertex of the play. Formally, a *strategy* of \mathcal{P}_i in an arena \mathcal{A} is a function $\sigma_i: \text{Hist}_i(\mathcal{A}) \rightarrow V$ such that for all histories $h \in \text{Hist}_i(\mathcal{A})$, $(\text{last}(h), \sigma_i(h)) \in E$. A *strategy profile* is a tuple $\sigma = (\sigma_i)_{i \in \llbracket n \rrbracket}$, where σ_i is a strategy of \mathcal{P}_i for all $i \in \llbracket n \rrbracket$. To highlight the role of \mathcal{P}_i , we sometimes write $\sigma = (\sigma_i, \sigma_{-i})$, where σ_{-i} denotes the strategy profile of the players other than \mathcal{P}_i .

A play $\pi = v_0v_1v_2\dots$ is *consistent* with a strategy σ_i of \mathcal{P}_i if for all $\ell \in \mathbb{N}$, $v_\ell \in V_i$ implies $v_{\ell+1} = \sigma_i(\pi_{\leq \ell})$. A play is *consistent* with a strategy profile if it is consistent with all strategies of the profile. Given an initial vertex v_0 and a strategy profile σ , there is a unique play $\text{Out}(\sigma, v_0)$ from v_0 that is consistent with σ , called the *outcome* of σ from v_0 .

We identify two classes of strategies of interest in this work. A strategy σ_i is *memoryless* if the moves it prescribes depend only on the current vertex, i.e., if for all $h, h' \in \text{Hist}_i(\mathcal{A})$, if $\text{last}(h) = \text{last}(h')$, then $\sigma_i(h) = \sigma_i(h')$. We view memoryless strategies as functions $V_i \rightarrow V$.

A strategy is *finite-memory* if it can be encoded by a Mealy machine, i.e., a finite automaton with outputs. A *Mealy machine* (for \mathcal{P}_i) is a tuple $\mathcal{M} = (M, m_{\text{init}}, \text{up}, \text{nxt}_i)$ where M is a finite set of memory states, m_{init} is an initial memory state, $\text{up}: M \times V \rightarrow M$ is a memory update function and $\text{nxt}_i: M \times V_i \rightarrow V$ is a next-move function.

To describe the strategy induced by a Mealy machine, we first define the iterated update function $\widehat{\text{up}}: V^* \rightarrow M$ by induction. We write ε for the empty word. We let $\widehat{\text{up}}(\varepsilon) = m_{\text{init}}$ and for all $wv \in V^*$, $\widehat{\text{up}}(wv) = \text{up}(\widehat{\text{up}}(w), v)$. The strategy $\sigma_i^{\mathcal{M}}$ induced by \mathcal{M} is defined, for all histories $h = h'v \in \text{Hist}_i(\mathcal{A})$, by $\sigma_i^{\mathcal{M}}(h) = \text{nxt}_i(\widehat{\text{up}}(h'), v)$.



(a) A weighted arena with several NEs. (b) An infinite arena where there is no \mathcal{P}_2 optimal strategy from v_∞ in the zero-sum shortest path game with $T = \{t\}$.

Figure 1 Two weighted arenas. Circles and squares respectively denote \mathcal{P}_1 and \mathcal{P}_2 vertices. Edge labels denote their weight and unlabelled edges have a weight of 1.

We say that a finite-memory strategy σ_i has *memory size* $b \in \mathbb{N}$ if there is some Mealy machine $(M, m_{\text{init}}, \text{up}, \text{nxt}_i)$ encoding σ_i with $|M| = b$ and b is the smallest such number.

► **Remark 1.** Some authors define the updates of Mealy machines using edges rather than vertices. Any vertex-update Mealy machine can directly be seen as an edge-update Mealy machine. The converse is not true. In particular, a vertex-update Mealy machine representation of a strategy can require a larger size than an equivalent edge-update Mealy machine.

Nash equilibria. Let $\mathcal{G} = (\mathcal{A}, (\text{cost}_i)_{i \in \llbracket n \rrbracket})$ be a game and v_0 be an initial vertex. Given a strategy profile $\sigma = (\sigma_i)_{i \in \llbracket n \rrbracket}$, we say that a strategy τ_i of \mathcal{P}_i is a *profitable deviation* (with respect to σ from v_0) if $\text{cost}_i(\text{Out}((\tau_i, \sigma_{-i}), v_0)) < \text{cost}_i(\text{Out}(\sigma, v_0))$. A *Nash equilibrium* (NE) from v_0 is a strategy profile such that no player has a profitable deviation. Equivalently, σ is an NE from v_0 if, for all $i \in \llbracket n \rrbracket$ and all plays π consistent with σ_{-i} starting in v_0 , $\text{cost}_i(\pi) \geq \text{cost}_i(\text{Out}(\sigma, v_0))$. In general, NEs with incomparable cost profiles may coexist.

► **Example 2.** Consider the shortest path game played on the arena depicted in Fig. 1a where $T_1 = \{t_{12}, t_1\}$ and $T_2 = \{t_{12}\}$. The memoryless strategy profile (σ_1, σ_2) with $\sigma_1(v_0) = t_{12}$ and $\sigma_2(v_1) = v_2$ is an NE from v_0 with cost profile $(3, 3)$. Another NE from v_0 would be the memoryless strategy profile (σ'_1, σ'_2) such that $\sigma'_1(v_0) = v_1$ and $\sigma'_2(v_1) = t_1$; the cost profile of its outcome is $(2, +\infty)$, which is incomparable with $(3, 3)$. ◻

Zero-sum games. In a zero-sum game, two players compete with opposing goals. Formally, a two-player zero-sum game is a two-player game $\mathcal{G} = (\mathcal{A}, (\text{cost}_1, \text{cost}_2))$ where \mathcal{A} is a two-player arena and $\text{cost}_2 = -\text{cost}_1$. We usually shorten the notation of a zero-sum game to $\mathcal{G} = (\mathcal{A}, \text{cost}_1)$ due to the definition.

Let $v_0 \in V$. If $\inf_{\sigma_1} \sup_{\sigma_2} \text{cost}_1(\text{Out}((\sigma_1, \sigma_2), v_0)) = \sup_{\sigma_2} \inf_{\sigma_1} \text{cost}_1(\text{Out}((\sigma_1, \sigma_2), v_0))$, where σ_i is quantified over the strategies of \mathcal{P}_i , we refer to the above as the *value* of v_0 and denote it by $\text{val}(v_0)$. A game is *determined* if the value is defined in all vertices.

A strategy σ_1 of \mathcal{P}_1 (resp. σ_2 of \mathcal{P}_2) is said to ensure $\alpha \in \mathbb{R}$ from a vertex v_0 if all plays π consistent with σ_1 (resp. σ_2) from v_0 are such that $\text{cost}_1(\pi) \leq \alpha$ (resp. $\text{cost}_1(\pi) \geq \alpha$). A strategy of \mathcal{P}_i is *optimal* from $v_0 \in V$ if it ensures $\text{val}(v_0)$ from v_0 . A strategy is a *uniform optimal* strategy if it ensures $\text{val}(v)$ from v for all $v \in V$. Optimal strategies do not necessarily exist, even if the value does.

► **Example 3.** Consider the two-player zero-sum game played on the weighted arena illustrated in Fig. 1b where the cost function of \mathcal{P}_1 is $\text{TS}_w^{\{t\}}$. Let $\alpha \in \mathbb{N} \setminus \{0\}$. It holds that $\text{val}(v_\alpha) = \alpha$. On the one hand, \mathcal{P}_1 can ensure a cost of α from v_α by moving leftward in the illustration. On the other hand, \mathcal{P}_2 can ensure a cost of α from v_α with the memoryless strategy that

moves from v_∞ to v_α . It follows that this same memoryless strategy of \mathcal{P}_2 ensures $\alpha + 1$ from v_∞ . We conclude that $\text{val}(v_\infty) = +\infty$. However, \mathcal{P}_2 cannot prevent t from being reached from v_∞ , despite its infinite value. Therefore, \mathcal{P}_2 does not have an optimal strategy. \square

If the goal of \mathcal{P}_1 is formulated by an objective Ω , we say that a strategy σ_1 of \mathcal{P}_1 (resp. σ_2 of \mathcal{P}_2) is *winning* from v_0 if all plays consistent with it from v_0 satisfy Ω (resp. $\text{Plays}(\mathcal{A}) \setminus \Omega$). The set of vertices from which \mathcal{P}_1 (resp. \mathcal{P}_2) has a winning strategy is called their *winning region* denoted by $W_1(\Omega)$ (resp. $W_2(\text{Plays}(\mathcal{A}) \setminus \Omega)$). A strategy σ_1 of \mathcal{P}_1 (resp. σ_2 of \mathcal{P}_2) is a *uniform winning* strategy if it is winning from all vertices in $W_1(\Omega)$ (resp. $W_2(\text{Plays}(\mathcal{A}) \setminus \Omega)$).

Given an n -player game $\mathcal{G} = (\mathcal{A}, (\text{cost}_i)_{i \in [n]})$ where $\mathcal{A} = ((V_i)_{i \in [n]}, E)$, we define the *coalition game (for \mathcal{P}_i)* as the game opposing \mathcal{P}_i to the coalition of the other players, formally defined as the two-player zero-sum game $\mathcal{G}_i = (\mathcal{A}_i, \text{cost}_i)$ where $\mathcal{A}_i = ((V_i, V \setminus V_i), E)$. We write \mathcal{P}_{-i} to refer to the coalition of players other than \mathcal{P}_i .

We also refer to two-player zero-sum games where the objective (resp. cost function) of \mathcal{P}_1 is a reachability objective (resp. a truncated sum cost function, a Büchi objective) as reachability (resp. shortest path, Büchi) games.

3 Zero-sum games: punishing strategies

In this section, we present results on strategies in zero-sum games. They are of interest for the classical punishment mechanism used to construct NEs (described in Sect. 1). Intuitively, this mechanism functions as follows: if some player deviates from the intended outcome of the NE, the other players coordinate as a coalition to prevent the player from having a profitable deviation. The strategy of the coalition used to sabotage the deviating player is called a *punishing strategy*.

We explain that we can always find memoryless punishing strategies. First, we recall classical results on reachability and Büchi games. Second, we describe memoryless punishing strategies for shortest path games. We fix a two-player arena $\mathcal{A} = ((V_1, V_2), E)$ and a target $T \subseteq V$ for the remainder of this section.

Reachability and Büchi games. Zero-sum reachability games enjoy *memoryless determinacy*: they are determined and for both players, there exist *memoryless uniform winning strategies*. Furthermore, any vertex of \mathcal{P}_2 that is winning for \mathcal{P}_2 has a successor in this winning region. Any strategy of \mathcal{P}_2 that selects only such successors can be shown to be winning from any vertex in their winning region. The statements above follow, e.g., from the proof of [21, Proposition 2.18]. We summarise this information in the following theorem.

► **Theorem 4.** *Both players have memoryless uniform winning strategies in reachability games. Let $\mathcal{G} = (\mathcal{A}, \text{Reach}(T))$, $W_2(\text{Safe}(T))$ be the winning region of \mathcal{P}_2 in \mathcal{G} , $v_0 \in W_2(\text{Safe}(T))$ and σ_2 be a strategy of \mathcal{P}_2 . If for all histories $h \in \text{Hist}_2(\mathcal{A})$ starting in v_0 containing only vertices of $W_2(\text{Safe}(T))$, we have $\sigma_2(h) \in W_2(\text{Safe}(T))$, then σ_2 is winning from v_0 .*

Büchi games also enjoy memoryless determinacy. It follows from the memoryless determinacy of parity games [13], a class of objectives subsuming Büchi objectives.

► **Theorem 5.** *Both players have memoryless uniform winning strategies in Büchi games.*

Shortest path games. Let $w: E \rightarrow \mathbb{N}$ be a weight function and $\mathcal{G} = (\mathcal{A}, \text{TS}_w^T)$ be a zero-sum shortest path game. First, we remark that \mathcal{G} is determined. It can be shown using the determinacy of games with open objectives [16]. Furthermore, \mathcal{P}_1 has a memoryless uniform

optimal strategy. Intuitively, this is because \mathcal{P}_1 has no need to remember the past, and only should follow a shortest path to a target from the current vertex. Although \mathcal{P}_2 does not necessarily have an optimal strategy (Ex. 3), it can be shown that there exists a family of \mathcal{P}_2 memoryless strategies labelled by non-negative integers that are winning from any vertex in the winning region of \mathcal{P}_2 in the reachability game $(\mathcal{A}, \text{Reach}(T))$ and that ensure the minimum of the integer and the value of the vertex from any other vertex. This information is summarised in the following theorem. Its proof is provided in the full paper [20].

► **Theorem 6.** *The game \mathcal{G} is determined. A memoryless uniform optimal strategy exists for \mathcal{P}_1 . For all $\alpha \in \mathbb{N}$, there exists a memoryless strategy σ_2^α of \mathcal{P}_2 such that, for all $v \in V$: (i) σ_2^α is winning from v for \mathcal{P}_2 in the game $(\mathcal{A}, \text{Reach}(T))$ if $v \in W_2(\text{Safe}(T))$ and (ii) σ_2^α ensures a cost of at least $\min\{\text{val}(v), \alpha\}$.*

4 Characterising Nash equilibria outcomes

We provide characterisations of plays that are outcomes of NEs in reachability, Büchi and shortest path games. These characterisations relate to the corresponding zero-sum games: they roughly state that a play is an NE outcome if and only if the cost incurred by a player from a vertex of the play is less than the value of said vertex in the coalition game opposing the player to the others. We provide a characterisation for reachability and Büchi games, and then a characterisation for shortest path games. We fix an arena $\mathcal{A} = ((V_i)_{i \in \llbracket n \rrbracket}, E)$ and targets $T_1, \dots, T_n \subseteq V$ for this entire section. Proofs of the results below are presented in the full paper [20].

Reachability and Büchi games. We first consider reachability and Büchi games: their respective NE outcome characterisations are close. Let $\mathcal{G} = (\mathcal{A}, (\Omega_i)_{i \in \llbracket n \rrbracket})$ be a reachability or Büchi game. We denote by $W_i(\Omega_i)$ the winning region of the first player of the coalition game $\mathcal{G}_i = (\mathcal{A}_i, \Omega_i)$, in which \mathcal{P}_i is opposed to the other players. The characterisation follows. It relies on the existence of punishing strategies. An identical characterisation for finite arenas can be found in [11].

► **Theorem 7.** *Assume \mathcal{G} is a reachability (resp. Büchi) game. Let $\pi = v_0 v_1 \dots$ be a play. Then π is the outcome of an NE from v_0 if and only if, for all $i \in \llbracket n \rrbracket \setminus \text{VisPI}(\pi)$ (resp. $i \in \llbracket n \rrbracket \setminus \text{InfPI}(\pi)$), $v_\ell \notin W_i(\text{Reach}(T_i))$ (resp. $v_\ell \notin W_i(\text{Büchi}(T_i))$) for all $\ell \in \mathbb{N}$.*

Shortest path games. Let $w: E \rightarrow \mathbb{N}$ be a weight function. We now consider a shortest path game $\mathcal{G} = (\mathcal{A}, (\text{TS}_w^{T_i})_{i \in \llbracket n \rrbracket})$. For any $v \in V$, we denote by $\text{val}_i(v)$ the value of v in the coalition game $\mathcal{G}_i = (\mathcal{A}_i, \text{TS}_w^{T_i})$. We keep the notation $W_i(\text{Reach}(T_i))$ of the previous section.

In reachability and Büchi games, Thm. 7 indicates that the value in coalition games (i.e., who wins) is sufficient to characterise NE outcomes. It is also the case in finite arenas for shortest path games [6, Theorem 15]. However, it is not in arbitrary arenas.

► **Example 8.** Let us consider the arena depicted in Fig. 1b and let $T_1 = \{t\}$ and $T_2 = \{v_0\}$. It holds that $\text{val}_1(v_0) = +\infty$ (it follows from $\text{val}_1(v_\infty) = +\infty$ which is shown in Ex. 3). Therefore, the cost of all suffixes of the play v_0^ω for \mathcal{P}_1 matches the value of their first vertex v_0 . However, for any strategy profile resulting in v_0^ω from v_0 , \mathcal{P}_1 has a profitable deviation in moving to v_∞ and using a reachability strategy to ensure a finite cost. ◻

A value-based characterisation fails because of vertices $v \in W_i(\text{Reach}(T_i))$ such that $\text{val}_i(v)$ is infinite. Despite the infinite value of such vertices, \mathcal{P}_i has a strategy such that their cost is finite no matter the behaviour of the others. To obtain a characterisation, we impose additional conditions on players whose targets are not visited that are related to reachability games. We obtain the following characterisation.

► **Theorem 9.** *Let $\pi = v_0v_1\dots$ be a play. Then π is an outcome of an NE from v_0 in \mathcal{G} if and only (i) for all $i \in \llbracket n \rrbracket \setminus \text{VisPI}(\pi)$ and $\ell \in \mathbb{N}$, we have $v_\ell \notin W_i(\text{Reach}(T_i))$ and (ii) for all $i \in \text{VisPI}(\pi)$ and all $\ell \leq r_i$, it holds that $\text{TS}_w^{T_i}(\pi_{\geq \ell}) \leq \text{val}_i(v_\ell)$ where $r_i = \min\{r \in \mathbb{N} \mid v_r \in T_i\}$.*

5 Finite-memory Nash equilibria in reachability games

In this section, we describe finite-memory strategy profiles for NEs with memory bounds that depend solely on the numbers of players in reachability and shortest path games. These finite-memory strategy profiles behave differently to those described for the characterisations in Thm. 7 and Thm. 9. Intuitively, following a deviation of \mathcal{P}_i , the coalition \mathcal{P}_{-i} does not necessarily switch to a punishing strategy for \mathcal{P}_i . Instead, they may attempt to keep following a suffix of the equilibrium's original outcome if the deviation does not appear to prevent it.

This section is structured as follows. We illustrate the constructions for reachability and shortest path games with examples in Sect. 5.1. In Sect. 5.2, we provide templates for finite-memory NEs and technical notions to define them. In Sect. 5.3, we show that we can derive, from any NE outcome, another with a simple structure and provide the general constructions for finite-memory NEs with memory size independent of the arena. The details of the last two sections are provided in the full paper [20].

We fix an arena $\mathcal{A} = ((V_i)_{i \in \llbracket n \rrbracket}, E)$, target sets $T_1, \dots, T_n \subseteq V$ and a weight function $w: E \rightarrow \mathbb{N}$ for the remainder of this section. We introduce a new operator in this section. Given two histories $h = v_0 \dots v_\ell$ and $h' = v_\ell v_{\ell+1} \dots v_r$, we let $h \cdot h' = v_0 \dots v_\ell v_{\ell+1} \dots v_r$; we say that $h \cdot h'$ is the *combination* of h and h' . The combination $h \cdot \pi$ of a history h and a play π such that $\text{last}(h) = \text{first}(\pi)$ is defined similarly.

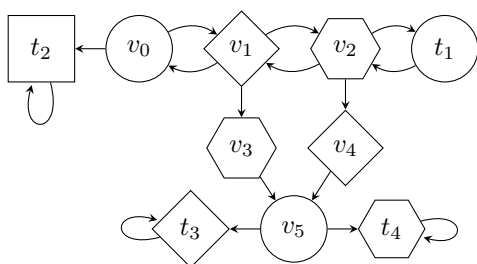
5.1 Examples

In this section, we illustrate the upcoming construction for finite-memory NEs for both settings of interest. We start with a reachability game.

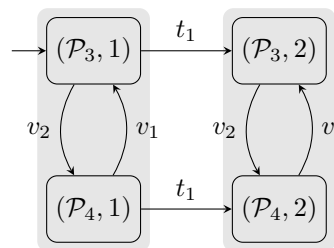
► **Example 10.** We consider the game on the arena depicted in Fig. 2a where the objective of \mathcal{P}_i is $\text{Reach}(\{t_i\})$ for $i \in \llbracket 4 \rrbracket$. We present a finite-memory NE with outcome $\pi = v_0v_1v_2t_1v_2v_1v_0t_2^\omega$ to illustrate the idea behind the upcoming construction.

First, observe that π can be seen as the combination of the simple history $\text{sg}_1 = v_0v_1v_2t_1$ and the simple lasso $\text{sg}_2 = t_1v_2v_1v_0t_2^\omega$. The simple history sg_1 connects the initial vertex to the first visited target, and the simple lasso sg_2 connects the first target to the second and contains the suffix of the play. Therefore, if we were not concerned with the stability of the equilibrium, the outcome π could be obtained by using a finite-memory strategy profile where all strategies are defined by a Mealy machine with state space $\llbracket 2 \rrbracket$. Intuitively, these strategies would follow sg_1 while remaining in their first memory state 1, then, when t_1 is visited, they would update their memory state to 2 and follow sg_2 .

We build on these simple Mealy machines with two states. We include additional information in each memory state. We depict a suitable Mealy machine state space and update scheme in Fig. 2b. The rectangles grouping together states (\mathcal{P}_3, j) and (\mathcal{P}_4, j)



(a) An arena. Circles, squares, diamonds and hexagons are resp. $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4$ vertices.



(b) An illustration of the update scheme of a Mealy machine. Transitions that do not change the memory state are omitted.

■ **Figure 2** A reachability game and a representation of a Mealy machine update scheme suitable for an NE from v_0 .

represent the memory state j of the simpler Mealy machine, for $j \in \llbracket 2 \rrbracket$. The additional information roughly encodes the last player to act among the players whose objective is not satisfied in π . More precisely, an update is performed from the memory state (\mathcal{P}_i, j) only if the vertex fed to the Mealy machine appears in \mathbf{sg}_j for $j \in \llbracket 2 \rrbracket$.

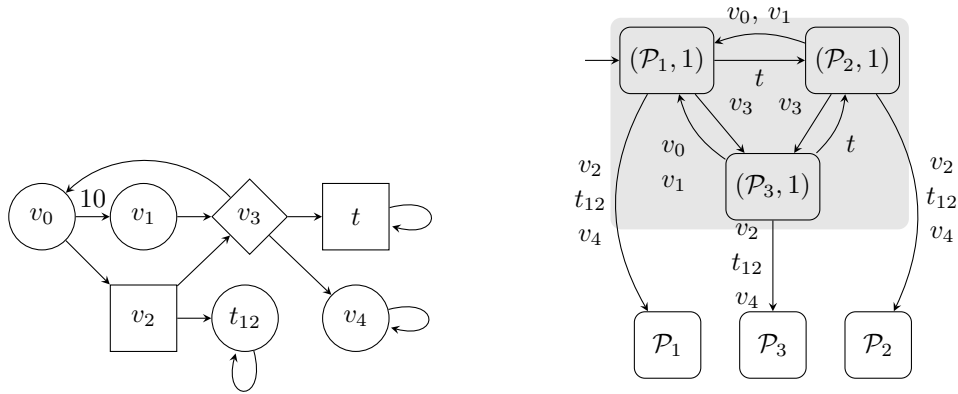
By construction, if \mathcal{P}_i (among \mathcal{P}_3 and \mathcal{P}_4) deviates and exits the set of vertices of \mathbf{sg}_j when in a memory state of the form (\cdot, j) , then the memory updates to (\mathcal{P}_i, j) and does not change until the play returns to some vertex of \mathbf{sg}_j (which is not possible here due to the structure of the graph, but may be in general). For instance, assume \mathcal{P}_3 moves from v_1 to v_3 after the history $h = v_0v_1v_2t_1v_2v_1$. Then the memory after h is in state $(\mathcal{P}_3, 2)$ and no longer changes from there on.

It remains to explain how the next-move function of the Mealy machine should be defined to ensure an NE. Essentially, for a state of the form (\mathcal{P}_i, j) and vertices in \mathbf{sg}_j , we assign actions as in the simpler two-state Mealy machine described previously. On the other hand, for a state of the form (\mathcal{P}_i, j) and a vertex not in \mathbf{sg}_j , we use a memoryless punishing strategy against \mathcal{P}_i . In this particular case, we need only specify what \mathcal{P}_1 should do in v_5 . Naturally, in memory state (\mathcal{P}_i, j) , \mathcal{P}_1 should move to the target of the other player. It is essential to halt memory updates for vertices v_3 and v_4 to ensure the correct player is punished.

We close this example with comments on the structure of the Mealy machine. Assume the memory state is of the form (\mathcal{P}_i, j) . If a deviation occurs and leads to a vertex of \mathbf{sg}_j other than the intended one, then the other players will continue trying to progress along \mathbf{sg}_j and do not specifically try punishing the deviating player. Similarly, if after a deviation leaving the set of vertices of \mathbf{sg}_j (from which point the memory is no longer updated until this set is rejoined), a vertex of \mathbf{sg}_j is visited again, then the players resume trying to progress along this history and memory updates resume. In other words, these finite-memory strategies do not pay attention to all deviations and do not have dedicated memory that commit to punishing deviating players for the remainder of a play after a deviation. \lrcorner

We now give an example for the shortest path case. The Mealy machines we propose are slightly larger in this case. We argue that it may be necessary to commit to a punishing strategy if the set of vertices of the history the players want to progress along is left. This requires additional memory states. Our example shows that it may be necessary to punish deviations from players whose targets are visited, as they can possibly improve their cost.

► **Example 11.** We consider the shortest path game on the weighted arena depicted in Fig. 3a where the target of \mathcal{P}_i is $T_i = \{t, t_{12}\}$ for $i \in \llbracket 2 \rrbracket$ and $T_3 = \{t\}$ for \mathcal{P}_3 . We argue that a finite-memory NE with outcome $\pi = v_0v_1v_3t^\omega$ from v_0 cannot be obtained by adapting the construction of Ex. 10. We provide an alternative construction that builds on the same ideas.



(a) A weighted arena. Edge labels indicate their weight. Unlabelled edges have a weight of 1. (b) An illustration of the update scheme of a Mealy machine. Transitions that do not change the memory state are omitted.

■ **Figure 3** A shortest path game and a representation of a Mealy machine update scheme suitable for some NE from v_0 .

In this case, π is a simple lasso, much like the second part of the play in the previous example. First, let us assume a Mealy machine similar to that of Ex. 10, i.e., such that it tries to progress along π whenever it is in one of its vertices. The update scheme of such a Mealy machine would be obtained by removing the transitions to states of the form \mathcal{P}_i from Fig. 3b (replacing them by self-loops).

If \mathcal{P}_3 uses a strategy based on such a Mealy machine, then \mathcal{P}_1 has a profitable deviation from v_0 : if \mathcal{P}_1 moves from v_0 to v_2 , then either \mathcal{P}_1 incurs a cost of 2 if \mathcal{P}_2 moves to t_{12} from v_2 or a cost of 3 if \mathcal{P}_2 moves to v_3 as \mathcal{P}_3 would then move to t by definition of the Mealy machine. To circumvent this issue, if \mathcal{P}_i exits the set of vertices of π , we update the memory to the punishment state \mathcal{P}_i . This results in the update scheme depicted in Fig. 3b. Next-move functions to obtain an NE can be defined as follows, in addition to the expected behaviour to obtain π : for \mathcal{P}_2 , $\text{nxt}_2((\mathcal{P}_1, 1), v_2) = \text{nxt}_2(\mathcal{P}_1, v_2) = v_3$ and for \mathcal{P}_3 , $\text{nxt}_3(\mathcal{P}_1, v_3) = v_4$.

Similarly to the previous example, players do not explicitly react to deviations that move to vertices of π ; if \mathcal{P}_3 deviates after reaching v_3 and moves back to v_0 , the memory of the other players does not update to state \mathcal{P}_3 . Intuitively, there is no need to switch to a punishing strategy for \mathcal{P}_3 as going back to the start of the intended outcome is more costly than conforming to it, preventing the existence of a profitable deviation.

This example differs slightly from the general construction below, which would decompose π into two parts: a history $v_0v_1v_3t$ from the initial vertex to the first target and the suffix t^ω of the play after all targets are visited. In full generality, this separation is needed [20]. \square

5.2 Segments and strategies

In Sect. 5.1, we illustrated that the finite-memory Nash equilibria we construct in reachability and shortest path games share a common structure. In this section, we provide the generic part of these Mealy machines. We first introduce decompositions of plays. We then partially define Mealy machines encoding strategies based on so-called simple decompositions.

Decomposing plays. We fix $\pi = v_0v_1\dots \in \text{Plays}(\mathcal{A})$ for this whole section. We first introduce some terminology. A play or history is *simple* if no vertex occurs twice within. A play is a *simple lasso* if it is of the form pc^ω where $pc \in \text{Hist}(\mathcal{A})$ is a simple history. A

segment of π is either a suffix $\pi_{\geq \ell}$ of π ($\ell \in \mathbb{N}$) or any history of the form $v_\ell \dots v_{\ell'}$ ($\ell \leq \ell'$). We denote segments by \mathbf{sg} to avoid distinguishing finite and infinite segments of plays in the following. A segment is *simple* if it is a simple history, a simple play or a simple lasso.

A (finite) *segment decomposition* of π is a sequence $\mathcal{D} = (\mathbf{sg}_1, \dots, \mathbf{sg}_k)$ where \mathbf{sg}_j is a history for all $j < k$, \mathbf{sg}_k is a suffix of π , $\text{last}(\mathbf{sg}_j) = \text{first}(\mathbf{sg}_{j+1})$ for all $j < k$ and $\pi = \mathbf{sg}_1 \cdot \dots \cdot \mathbf{sg}_k$. We assume that among the histories of a decomposition, there are none of the form $h = v$, i.e., there are no trivial segments. The segment decomposition \mathcal{D} is *simple* if all segments within are simple. If there is some NE outcome with a given cost profile, we show that there is an NE outcome with a preferable cost profile that admits a simple segment decomposition. To obtain finite-memory NEs, we build on NE outcomes with a simple segment decomposition.

Finite-memory decomposition-based strategies. Let $\pi = v_0 v_1 \dots \in \text{Plays}(\mathcal{A})$ be a play that admits a simple segment decomposition $\mathcal{D} = (\mathbf{sg}_1, \dots, \mathbf{sg}_k)$. We partially define a Mealy machine that serves as the basis for the finite-memory NEs described in the next section.

The memory state space is made of pairs of the form (\mathcal{P}_i, j) for some $j \in \llbracket k \rrbracket$. We do not consider all such pairs, e.g., it is not necessary in Ex. 10. Therefore, we parameterise our construction by a non-empty set of players $I \subseteq \llbracket n \rrbracket$. We consider the memory state space $M^{I, \mathcal{D}} = \{\mathcal{P}_i \mid i \in I\} \times \llbracket k \rrbracket$. The initial state $m_{\text{init}}^{I, \mathcal{D}}$ is any state of the form $(\mathcal{P}_i, 1) \in M^{I, \mathcal{D}}$.

The update function $\text{up}^{I, \mathcal{D}}$ behaves similarly to Fig. 2b. It keeps track of the last player in I to have moved and the current segment. Formally, for any $(\mathcal{P}_i, j) \in M^{I, \mathcal{D}}$ and vertex v occurring in \mathbf{sg}_j , we let $\text{up}^{I, \mathcal{D}}((\mathcal{P}_i, j), v) = (\mathcal{P}_{i'}, j')$ where (i) i' is such that $v \in V_{i'}$ if $v \in \bigcup_{i'' \in I} V_{i''}$ and otherwise $i' = i$, and (ii) $j' = j + 1$ if $j < k$ and $v = \text{last}(\mathbf{sg}_j)$ and $j' = j$ otherwise. Updates from (\mathcal{P}_i, j) for a vertex that does not appear in \mathbf{sg}_j are left undefined.

The next-move function $\text{nxt}_i^{I, \mathcal{D}}$ of \mathcal{P}_i proposes the next vertex of the current segment. Formally, given a memory state $(\mathcal{P}_{i'}, j) \in M^{I, \mathcal{D}}$ and a vertex $v \in V_i$ that occurs in \mathbf{sg}_j , we let $\text{nxt}_i^{I, \mathcal{D}}((\mathcal{P}_{i'}, j), v)$ be the vertex occurring after v in \mathbf{sg}_{j+1} if $j < k$ and $v = \text{last}(\mathbf{sg}_j)$, and otherwise we let it be the vertex occurring after v in \mathbf{sg}_j . Like updates, the next-move function is left undefined in memory states (\mathcal{P}_i, j) for a vertex that does not appear in \mathbf{sg}_j .

5.3 Nash equilibria

We now present finite-memory NEs with memory bounds depending only on the number of players. We first derive, given an NE outcome, another NE outcome that admits a simple decomposition. We impose additional technical properties on these decompositions to define NEs with strategies based on them. We then define finite-memory strategies based on these simple decompositions by extending the partial definition above to obtain finite-memory NEs. We deal with reachability games then shortest paths games.

Simplifying outcomes. We explain that from any NE outcome in a shortest path game, we can derive another NE outcome with a preferable cost profile that admits a simple segment decomposition. The result extends to reachability games. We consider two cases.

First, we consider NE outcomes such that all players who see their target have the initial vertex of the outcome in it, generalising the case where no players see their targets. From these outcomes, we can directly derive an NE outcome that is a simple lasso or simple play.

► **Lemma 12.** *Let $\pi' \in \text{Plays}(\mathcal{A})$ be the outcome of an NE from $v_0 \in V$ in a shortest path game $\mathcal{G} = (\mathcal{A}, (\text{TS}_w^{T_i})_{i \in \llbracket n \rrbracket})$ such that $\text{VisPos}(\pi') \subseteq \{0\}$. There exists an NE outcome $\pi \in \text{Plays}(\mathcal{A})$ from v_0 with the same cost profile as π' that is a simple lasso or a simple play and such that $\text{VisPos}(\pi) \subseteq \{0\}$. In particular, π has the simple segment decomposition (π) .*

We now consider NE outcomes such that some player sees their target later than in the initial vertex. In this case, we can derive an NE outcome with a simple decomposition such that the simple histories of the decomposition connect the first target elements that are visited. We impose a technical condition on these simple histories, to ensure that no player has a profitable deviation by skipping ahead in a segment.

► **Lemma 13.** *Let π' be the outcome of an NE from $v_0 \in V$ in a shortest path game $\mathcal{G} = (\mathcal{A}, (\text{TS}_w^{T_i})_{i \in \llbracket n \rrbracket})$. Assume that $|\text{VisPos}(\pi') \setminus \{0\}| = k > 0$. There exists an NE outcome π from v_0 with $\text{VisPos}(\pi) \setminus \{0\} = \{\ell_1 < \dots < \ell_k\}$ that admits a simple segment decomposition $(\text{sg}_1, \dots, \text{sg}_{k+1})$ such that (i) $(\text{sg}_1, \dots, \text{sg}_k \cdot \text{sg}_{k+1})$ is also a simple decomposition of π ; (ii) for all $j \in \llbracket k \rrbracket$, $\text{sg}_1 \cdot \dots \cdot \text{sg}_j = \pi_{\leq \ell_j}$; (iii) for all $j \in \llbracket k \rrbracket$, $w(\text{sg}_j)$ is minimum among all histories that share their first and last vertex with sg_j and traverse a subset of the vertices occurring in sg_j ; and (iv) for all $i \in \llbracket n \rrbracket$, $\text{TS}_w^{T_i}(\pi) \leq \text{TS}_w^{T_i}(\pi')$.*

Reachability games. We fix a reachability game $\mathcal{G} = (\mathcal{A}, (\text{Reach}(T_i))_{i \in \llbracket n \rrbracket})$. We construct finite-memory NEs by extending the partially-defined Mealy machines of Sect. 5.2 and generalising the strategies presented in Ex. 10. The main result is the following.

► **Theorem 14.** *Let σ' be an NE from a vertex v_0 . There exists a finite-memory NE σ from v_0 such that $\text{VisPl}(\text{Out}(\sigma, v_0)) = \text{VisPl}(\text{Out}(\sigma', v_0))$ where each strategy of σ has a memory size of at most n^2 . Precisely, a memory size of $\max\{1, n - |\text{VisPl}(\text{Out}(\sigma', v_0))|\} \cdot \max\{1, |\text{VisPos}(\text{Out}(\sigma', v_0)) \setminus \{0\}|\}$ suffices.*

Proof sketch. Consider an NE outcome π and its simple decomposition $\mathcal{D} = (\text{sg}_1, \dots, \text{sg}_k)$ provided by Lem. 12 and Lem. 13 (point (i)) for $\text{Out}(\sigma', v_0)$. The general idea of the construction is to use the state space $M^{I, \mathcal{D}}$ where $I \subseteq \llbracket n \rrbracket$ is the set of players who do not see their targets if it is non-empty, or a single arbitrary player if all players see their target. Let $i' \in I$ and $j \in \llbracket k \rrbracket$. We extend $\text{up}^{I, \mathcal{D}}$ to leave unchanged the memory state if in state $(\mathcal{P}_{i'}, j)$ whenever the current vertex is not in sg_j . In this same situation, the next-move functions $\text{next}_i^{I, \mathcal{D}}$ are extended to assign moves from a uniform memoryless winning strategy of the second player in the coalition game $\mathcal{G}_{i'} = (\mathcal{A}_{i'}, \text{Reach}(T_{i'}))$ (which exists by Thm. 4). The equilibrium's stability is a consequence of Thm. 7 and the second statement of Thm. 4. ◀

We remark that Thm. 14 provides a memory bound that is *linear in the number of players* when no players see their target and when all players see their target.

► **Corollary 15.** *If there exists an NE from v_0 such that no (resp. all) players see their target in its outcome, then there is a finite-memory NE from v_0 such that no (resp. all) players see their target in its outcome such that all strategies have a memory size of at most n .*

Shortest path games. We now fix a shortest path game $\mathcal{G} = (\mathcal{A}, (\text{TS}_w^{T_i})_{i \in \llbracket n \rrbracket})$. We provide an alternative generalisation of the partially-defined Mealy machines described in Sect. 5.2, this time generalising the strategies provided in Ex. 11. Ex. 11 shows that only altering the construction of Thm. 14 to also monitor (and punish) players whose targets are visited is not sufficient. To overcome this, we change the approach so players commit to punishing any player who exits the current segment of the intended outcome.

► **Theorem 16.** *Let σ' be an NE from a vertex v_0 . There exists a finite-memory NE σ from v_0 such that $\text{VisPl}(\text{Out}(\sigma, v_0)) = \text{VisPl}(\text{Out}(\sigma', v_0))$ and, for all $i \in \llbracket n \rrbracket$, $\text{TS}_w^{T_i}(\text{Out}(\sigma, v_0)) \leq \text{TS}_w^{T_i}(\text{Out}(\sigma', v_0))$ where each strategy of σ has a memory size of at most $n^2 + 2n$. Precisely, a memory size of $n \cdot (|\text{VisPos}(\text{Out}(\sigma', v_0)) \setminus \{0\}| + 2)$ suffices.*

Proof sketch. The construction is similar to that of Thm. 14. We obtain a suitable NE outcome π and its decomposition \mathcal{D} in the same way, and build on $M^{I,\mathcal{D}}$, $\text{up}^{I,\mathcal{D}}$, $\text{next}_i^{I,\mathcal{D}}$ with $I = \llbracket n \rrbracket$. If we apply Lem. 13 here, we consider the first decomposition the lemma provides and not the one from point (i) of the lemma. Intuitively, merging the last two segments, as done in the decomposition from Lem. 13, point (i), prevents players from reacting to deviations within the merged segment and could enable a profitable deviation.

In this case, instead of freezing memory updates if the current segment is left when the memory state is of the form (\mathcal{P}_i, j) , the memory switches to a memory state \mathcal{P}_i that is never left. This switch can only occur if \mathcal{P}_i deviates. The next-move function, for this memory state, assigns moves from a punishing strategy obtained from the coalition game $\mathcal{G}_i = (\mathcal{A}_i, \text{TS}_w^{T_i})$ by Thm. 6, chosen to hinder \mathcal{P}_i , ensuring that in case of a deviation, \mathcal{P}_i 's cost is at least that of the original outcome.

The conditions imposed on outcomes of Lem. 13 (notably condition (iii)) and the characterisation of Thm. 9 imply the correctness of this construction. Condition (iii) of Lem. 13 ensures that a player cannot reach their target with a lesser cost by traversing the vertices within a segment in another order, whereas the characterisation of Thm. 9 guarantees that the punishing strategies sabotage deviating players sufficiently. ◀

In this case, Thm. 16 provides the memory bound $2n$ if no players visit their target. However, the construction of Thm. 9 applies to such NEs in shortest path games.

▶ **Corollary 17.** *If there exists an NE from v_0 such that no players see their target in its outcome, then there is a finite-memory NE from v_0 such that no players see their target in its outcome such that all strategies have a memory size of at most n .*

6 Finite-memory Nash equilibria in Büchi games

We now present finite-memory NEs for Büchi games. We illustrate in Sect. 6.1 that the constructions for reachability and shortest path games do not extend directly to Büchi games. We build on the techniques of Sect. 5.2 to provide finite-memory NEs in Sect. 6.2. We fix an arena $\mathcal{A} = ((V_i)_{i \in \llbracket n \rrbracket}, E)$, targets $T_1, \dots, T_n \subseteq V$ and the Büchi game $\mathcal{G} = (\mathcal{A}, (\text{Büchi}(T_i))_{i \in \llbracket n \rrbracket})$ for this entire section. Proofs and details of this section are presented in the full version of the paper [20].

6.1 Examples

For reachability and shortest path games, we relied on simple segment decompositions between consecutive targets along some NE outcome to obtain finite-memory NEs. Our strategies based on these decompositions do not explicitly punish players who deviate. We show that this can be problematic when dealing with Büchi objectives.

▶ **Example 18.** Consider the game on the arena depicted in Fig. 4a where the objectives of \mathcal{P}_1 and \mathcal{P}_2 are $\text{Büchi}(\{v_1\})$ and $\text{Büchi}(\{v_2\})$ respectively. The play $v_0 v_1 v_2^\omega$ is the outcome of an NE by Thm. 7. To mimic the construction underlying Thm. 14 and Thm. 16, we would consider a finite-memory strategy based on the decomposition $\mathcal{D} = (v_0 v_1 v_2, v_2^\omega)$. However, if \mathcal{P}_2 uses such a strategy, \mathcal{P}_1 would enforce their objective via the memoryless strategy σ_1 such that $\sigma_2(v_1) = v_0$, resulting in the outcome $(v_0 v_1)^\omega$, as \mathcal{P}_2 would not punish the deviation. ◻

In the previous example, the issue with the proposed decomposition lies with the occurrence of a target of \mathcal{P}_2 , whose objective is not satisfied in the intended outcome, within some segment of the decomposition. To circumvent this issue, we construct strategies that



(a) An arena where a direct segment-based approach fails to obtain an NE.

(b) An arena on which players should commit to punishing strategies once a segment is left.

■ **Figure 4** Two arenas. Circle, squares and diamonds are resp. \mathcal{P}_1 , \mathcal{P}_2 and \mathcal{P}_3 vertices.

follow two phases in the following section. In their first phase, these strategies punish any deviations from the intended outcome. For their second phase, we adapt the strategies of Section 5.2. To ensure no profitable deviations may appear in the second phase, we start it at a point of the intended outcome from which no more targets of losing players occur.

We close this section by illustrating that the punishing mechanism used for finite-memory NEs in reachability games does not suffice, i.e., players must commit to their punishing strategies once some player exits the current segment in the second phase mentioned above.

► **Example 19.** Consider the game on the arena depicted in Fig. 4b where the objectives of \mathcal{P}_1 , \mathcal{P}_2 and \mathcal{P}_3 are Büchi($\{v_1\}$) and Büchi($\{v_2, v_4\}$) and Büchi($\{v_4\}$) respectively. The play $\pi = (v_0 v_1 v_3)^\omega$ is the outcome of an NE by Thm. 7. Consider a \mathcal{P}_1 strategy based on the decomposition (π) that uses the punishment mechanism we introduced for reachability games. Then the behaviour of \mathcal{P}_1 does not change if \mathcal{P}_2 moves from v_0 to v_2 instead of v_1 : \mathcal{P}_1 would move from v_1 to v_3 and then to v_0 . It follows that \mathcal{P}_2 would have a profitable deviation no matter the strategy of \mathcal{P}_3 .

To obtain an NE where all players use strategies based on the decomposition (π), \mathcal{P}_1 , must commit to a punishing strategy for \mathcal{P}_2 if v_2 is visited. For \mathcal{P}_2 and \mathcal{P}_3 we consider the memoryless strategies σ_2 and σ_3 such that $\sigma_2(v_0) = \sigma_3(v_2) = v_1$. It is easy to check that this is an NE. \lrcorner

6.2 Finite-memory Nash equilibria

In this section, we establish the counterpart of Thm. 14 and Thm. 16 for Büchi games. It is split in two statements (Thm. 21 and Thm. 23) that depend on the form of the outcome of the considered NE. Each case is considered in a dedicated section. First, we consider NE outcomes with a vertex that occurs infinitely often within. We then show the result for NE outcomes without infinitely occurring vertices. For both cases, we first provide NE outcomes with a simple structure and then construct corresponding finite-memory NEs.

We consider alternative segment decompositions in this section. These decompositions differ from those defined in Sect. 5 in the following way. First, we allow *infinite* segment decompositions and tolerate decompositions such that their first segment is trivial. We extend the definition of simple segment to include simple cycles.

Throughout this section, we assume without loss of generality that any considered NE outcome π is such that $\text{InfPI}(\pi)$ is not empty. This can be ensured by adding a new player for whom all vertices are targets if necessary.

Outcomes with an infinitely occurring vertex. The first case we consider is a generalisation of the finite-arena case: in a finite arena, all plays contain some infinitely occurring vertex. To obtain finite-memory NEs, we use the two-phase mechanism presented previously with an adaptation of the decomposition-based finite-memory strategies of Section 5.2 that can handle infinite ultimately periodic decompositions.

The following lemma provides NE outcomes with a simple structure on which we rely to define finite-memory NEs.

► **Lemma 20.** *Let π' be the outcome of an NE from $v_0 \in V$ in the Büchi game \mathcal{G} such that some vertex occurs infinitely often in π' and let $k = |\text{InfPI}(\pi')|$. Then there exists an NE outcome π from v_0 with $\text{InfPI}(\pi) = \text{InfPI}(\pi')$ such that π admits an infinite simple segment decomposition $(\text{sg}_0, \text{sg}_1, \dots)$ such that (i) for all $j \geq 1$ and all $i \in \llbracket n \rrbracket \setminus \text{InfPI}(\pi)$, no vertex of T_i occurs in sg_j and (ii) for all $j \geq 1$, $\text{sg}_j = \text{sg}_{j+k}$.*

We now state the main theorem of this section.

► **Theorem 21.** *Let σ' be an NE from a vertex v_0 such that some vertex occurs infinitely often in its outcome. There exists a finite-memory NE σ from v_0 such that $\text{InfPI}(\text{Out}(\sigma, v_0)) = \text{InfPI}(\text{Out}(\sigma', v_0))$. If \mathcal{A} is finite, a memory size of at most $|V| + n^2 + n$ suffices.*

Proof sketch. Let π and $\mathcal{D} = (\text{sg}_0, \text{sg}_1, \dots)$ be given by Lem. 20 for $\text{Out}(\sigma', v_0)$ and let $k = |\text{InfPI}(\pi)|$. We obtain finite-memory NEs via the two-phase finite-memory strategies described in Sect. 6.1. For the first phase, we follow the history sg_0 . For the second phase, we switch to a strategy that is based on the decomposition $\mathcal{D}' = (\text{sg}_1, \text{sg}_2, \dots)$. Although this decomposition is infinite, we can construct a finite-memory strategy based on \mathcal{D} by exploiting its ultimately periodic nature. To achieve this, we alter the definitions of Sect. 5.2: when reading $\text{last}(\text{sg}_k)$ in memory states of the form (\mathcal{P}_i, k) , we update the memory to an appropriate memory state of the form $(\mathcal{P}_{i'}, 1)$.

By completing the behaviour described above with switches to memoryless punishing strategies (Thm. 5) if sg_0 is not accurately simulated or if a player exits the current segment, we obtain a finite-memory NE. The stability of the NE follows from Thm. 7 for deviations that induce the use of punishing strategies and the property that no targets of losing players occur in segments sg_j , $j \geq 1$ for other deviations. ◀

With the classical approach to derive NEs from outcomes with a finite representation (Sect. 1), we can also design finite-memory NEs for outcomes obtained by Lem. 20. If $|V|$ is finite, the resulting strategies of this approach have a memory size of at most $(|V| + 2)n$. It follows our construction is preferable if there are few players compared to vertices.

Outcomes without an infinitely occurring vertex. We now deal with NE outcomes that can only appear in infinite arenas. We once again rely on a two-phase mechanism where the first phase is unchanged. The second phase is loosely based on an infinite decomposition. Intuitively, we allocate infinitely many disjoint segments to a same group of memory state. Due to this, players may not react to someone exiting the current segment.

The following lemma is the counterpart of Lemma 20 for this case.

► **Lemma 22.** *Let π' be the outcome of an NE from $v_0 \in V$ in the Büchi game \mathcal{G} such that no vertex occurs infinitely often in π' . Then there exists an NE outcome π from v_0 with $\text{InfPI}(\pi) = \text{InfPI}(\pi')$ such that π admits an infinite simple segment decomposition $(\text{sg}_0, \text{sg}_1, \dots)$ such that (i) for all $j \geq 1$ and all $i \in \llbracket n \rrbracket \setminus \text{InfPI}(\pi)$, no vertex of T_i occurs in sg_j and (ii) for all $j \neq j'$, sg_j and $\text{sg}_{j'}$ have no vertices in common if j and j' have the same parity.*

We can now state the last theorem of this section.

► **Theorem 23.** *Let σ' be an NE from a vertex v_0 such that no vertex occurs infinitely often in its outcome. There exists a finite-memory NE σ from v_0 such that $\text{InfPI}(\text{Out}(\sigma, v_0)) = \text{InfPI}(\text{Out}(\sigma', v_0))$.*

Proof sketch. Let π be the play and $\mathcal{D} = (\mathbf{sg}_0, \mathbf{sg}_1, \dots)$ be the decomposition of π given by Lem. 22 for $\text{Out}(\sigma', v_0)$. We rely once again on strategies with two phases. The first phase is defined exactly as for Thm. 21. For the second phase, we also adapt the definitions of Sect. 5.2. The update and next-move function in the original definitions are defined for each memory state of the form (\mathcal{P}_i, j) based on the segment \mathbf{sg}_j . In this case, we define the update and next-move functions in memory states of the form $(\mathcal{P}_i, 1)$ (resp. $(\mathcal{P}_i, 2)$) based on all odd segments (resp. all even segments besides \mathbf{sg}_0) of \mathcal{D} simultaneously, such that when the end of an even segment is reached in a memory state of the form $(\mathcal{P}_i, 2)$, the memory is updated to a state of the form $(\mathcal{P}_{i'}, 1)$. The fact all odd (resp. even) segments traverse pairwise disjoint set of vertices ensures that the next-move function is well-defined.

If at some point in the second phase, a vertex that does not occur in an odd segment is read in a memory state $(\mathcal{P}_i, 1)$, the memory is updated to a punishing state \mathcal{P}_i , such that players attempt to punish \mathcal{P}_i with a memoryless strategy (Thm. 5). We proceed similarly for the even case. The resulting finite-memory strategy profile is an NE from v_0 . On the one hand, any deviation such that the memory never updates to a punishing state must only have vertices that occur in segment \mathbf{sg}_j with $j \neq 0$ in the limit. By choice of \mathcal{D} , this deviation cannot be profitable. Otherwise, it can be argued that the punishing strategy does in fact sabotage the deviating player, so long as their objective is not satisfied in π , by Thm. 7. ◀

References

- 1 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 921–962. Springer, 2018. doi:10.1007/978-3-319-10575-8_27.
- 2 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure Nash equilibria in concurrent deterministic games. *Log. Methods Comput. Sci.*, 11(2), 2015. doi:10.2168/LMCS-11(2:9)2015.
- 3 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Games where you can play optimally with arena-independent finite memory. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/lmcs-18(1:11)2022.
- 4 Patricia Bouyer, Mickael Randour, and Pierre Vandenhove. Characterizing omega-regularity through finite-memory determinacy of games on infinite graphs. *TheoretCS*, 2, 2023. doi:10.46298/theoretics.23.1.
- 5 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, Jean-François Raskin, and Marie van den Bogaard. The complexity of subgame perfect equilibria in quantitative reachability games. *Log. Methods Comput. Sci.*, 16(4), 2020. URL: <https://lmcs.episciences.org/6883>.
- 6 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, and Nathan Thomasset. On relevant equilibria in reachability games. *J. Comput. Syst. Sci.*, 119:211–230, 2021. doi:10.1016/j.jcss.2021.02.009.
- 7 Thomas Brihaye, Julie De Pril, and Sven Schewe. Multiplayer cost games with simple Nash equilibria. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science, International Symposium, LFCS 2013, San Diego, CA, USA, January 6-8, 2013. Proceedings*, volume 7734 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2013. doi:10.1007/978-3-642-35722-0_5.
- 8 Thomas Brihaye, Aline Goeminne, James C. A. Main, and Mickael Randour. Reachability games and friends: A journey through the lens of memory and complexity (invited talk). In Patricia Bouyer and Srikanth Srinivasan, editors, *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2023, December 18-20, 2023, IIT Hyderabad, Telangana, India*, volume 284 of *LIPICs*, pages 1:1–1:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.FSTTCS.2023.1.




- 9 Véronique Bruyère. Computer aided synthesis: A game-theoretic approach. In Émilie Charlier, Julien Leroy, and Michel Rigo, editors, *Developments in Language Theory - 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2017. doi:10.1007/978-3-319-62809-7_1.
- 10 Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Secure equilibria in weighted games. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 26:1–26:26. ACM, 2014. doi:10.1145/2603088.2603109.
- 11 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.121.
- 12 Julie De Pril. *Equilibria in Multiplayer Cost Games*. PhD thesis, UMONS, 2013. URL: http://math.umons.ac.be/staff/ancien/DePril.Julie/thesis_Julie_DePril.pdf.
- 13 E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. In *FOCS*, pages 328–337. IEEE Computer Society, 1988. doi:10.1109/SFCS.1988.21949.
- 14 Nathanaël Fijalkow, Nathalie Bertrand, Patricia Bouyer-Decitre, Romain Brenguier, Arnaud Carayol, John Fearnley, Hugo Gimbert, Florian Horn, Rasmus Ibsen-Jensen, Nicolas Markey, Benjamin Monmege, Petr Novotný, Mickael Randour, Ocan Sankur, Sylvain Schmitz, Olivier Serre, and Mateusz Skomra. Games on graphs. *CoRR*, abs/2305.10546, 2023. doi:10.48550/arXiv.2305.10546.
- 15 James Friedman. A non-cooperative equilibrium for supergames. *Review of Economic Studies*, 38(1):1–12, 1971. URL: <https://EconPapers.repec.org/RePEc:oup:restud:v:38:y:1971:i:1:p:1-12>.
- 16 David Gale and Frank M Stewart. Infinite games with perfect information. *Contributions to the Theory of Games*, 2(245-266):2–16, 1953.
- 17 Hugo Gimbert and Wieslaw Zielonka. Games where you can play optimally without any memory. In *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, pages 428–442, 2005. doi:10.1007/11539452_33.
- 18 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.
- 19 Stéphane Le Roux and Arno Pauly. Extending finite-memory determinacy to multi-player games. *Inf. Comput.*, 261:676–694, 2018. doi:10.1016/j.ic.2018.02.024.
- 20 James C. A. Main. Arena-independent memory bounds for Nash equilibria in reachability games. *CoRR*, abs/2310.02142, 2023. doi:10.48550/ARXIV.2310.02142.
- 21 René Mazala. Infinite games. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*, pages 23–42. Springer, 2001. doi:10.1007/3-540-36387-4_2.
- 22 John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950. doi:10.1073/pnas.36.1.48.
- 23 Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. The MIT Press, 1994.
- 24 Mickael Randour. Automated synthesis of reliable and efficient systems through game theory: A case study. In *Proc. of ECCS 2012*, Springer Proceedings in Complexity XVII, pages 731–738. Springer, 2013. doi:10.1007/978-3-319-00395-5_90.

- 25 Michael Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2006. doi:10.1007/11944836_21.
- 26 Michael Ummels. The complexity of Nash equilibria in infinite multiplayer games. In Roberto M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2008. doi:10.1007/978-3-540-78499-9_3.
- 27 Michael Ummels and Dominik Wojtczak. The complexity of Nash equilibria in limit-average games. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2011. doi:10.1007/978-3-642-23217-6_32.

Weighted HOM-Problem for Nonnegative Integers

Andreas Maletti   

Institute of Computer Science, Leipzig University, Germany

Andreea-Teodora Nász   

Institute of Computer Science, Leipzig University, Germany

Erik Paul  

Institute of Computer Science, Leipzig University, Germany

Abstract

The HOM-problem asks whether the image of a regular tree language under a given tree homomorphism is again regular. It was recently shown to be decidable by GODOY, GIMÉNEZ, RAMOS, and ÁLVAREZ. In this paper, the \mathbb{N} -weighted version of this problem is considered and its decidability is proved. More precisely, it is decidable in polynomial time whether the image of a regular \mathbb{N} -weighted tree language under a nondeleting, nonerasing tree homomorphism is regular.

2012 ACM Subject Classification Theory of computation \rightarrow Quantitative automata; Theory of computation \rightarrow Computability; Theory of computation \rightarrow Tree languages; Theory of computation \rightarrow Grammars and context-free languages

Keywords and phrases Weighted Tree Automaton, Decision Problem, Subtree Equality Constraint, Tree Homomorphism, HOM-Problem, Weighted Tree Grammar, Weighted HOM-Problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.51

Related Version *Previous Version*: <https://arxiv.org/abs/2311.11067>

1 Introduction

The prominent model of finite-state string automata has seen a variety of extensions in the past few decades. Notably, their qualitative evaluation was generalized to a quantitative one to yield the weighted automata of [26]. These automata are able to neatly represent process factors such as costs, consumption of resources or time, and probabilities related to the input, and have been extensively studied [25]. Semirings [17, 18] present themselves as a well suited algebraic structure for evaluating the weights because of their generality as well as their reasonable computational efficiency that is derived from distributivity.

Parallel to this development, finite-state string automata have been generalized to process other forms of inputs such as infinite words [23], graphs [3] and trees [5]. Finite-state tree automata and the *regular tree languages* they generate have been widely researched since their introduction in [7, 27, 28]. These models prove to be useful in a variety of application areas including natural language processing [19], image generation [8], and compiler construction [29]. Many applications require both features: trees as more expressive input structure and quantitative evaluation. This led to the development of several weighted tree automata (WTA) models. An extensive overview can be found in [12, Chapter 9].

Finite-state tree automata have serious limitations; notably, they cannot guarantee that two specific subtrees are always equal in the accepted trees provided that those subtrees can be arbitrarily large. Similarly finite-state string automata cannot ensure that the number of a 's and b 's in the accepted words is equal. These restrictions are well-known [13], and the mentioned drawback was addressed in [21], where an extension was proposed that can explicitly require certain subtrees to be equal or different. This extension is very convenient in the study of tree transformations [12] that can duplicate subtrees, and it is also the primary tool used in the seminal paper [15] to prove the decidability of the *HOM-problem*.

 © Andreas Maletti, Andreea-Teodora Nász, and Erik Paul;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 51; pp. 51:1–51:17



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



The HOM-problem, a previously long-standing open question in the study of tree languages, asks whether the image of a regular tree language under a given tree homomorphism is also regular. The image need not be regular since tree homomorphisms can also duplicate subtrees. Indeed, if this duplication ability is removed from the tree homomorphism (e.g., linear tree homomorphisms), then the image is always regular [13]. The HOM-problem was recently solved in [15, 16], where the image is represented by a tree automaton with constraints, for which it is then determined whether it generates a regular tree language. Later the HOM-problem was shown to be EXPTIME-complete [6].

In the weighted case, decidability of the HOM-problem remains wide open. Previous research on the preservation of regularity in the weighted setting [4, 9, 10, 11] focuses on cases that explicitly exclude the duplication power of the homomorphism. Recently, the weighted HOM-problem over zero-sum free semirings was addressed, but only solved for significantly restricted inputs [22]. In the present work, we prove that the HOM-problem for regular weighted tree languages over the semiring \mathbb{N} of nonnegative integers can be decided in polynomial time. The proof outline is as follows: Consider such a regular \mathbb{N} -weighted tree language and a nondeleting, nonerasing tree homomorphism. First, we represent this image efficiently using an extension (WTGh) of weighted tree automata that permits constraints [20]. Next, we ask whether this WTGh generates a regular weighted tree language. This semantic problem is reduced to an easier, **essentially** syntactic property: the *large duplication property*. In turn, this allows us to prove decidability of the weighted HOM-problem in polynomial time by directly proving it for the large duplication property. If the WTGh for the homomorphic image does not have this property, then we give an effective construction of an equivalent \mathbb{N} -weighted tree automaton without constraints (albeit in exponential time), thus proving its regularity. Otherwise, we use a pumping lemma presented in [20] and isolate a strictly non-regular part from the WTGh. The most challenging part of our proof and our main technical contribution is showing that the remaining trees in the homomorphic image cannot compensate for the non-regular behavior of this isolated part. For this, we employ RAMSEY's theorem [24] to identify a witness for the non-regularity of the whole weighted tree language.

Compared to the unweighted case where the HOM-problem is EXPTIME-complete [6], the \mathbb{N} -weighted HOM-problem can be decided in polynomial time. Both proofs reduce the (non)regularity of the homomorphic image in question to a decidable property of the tree grammar with constraints representing it; however, the unweighted regularity notion is very different from the corresponding notion for weighted tree languages over \mathbb{N} . Unlike the BOOLEAN semiring \mathbb{B} , which corresponds to the unweighted case, the semiring \mathbb{N} can be embedded into a field, which allows us to apply methods of linear algebra. The *large duplication property*, to which we successfully reduce the \mathbb{N} -weighted HOM-problem, is certainly necessary, but insufficient in the unweighted case. This is due to the fact that the BOOLEAN semiring is idempotent, which permits covering an irregular tree language with the help of a regular one (e.g., the union $L \cup T_\Sigma = T_\Sigma$ of an irregular tree language L with the regular tree language T_Σ of all trees is again regular). We will prove that such covers cannot happen in the semiring \mathbb{N} of nonnegative integers. As a consequence, the *large duplication property* is necessary and sufficient for non-regularity of weighted tree languages over \mathbb{N} . In summary, our overall strategy for approaching the HOM-problem is similar to [15], but the required notions and details of the proofs significantly differ.

Tree automata and grammars with constraints are applied in domains such as automated deduction [2] or security verification [1]. In this context, studying quantitative extensions of these models is naturally relevant. Tree structures are also central in XML, and homomorphic transformations on trees allow us to modify the codes while preserving the hierarchical structure. Moreover, the HOM-problem plays a role in the context of term rewriting [14]: For

a term rewrite system, the set of normal forms (i.e., terms to which no rule can be applied) can be expressed as the complement of a homomorphic image; a better understanding of these images can help generalize known results in this field.

2 Preliminaries

We denote the set of nonnegative integers by \mathbb{N} . For $i, j \in \mathbb{N}$ we let $[i, j] = \{k \in \mathbb{N} \mid i \leq k \leq j\}$ and $[j] = [1, j]$. Let Z be an arbitrary set. The cardinality of Z is denoted by $|Z|$, and the set of words over Z (i.e., the set of ordered finite sequences of elements of Z) is denoted by Z^* .

Trees, Substitutions, and Contexts

A *ranked alphabet* (Σ, rk) consists of a finite set Σ and a mapping $\text{rk}: \Sigma \rightarrow \mathbb{N}$ that assigns a rank to each symbol of Σ . If there is no risk of confusion, then we denote the ranked alphabet (Σ, rk) by Σ alone. We write $\sigma^{(k)}$ to indicate that $\text{rk}(\sigma) = k$. Moreover, for every $k \in \mathbb{N}$ we let $\Sigma_k = \text{rk}^{-1}(k)$ and $\text{rk}(\Sigma) = \max\{k \in \mathbb{N} \mid \Sigma_k \neq \emptyset\}$ be the maximal rank of symbols of Σ . Let $X = \{x_i \mid i \in \mathbb{N}\}$ be a countable set of (formal) variables. For every $n \in \mathbb{N}$, we let $X_n = \{x_i \mid i \in [n]\}$. Given a ranked alphabet Σ and a set Z , the set $T_\Sigma(Z)$ of Σ -trees indexed by Z is the smallest set such that $Z \subseteq T_\Sigma(Z)$ and $\sigma(t_1, \dots, t_k) \in T_\Sigma(Z)$ for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \dots, t_k \in T_\Sigma(Z)$. We abbreviate $T_\Sigma(\emptyset)$ simply by T_Σ , and any subset $L \subseteq T_\Sigma$ is called a *tree language*.

Let Σ be a ranked alphabet, Z a set, and consider a tree $t \in T_\Sigma(Z)$. The set $\text{pos}(t)$ of *positions of t* is defined by $\text{pos}(z) = \{\varepsilon\}$ for all $z \in Z$ and by $\text{pos}(\sigma(t_1, \dots, t_k)) = \{\varepsilon\} \cup \{iw \mid i \in [k], w \in \text{pos}(t_i)\}$ for all $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \dots, t_k \in T_\Sigma(Z)$. With their help, we define the *size* ‘ $\text{size}(t)$ ’ and *height* ‘ $\text{ht}(t)$ ’ of t as $\text{size}(t) = |\text{pos}(t)|$ and $\text{ht}(t) = \max_{w \in \text{pos}(t)} |w|$. Positions are partially ordered by the standard prefix order \leq on $[\text{rk}(\Sigma)]^*$, and they are totally ordered by the ascending lexicographic order \preceq on $[\text{rk}(\Sigma)]^*$, in which prefixes are larger; i.e., ε is the largest element. More precisely, for $v, w \in \text{pos}(t)$ if there exists $u \in [\text{rk}(\Sigma)]^*$ with $vu = w$, then we write $v \leq w$, call v a *prefix* of w , and let $v^{-1}w = u$ because u is uniquely determined if it exists. Provided that $u = u_1 \cdots u_n$ with $u_1, \dots, u_n \in [\text{rk}(\Sigma)]$ we also define the *path* $[v, \dots, w]$ from v to w as the sequence $(v, vu_1, vu_1u_2, \dots, w)$ of positions. Any two positions that are \leq -incomparable are called *parallel*.

Given $t, t' \in T_\Sigma(Z)$ and $w \in \text{pos}(t)$, the *label* $t(w)$ of t at w , the *subtree* $t|_w$ of t at w , and the *substitution* $t[t']_w$ of t' into t at w are defined by $z(\varepsilon) = z|_\varepsilon = z$ and $z[t']_\varepsilon = t'$ for all $z \in Z$ and by $t(\varepsilon) = \sigma$, $t(iw') = t_i(w')$, $t|_\varepsilon = t$, $t|_{iw'} = t_i|_{w'}$, $t[t']_\varepsilon = t'$, and $t[t']_{iw'} = \sigma(t_1, \dots, t_{i-1}, t_i[t']_{w'}, t_{i+1}, \dots, t_k)$ for all trees $t = \sigma(t_1, \dots, t_k)$ with $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, $t_1, \dots, t_k \in T_\Sigma(Z)$, all $i \in [k]$, and all $w' \in \text{pos}(t_i)$. For all sets $S \subseteq \Sigma \cup Z$ of symbols, we let $\text{pos}_S(t) = \{w \in \text{pos}(t) \mid t(w) \in S\}$, and we write $\text{pos}_s(t)$ instead of $\text{pos}_{\{s\}}(t)$ for every $s \in \Sigma \cup Z$. The set of variables occurring in t is $\text{var}(t) = \{x \in X \mid \text{pos}_x(t) \neq \emptyset\}$. Finally, consider $n \in \mathbb{N}$ and a mapping $\theta': X_n \rightarrow T_\Sigma(Z)$. Then by substitution, θ' induces a mapping $\theta: T_\Sigma(Z) \rightarrow T_\Sigma(Z)$ defined by $\theta(x) = \theta'(x)$ for every $x \in X_n$, $\theta(z) = z$ for every $z \in Z \setminus X_n$, and $\theta(\sigma(t_1, \dots, t_k)) = \sigma(\theta(t_1), \dots, \theta(t_k))$ for all $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \dots, t_k \in T_\Sigma(Z)$. For $t \in T_\Sigma(Z)$, we denote $\theta(t)$ by $t\theta$ or, more commonly, by $t[x_1 \leftarrow \theta'(x_1), \dots, x_n \leftarrow \theta'(x_n)]$.

Let $\square \notin \Sigma$. A *context* is a tree $C \in T_\Sigma(\square)$ with $\text{pos}_\square(C) \neq \emptyset$. More specifically, we call C an *n -context* if $n = |\text{pos}_\square(C)|$. For an n -context C and $t_1, \dots, t_n \in T_\Sigma$, we define the substitution $C[t_1, \dots, t_n]$ as follows. Let $\text{pos}_\square(C) = \{w_1, \dots, w_n\}$ be the occurrences of \square in C in lexicographic order $w_1 \prec \dots \prec w_n$. Then we let $C[t_1, \dots, t_n] = C[t_1]_{w_1} \cdots [t_n]_{w_n}$.

Tree Homomorphisms and Weighted Tree Grammars

Given ranked alphabets Σ and Γ , let $h': \Sigma \rightarrow T_\Gamma(X)$ be a mapping with $h'(\sigma) \in T_\Gamma(X_k)$ for all $k \in \mathbb{N}$ and $\sigma \in \Sigma_k$. We extend h' to $h: T_\Sigma \rightarrow T_\Gamma$ by $h(\alpha) = h'(\alpha) \in T_\Gamma(X_0) = T_\Gamma$ for all $\alpha \in \Sigma_0$ and $h(\sigma(t_1, \dots, t_k)) = h'(\sigma)[x_1 \leftarrow h(t_1), \dots, x_k \leftarrow h(t_k)]$ for all $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \dots, t_k \in T_\Sigma$. The mapping h is called the *tree homomorphism induced by h'* , and we identify h' and its induced tree homomorphism h . For the complexity analysis of our decision procedure, we define the size of h as $\text{size}(h) = \sum_{\sigma \in \Sigma} \text{size}(h(\sigma))$. We call h *nonerasing* (respectively, *nondeleting*) if $h'(\sigma) \notin X$ (respectively, $\text{var}(h'(\sigma)) = X_k$) for all $k \in \mathbb{N}$ and $\sigma \in \Sigma_k$. In this contribution, we will only consider nonerasing and nondeleting tree homomorphisms $h: T_\Sigma \rightarrow T_\Gamma$, which are therefore *input finitary*; i.e., the preimage $h^{-1}(u)$ is finite for every $u \in T_\Gamma$ since $|t| \leq |u|$ for every $t \in h^{-1}(u)$. Any mapping $A: T_\Sigma \rightarrow \mathbb{N}$ is called *\mathbb{N} -weighted tree language*, and we define the weighted tree language $h_A: T_\Gamma \rightarrow \mathbb{N}$ for every $u \in T_\Gamma$ by $h_A(u) = \sum_{t \in h^{-1}(u)} A(t)$ and call it the *image of A under h* . This definition relies on the tree homomorphism to be input-finitary; otherwise the defining sum is not finite, so the value $h_A(u)$ is not necessarily well-defined.

A *weighted tree grammar with equality constraints* (WTGc) [20] is a tuple $(Q, \Sigma, F, P, \text{wt})$, in which Q is a finite set of *states*, Σ is a ranked alphabet of *input symbols*, $F: Q \rightarrow \mathbb{N}$ assigns a *final weight* to every state, P is a finite set of *productions* of the form (ℓ, q, E) with $\ell \in T_\Sigma(Q) \setminus Q$, $q \in Q$, and finite subset $E \subseteq \mathbb{N}^* \times \mathbb{N}^*$, and $\text{wt}: P \rightarrow \mathbb{N}$ assigns a *weight* to every production. A production $p = (\ell, q, E) \in P$ is usually written $p = \ell \xrightarrow{E} q$ or $p = \ell \xrightarrow[\text{wt}(p)]{E} q$, and the tree ℓ is called its *left-hand side*, q is its *target state*, and E are its *equality constraints*, respectively. Equality constraints $(v, v') \in E$ are also written as $v = v'$. A state $q \in Q$ is *final* if $F(q) \neq 0$.

Next, we recall the *derivation semantics* of WTGc from [20]. Let $(v, v') \in \mathbb{N}^* \times \mathbb{N}^*$ be an equality constraint and $t \in T_\Sigma$. The tree t satisfies (v, v') if and only if $v, v' \in \text{pos}(t)$ and $t|_v = t|_{v'}$, and for a finite set $C \subseteq \mathbb{N}^* \times \mathbb{N}^*$ of equality constraints, we write $t \models C$ if t satisfies all $(v, v') \in C$. Let $G = (Q, \Sigma, F, P, \text{wt})$ be a WTGc. A *sentential form* (for G) is a tree $\xi \in T_\Sigma(Q)$. Given an input tree $t \in T_\Sigma$, sentential forms $\xi, \zeta \in T_\Sigma(Q)$, a production $p = \ell \xrightarrow{E} q \in P$, and a position $w \in \text{pos}(\xi)$, we write $\xi \Rightarrow_{G,t}^{p,w} \zeta$ if $\xi|_w = \ell$, $\zeta = \xi|_w$, and $t|_w \models E$; i.e., the equality constraints E are fulfilled on $t|_w$. A sequence $d = (p_1, w_1) \cdots (p_n, w_n) \in (P \times \mathbb{N}^*)^*$ is a *derivation* (of G) for t if there exist $\xi_0, \dots, \xi_n \in T_\Sigma(Q)$ such that $\xi_0 = t$ and $\xi_{i-1} \Rightarrow_{G,t}^{p_i, w_i} \xi_i$ for all $i \in [n]$. We call d *left-most* if additionally $w_1 \prec w_2 \prec \cdots \prec w_n$. Note that the sentential forms ξ_0, \dots, ξ_n are uniquely determined if they exist, and for any derivation d for t there exists a unique permutation of d that is a left-most derivation for t . We call d *complete* if $\xi_n \in Q$, and in this case we also call it a derivation *to ξ_n* . The set of all complete left-most derivations for t to $q \in Q$ is denoted by $D_G^q(t)$. A complete derivation to some final state is called *accepting*. If for every $p \in P$, there exists a tree $t \in T_\Sigma$, a final state q and a derivation $(p_1, w_1) \cdots (p_m, w_m) \in D_G^q(t)$ such that $F(q) \cdot \prod_{i=1}^m \text{wt}(p_i) \neq 0$ and $p \in \{p_1, \dots, p_m\}$; i.e. if every production is used in an accepting derivation with nonzero weights, then G is *trim*.

Let $d = (p_1, w_1) \cdots (p_n, w_n) \in D_G^q(t)$ for some $t \in T_\Sigma$ and $i \in [n]$. Moreover, let $\{j_1, \dots, j_\ell\}$ be the set $\{j \in [n] \mid w_i \leq w_j\}$ with the indices $j_1 < \cdots < j_\ell$ of those positions of which w_i is a prefix. We refer to $(p_{j_1}, w_i^{-1}w_{j_1}), \dots, (p_{j_\ell}, w_i^{-1}w_{j_\ell})$ as the *derivation for $t|_{w_i}$ incorporated in d* . Conversely, for $w \in \mathbb{N}^*$ we abbreviate the derivation $(p_1, w_1) \cdots (p_n, w_n)$ by wd .

The *weight* of a derivation $d = (p_1, w_1) \cdots (p_n, w_n)$ is defined as $\text{wt}_G(d) = \prod_{i=1}^n \text{wt}(p_i)$. The weighted tree language generated by G , written $\llbracket G \rrbracket: T_\Sigma \rightarrow \mathbb{N}$, is defined for all $t \in T_\Sigma$ by $\llbracket G \rrbracket(t) = \sum_{q \in Q, d \in D_G^q(t)} F(q) \cdot \text{wt}_G(d)$. For $t \in T_\Sigma$ and $q \in Q$, we will often use the

value $\text{wt}_G^q(t)$ defined as $\text{wt}_G^q(t) = \sum_{d \in D_G^q(t)} \text{wt}_G(d)$. Using distributivity, $\llbracket G \rrbracket(t)$ then simplifies to $\llbracket G \rrbracket(t) = \sum_{q \in Q} F(q) \cdot \text{wt}_G^q(t)$. We call two WTGc *equivalent* if they generate the same weighted tree language.

We call a WTGc $(Q, \Sigma, F, P, \text{wt})$ a *weighted tree grammar* (WTG) if $E = \emptyset$ for every production $\ell \xrightarrow{E} q \in P$; i.e., no production utilizes equality constraints. Instead of $\ell \xrightarrow{\emptyset} q$ we also simply write $\ell \rightarrow q$. Moreover, we call a WTGc a *weighted tree automaton with equality constraints* (WTAc) if $\text{pos}_\Sigma(\ell) = \{\varepsilon\}$ for every production $\ell \xrightarrow{E} q \in P$, and a *weighted tree automaton* (WTA) if it is both a WTG and a WTAc. The classes of WTGc and WTAc are equally expressive, and they are strictly more expressive than the class of WTA [20]. We call a weighted tree language *regular* if it is generated by a WTA and *constraint-regular* if it is generated by a WTGc. Productions with weight 0 are obviously useless, so we may assume that $\text{wt}(p) \neq 0$ for every production p . Finally, we define the size of a WTGc as follows.

► **Definition 1.** *Let $G = (Q, \Sigma, F, P, \text{wt})$ be a WTGc and $p = \ell \xrightarrow{E} q \in P$ be a production. We define the height of p as $\text{ht}(p) = \text{ht}(\ell)$ and its size as $\text{size}(p) = \text{size}(\ell)$, the height of P as $\text{ht}(P) = \max_{p \in P} \text{ht}(p)$ and its size as $\text{size}(P) = \sum_{p \in P} \text{size}(p)$, and finally the height of G as $\text{ht}(G) = |Q| \cdot \text{ht}(P)$ and its size as $\text{size}(G) = |Q| + \text{size}(P)$.*

It is known [20] that WTGc of a particular shape can represent homomorphic images of regular weighted tree languages. This subclass of WTGc will be central in our work.

► **Definition 2.** *A WTGc $(Q, \Sigma, F, P, \text{wt})$ is classic if every production $p = \ell \xrightarrow{E} q \in P$ satisfies $E \subseteq \text{pos}_Q(\ell)^2$; i.e., all equality constraints point to the Q -labeled positions of its left-hand side. Without loss of generality, we can assume that every set E of equality constraints is reflexive, symmetric, and transitive, that is, an equivalence relation on a subset $D \subseteq \text{pos}_Q(\ell)$, so not all occurrences of states need to be constrained.*

A classic WTGc is eq-restricted if it has a so-called sink state $\perp \in Q \setminus F$ such that (i) $\sigma(\perp, \dots, \perp) \rightarrow_1 \perp$ belongs to P for all $\sigma \in \Sigma$, and no other productions target \perp , and (ii) for every production $\ell \xrightarrow{E} q$ with $q \neq \perp$, if $\text{pos}_Q(\ell) = \{p_1, \dots, p_n\}$ and $q_i = \ell(p_i)$ for $i \in [n]$, the following conditions hold:

1. *For each $i \in [n]$, the set $\{q_j \mid p_j \in [p_i]_{\equiv_E}\} \setminus \{\perp\}$ is a singleton.*
2. *There exists exactly one $p_j \in [p_i]_{\equiv_E}$ such that $q_j \neq \perp$.*

In other words, an eq-restricted WTGc G has a designated nonfinal sink state $\perp \in Q$ such that $F(\perp) = 0$ as well as $p_\gamma = \gamma(\perp, \dots, \perp) \rightarrow \perp \in P$ and $\text{wt}(p_\gamma) = 1$ for every $\gamma \in \Gamma$. In addition, every production $p = \ell \xrightarrow{E} q \in P$ satisfies the following two properties. First, $E \subseteq \text{pos}_Q(\ell)^2$; i.e., all equality constraints point to the Q -labeled positions of its left-hand side. Second, $\ell(v) = \perp$ and $\ell(w) \neq \perp$ for every $v \in [w']_E \setminus \{w\}$ and w' occurring in E , where $w = \min_{\prec} [w']_E$; i.e., all but the lexicographically least position in each equivalence class of E are guarded by state \perp . Essentially, an eq-restricted WTGc G performs its checks (and charges weights) exclusively on the lexicographically least occurrences of equality-constrained subtrees. All the other subtrees, which by means of the constraint are forced to coincide with another subtree, are simply ignored by the WTGc, which formally means that they are processed in the designated sink state \perp . In the following, we will use \perp to denote such a sink state, and write $Q \cup \{\perp\}$ to explicitly indicate its presence.

To simplify our terminology, we will refer to eq-restricted WTGc simply as *WTGh*.

► **Theorem 3** (see [20, Theorem 5]). *Let $G = (Q, \Sigma, F, P, \text{wt})$ be a trim WTA and $h: T_\Sigma \rightarrow T_\Gamma$ be a nondeleting and nonerasing tree homomorphism. Then there exists a trim WTGh G' with $\llbracket G' \rrbracket = h_{\llbracket G \rrbracket}$. Moreover, $\text{size}(G') \in \mathcal{O}(\text{size}(G) \cdot \text{size}(h))$ and $\text{ht}(G') \in \mathcal{O}(\text{size}(h))$.*

► **Example 4.** Let $G = (Q \cup \{\perp\}, \Gamma, F, P, \text{wt})$ with $Q = \{q, q_f\}$, $\Gamma = \{\alpha^{(0)}, \gamma^{(1)}, \delta^{(3)}\}$, $F(q) = F(\perp) = 0$ and $F(q_f) = 1$, and the following set P of productions.

$$\left\{ \alpha \rightarrow_1 q, \gamma(q) \rightarrow_2 q, \delta(q, \gamma(\perp), q) \xrightarrow{1=2\perp}_1 q_f, \alpha \rightarrow_1 \perp, \gamma(\perp) \rightarrow_1 \perp, \delta(\perp, \perp, \perp) \rightarrow_1 \perp \right\}$$

The WTGc G is a WTGh. It generates the homomorphic image $\llbracket G \rrbracket = h_A$ for the tree homomorphism h induced by the mapping $\alpha \mapsto \alpha$, $\gamma \mapsto \gamma(x_1)$, and $\sigma \mapsto \delta(x_2, \gamma(x_2), x_1)$ applied to the regular weighted tree language $A: T_\Sigma \rightarrow \mathbb{N}$ given by $A(t) = 2^{|\text{pos}_\gamma(t)|}$ for every $t \in T_\Sigma$ with $\Sigma = \{\alpha^{(0)}, \gamma^{(1)}, \sigma^{(2)}\}$. The weighted tree language $\llbracket G \rrbracket$ is itself not regular because its support is clearly not a regular tree language.

The restrictions in the definition of a WTGh allow us to trim it effectively.

► **Lemma 5.** *Let $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ be a WTGh. An equivalent, trim WTGh G' can be constructed in polynomial time.*

Proof. First, recall that we may assume $\text{wt}(p) \neq 0$ for every $p \in P$ because $\text{wt}_G(d) = 0$ for every derivation d of G that contains a production p with $\text{wt}(p) = 0$. For the proof, we employ a simple reachability algorithm. For every $n \in \mathbb{N}$ and $U \subseteq Q$, let

$$Q_0 = \emptyset \quad Q_{n+1} = Q_n \cup \bigcup_{\substack{(\ell \xrightarrow{E} q) \in P \\ \ell \in T_\Sigma(Q_n)}} \{q\} \quad \Pi_U = \bigcup_{\substack{(\ell \xrightarrow{E} q) \in P \\ \ell \in T_\Sigma(U)}} \{(q, q') \in U^2 \mid \text{pos}_{q'}(\ell) \neq \emptyset\} .$$

Since Q is finite, there exists N with $Q_N = Q_{N+1}$. Let $Q' = Q_N$. A straightforward proof shows that $q \in Q'$ if and only if for some $t \in T_\Sigma$ there exists $d \in D_G^q(t)$ with $\text{wt}_G(d) \neq 0$. To ensure the reachability of a final state, we let \triangleleft be the smallest reflexive and transitive relation on Q' that contains $\Pi_{Q'}$. Then $P' = \{\ell \xrightarrow{E} q \in P \mid q \in Q', \exists q_f \in Q': F(q_f) \neq 0, q_f \triangleleft q\}$, and the desired WTGh is simply $G' = (Q \cup \{\perp\}, \Sigma, F, P', \text{wt}|_{P'})$. ◀

3 Substitutions in the Presence of Equality Constraints

This short section recalls from [20] some definitions together with a pumping lemma for WTGh, which will be essential for deciding the integer-weighted HOM-problem. First, we need to refine the substitution of trees such that it complies with existing constraints.

► **Definition 6** (see [20] and cf. [15]). *Let $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ be a WTGh. Moreover, let $q, q' \in Q$, $t, t' \in T_\Sigma$, and $d \in D_G^q(t)$ as well as $d' \in D_G^{q'}(t')$ such that $q \neq \perp \neq q'$ and $d = \underline{d}(p, \varepsilon)$ uses $p = c[q_1, \dots, q_k] \xrightarrow{E, \emptyset} q \in P$ as its final production. For every $i \in [k]$ let $w_i = \text{pos}_{x_i}(c)$ and d_i be the unique left-most derivation for $t_i = t|_{\text{pos}_{x_i}(c)}$ incorporated in d . Finally, for every $u \in T_\Sigma$ let d_u^\perp be the unique left-most derivation for u to \perp . For every $w \in \text{pos}(t)$ at which the production used in d targets q' , we recursively define the derivation substitution $d[\underline{d}']_w$ of d' into d at w and the resulting tree $t[\underline{t}']_w^d$ as follows. If $w = \varepsilon$, then $d[\underline{d}']_\varepsilon = d'$ and $t[\underline{t}']_\varepsilon^d = t'$. Otherwise $w = w_j \underline{w}$ for some $j \in [k]$ and we have*

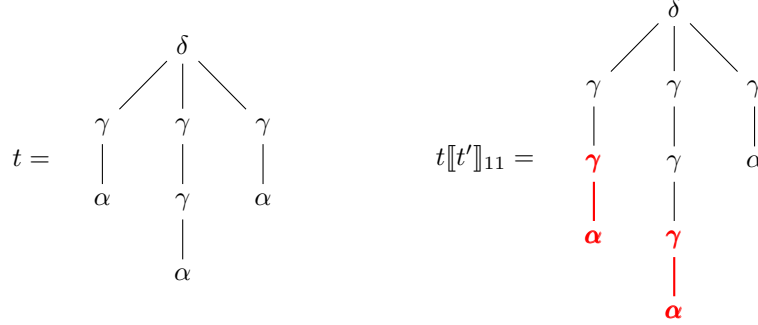
$$d[\underline{d}']_w = d'_1 \cdots d'_k(p, \varepsilon) \quad \text{and} \quad t[\underline{t}']_w^d = c[t'_1, \dots, t'_k] ,$$

where for each $i \in [k]$ we have

- if $i = j$ (i.e., w_i is a prefix of w), then $d'_i = w_i(d_i[\underline{d}']_w)$ and $t'_i = t_i[\underline{t}']_w^{d_i}$,
- if $q_i = \perp$ and $w_i \in [w_j]_{\equiv_E}$ (i.e., it is a position that is equality restricted to w_j), then $d'_i = w_i d_u^\perp$ and $t'_i = u$ with $u = t_j[\underline{t}']_w^{d_j}$, and
- otherwise $d'_i = w_i d_i$ and $t'_i = t_i$ (i.e., derivation and tree remain unchanged).

It is straightforward to verify that $d[\underline{d}']_w$ is a complete left-most derivation for $t[\underline{t}']_w^d$ to q .

► **Example 7.** Consider the WTGh G of Example 4 and the following tree t it generates into which we want to substitute the tree $t' = \gamma(\alpha)$ at position $w = 11$.



We consider the following complete left-most derivation for t to q_f .

$$d = (\alpha \rightarrow q, 11) (\gamma(q) \rightarrow q, 1) (\alpha \rightarrow \perp, 211) (\gamma(\perp) \rightarrow \perp, 21) \\ (\alpha \rightarrow q, 31) (\gamma(q) \rightarrow q, 3) (\delta(q, \gamma(\perp), q) \xrightarrow{1=21} q_f, \varepsilon)$$

Moreover, let $d' = (\alpha \rightarrow q, 1) (\gamma(q) \rightarrow q, \varepsilon)$ and $d'_\perp = (\alpha \rightarrow \perp, 1) (\gamma(\perp) \rightarrow \perp, \varepsilon)$. With the notation of Definition 6, in the first step we have $v_1 = 1$, $v_2 = 21$, $v_3 = 3$, $d_1 = d_3 = d'$, $d_2 = d'_\perp$, and $\hat{w} = v_1^{-1}w = 1$. Respecting the only constraint $1 = 21$, we set $d'_1 = d_1 \llbracket d' \rrbracket_{\hat{w}} = d' \llbracket d' \rrbracket_1$, $d'_2 = d_2 \llbracket d'_\perp \rrbracket_{\hat{w}} = d'_\perp \llbracket d'_\perp \rrbracket_1$, and $d'_3 = d_3 = d'$. Eventually, $d'_1 = (\alpha \rightarrow q, 11) (\gamma(q) \rightarrow q, 1) (\gamma(q) \rightarrow q, \varepsilon)$ and $d'_2 = (\alpha \rightarrow \perp, 11) (\gamma(\perp) \rightarrow \perp, 1) (\gamma(\perp) \rightarrow \perp, \varepsilon)$. Hence, we obtain the following derivation $d \llbracket d' \rrbracket_{11}$ for our new tree $t \llbracket t' \rrbracket_{11}$.

$$d \llbracket d' \rrbracket_{11} = (\alpha \rightarrow q, 111) (\gamma(q) \rightarrow q, 11) (\gamma(q) \rightarrow q, 1) (\alpha \rightarrow \perp, 2111) (\gamma(\perp) \rightarrow \perp, 211) \\ (\gamma(\perp) \rightarrow \perp, 21) (\alpha \rightarrow q, 31) (\gamma(q) \rightarrow q, 3) (\delta(q, \gamma(\perp), q) \xrightarrow{1=21} q_f, \varepsilon)$$

Although $t|_{31} = \alpha$ also coincides with the subtree $t|_{11} = \alpha$ we replaced, these two subtrees are not equality-constrained, so the simultaneous substitution does not affect $t|_{31}$.

The substitution of Definition 6 allows us to prove a pumping lemma for the class of WTGh: If d is an accepting derivation of a WTGh $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ for a tree t with $\text{ht}(t) > \text{ht}(G)$, then there exist at least $|Q \setminus \{\perp\}| + 1$ positions $w_1 > \dots > w_{|Q|+1}$ in t at which d applies productions with non-sink target states. By the pigeonhole principle, there thus exist two positions $w_i > w_j$ in t at which d applies productions with the same non-sink target state. Employing the substitution we just defined, we can substitute $t|_{w_j}$ into w_i and obtain a derivation of G for $t \llbracket t|_{w_j} \rrbracket_{w_i}$. This process can be repeated to obtain an infinite sequence of trees strictly increasing in size. Formally, the following lemma was proved in [20].

► **Lemma 8** ([20, Lemma 4]). *Let $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ be a WTGh. Consider some tree $t \in T_\Sigma$ and non-sink state $q \in Q \setminus \{\perp\}$ such that $\text{ht}(t) > \text{ht}(G)$ and $D_G^q(t) \neq \emptyset$. Then there are infinitely many pairwise distinct trees t_0, t_1, \dots such that $D_G^q(t_i) \neq \emptyset$ for all $i \in \mathbb{N}$.*

► **Example 9.** Recall the WTGh G of Example 4. We have $\text{ht}(P) = 2$ and $\text{ht}(G) = 4$, but for simplicity, we choose the smaller tree $t = \delta(\gamma(\alpha), \gamma(\gamma(\alpha)), \gamma(\alpha))$, which we also considered in Example 7, since it also allows pumping. The derivation d presented in Example 7 for t applies the productions $(\alpha \rightarrow q)$ at 11 and $(\gamma(q) \rightarrow q)$ at 1, so we substitute $t|_1 = \gamma(\alpha)$ at 11 to obtain $t \llbracket \gamma(\alpha) \rrbracket_{11}$. In fact, this is exactly the substitution we illustrated in Example 7.

4 The Decision Procedure

Let us now turn to the \mathbb{N} -weighted version of the HOM-problem. In the following, we show that the regularity of the homomorphic image of a regular \mathbb{N} -weighted tree language is decidable in polynomial time. More precisely, we prove the following theorem.

► **Theorem 10.** *The weighted HOM-problem over \mathbb{N} is polynomial; i.e. for fixed ranked alphabets Γ and Σ , given a trim WTA A over Γ , and a nondeleting, nonerasing tree homomorphism $h: T_\Gamma \rightarrow T_\Sigma$, it is decidable in polynomial time whether $h_{\llbracket A \rrbracket}$ is regular.*

In the beginning, the proof of Theorem 10 resembles the unweighted case [15]: Given a regular weighted tree language A (represented by a trim WTA) and a tree homomorphism h , we first construct a trim WTGH G for its image $\llbracket G \rrbracket = h_A$ applying Theorem 3. We then show that $\llbracket G \rrbracket$ is regular if and only if the equality constraints used in G only act on subtrees of height at most $\text{ht}(G)$. In other words, if there exists a production $\ell \xrightarrow{E} q$ in G such that for some equality constraint $(u, v) \in E$ with non-sink state $q = \ell(u)$ there exists a tree $t \in T_\Sigma$ with $\text{ht}(t) > \text{ht}(G)$ and $D_G^q(t) \neq \emptyset$, then $\llbracket G \rrbracket$ is not regular, and if no such production exists, then $\llbracket G \rrbracket$ is regular. There are thus three parts to our proof. First, we show that the existence of such a production is decidable in polynomial time. Then we show that $\llbracket G \rrbracket$ is regular if no such production exists. Finally, we show that $\llbracket G \rrbracket$ is not regular if such a production exists. The latter part employs RAMSEY's theorem [24] and is the most significant technical contribution in our paper. For convenience, we attach a name to the property described here.

► **Definition 11.** *Let $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ be a trim WTGH. We say that G has the large duplication property if there exist a production $\ell \xrightarrow{E} q \in P$, an equality constraint $(u, v) \in E$ with $\ell(u) \neq \perp = \ell(v)$, and a tree $t \in T_\Sigma$ such that $\text{ht}(t) > \text{ht}(G)$ and $D_G^{\ell(u)}(t) \neq \emptyset$.*

We start with the decidability of the large duplication property.

► **Lemma 12.** *Consider a fixed ranked alphabet Σ . The following is decidable in polynomial time: Given a trim WTGH G , does it satisfy the large duplication property?*

Proof. Let $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ and construct the directed graph $G = (Q, E)$ with edges $E = \bigcup_{\ell \xrightarrow{E} q \in P} \{(q', q) \mid q' \in Q, \text{pos}_{q'}(\ell) \neq \emptyset\}$. Clearly, the large duplication property is equivalent to the condition that there exists a production $\ell \xrightarrow{E} q \in P$, an equality constraint $(u, v) \in E$ with $\ell(u) \neq \perp = \ell(v)$, and a state $q' \in Q \setminus \{\perp\}$ such that there exists a cycle from q' to q' in G and a path from q' to $\ell(u)$ in G . This equivalent condition can be checked in polynomial time. The equivalence of the two statements is easy to establish. If the large duplication property holds, then the pumping lemma [20, Lemma 4] exhibits the required cycle and path. Conversely, if the cycle and path exist, then the pumping lemma [20, Lemma 4] can be used to derive arbitrarily tall trees for which a derivation exists. ◀

Next, we show that if a WTGH G does not satisfy the large duplication property, then its generated weighted tree language $\llbracket G \rrbracket$ is regular. To this end, we construct the *linearization* of G . The linearization of a WTGH G is a WTG that simulates all derivations of G which only ensure the equality of subtrees of height at most $\text{ht}(G)$. For this, we replace every production $\ell \xrightarrow{E} q$ in G by the collection of all productions $\ell' \rightarrow q$ which can be obtained by instantiating E , i.e., substituting each position constrained by E with a compatible tree of height at most $\text{ht}(G)$ that satisfies E . Note that positions in ℓ that are unconstrained by E are unaffected by these substitutions. Formally, we define the linearization following [15, Definition 7.1].

- **Definition 13.** Let $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ be a WTGh. The linearization $\text{lin}(G)$ of G is the WTG $\text{lin}(G) = (Q \cup \{\perp\}, \Sigma, F, P_{\text{lin}}, \text{wt}_{\text{lin}})$, where P_{lin} and wt_{lin} are defined as follows. For $\ell' \in T_{\Sigma}(Q) \setminus Q$ and $q \in Q$, we let $(\ell' \rightarrow q) \in P_{\text{lin}}$ if and only if there exist a production $(\ell \xrightarrow{E} q) \in P$, positions $w_1, \dots, w_k \in \text{pos}_{Q \cup \{\perp\}}(\ell)$, and trees $t_1, \dots, t_k \in T_{\Sigma}$ with
- $\{w_1, \dots, w_k\} = \bigcup_{w \in \text{pos}_{\perp}(\ell)} [w]_E$; i.e., E constrains exactly the positions w_1, \dots, w_k ,
 - $t_i = t_j$ if $(w_i, w_j) \in E$ for all $i, j \in [k]$,
 - $\ell' = \ell[t_1]_{w_1} \cdots [t_k]_{w_k}$, and
 - $D_G^{\ell(w_i)}(t_i) \neq \emptyset$ and $\text{ht}(t_i) \leq \text{ht}(G)$ for all $i \in [k]$.

For every such production $\ell' \rightarrow q$ we define $\text{wt}_{\text{lin}}(\ell' \rightarrow q)$ as the sum over all weights

$$\text{wt}(\ell \xrightarrow{E} q) \cdot \prod_{i \in [k]} \text{wt}_G^{\ell(w_i)}(t_i)$$

for all $(\ell \xrightarrow{E} q) \in P$, $w_1, \dots, w_k \in \text{pos}_{Q \cup \{\perp\}}(\ell)$, and $t_1, \dots, t_k \in T_{\Sigma}$ as above.

If a trim WTGh G does not satisfy the large duplication property, then every equality constraint in every derivation of G only ensures the equality of subtrees of height at most $\text{ht}(G)$. Thus, $\text{lin}(G)$ and G generate the same weighted tree language $\llbracket G \rrbracket = \llbracket \text{lin}(G) \rrbracket$, which is then regular because $\text{lin}(G)$ is a WTG. Thus we summarize:

- **Proposition 14.** Let G be a trim WTGh and suppose that G does not satisfy the large duplication property. Then $\llbracket G \rrbracket$ is a regular weighted tree language.

Finally, we show that if a WTGh $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ satisfies the large duplication property, then $\llbracket G \rrbracket$ is not regular. For this, we first show that if G satisfies the large duplication property, then we can decompose it into two WTGh G_1 and G_2 such that $\llbracket G \rrbracket = \llbracket G_1 \rrbracket + \llbracket G_2 \rrbracket$ and at least one of $\llbracket G_1 \rrbracket$ and $\llbracket G_2 \rrbracket$ is not regular. To conclude the desired statement, we then show that the sum $\llbracket G \rrbracket = \llbracket G_1 \rrbracket + \llbracket G_2 \rrbracket$ is also not regular. For the decomposition, consider the following idea. Assume that there exists a production $p = (\ell \xrightarrow{E} q) \in P$ as in the large duplication property such that $F(q) \neq 0$. Then we create two copies G_1 and G_2 of G as follows. In G_1 we set all final weights to 0, add a new state f with final weight $F(q)$, and add the new production $(\ell \xrightarrow{E} f)$ with the same weight as p . On the other hand, in G_2 we set the final weight of q to 0, add a new state f with final weight $F(q)$, and for every production $p' = (\ell' \xrightarrow{E'} q) \in P$ except p , we add the new production $\ell' \xrightarrow{E'} f$ to G_2 with the same weight as p' . Then $\llbracket G \rrbracket = \llbracket G_1 \rrbracket + \llbracket G_2 \rrbracket$ because every derivation of G whose last production is p is now a derivation of G_1 to f , and every other derivation is either directly a derivation of G_2 or, in case of other derivations to q , is a derivation of G_2 to f .

By our assumption on the production $p = (\ell \xrightarrow{E} q)$, there exist a tall tree $t \in T_{\Sigma}$ with $\text{ht}(t) > \text{ht}(G)$ and a constraint $(u, v) \in E$ with $\ell(u) \neq \perp = \ell(v)$ and $D_G^{\ell(u)}(t) \neq \emptyset$. Thus, every tree t' generated by G_1 satisfies $t'|_u = t'|_v$, and by Lemma 8, there exist infinitely many pairwise distinct trees with a derivation to $\ell(u)$. The support (i.e., set of nonzero weighted trees) of $\llbracket G_1 \rrbracket$ is therefore not a regular tree language. This implies that $\llbracket G_1 \rrbracket$ is not regular, as the support of every regular weighted tree language over \mathbb{N} is a regular tree language [12].

In general, we cannot expect that a production $\ell \xrightarrow{E} q$ as in the large duplication property exists that already targets a final state. We therefore “grow” productions from the top, beginning with a production whose target state is final, by substituting Q -labeled positions with left-hand sides of other productions until we have “synthesized” a production which satisfies the large duplication property. We then construct G_1 by adding this newly formed

production as a production to a new state f . We construct G_2 simply to ensure that it simulates all derivations of G that are not already accounted for by G_1 . Formally, we show the following lemma.

- **Lemma 15.** *Let $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ be a trim WTGh that satisfies the large duplication property. Then there exist two trim WTGh $G_1 = (Q_1 \cup \{\perp\}, \Sigma, F_1, P_1, \text{wt}_1)$ and $G_2 = (Q_2 \cup \{\perp\}, \Sigma, F_2, P_2, \text{wt}_2)$ such that $\llbracket G \rrbracket = \llbracket G_1 \rrbracket + \llbracket G_2 \rrbracket$ and for some $f \in Q_1$ we have*
- $F_1(f) \neq 0$ and $F_1(q) = 0$ for all $q \in Q_1 \setminus \{f\}$, and
 - *there exists exactly one production $p_f = (\ell_f \xrightarrow{E_f} f) \in P_1$ with target state f , and for this production there exists $(u, v) \in E_f$ with $\ell_f(u) \neq \ell_f(v) = \perp$ and an infinite sequence of pairwise distinct trees $t_0, t_1, t_2, \dots \in T_\Sigma$ such that $D_{G_1}^{\ell_f(u)}(t_i) \neq \emptyset$ for all $i \in \mathbb{N}$.*

Proof. Let $p = (\ell \xrightarrow{E} q) \in P$ be a production as in the large duplication property. Since G is trim, there exist a tree $t' \in T_\Sigma$, a final state $q_f \in Q$ with $F(q_f) \neq 0$, a derivation $d = (p_1, w_1) \cdots (p_m, w_m) \in D_G^{q_f}(t')$, and $i \in [m]$ such that $p_i = p$. In other words, there is a derivation utilizing production p . We let $p_j = \ell_j \xrightarrow{E_j} q_j$ for every $j \in [m]$, and let $w_{i_1} > \cdots > w_{i_k}$ be the sequence of prefixes of w_i among the positions $\{w_1, \dots, w_m\}$ in strictly descending order with respect to the prefix order. In particular, we have $w_{i_1} = w_i$ and $w_{i_k} = \varepsilon$.

For a position w and a set E' of constraints, we define $wE' = \{(wu, wv) \mid (u, v) \in E'\}$. We want to join the left-hand sides of the productions p_{i_1}, \dots, p_{i_k} to a new production $\ell_{i_k}[\ell_{i_{k-1}}]_{w_{i_{k-1}}} \cdots [\ell_{i_1}]_{w_{i_1}} \xrightarrow{E_f} q_f$ with $E_f = \bigcup_{j \in [k]} w_{i_j} E_{i_j}$. However, we need to ensure that w_{i_1}, \dots, w_{i_k} do not occur in E_f . Therefore, we assume that p, t', q_f, d , and i above are chosen such that w_i is of minimal length among all possible choices. Then we see as follows that w_{i_1}, \dots, w_{i_k} do not occur in E_f .

Let $(u, v) \in E$ with $\ell(u) \neq \ell(v) = \perp$ and $t \in T_\Sigma$ with $\text{ht}(t) > \text{ht}(G)$ and $D_G^{\ell(u)}(t) \neq \emptyset$. Suppose there exists $j \in [k]$ such that w_{i_j} occurs in E_f . Then there exists $(u', v') \in E_{i_{j+1}}$ with $w_{i_j} = w_{i_{j+1}} u'$. Then the tree $t' \llbracket t \rrbracket_{w_i u} w_{i_j}$ shows us that $p_{i_{j+1}}$ is also a production as in the large duplication property, but $|w_{i_{j+1}}| < |w_i|$, so w_i is not of minimal length.

We define $G_1 = (Q_1 \cup \{\perp\}, \Sigma, F_1, P_1, \text{wt}_1)$ as follows. Let $f \notin Q \cup \{\perp\}$ be a new state. We set $Q_1 = Q \cup \{f\}$, $F_1(f) = F(q_f)$, and $F_1(q') = 0$ for all $q' \in Q$. For the production $p_f = (\ell_{i_k}[\ell_{i_{k-1}}]_{w_{i_{k-1}}} \cdots [\ell_{i_1}]_{w_{i_1}} \xrightarrow{E_f} f)$ with $E_f = \bigcup_{j \in [k]} w_{i_j} E_{i_j}$, we let $P_1 = P \cup \{p_f\}$, $\text{wt}_1(p_f) = \prod_{j \in [k]} \text{wt}(p_{i_j})$, and $\text{wt}_1(p') = \text{wt}(p')$ for all $p' \in P$. Then G_1 simulates all derivations of G with productions p_{i_1}, \dots, p_{i_k} at the positions w_{i_1}, \dots, w_{i_k} , respectively. For the existence of the infinite sequence of trees, let $(u, v) \in E$ with $\ell(u) \neq \ell(v) = \perp$ and $t \in T_\Sigma$ with $\text{ht}(t) > \text{ht}(G)$ and $D_G^{\ell(u)}(t) \neq \emptyset$. By Lemma 8, there exists an infinite sequence $t_0, t_1, t_2, \dots \in T_\Sigma$ of pairwise distinct trees with $D_G^{\ell(u)}(t_i) \neq \emptyset$ for all $i \in \mathbb{N}$. Since $D_G^{\ell(u)}(t_i) \subseteq D_{G_1}^{\ell(u)}(t_i)$ for all $i \in \mathbb{N}$, this is the desired sequence. We conclude the definition of G_1 by noting that $(w_i u, w_i v) \in E_f$ and that the left-hand side ℓ_f of p_f satisfies $\ell_f(w_i u) = \ell(u)$.

Next, we construct G_2 such that it simulates all remaining derivations of G in the following sense. If d is a derivation of G to a state different from q_f , then it is a derivation of G_2 to that same state. If d is a derivation of G to q_f but its last production is not p_{i_k} , then it is simulated by a derivation of G_2 to a new state f . If d is a derivation of G and its last production is p_{i_k} but the production at $w_{i_{k-1}}$ is not $p_{i_{k-1}}$, then it again is simulated by a derivation of G_2 to f , and so on. To have a more compact definition for G_2 , we use the symbol \square to denote a tree of height 0 and a term $\square[\ell_{i_k}]_{w_{i_k}} \cdots [\ell_{i_{j+1}}]_{w_{i_{j+1}}} [\ell']_{w_{i_j}}$ for $j = k$ is to be read as $\square[\ell']_{w_{i_j}}$. We let $f \notin Q \cup \{\perp\}$ be a new state and define $G_2 = (Q_2 \cup \{\perp\}, \Sigma, F_2, P_2, \text{wt}_2)$ by $Q_2 = Q \cup \{f\}$, $F_2(q_f) = 0$, $F_2(f) = F(q_f)$, and $F_2(q') = F(q')$ for all $q' \in Q \setminus \{q_f\}$.



■ **Figure 1** The tree t' and the new production p_f .

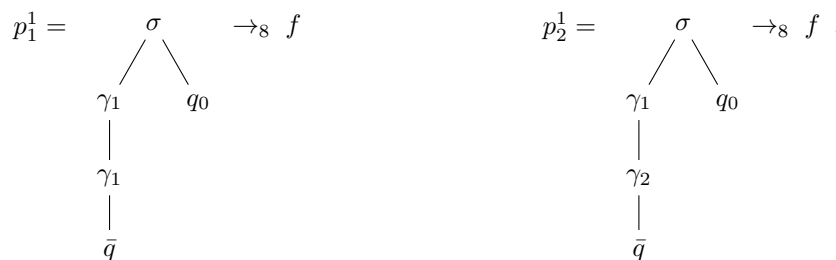
For the set P_2 of productions, we let

$$P_2 = P \cup \bigcup_{j \in [k]} \left\{ \square[\ell_{i_k}]_{w_{i_k}} \cdots [\ell_{i_{j+1}}]_{w_{i_{j+1}}} [\ell']_{w_{i_j}} \xrightarrow{E_f} f \mid p' = (\ell' \xrightarrow{E'} q_{i_j}) \in P \setminus \{p_{i_j}\}, \right. \\ \left. E_f = w_{i_j} E' \cup \bigcup_{j'=j+1}^k w_{i_{j'}} E_{i_{j'}} \right\} .$$

For a production $p_f = \square[\ell_{i_k}]_{w_{i_k}} \cdots [\ell_{i_{j+1}}]_{w_{i_{j+1}}} [\ell']_{w_{i_j}} \xrightarrow{E_f} f$ constructed from p' as above we let $\text{wt}_2(p_f) = \text{wt}(p') \cdot \prod_{j'=j+1}^k \text{wt}(p_{i_{j'}})$ and for every $p' \in P$ we let $\text{wt}_2(p') = \text{wt}(p')$. Then we have $\llbracket G \rrbracket(t) = \llbracket G_1 \rrbracket(t) + \llbracket G_2 \rrbracket(t)$ for every $t \in T_\Sigma$. Note that trimming G_1 and G_2 will not remove any of the newly added productions under the assumption that G is trim. ◀

► **Example 16.** We present an example for the decomposition in Lemma 15. Consider the trim WTGh $G = (Q \cup \{\perp\}, \Sigma, P, F, \text{wt})$ with $Q = \{q_0, \bar{q}, q_f\}$, $\Sigma = \{\alpha^{(0)}, \gamma^{(1)}, \sigma^{(2)}, \gamma_1^{(1)}, \gamma_2^{(1)}\}$, final weights $F(q_f) = 1$ and $F(q_0) = F(\bar{q}) = F(\perp) = 0$, and the set $P = P_\perp \cup P'$ defined by $P' = \{ \alpha \rightarrow_1 q_0, \gamma(q_0) \rightarrow_1 q_0, \sigma(q_0, \perp) \xrightarrow{1=2}_2 \bar{q}, \gamma_1(\bar{q}) \rightarrow_2 \bar{q}, \gamma_2(\bar{q}) \rightarrow_2 \bar{q}, \sigma(\bar{q}, q_0) \rightarrow_2 q_f \}$ and the usual productions targeting \perp in P_\perp . Trees of the form $\gamma(\cdots(\gamma(\alpha))\cdots)$ of arbitrary height are subject to the constraint $1 = 2$, so G satisfies the large duplication property.

We consider t' as in Figure 1 and use its (unique) derivation in G . Following the approach sketched above, we choose a new state f and define $G_1 = (Q \cup \{f\} \cup \{\perp\}, \Sigma, F_1, P_1, \text{wt}_1)$, where $F_1(f) = 1$ and $F_1(q) = 0$ for every $q \in Q \cup \{\perp\}$, and $P_1 = P \cup \{p_f\}$ with the new production p_f depicted in Figure 1, which joins all the productions of G used to derive t' , from the one evoking the large duplication property to the one targeting a final state. It remains to construct a WTGh G_2 such that $\llbracket G \rrbracket = \llbracket G_1 \rrbracket + \llbracket G_2 \rrbracket$. All productions of G still occur in G_2 , but q_f is not final anymore. Instead, we add a state f with $F_2(f) = F(q_f) = 1$ and make sure that this state adopts all other accepting derivations that formerly led to q_f . For this, we handle first the derivations that coincide with the derivation for t' at the juncture positions ε and 1, but not at 2. This leads to the following new productions p_1^1 and p_2^1 :



51:12 Weighted HOM-Problem for Nonnegative Integers

Next we cover the derivations that differ from the derivation for t' at the position 1 but coincide with it at the root. This leads to the new productions

$$p_1^2 = \begin{array}{c} \sigma \\ \swarrow \quad \searrow \\ \sigma \quad q_0 \\ \swarrow \quad \searrow \\ q_0 \quad \perp \end{array} \xrightarrow[4]{11=12} f \qquad p_2^2 = \begin{array}{c} \sigma \\ \swarrow \quad \searrow \\ \gamma_2 \quad q_0 \\ | \\ q_0 \end{array} \rightarrow_4 f .$$

Apart from the production incorporated at the root of p_f , no other production of G targets q_f directly, so no more productions are added to P_2 .

Finally, we define the WTGh $G_2 = (Q \cup \{f\} \cup \{\perp\}, \Sigma, F_2, P_2, \text{wt}_2)$ with $F_2(f) = F(q_f) = 1$, $F_2(q_f) = F_2(q_0) = F_2(\bar{q}) = F_2(\perp) = 0$, and $P_2 = P \cup \{p_1^1, p_2^1\} \cup \{p_1^2, p_2^2\}$.

It remains to show that the existence of a decomposition $\llbracket G \rrbracket = \llbracket G_1 \rrbracket + \llbracket G_2 \rrbracket$ as in Lemma 15 implies the non-regularity of $\llbracket G \rrbracket$. For this, we employ the following idea. Consider a ranked alphabet Σ containing a letter σ of rank 2, a WTA $G' = (Q, \Sigma, F, P, \text{wt})$ over Σ (which exemplifies G_2), and a sequence $t_0, t_1, t_2, \dots \in T_\Sigma$ of pairwise distinct trees. At this point, we assume that P contains all possible productions, but we may have $\text{wt}(p) = 0$ for $p \in P$. Using the initial algebra semantics [12], we can find a matrix representation for the weights assigned by G' to trees of the form $\sigma(t_i, t_j)$ as follows. We enumerate the states $Q = \{q_1, \dots, q_n\}$ and for every $i \in \mathbb{N}$ define a (column) vector $\nu_i \in \mathbb{N}^n$ by $(\nu_i)_k = \text{wt}_{G'}^{q_k}(t_i)$ for $k \in [n]$. Furthermore, we define a matrix $N \in \mathbb{N}^{n \times n}$ by $N_{kh} = \sum_{q \in Q} F(q) \cdot \text{wt}(\sigma(q_k, q_h) \rightarrow q)$ for $k, h \in [n]$. Then $\llbracket G' \rrbracket(\sigma(t_i, t_j)) = \nu_i^T N \nu_j$ for all $i, j \in \mathbb{N}$, where ν_i^T is the transpose of ν_i .

We employ this matrix representation to show that the sum of $\llbracket G' \rrbracket$ and the (non-regular) characteristic function 1_L of the tree language $L = \{\sigma(t_i, t_i) \mid i \in \mathbb{N}\}$ is not regular. We proceed by contradiction and assume that $\llbracket G' \rrbracket + 1_L$ is regular. Thus we can find an analogous matrix representation using a matrix N' and vectors ν'_i for $\llbracket G' \rrbracket + 1_L$. Since the trees t_0, t_1, t_2, \dots are pairwise distinct, we can write

$$(\llbracket G' \rrbracket + 1_L)(\sigma(t_i, t_j)) = (\nu'_i)^T N' \nu'_j = \llbracket G' \rrbracket(\sigma(t_i, t_j)) + \delta_{ij} = \nu_i^T N \nu_j + \delta_{ij}$$

for all $i, j \in \mathbb{N}$, where δ_{ij} denotes the KRONECKER delta. The vectors ν'_i and ν_i contain nonnegative integers, so we may consider the concatenated vectors $\langle \nu'_i, \nu_i \rangle$ as vectors of \mathbb{Q}^m where $m \in \mathbb{N}$ is the sum of number of states of G' and of the WTA we assumed recognizes $\llbracket G' \rrbracket + 1_L$. Since \mathbb{Q}^m is a finite dimensional \mathbb{Q} -vector space, the \mathbb{Q} -vector space spanned by the family $(\langle \nu'_i, \nu_i \rangle)_{i \in \mathbb{N}}$ is also finite dimensional. We may thus select a finite generating set from $(\langle \nu'_i, \nu_i \rangle)_{i \in \mathbb{N}}$. For simplicity, we assume that $\langle \nu'_1, \nu_1 \rangle, \dots, \langle \nu'_K, \nu_K \rangle$ form such a generating set. Thus there exist $a_1, \dots, a_K \in \mathbb{Q}$ with $\langle \nu'_{K+1}, \nu_{K+1} \rangle = \sum_{i \in [K]} a_i \langle \nu'_i, \nu_i \rangle$. Applying the usual distributivity laws for matrix multiplication, we reach a contradiction as follows.

$$\begin{aligned} (\llbracket G' \rrbracket + 1_L)(\sigma(t_{K+1}, t_{K+1})) &= (\nu'_{K+1})^T N' \nu'_{K+1} = \sum_{i \in [K]} a_i (\nu'_i)^T N' \nu'_{K+1} \\ &= \sum_{i \in [K]} a_i \nu_i^T N \nu_{K+1} = \nu_{K+1}^T N \nu_{K+1} = \llbracket G' \rrbracket(\sigma(t_{K+1}, t_{K+1})) \end{aligned}$$

For the general case, we do not want to assume that $\llbracket G_2 \rrbracket$ is regular, so we cannot assume to have a matrix representation as we had for $\llbracket G' \rrbracket$ above. In order to make our idea work, we identify a set of trees for which the behavior of $\llbracket G_1 \rrbracket + \llbracket G_2 \rrbracket$ resembles that of $\llbracket G' \rrbracket + 1_L$; more precisely, we construct a context C and a sequence t_0, t_1, t_2, \dots of pairwise distinct trees

such that $(\llbracket G_1 \rrbracket + \llbracket G_2 \rrbracket)(C(t_i, t_j)) = \nu_i^{(1)} N \nu_j^{(2)} + \delta_{ij} \mu_i$ for all $i, j \in \mathbb{N}$ and additionally, $\mu_i > 0$ for all $i \in \mathbb{N}$. This representation then allows us to perform linear algebra computations in order to prove that $\llbracket G_1 \rrbracket + \llbracket G_2 \rrbracket$ is non-regular. Unfortunately, working with a 2-context C may be insufficient if G_1 uses constraints of the form $\{v = v', v' = v''\}$, where more than two positions are constrained to be pairwise equivalent. Therefore, we have to consider more general n -contexts C and then identify a sequence of trees such that the equation above is satisfied on $C(t_i, t_j, t_j, \dots, t_j)$.

Isolating this desired sequence of trees is the most technically involved proof in our paper. We illustrate the effect of this selection in Example 19 below. Along the way, we will use the following version of RAMSEY's theorem [24]. For a set X , we denote by $\binom{X}{2}$ the set of all subsets of X of size 2.

► **Theorem 17.** *Let $k \geq 1$ be an integer and $f: \binom{\mathbb{N}}{2} \rightarrow [k]$ a mapping. There exists an infinite subset $E \subseteq \mathbb{N}$ such that $f|_{\binom{E}{2}} \equiv i$ for some $i \in [k]$.*

► **Lemma 18.** *Let $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ be a trim WTGh. If G satisfies the large duplication property, then there exists an integer $r \geq 2$, an r -context $C \in T_\Sigma(\square)$, trees $(t_i)_{i \in \mathbb{N}} \subseteq T_\Sigma$, an integer $m \in \mathbb{N}$, row vectors $(\nu_i^{(1)})_{i \in \mathbb{N}} \subseteq \mathbb{N}^m$, column vectors $(\nu_i^{(2)})_{i \in \mathbb{N}} \subseteq \mathbb{N}^m$, a matrix $N \in \mathbb{N}^{m \times m}$, and weights $(\mu_i)_{i \in \mathbb{N}} \subseteq \mathbb{N} \setminus \{0\}$ with $\llbracket G \rrbracket(C(t_k, t_h, t_h, \dots, t_h)) = \nu_k^{(1)} N \nu_h^{(2)} + \delta_{kh} \mu_k$ for all $k, h \in \mathbb{N}$.*

Proof. By Lemma 15 there exist two trim WTGh $G_1 = (Q_1 \cup \{\perp\}, \Sigma, F_1, P_1, \text{wt}_1)$ and $G_2 = (Q_2 \cup \{\perp\}, \Sigma, F_2, P_2, \text{wt}_2)$ with $\llbracket G \rrbracket(t) = \llbracket G_1 \rrbracket(t) + \llbracket G_2 \rrbracket(t)$ for all $t \in T_\Sigma$. Additionally, there exists $f \in Q_1$ with $F_1(f) \neq 0$ and $F_1(q) = 0$ for all $q \in Q_1 \setminus \{f\}$ and there exists exactly one production $p_f = (\ell_f \xrightarrow{E_f} f) \in P_1$ whose target state is f . Finally, for this production p_f there exists $(u^{(1)}, v^{(1)}) \in E_f$ with $\ell_f(u^{(1)}) \neq \ell_f(v^{(1)}) = \perp$ and an infinite sequence $t_0, t_1, t_2, \dots \in T_\Sigma$ of pairwise distinct trees with $D_{G_1}^{\ell_f(u^{(1)})}(t_i) \neq \emptyset$ for all $i \in \mathbb{N}$.

Let $t \in T_\Sigma$ be such that $D_{G_1}^f(t) \neq \emptyset$, and let $u_1^{(1)}, \dots, u_r^{(1)}$ be an enumeration of all positions that are equality-constrained to $u^{(1)}$ via E_f , where we assume that $u_1^{(1)} = u^{(1)}$. We define a context $C = t[\square]_{u_1^{(1)}} \cdots [\square]_{u_r^{(1)}}$. Then $\llbracket G_1 \rrbracket(C(t_i, t_j, t_j, \dots, t_j)) > 0$ iff $i = j$.

Let us establish some additional notations. Let $k, h \in \mathbb{N}$ and assume there is $q \in Q_2$ with $F_2(q) \neq 0$ and $d = (p_1, w_1) \cdots (p_m, w_m) \in D_{G_2}^q(C(t_k, t_h, t_h, \dots, t_h))$. Let $p_i = \ell_i \xrightarrow{E_i} q_i$ for every $i \in [m]$, and for a set $X \subseteq \text{pos}(C(t_k, t_h, t_h, \dots, t_h))$, we let $i_1 < \dots < i_n$ be such that w_{i_1}, \dots, w_{i_n} is an enumeration of $\{w_1, \dots, w_m\} \cap X$; i.e., all positions in X to which d applies productions. We set $d|_X = (p_{i_1}, w_{i_1}) \cdots (p_{i_n}, w_{i_n})$, $\text{wt}_2(d|_X) = \prod_{j \in [n]} \text{wt}_2(p_{i_j})$, and $D_{kh} = \{d'|_{\text{pos}(C)} \mid \exists q' \in Q_2: F_2(q') \neq 0, d' \in D_{G_2}^{q'}(C(t_k, t_h, t_h, \dots, t_h))\}$.

We now employ RAMSEY's theorem in the following way. For $k, h \in \mathbb{N}$ with $k < h$, we consider the mapping $\{k, h\} \mapsto D_{kh}$. This mapping has a finite range as every D_{kh} is a set of finite words over the alphabet $P_2 \times \text{pos}(C)$ of length at most $\text{size}(C)$. Thus, by RAMSEY's theorem, we obtain a subsequence $(t_{i_j})_{j \in \mathbb{N}}$ with $D_{i_k i_h} = D_{<}$ for all $k, h \in \mathbb{N}$ and some set $D_{<}$. For simplicity, we assume $D_{kh} = D_{<}$ for all $k, h \in \mathbb{N}$ with $k < h$. Similarly, we select a further subsequence and assume $D_{kh} = D_{>}$ for all $k, h \in \mathbb{N}$ with $k > h$. Finally, the mapping $k \mapsto D_{kk}$ also has a finite range, so by the pigeonhole principle, we may select a further subsequence and assume that $D_{kk} = D_{=}$ for all $k \in \mathbb{N}$ and some set $D_{=}$. In the following, we show that $D_{<} = D_{>} \subseteq D_{=}$.

For now, we assume $D_{<} \neq \emptyset$, let $(p_1, w_1) \cdots (p_m, w_m) \in D_{<}$, and let $p_i = \ell_i \xrightarrow{E_i} q_i$ for every $i \in [m]$. Also, we define $C_{kh} = C(t_k, t_h, t_h, \dots, t_h)$, $C_{k\square} = C(t_k, \square, \square, \dots, \square)$, and $C_{\square h} = C(\square, t_h, t_h, \dots, t_h)$ for $k, h \in \mathbb{N}$. We show that every constraint from every E_i is satisfied on all C_{kh} with $k, h \geq 1$, not just for $k < h$. More precisely, let $i \in [m]$, $(u', v') \in E_i$,

and $(u, v) = (w_i u', w_i v')$. We show $C_{kh}|_u = C_{kh}|_v$ for all $k, h \geq 1$. Note that by assumption, $C_{kh}|_u = C_{kh}|_v$ is true for all $k, h \in \mathbb{N}$ with $k < h$. We show our statement by a case distinction depending on the position of u and v in relation to the positions $u_1^{(1)}, \dots, u_r^{(1)}$.

1. If both u and v are parallel to $u^{(1)}_1$, then $C_{ij}|_u$ and $C_{ij}|_v$ do not depend on i . Thus, $C_{0j}|_u = C_{0j}|_v$ for all $j \geq 1$ implies the statement.
2. If u is in prefix-relation with $u^{(1)}_1$ and v is parallel to $u^{(1)}_1$, then $C_{ij}|_v$ does not depend on i . If $u \leq u^{(1)}_1$, then by our assumption that $(t_i)_{i \in \mathbb{N}}$ are pairwise distinct, we obtain the contradiction $C_{02}|_v = C_{02}|_u \neq C_{12}|_u = C_{12}|_v$, where $C_{02}|_v = C_{12}|_v$ should hold. Thus, we have $u^{(1)}_1 \leq u$ and in particular, $C_{ij}|_u$ does not depend on j . Thus, for all $i, j \geq 1$ we obtain $C_{ij}|_u = C_{i,i+1}|_u = C_{i,i+1}|_v = C_{0,i+1}|_v = C_{0,i+1}|_u = C_{0j}|_u = C_{0j}|_v = C_{ij}|_v$. If v is in prefix-relation with $u^{(1)}_1$ and u is parallel to $u^{(1)}_1$, then we come to the same conclusion by formally exchanging u and v in this argumentation.
3. If u and v are both in prefix-relation with $u^{(1)}_1$, then u and v being parallel to each other implies $u^{(1)}_1 \leq u$ and $u^{(1)}_1 \leq v$. In particular, both u and v are parallel to all $u_2^{(1)}, \dots, u_r^{(1)}$. Thus, we obtain, as in the first case, that $C_{ij}|_u$ and $C_{ij}|_v$ do not depend on j and the statement follows from $C_{i,i+1}|_u = C_{i,i+1}|_v$ for all $i \in \mathbb{N}$.

Let $k, h \geq 1$ and $d_C \in D_{<}$, and let $q \in Q_2$, $d_{k,k+1} \in D_{G_2}^q(C_{k,k+1})$, and $d_{h-1,h} \in D_{G_2}^q(C_{h-1,h})$ such that $d_C = d_{k,k+1}|_{\text{pos}(C)} = d_{h-1,h}|_{\text{pos}(C)}$. Then for $d_k = d_{k,k+1}|_{\text{pos}(C_{k,k+1}) \setminus \text{pos}(C_{\square,k+1})}$ and $d_h = d_{h-1,h}|_{\text{pos}(C_{h-1,h}) \setminus \text{pos}(C_{h-1,\square})}$, we can reorder $d = d_k d_h d_C$ to a complete left-most derivation of G_2 for C_{kh} , as all equality constraints from d_k are satisfied by the assumption on $d_{k,k+1}$, all equality constraints from d_h are satisfied by the assumption on $d_{h-1,h}$, and all equality constraints from d_C are satisfied by our case distinction. Considering the special cases $k = 2, h = 1$, and $k = h = 1$, and the definitions of $D_{>}$ and $D_{=}$, we obtain $d_C \in D_{21} = D_{>}$ and $d_C \in D_{11} = D_{=}$, and hence, $D_{<} \subseteq D_{>}$ and $D_{<} \subseteq D_{=}$.

The converse inclusion $D_{>} \subseteq D_{<}$ follows with an analogous reasoning. In conclusion, we obtain $D_{<} = D_{>} \subseteq D_{=}$. By the reasoning above, the case $D_{<} = \emptyset$ we excluded earlier is only possible if also $D_{>} = \emptyset$, in which case we again have $D_{<} = D_{>} \subseteq D_{=}$.

Let d_1, \dots, d_n be an enumeration of $D_{<}$, $i \in [n]$, and $k \in \mathbb{N}$. We define the sets

$$\begin{aligned} D_{i,k}^{(1)} &= \{d|_{\text{pos}(C_{k,k+1}) \setminus \text{pos}(C_{\square,k+1})} \mid d \in D_{G_2}^q(C_{k,k+1}), d_i = d|_{\text{pos}(C)}, q \in Q_2\} \\ D_{i,k}^{(2)} &= \{d|_{\text{pos}(C_{k+1,k}) \setminus \text{pos}(C_{k+1,\square})} \mid d \in D_{G_2}^q(C_{k+1,k}), d_i = d|_{\text{pos}(C)}, q \in Q_2\} \end{aligned}$$

and the corresponding weights $\nu_{i,k}^{(1)} = \sum_{d \in D_{i,k}^{(1)}} \text{wt}_2(d)$ and $\nu_{i,k}^{(2)} = \sum_{d \in D_{i,k}^{(2)}} \text{wt}_2(d)$. Let q_i be the target state of the last production in d_i and define $\nu_i = F_2(q_i) \cdot \text{wt}_2(d_i)$. Then for all $k, h \in \mathbb{N}$ we have $\llbracket G_2 \rrbracket(C_{kh}) = \sum_{i \in [n]} (\nu_{i,k}^{(1)} \cdot \nu_i \cdot \nu_{i,h}^{(2)}) + \delta_{kh} \mu'_k$ for nonnegative $(\mu'_j)_{j \in \mathbb{N}}$, which stem from the fact that potentially $D_{=} \setminus D_{<} \neq \emptyset$. We arrange the weights $\nu_{i,k}^{(1)}$ into a row vector $\nu_k^{(1)}$, and the weights $\nu_{i,h}^{(2)}$ into a column vector $\nu_h^{(2)}$, and the weights ν_i into a diagonal matrix N such that $\llbracket G_2 \rrbracket(C_{kh}) = \nu_k^{(1)} N \nu_h^{(2)} + \delta_{kh} \mu'_k$. Finally, recall that $\llbracket G_1 \rrbracket(C_{kh}) > 0$ iff $k = h$ for all $k, h \in \mathbb{N}$. Thus we set $\mu_k = \mu'_k + \llbracket G_1 \rrbracket(C_{kk})$ and obtain $\llbracket G \rrbracket(C_{kh}) = \llbracket G_2 \rrbracket(C_{kh}) + \llbracket G_1 \rrbracket(C_{kh}) = \nu_k^{(1)} N \nu_h^{(2)} + \delta_{kh} \mu_k$ with $\mu_k > 0$ for all $k, h \in \mathbb{N}$. ◀

Before concluding the correctness of our decision procedure for the weighted HOM-problem, we want to exemplify how the Lemma 12 acts on a simple weighted tree language.

► **Example 19.** Consider the WTGH $G = (\{q, q_f, \perp\}, \{a^{(0)}, g^{(1)}, f^{(2)}\}, F, P, \text{wt})$ with final weights $F(q_f) = 1, F(q) = F(\perp) = 0$ and the following productions:

$$P = \{a \rightarrow_1 q, g(q) \rightarrow_2 q, f(q, \perp) \xrightarrow{1=2}_1 q_f, f(q, g(\perp)) \xrightarrow{1=2}_1 q_f\} \cup P_{\perp}$$

where $P_{\perp} = \{a \rightarrow_1 \perp, g(\perp) \rightarrow_1 \perp, f(\perp, \perp) \rightarrow_1 \perp\}$. The production $f(q, \perp) \xrightarrow{1=2}_1 q_f$ and the tree $g^{\text{ht}(G)}(a)$ satisfy the conditions in the large duplication property, so let G_1 denote the WTGh constructed according to Lemma 15 which simulates all derivations of G that use this production at ε . Consider the sequence $t_i = g^{i+\text{ht}(G)}(a)$ for $i \in \mathbb{N}$. The context $C = f(\square, \square)$ satisfies $\llbracket G_1 \rrbracket(C(t_i, t_j)) \neq 0$ iff $i = j$. In order to reproduce the linear-algebra argument from the special case of $\llbracket G' \rrbracket + 1_L$ described above, we need a matrix representation for the remaining part $\llbracket G_2 \rrbracket$, possibly with an additional factor δ_{ij} . In terms of the weights computed by G_2 , we can achieve this by the condition that $\llbracket G_2 \rrbracket(C(t_i, t_j)) \neq 0$ either for all $i, j \in \mathbb{N}$, or for none, or only if $i = j$. However, because of the production $f(q, g(\perp)) \xrightarrow{1=2}_1 q_f$, for each i we have $\llbracket G_2 \rrbracket(C(t_i, t_{i+1})) \neq 0$ and $\llbracket G_2 \rrbracket(C(t_i, t_j)) = 0$ for all $j \neq i + 1$. To fix this issue, we may select the subsequence $(t_{2i})_{i \in \mathbb{N}}$: In that case, we have $\llbracket G_2 \rrbracket(C(t_{2i}, t_{2j})) = 0$ for all $i, j \in \mathbb{N}$, and the matrix representation for $\llbracket G_2 \rrbracket$ is trivial.

Let us now conclude the decidability of the \mathbb{N} -weighted HOM-problem.

► **Theorem 20.** *Let $G = (Q \cup \{\perp\}, \Sigma, F, P, \text{wt})$ be a trim WTGh. If G satisfies the large duplication property, then $\llbracket G \rrbracket$ is not regular.*

Proof. Let $C \in T_{\Sigma}(\square)$, $(t_i)_{i \in \mathbb{N}} \subseteq T_{\Sigma}$, $m \in \mathbb{N}$, $(\nu_i^{(1)})_{i \in \mathbb{N}}, (\nu_i^{(2)})_{i \in \mathbb{N}} \subseteq \mathbb{N}^m$, $N \in \mathbb{N}^{m \times m}$, and $(\mu_i)_{i \in \mathbb{N}} \subseteq \mathbb{N} \setminus \{0\}$ be as in Lemma 18, i.e., $\llbracket G \rrbracket(C(t_k, t_h, t_h, \dots, t_h)) = \nu_k^{(1)} N \nu_h^{(2)} + \delta_{kh} \mu_k$ for all $k, h \in \mathbb{N}$. If $\llbracket G \rrbracket$ is regular, then we can assume a representation for all $k, h \in \mathbb{N}$ as $\llbracket G \rrbracket(C(t_k, t_h, t_h, \dots, t_h)) = g(\kappa_k, \kappa_h, \kappa_h, \dots, \kappa_h)$, where κ_h is a finite vector of weights over \mathbb{N} where each entry corresponds to the sum of all derivations for t_h to a specific state of a WTA, and g is a multilinear map encoding the weights of the derivations for $C(\square, \square, \dots, \square)$ depending on the specific input states at the \square -nodes and the target state at the root ε . We choose K such that the concatenated vectors $\langle \kappa_1, \nu_1^{(1)} \rangle, \dots, \langle \kappa_K, \nu_K^{(1)} \rangle$ form a generating set of the \mathbb{Q} -vector space spanned by $(\langle \kappa_i, \nu_i^{(1)} \rangle)_{i \in \mathbb{N}}$. Then there are coefficients $a_1, \dots, a_K \in \mathbb{Q}$ with $\kappa_{K+1} = \sum_{i \in [K]} a_i \kappa_i$ and $\nu_{K+1}^{(1)} = \sum_{i \in [K]} a_i \nu_i^{(1)}$. Thus, we reach our contradiction by

$$\begin{aligned} \nu_{K+1}^{(1)} N \nu_{K+1}^{(2)} + \mu_{K+1} &= g(\kappa_{K+1}, \kappa_{K+1}, \dots, \kappa_{K+1}) = \sum_{i \in [K]} a_i g(\kappa_i, \kappa_{K+1}, \dots, \kappa_{K+1}) \\ &= \sum_{i \in [K]} a_i \nu_i^{(1)} N \nu_{K+1}^{(2)} = \nu_{K+1}^{(1)} N \nu_{K+1}^{(2)}. \quad \blacktriangleleft \end{aligned}$$

5 Conclusion

In this contribution, we proved that the \mathbb{N} -weighted HOM-problem is decidable. Formally, given a regular weighted tree language A over \mathbb{N} and a nondeleting, nonerasing tree homomorphism h as input, it is decidable in polynomial time whether the homomorphic image h_A is again regular. This was achieved by reducing the HOM-problem to the newly introduced large duplication property, which formalizes the non-regular behavior of the investigated weighted tree language h_A , and then showing that this property is decidable.

Initially, h_A is represented by a generalized tree grammar (WTGh) as introduced in [20]. Such a device expresses the duplication of subtrees performed by h by means of explicit equality constraints. This WTGh is trimmed and tested directly for the large duplication property. If it does not satisfy this property, we construct an equivalent weighted tree grammar without constraints, which proves regularity of the generated weighted tree language. However, if the trim WTGh for h_A does satisfy the large duplication property, then no equivalent weighted

tree grammar exists. To prove this, we first identify a special sequence of productions, isolate it from the remainder of the WTGc, and then prove that it induces a non-regularity which cannot be compensated by the remaining derivations of the WTGh.




We require h to be nondeleting and nonerasing simply to ensure that h_A is well-defined in general. These properties have no impact on the correctness of the reduction or the computational complexity of the large duplication property, to which we reduce the \mathbb{N} -weighted HOM-problem. Indeed, our decision procedure for this problem is polynomial, while the unweighted HOM-problem is EXPTIME-complete [6]. In the \mathbb{N} -weighted setting we proved that the large duplication property is sufficient for non-regularity; this is the main technical difficulty and utilizes RAMSEY's theorem to identify a sequence of trees that acts as a witness for the non-regularity of the homomorphic image. A matrix representation that resembles the initial algebra semantics is then utilized to prove non-regularity. In the unweighted case the large duplication property is clearly necessary, but not sufficient. This difference is caused by the different algebraic structures of the underlying semirings. Whereas the semiring \mathbb{N} embeds into a field, the BOOLEAN semiring is idempotent, which can be used to cover non-regular behavior with regular behavior making it irrelevant. Essentially we proved that such covers are impossible in \mathbb{N} , which simplifies the execution of the decision procedure and allows us to prove polynomial-time decidability of the \mathbb{N} -weighted HOM-problem.

References

- 1 Adel Bouhoula and Florent Jacquemard. Tree automata, implicit induction and explicit destructors for security protocol verification. Technical report, Research Report LSV-07-10, 2007.
- 2 Adel Bouhoula and Florent Jacquemard. Automated induction with constrained tree automata. In *International Joint Conference on Automated Reasoning*, pages 539–554. Springer, 2008.
- 3 Symeon Bozapalidis and Antonios Kalampakas. Graph automata. *Theoret. Comput. Sci.*, 393(1-3):147–165, 2008. doi:10.1016/j.tcs.2007.11.022.
- 4 Symeon Bozapalidis and George Rahonis. On the closure of recognizable tree series under tree homomorphisms. *J. Autom. Lang. Comb.*, 10(2–3):185–202, 2005. doi:10.25596/jalc-2005-185.
- 5 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata — Techniques and Applications. <https://jacquema.gitlabpages.inria.fr/files/tata.pdf>, 2007.
- 6 Carles Creus, Adrià Gascón, Guillem Godoy, and Lander Ramos. The HOM problem is EXPTIME-complete. *SIAM J. Comput.*, 45(4):1230–1260, 2016. doi:10.1137/140999104.
- 7 John Doner. Tree acceptors and some of their applications. *J. Comput. System Sci.*, 4(5):406–451, 1970. doi:10.1016/S0022-0000(70)80041-1.
- 8 Frank Drewes. *Grammatical picture generation: A tree-based approach*. Springer, 2006. doi:10.1007/3-540-32507-7.
- 9 Zoltán Ésik and Werner Kuich. Formal tree series. *J. Autom. Lang. Comb.*, 8(2):219–285, 2003. doi:10.25596/jalc-2003-219.
- 10 Zoltán Fülöp, Andreas Maletti, and Heiko Vogler. Preservation of recognizability for synchronous tree substitution grammars. In *Proc. Workshop Applications of Tree Automata in Natural Language Processing*, pages 1–9. ACL, 2010. URL: <https://aclanthology.org/W10-2501>.
- 11 Zoltán Fülöp, Andreas Maletti, and Heiko Vogler. Weighted extended tree transducers. *Fundam. Inform.*, 111(2):163–202, 2011. doi:10.3233/FI-2011-559.
- 12 Zoltán Fülöp and Heiko Vogler. Weighted tree automata and tree transducers. In *Handbook of Weighted Automata*, chapter 9, pages 313–403. Springer, 2009. doi:10.1007/978-3-642-01492-5_9.

- 13 Ferenc Gécseg and Magnus Steinby. Tree automata. Technical Report 1509.06233, arXiv, 2015. URL: <https://arxiv.org/pdf/1509.06233.pdf>.
- 14 Rémy Gilleron and Sophie Tison. Regular tree languages and rewrite systems. *Fundamenta informaticae*, 24(1-2):157–175, 1995.
- 15 Guillem Godoy and Omer Giménez. The HOM problem is decidable. *J. ACM*, 60(4):1–44, 2013. doi:10.1145/2508028.2501600.
- 16 Guillem Godoy, Omer Giménez, Lander Ramos, and Carme Àlvarez. The HOM problem is decidable. In *Proc. 42nd ACM Symp. Theory of Computing*, pages 485–494. ACM, 2010.
- 17 Jonathan S. Golan. *Semirings and their Applications*. Kluwer Academic, Dordrecht, 1999. doi:10.1007/978-94-015-9333-5.
- 18 Udo Hebisch and Hanns J. Weinert. *Semirings — Algebraic Theory and Applications in Computer Science*. World Scientific, 1998. doi:10.1142/3903.
- 19 Dan Jurafsky and James H. Martin. *Speech and language processing*. Prentice Hall, 3rd edition, 2023. URL: <https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf>.
- 20 Andreas Maletti and Andreea-Teodora Nász. Weighted tree automata with constraints. *Theory Comput. Syst.*, 2023. to appear. URL: <https://arxiv.org/pdf/2302.03434.pdf>.
- 21 J. Mongy-Steen. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Université de Lille, 1981.
- 22 Andreea-Teodora Nász. Solving the weighted HOM-problem with the help of unambiguity. In *Proc. 16th Int. Conf. Automata and Formal Languages*, volume 386 of *EPTCS*, pages 200–214. Open Publishing Association, 2023. doi:10.4204/EPTCS.386.16.
- 23 Dominique Perrin. Recent results on automata and infinite words. In *Proc. 11th Int. Symp. Mathematical Foundations of Computer Science*, volume 176 of *LNCS*, pages 134–148. Springer, 1984. doi:10.1007/BFb0030294.
- 24 F. P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc.*, 30, 1930. doi:10.1112/plms/s2-30.1.264.
- 25 Arto Salomaa and Matti Soittola. *Automata-theoretic aspects of formal power series*. Springer, 1978. doi:10.1007/978-1-4612-6264-0.
- 26 Marcel Paul Schützenberger. On the definition of a family of automata. *Inform. and Control*, 4(2-3):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.
- 27 James W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *J. Comput. Syst. Sci.*, 1(4):317–322, 1967. doi:10.1016/S0022-0000(67)80022-9.
- 28 James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Systems Theory*, 2(1):57–81, 1968. doi:10.1007/BF01691346.
- 29 Reinhard Wilhelm, Helmut Seidl, and Sebastian Hack. *Compiler Design*. Springer, 2013. doi:10.1007/978-3-642-17540-4.

Worst-Case and Smoothed Analysis of the Hartigan–Wong Method for k -Means Clustering

Bodo Manthey   

Faculty of Electrical Engineering, Mathematics, and Computer Science, University of Twente, Enschede, The Netherlands

Jesse van Rhijn   

Faculty of Electrical Engineering, Mathematics, and Computer Science, University of Twente, Enschede, The Netherlands

Abstract

We analyze the running time of the Hartigan–Wong method, an old algorithm for the k -means clustering problem. First, we construct an instance on the line on which the method can take $2^{\Omega(n)}$ steps to converge, demonstrating that the Hartigan–Wong method has exponential worst-case running time even when k -means is easy to solve. As this is in contrast to the empirical performance of the algorithm, we also analyze the running time in the framework of smoothed analysis. In particular, given an instance of n points in d dimensions, we prove that the expected number of iterations needed for the Hartigan–Wong method to terminate is bounded by $k^{12kd} \cdot \text{poly}(n, k, d, 1/\sigma)$ when the points in the instance are perturbed by independent d -dimensional Gaussian random variables of mean 0 and standard deviation σ .

2012 ACM Subject Classification Theory of computation \rightarrow Randomness, geometry and discrete structures; Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Discrete optimization

Keywords and phrases k -means clustering, smoothed analysis, probabilistic analysis, local search, heuristics

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.52

Related Version *Full Version:* <https://arxiv.org/abs/2309.10368>

Funding *Jesse van Rhijn:* Supported by NWO grant OCENW.KLEIN.176.

1 Introduction

Clustering is an important problem in computer science, from both a practical and a theoretical perspective. On the practical side, identifying clusters of similar points in large data sets has relevance to fields ranging from physics to biology to sociology. Recent advances in machine learning and big data have made the need for efficient clustering algorithms even more apparent. On the theoretical side, clustering problems continue to be a topic of research from the perspective of approximation algorithms, heuristics, and computational geometry.

Perhaps the best-studied clustering problem is that of k -means clustering. In this problem, one is given a finite set of points $\mathcal{X} \subseteq \mathbb{R}^d$ and an integer k . The goal is to partition the points into k subsets, such that the sum of squared distances of each point to the centroid of its assigned cluster, also called its cluster center, is minimized.

Despite great effort to devise approximation algorithms for k -means clustering, the method of choice remains Lloyd’s method [11]. This method starts with an arbitrary choice of centers, and assigns each point to its closest center. The centers are then moved to the centroids of each cluster. In the next iteration, each point is again reassigned to its closest center, and the process repeats.



It is not hard to show that this process strictly decreases the objective function whenever either a cluster center changes position, or a point is reassigned. Hence, no clustering can show up twice during an execution of this algorithm. Since the number of partitions of n points into k sets is at most k^n , the process must eventually terminate.

Although Lloyd’s method has poor approximation performance both in theory and in practice [2], its speed has kept it relevant to practitioners. This is in startling contrast to its worst-case running time, which is exponential in the number of points [15].

To close the gap between theory and practice, Arthur et al. have shown that Lloyd’s method terminates in expected polynomial time on perturbed point sets, by means of a smoothed analysis [1]. This provides some theoretical justification for the use of Lloyd’s method in practice.

Another, less well-known heuristic for clustering is the Hartigan–Wong method [8]. In this method, one proceeds point-by-point. Given an arbitrary clustering, one checks whether there exists a point that can be reassigned to a different cluster, such that the objective function decreases. If such a point exists, it is reassigned to this new cluster. If no such points exist, the algorithm terminates and the clustering is declared locally optimal.

Although at first sight the Hartigan–Wong method might seem like a simpler version of Lloyd’s method, it is qualitatively different. If Lloyd’s method reassigns a point x from cluster i to cluster j , then x must be closer to the center of cluster j than to that of cluster i . In the Hartigan–Wong method, this is not true; x may be reassigned even when there are no cluster centers closer to x than its current center. This can be beneficial, as Telgarsky & Vattani showed that the Hartigan–Wong method is more powerful than Lloyd’s method [14].

To be precise, every local optimum of the Hartigan–Wong method is also a local optimum of Lloyd’s method, while the converse does not hold. Telgarsky & Vattani moreover performed computational experiments, which show that the Hartigan–Wong method not only tends to find better clusterings than Lloyd’s, but also has a similar running time on practical instances. Despite these promising results, theoretical knowledge of the Hartigan–Wong method is lacking.

In this paper, we aim to advance our understanding of this heuristic. Our contributions are twofold. First, we construct an instance on the line on which the Hartigan–Wong method can take $2^{\Omega(n)}$ iterations to terminate. Considering that k -means clustering can be solved exactly in polynomial time in $d = 1$, this shows that the worst-case running time of the Hartigan–Wong method is very poor even on easy instances. This is in contrast to Lloyd’s method, where all known non-trivial lower bounds require $d \geq 2$.

► **Theorem 1.** *For each $m \in \mathbb{N}_{\geq 2}$ there exists an instance of k -means clustering on the line with $n = 4m - 3$ points and $k = 2m - 1$ clusters on which the Hartigan–Wong method can take $2^{\Omega(n)}$ iterations to converge to a local optimum.*

Second, we attempt to reconcile Theorem 1 with the observed practical performance of the Hartigan–Wong method. We perform a smoothed analysis of its running time, in which each point in an arbitrary instance is independently perturbed by a Gaussian random variable of variance σ^2 .

► **Theorem 2.** *Let $n, k, d \in \mathbb{N}$, and assume $4kd \leq n$. Fix a set of n points $\mathcal{Y} \subseteq [0, 1]^d$, and assume that each point in \mathcal{Y} is independently perturbed by a d -dimensional Gaussian random variable with mean 0 and standard deviation σ , yielding a new set of points \mathcal{X} . Then the expected running time of the Hartigan–Wong method on \mathcal{X} is bounded by*

$$O\left(\frac{k^{12kd+5} d^{12} n^{12.5+\frac{1}{d}} \ln^{4.5}(nkd)}{\sigma^4}\right) = k^{12kd} \cdot \text{poly}(n, k, d, 1/\sigma).$$

Although we do not attain a polynomial smoothed running time in all problem parameters, we note that for Lloyd’s method one of the first smoothed analyses yielded a similar k^{kd} $\text{poly}(n, 1/\sigma)$ bound. This was later improved to $\text{poly}(n, k, d, 1/\sigma)$. We therefore regard Theorem 2 as a first step to settling the conjecture by Telgarsky & Vattani that the Hartigan–Wong method, like Lloyd’s method, should have polynomial smoothed running time.

We note that Theorem 1 shows that there exists an instance on which there exists some very specific sequence of iterations that has exponential length. In essence, this means that the exponential running time is only shown for a very specific pivot rule for choosing which point to reassign to which cluster in each iteration. By contrast, Theorem 2 holds for *any* pivot rule, not simply for any particular choice.

2 Preliminaries and Notation

Given vectors $x, y \in \mathbb{R}^d$, we write $\langle x, y \rangle$ for the standard Euclidean inner product on \mathbb{R}^d , and $\|x\| = \sqrt{\langle x, x \rangle}$ for the standard norm.

Given a set of k clusters $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_k\}$, a configuration of a cluster $\mathcal{C}_i \in \mathcal{C}$ is an assignment of a set of points to \mathcal{C}_i . We will denote the clusters by calligraphic letters, and their configurations by regular letters; i.e., the configuration of \mathcal{C}_i will be denoted C_i . This distinction is sometimes useful. For the majority of this paper, however, we will not make this distinction explicitly, and will refer to both a cluster and its configuration interchangeably by regular letters.

Given a finite set of points $S \subseteq \mathbb{R}^d$, we define the center of mass of S as

$$\text{cm}(S) = \frac{1}{|S|} \sum_{x \in S} x.$$

With this definition, we can formally define the objective function of k -means. Let $C = \{C_i\}_{i=1}^k$ be a partition of a finite set of points $\mathcal{X} \subseteq \mathbb{R}^d$. Then the objective function of k -means is

$$\Phi(C) = \sum_{i=1}^k \sum_{x \in C_i} \|x - \text{cm}(C_i)\|^2 = \sum_{i=1}^k \Phi(C_i),$$

where we define $\Phi(C_i) = \sum_{x \in C_i} \|x - \text{cm}(C_i)\|^2$. We will also refer to $\Phi(C)$ as the potential function.

For both the worst-case and smoothed complexity bounds, we need to analyze the improvement of a single iteration. Thus, we need a simple expression for this quantity. Lemmas 3 and 4 allow us to obtain such an expression. These results were already obtained by Telgarsky & Vattani [14].

► **Lemma 3** (Telgarsky & Vattani [14]). *Let S and T be two disjoint nonempty sets of points in \mathbb{R}^d . Then*

$$\Phi(S \cup T) - \Phi(S) - \Phi(T) = \frac{|S| \cdot |T|}{|S| + |T|} \cdot \|\text{cm}(S) - \text{cm}(T)\|^2.$$

► **Lemma 4** (Telgarsky & Vattani [14]). *Let S and T be two disjoint nonempty sets of points in \mathbb{R}^d with $|S| > 1$. Suppose we move a point $x \in S$ from S to T . Then*

$$\Phi(S \setminus \{x\}) + \Phi(T \cup \{x\}) - \Phi(T) - \Phi(S) = \frac{|T|}{|T| + 1} \|\text{cm}(T) - x\|^2 - \frac{|S|}{|S| - 1} \|\text{cm}(S) - x\|^2.$$

52:4 Worst-Case and Smoothed Analysis of the Hartigan–Wong Method

Let C be some clustering of \mathcal{X} . Suppose in some iteration of the Hartigan–Wong method, we move $x \in C_i$ to C_j . Let the gain of this iteration be denoted $\Delta_x(C_i, C_j)$. Then Lemma 4 tells us that

$$\Delta_x(C_i, C_j) = \frac{|C_i|}{|C_i| - 1} \|x - \text{cm}(C_i)\|^2 - \frac{|C_j|}{|C_j| + 1} \|x - \text{cm}(C_j)\|^2.$$

At first sight, it seems like Lemma 4 leaves open the possibility that a cluster is left empty. The following lemma shows that this can never happen.

► **Lemma 5.** *No iteration can leave a cluster empty.*

Proof. Suppose before an iteration, $C_i = \{x\}$ for some $x \in X$, and after the iteration $C'_i = \emptyset$ and $C'_j = C_j \cup \{x\}$, i.e. x is moved from cluster i to cluster j . The gain of this iteration is then (Lemma 3)

$$\Phi(C_i) + \Phi(C_j) - \Phi(\emptyset) - \Phi(C_j \cup \{x\}) = \Phi(C_j) - \Phi(C_j \cup \{x\}) = -\frac{|C_j|}{|C_j| + 1} \|x - \text{cm}(C_j)\|^2 \leq 0,$$

since $\text{cm}(C_i) = x$ and $\Phi(\emptyset) = 0$. Since every iteration must improve the clustering, this concludes the proof. ◀

3 Exponential Lower Bound

In this section, we construct a family of k -means instances on the line on which the Hartigan–Wong method can take an exponential number of iterations before reaching a local optimum. To be precise, we prove the following theorem.

► **Theorem 1 (Restated).** *For each $m \in \mathbb{N}_{\geq 2}$ there exists an instance of k -means clustering on the line with $n = 4m - 3$ points and $k = 2m - 1$ clusters on which the Hartigan–Wong method can take $2^{\Omega(n)}$ iterations to converge to a local optimum.*

The construction we employ is similar to the construction used by Vattani for Lloyd’s method [15]. However, the Hartigan–Wong method only reassigns a single point in each iteration, and we are free to choose which point we reassign. Moreover, we are even free to choose which cluster we move a point to if there are multiple options. This allows us to simplify the construction and embed it in a single dimension, rather than the plane used by Vattani.

We define a set of m gadgets G_i , $i \in \{0, \dots, m - 1\}$. Each gadget except for the “leaf” gadget G_0 consists of four points, and has two clusters $G_i(C_0)$ and $G_i(C_1)$ associated with it. Moreover, each gadget except G_0 has three distinguished states, called “morning”, “afternoon”, and “asleep”. The leaf gadget only has two states, “awake” and “asleep”.

During the morning state, a gadget G_i watches G_{i-1} . If G_{i-1} falls asleep, then it is awoken by G_i ; this is achieved by moving a point of G_i to one of the clusters of G_{i-1} . This allows G_{i-1} to perform a sequence of iterations, which ends with G_{i-1} back in its morning state.

Meanwhile, G_i performs a sequence of iterations that transition it to its afternoon state. During the afternoon state, it once more watches G_{i-1} . When the latter falls asleep, G_i once again wakes G_{i-1} , and transitions itself to its asleep state.

The leaf gadget G_0 , as it does not watch any gadgets, only ever awakens and immediately falls asleep again.

We end the sequence of iterations once gadget $m - 1$ falls asleep. Observe that with this construction, G_i falls asleep twice as often as G_{i+1} . With the condition that G_{m-1} falls asleep once, we obtain a sequence of at least 2^{m-1} iterations. With $n = 4m - 3$, this yields Theorem 1.

For space reasons, we only describe the instance and the exponential-length sequence here. The proof that this sequence is improving, which completes the proof of Theorem 1, is deferred to the full version.

Formal Construction

We now give a detailed construction of a unit gadget, G . All gadgets except for G_0 are scaled and translated versions of G . The unit gadget is a tuple $G = (S, \mathcal{C}_0, \mathcal{C}_1)$, where $S = \{a, b, p, q\} \subseteq \mathbb{R}$, and \mathcal{C}_0 and \mathcal{C}_1 are two clusters. The positions of the points in S are given in Table 1. In addition, the gadget is depicted schematically in Figures 1 and 2. Note that the relative positions of the points in these figures do not correspond to Table 1, but are chosen for visual clarity.

■ **Table 1** Positions of the points in $S(G)$, the leaf point f , and the translation vector t_0 between gadgets G_1 and G_2 .

Point	a	b	p	q	f	t_0
Position	9	6	5	13	0	8

We remark that the points in Table 1 are not simply chosen by trial-and-error. As will be explained shortly, we can obtain from our construction a series of inequalities that must be satisfied by the points in S . We then obtained these points by solving the model

$$\begin{aligned} \min \quad & a^2 + b^2 + p^2 + q^2 + f^2 + t_0^2 \\ \text{s.t.} \quad & \text{each move decreases the clustering cost,} \\ & a, b, p, q, f, t_0 \in \mathbb{Z} \end{aligned}$$

using Gurobi [7]. The first constraint here amounts to satisfying a series of inequalities of the form $\Delta_x(A, B) > 0$ for $x \in S(G)$ and A, B subsets of the points in a gadget and its neighboring gadgets. For space reasons, we defer their derivation and verification to the full version. The objective function here is purely chosen so that Gurobi prefers to choose small integers in the solution.

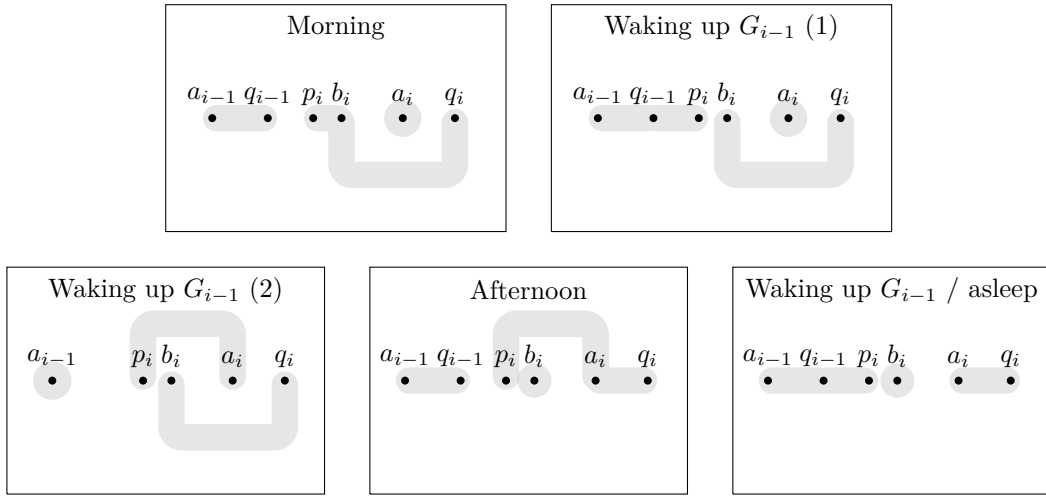
To construct G_i from the unit gadget (for $i \geq 1$), we scale the unit gadget by a factor 5^{i-1} , and translate it by $t_i = \sum_{j=0}^{i-1} 5^j t_0$, where $t_0 = 8$. Since each gadget only ever exchanges points with its neighbors in the sequence we are about to construct, it will suffice in proving Theorem 1 to consider only iterations involving G_i, G_{i-1} and G_{i+1} for some fixed $i > 2$. For the leaf gadget, we simply have $G_0 = (S_0, \mathcal{C}_0)$, where $S_0 = \{f\} = \{0\}$.

Before we go on to construct an improving sequence of exponential length, we define the earlier-mentioned states. For ease of notation, we will refer to the points of G_i as a_i, b_i , and so on, and to the clusters of G_i as $\mathcal{C}_0(G_i)$ and $\mathcal{C}_1(G_i)$. Then we say the state of $G_{i>0}$ is:

- asleep, if $\mathcal{C}_0(G_i) = \{b_i\}$ and $\mathcal{C}_1(G_i) = \{a_i, q_i\}$ (in this state, p_i is in some cluster of G_{i-1});
- morning, if $\mathcal{C}_0(G_i) = \{p_i, q_i, b_i\}$ and $\mathcal{C}_1(G_i) = \{a_i\}$;
- afternoon, if $\mathcal{C}_0(G_i) = \{b_i\}$ and $\mathcal{C}_1(G_i) = \{p_i, q_i, a_i\}$.

For the leaf gadget, we say its state is:

- asleep, if $\mathcal{C}_0(G_0) = \{f\}$;
- awake, otherwise.



■ **Figure 1** Schematic depiction of the interactions between G_i and G_{i-1} during the morning and afternoon phases of G_i .

We now explicitly determine a sequence of iterations of exponential length. In the proof of Theorem 1, we show that this sequence is improving. To analyze the sequence, we consider the perspective of G_i as it wakes up G_{i-1} and falls asleep; and then as it is awoken by G_{i+1} . We first consider only the case that $G_{i-1} \neq G_0$. See Figure 1 and Figure 2 for a schematic depiction of the sequence described below.

Morning

We start with G_i in the morning state, and G_{i-1} asleep. To wake up G_{i-1} , the point p_i moves to $\mathcal{C}_1(G_{i-1})$, which currently contains a_{i-1} and q_{i-1} . This triggers the wakeup phase of G_{i-1} ; we will analyze this phase later from the perspective of G_i . When the wakeup phase completes, $\mathcal{C}_1(G_{i-1})$ contains a_{i-1} and p_i , and p_i moves to $\mathcal{C}_1(G_i)$. Subsequently q_i moves from $\mathcal{C}_0(G_i)$ to $\mathcal{C}_1(G_i)$. Observe that this puts G_i into the afternoon state.

Afternoon

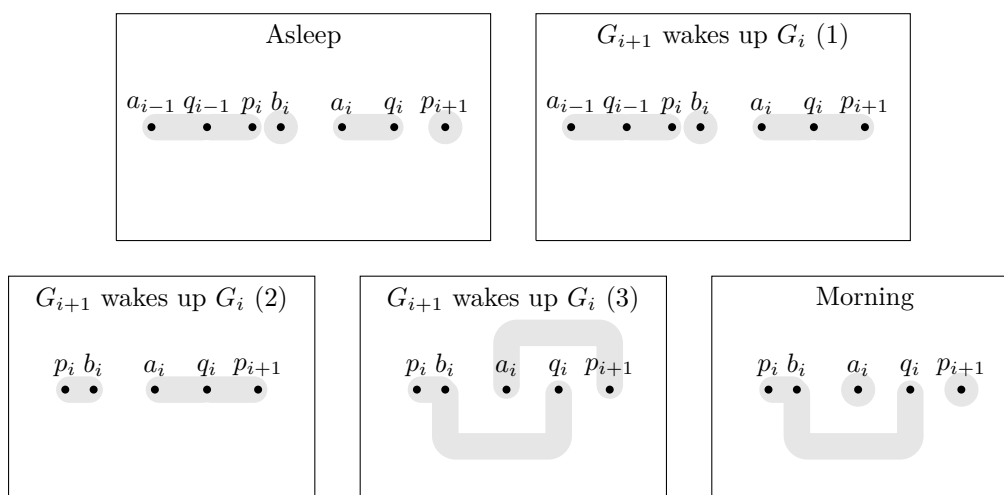
In this state, G_i is once again watching G_{i-1} . Once the latter falls asleep, p_i moves from $\mathcal{C}_1(G_i)$ to $\mathcal{C}_1(G_{i-1})$, which triggers another wakeup phase of G_{i-1} . Additionally, this move causes G_i to fall asleep. Thus, at the end of the wakeup phase of G_{i-1} , we have G_{i+1} wake up G_i .

Waking up

First, the point p_{i+1} joins $\mathcal{C}_1(G_i)$. Next, p_i moves from $\mathcal{C}_1(G_{i-1})$ to $\mathcal{C}_0(G_i)$. Then, q_i moves from $\mathcal{C}_1(G_i)$ to $\mathcal{C}_0(G_i)$, and finally, p_{i+1} leaves $\mathcal{C}_1(G_i)$, and joins either $\mathcal{C}_1(G_{i+1})$ (if G_{i+1} was in the morning state when waking up G_i) or $\mathcal{C}_0(G_{i+1})$ (if G_{i+1} was in the afternoon state; in this case, the move of p_{i+1} occurs during the wakeup phase of G_{i+1}).

Leaf gadget

The leaf gadget does not watch or wake up any other gadgets. It only wakes up when p_1 moves into $\mathcal{C}_0(G_0)$, and falls asleep again when p_1 moves back to a cluster of G_1 .



■ **Figure 2** Schematic depiction of the interactions between G_i , G_{i-1} and G_{i+1} during the wakeup phase of G_i . Note that the final state of G_i corresponds to the first state depicted in Figure 1.

Initialization

The sequence starts with all gadgets in the asleep, except for G_{m-1} , which is in its morning state.

At every step, we have the gadget with the smallest index that is not asleep wake up the gadget that it is watching. From this sequence of iterations, we can retrieve a series of inequalities, each of which encodes the condition that the gain of every iteration must be positive. To prove Theorem 1, we must show that the points in Table 1 satisfy these inequalities.

An implementation of the sequence described above is provided in the following link: <https://pastebin.com/raw/McdArCWg>.

4 Smoothed Analysis

For a smoothed analysis, the first hope might be to straightforwardly adapt a smoothed analysis of Lloyd's algorithm, e.g. that of Arthur, Manthey and Röglin [1]. On closer inspection, however, such analyses strongly rely on a couple of properties of Lloyd's method that are not valid in the Hartigan–Wong method.

First, in Lloyd's algorithm the hyperplane that bisects two cluster centers also separates their corresponding clusters, since every point is always assigned to the cluster center closest to itself. Second, the two stages of Lloyd's algorithm, moving the cluster centers and reassigning points, both decrease the potential. Neither of these properties are satisfied by iterations of the Hartigan–Wong method. Hence, any analysis that relies on either property cannot be easily repurposed.

Instead, we will use a different technique, more closely related to the analysis of the Flip heuristic for Max-Cut with squared Euclidean distances by Etscheid and Röglin [6]. The main result we will work towards in this section is stated in Theorem 2.

4.1 Technical Preliminaries

Let $\mathcal{Y} \subseteq [0, 1]^d$ be a set of n points. Throughout the remainder, we will denote by \mathcal{X} the set of points obtained by perturbing each point in \mathcal{Y} independently by a d -dimensional Gaussian vector of mean 0 and standard deviation $\sigma \leq 1$. Note that this last assumption is not actually a restriction. If $\sigma > 1$, we scale down the set \mathcal{Y} so that $\mathcal{Y} \subseteq [0, 1/\sigma]^d$, and subsequently perturb the points by Gaussian variables with $\sigma = 1$. Since the number of iterations required to terminate is invariant under scaling of the input point set, this is equivalent to the original instance.

Our analysis is based on the standard technique of proving that it is unlikely that a sequence of iterations decreases the potential function by a small amount. For this technique to work, we additionally require the potential function to be bounded from above and from below with sufficiently high probability. Since it is obvious that the potential is non-negative for any clustering, it is enough to guarantee that the perturbed point set \mathcal{X} lies within the hypercube $[-D/2, D/2]^d$ for some finite D . To that end, we have the following lemma.

► **Lemma 6.** *Let $D = \sqrt{2n \ln(nkd)}$. Then $\mathbb{P}(\mathcal{X} \not\subseteq [-D/2, D/2]^d) \leq k^{-n}$.*

Similar results to Lemma 6 can be found in previous works on the smoothed analysis of algorithms on Gaussian-perturbed point sets [13, 1]. The only difference in our version is the value of D . Hence, we omit the proof.

Lemma 6 allows us to assume that all points lie within $[-D/2, D/2]^d$ after the perturbation. Formally, we must take into account the failure event that any point lies outside this hypercube. However, since the probability of this event is at most k^{-n} , this adds only a negligible $+1$ to the smoothed complexity bound which we prove in Theorem 2. We therefore ignore the failure event in the sequel.

We need to show that we can approximate the gain of an iteration if we have a good approximation to the cluster centers. Recall that $\Delta_x(C_i, C_j)$ is the gain of moving a point x from C_i to C_j . Since we wish to use approximations to the centers of C_i and C_j , it is convenient to define the variable

$$\Delta_x^{|C_i|, |C_j|}(a, b) = \frac{|C_i|}{|C_i| - 1} \|x - a\|^2 - \frac{|C_j|}{|C_j| + 1} \|x - b\|^2.$$

This variable is the gain that would be incurred if the centers of C_i and C_j , with fixed sizes $|C_i|$ and $|C_j|$, were a and b . Indeed, note that $\Delta_x^{|C_i|, |C_j|}(\text{cm}(C_i), \text{cm}(C_j)) = \Delta_x(C_i, C_j)$. When their intended values are clear from context, we will often omit the superscripts $|C_i|$ and $|C_j|$ from $\Delta_x^{|C_i|, |C_j|}(a, b)$.

4.2 Approximating Iterations

Before we begin with the analysis, we provide a rough outline. Suppose we tile the hypercube $[-D/2, D/2]^d$ with a rectangular grid of spacing ϵ . Then any point in $[-D/2, D/2]^d$ is at a distance of at most $\sqrt{d}\epsilon$ from some grid point. Since we need the positions of the cluster centers $c_i = \text{cm}(C_i)$ for $i \in [k]$, we guess k grid points c'_i for their positions. If we guess correctly, meaning c'_i is the grid point closest to c_i for each $i \in [k]$, then we can approximate the gain Δ of an iteration by replacing the cluster centers with these grid points in the formula for Δ (Lemma 7).

The price for this approximation is a union bound over all choices of the grid points. However, we can compensate for this by noticing that, when we move a point between clusters, we know exactly how the cluster centers move. Thus, if the guessed grid points

are good approximations, we can obtain new good approximations by moving them the same amount. Thus, we only need to guess once, and can use this guess for a sequence of iterations. Then we can bound the probability that all iterations in this sequence yield a small improvement.

► **Lemma 7.** *Suppose the point x moves from cluster i to cluster j . Let C_i and C_j denote the configurations of these clusters before this move, and let $c_i = \text{cm}(C_i)$ and $c_j = \text{cm}(C_j)$. Let c'_i and c'_j be two points such that $\|c_i - c'_i\|, \|c_j - c'_j\| \leq \epsilon$ for some $0 \leq \epsilon \leq \sqrt{d}D$. Then*

$$|\Delta_x(C_i, C_j) - \Delta_x(c'_i, c'_j)| \leq 9\sqrt{d}D\epsilon,$$

In particular, $\Delta_x(C_i, C_j) \in (0, \epsilon]$ implies $|\Delta_x(c'_i, c'_j)| \leq 10\sqrt{d}D\epsilon$.

Proof. Observe that

$$\|x - c_i\|^2 = \|x - c'_i + c'_i - c_i\|^2 = \|x - c'_i\|^2 + \|c_i - c'_i\|^2 + 2\langle c'_i - c_i, x - c'_i \rangle.$$

Thus,

$$\begin{aligned} \Delta_x(C_i, C_j) &= \Delta_x(c'_i, c'_j) + \frac{|C_i|}{|C_i| - 1} (\|c_i - c'_i\|^2 + 2\langle c'_i - c_i, x - c'_i \rangle) \\ &\quad - \frac{|C_j|}{|C_j| + 1} (\|c_j - c'_j\|^2 + 2\langle c'_j - c_j, x - c'_j \rangle). \end{aligned}$$

By the Cauchy-Schwarz inequality, $|\langle c'_i - c_i, x - c'_i \rangle| \leq \epsilon \cdot \|x - c'_i\|$. Since all points are contained in $[-D/2, D/2]^d$, it holds that $c_i \in [-D/2, D/2]^d$. From this fact and the assumption that $\|c_i - c'_i\| \leq \epsilon \leq \sqrt{d}D$, it follows that $\|x - c'_i\| \leq \sqrt{d}D$.

Moving $\Delta_x(c'_i, c'_j)$ to the left and taking an absolute value, we then obtain

$$|\Delta_x(C_i, C_j) - \Delta_x(c'_i, c'_j)| \leq \left(\frac{|C_i|}{|C_i| - 1} + \frac{|C_j|}{|C_j| + 1} \right) \cdot 3\sqrt{d}D\epsilon.$$

To finish the proof, observe that by Lemma 5 the first term inside the parentheses is at most 2, while the second term is bounded by 1. We then have that $\Delta_x(C_i, C_j) \in (0, \epsilon]$ implies $\Delta_x(c'_i, c'_j) \in (-9\sqrt{d}D\epsilon, (9\sqrt{d}D + 1)\epsilon]$, which yields the lemma. ◀

In the following, we fix a set $A \subseteq \mathcal{X}$ of active points which will move during a sequence of the Hartigan–Wong method. We also fix the configuration of the active points, the sizes of the clusters $|C_1|$ and $|C_2|$ at the start of the sequence, and the order $\pi : A \rightarrow [|A|]$ in which the points move. Observe that these data also fix the sizes of the clusters whenever a new point moves.

While performing a sequence of iterations, the cluster centers move. Hence, even if we have a good approximation to a cluster center, it may not remain a good approximation after the iteration. However, if we know which points are gained and lost by each cluster, then we can compute new good approximations to the cluster centers from the old approximations. The following lemma captures this intuition.

► **Lemma 8.** *Let t_1, t_2 be two iterations of the Hartigan–Wong method in a sequence in which the points $A \subseteq \mathcal{X}$ move, with $t_1 < t_2$. Suppose in the iterations t_1 through $t_2 - 1$, cluster i loses the points S_- and gains the points S_+ . Let $c_i(t)$ denote the cluster center of cluster i before t takes place, and let C_i^t denote its configuration before t . Let $c'_i(t_1) \in \mathbb{R}^d$, and $c'_i(t_2) = \frac{|C_i^{t_1}|}{|C_i^{t_2}|} c'_i(t_1) + \frac{1}{|C_i^{t_2}|} \left(\sum_{x \in S_+} x - \sum_{x \in S_-} x \right)$. Then*

$$\|c'_i(t_2) - c_i(t_2)\| = \frac{|C_i^{t_1}|}{|C_i^{t_2}|} \cdot \|c'_i(t_1) - c_i(t_1)\|.$$

Moreover, if $\|c'_i(0) - c_i(0)\| \leq \epsilon$, then $\|c'_i(t_j) - c_i(t_j)\| \leq 2|A|\epsilon$ for all $j \in [|A|]$.

Proof. Since the center of a cluster is defined as its center of mass, we can write

$$|C_i^{t_2}| \text{cm}(C_i^{t_2}) = \sum_{x \in C_i^{t_1} \cup S_+ \setminus S_-} x = |C_i^{t_1}| \text{cm}(C_i^{t_1}) + \sum_{x \in S_+} x - \sum_{x \in S_-} x.$$

Thus,

$$|C_i^{t_2}| c_i(t_2) = |C_i^{t_1}| c_i(t_1) + \sum_{x \in S_+} x - \sum_{x \in S_-} x.$$

Observe then that

$$\|c_i'(t_2) - c_i(t_2)\| = \frac{|C_i^{t_1}|}{|C_i^{t_2}|} \cdot \|c_i'(t_1) - c_i(t_1)\|.$$

This proves the first claim. To prove the second claim, we set $t_1 = 0$ and $t_2 = t_j$ for some $j \in [|A|]$ to obtain

$$\|c_i(t_j) - c_i'(t_j)\| = \frac{|C_i^0|}{|C_i^{t_j}|} \cdot \|c_i(0) - c_i'(0)\| \leq (|A| + 1)\epsilon \leq 2|A|\epsilon,$$

since at most $|A|$ points are active during any subsequence. \blacktriangleleft

4.3 Analyzing Sequences

We now know that we can closely approximate the gain of a sequence of iterations, provided that we have good approximations to the cluster centers at the start of the sequence. The next step is then to show that there is only a small probability that such an approximate sequence improves the potential by a small amount. For that, we first require the following technical lemma.

► **Lemma 9.** *Let X be a d -dimensional Gaussian random variable with arbitrary mean μ and standard deviation $\sigma \leq 1$, and let $Z = a\|X\|^2 + \langle v, X \rangle$ for fixed $a \in \mathbb{R} \setminus \{0\}$ and $v \in \mathbb{R}^d$. Then the probability that Z falls in an interval of size $\epsilon \leq 1$ is bounded from above by $O\left(\frac{1}{|a|^{1/4} \sqrt{d}} \sqrt{\frac{\epsilon}{\sigma^2}}\right)$.*

Proof. Let $Z_i = aX_i^2 + v_i X_i$, so that $Z = \sum_{i=1}^d Z_i$. We define the auxiliary variable $\bar{Z}_i = Z_i + v_i^2/(4a)$ and set $\bar{Z} = \sum_{i=1}^d \bar{Z}_i$. Since a and v are fixed, the densities of Z and \bar{Z} are identical up to translation, and so we can analyze \bar{Z} instead. Observe that $\bar{Z}_i/a = (X_i + \frac{v_i}{2a})^2$. Thus, \bar{Z}/a is equal in distribution to $\|Y\|^2$, where Y is a d -dimensional Gaussian variable with mean $\mu + v/(4a)$ and variance σ^2 . We see then that \bar{Z}/a has the density of a non-central chi-squared distribution.

For $\lambda \geq 0$, denote by $f(x, \lambda, d)$ the non-central d -dimensional chi-squared density with non-centrality parameter λ and standard deviation σ . Then [10]

$$f(x, \lambda, d) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} f(x, 0, d + 2i).$$

Now observe that $f(x, 0, d)$ is bounded from above by $O\left(1/(\sqrt{d}\sigma^2)\right)$ for $d \geq 2$. We can thus compute for an interval I of size ϵ

$$\mathbb{P}(\|Y\|^2 \in I) = \int_I f(x, \lambda, d) \leq c \cdot \frac{\epsilon}{\sqrt{d}\sigma^2},$$

for some $c > 0$. Moreover, since probabilities are bounded from above by 1, we can replace the right-hand side by

$$O\left(\frac{\sqrt{\epsilon}}{\sqrt[4]{d}\sigma}\right).$$

Adding in the scaling factor of $1/|a|$ then yields the lemma for $d \geq 2$,

For $d = 1$, we have

$$f(x, 0, 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \frac{e^{-\frac{x^2}{2\sigma^2}}}{\sqrt{x/\sigma^2}}.$$

Let I be an interval of size ϵ . Then

$$\mathbb{P}(\|Y\|^2 \in I) = \int_I f(x, \lambda, 1) dx \leq \sum_{i=1}^{\infty} \frac{e^{-\lambda/2}(\lambda/2)^i}{i!} \int_I f(x, 0, 1 + 2i) dx + \int_I f(x, 0, 1) dx.$$

The first term is bounded by $O(\sqrt{\epsilon}/\sigma)$ by the same argument we used for $d \geq 2$. For the second term, we use the expression for $f(x, 0, 1)$ above to bound the integral as

$$\int_I f(x, 0, 1) dx \leq \frac{1}{\sqrt{2\pi\sigma^2}} \int_0^\epsilon \frac{e^{-\frac{x^2}{2\sigma^2}}}{\sqrt{x/\sigma}} dx = O(\sqrt{\epsilon}/\sigma).$$

This proves the lemma for $d = 1$ when we again add in the scaling factor $1/|a|$. \blacktriangleleft

With Lemma 9, we can show that a single fixed approximate iteration is unlikely to yield a small improvement.

► Lemma 10. *Let $a, b \in \mathbb{R}^d$ be fixed. Let $\Delta_x(a, b)$ be the improvement of the first move of x in S , if the cluster centers in this iteration are located at a and b . Let I be an interval of size $\epsilon \leq 1$. Then*

$$\mathbb{P}(\Delta_x(a, b) \in I) = O\left(\frac{n}{\sqrt[4]{d}} \cdot \sqrt{\frac{\epsilon}{\sigma^2}}\right).$$

Proof. By Lemma 4, we have

$$\begin{aligned} \Delta_x(a, b) &= \frac{|C_i|}{|C_i| - 1} \|x - a\|^2 - \frac{|C_j|}{|C_j| + 1} \|x - b\|^2 \\ &= \left(\frac{|C_i|}{|C_i| - 1} - \frac{|C_j|}{|C_j| + 1}\right) \|x\|^2 + \left\langle 2\left(\frac{|C_j|}{|C_j| + 1}b - \frac{|C_i|}{|C_i| - 1}a\right), x \right\rangle \\ &\quad + \frac{|C_i|}{|C_i| - 1} \|a\|^2 - \frac{|C_j|}{|C_j| + 1} \|b\|^2, \end{aligned}$$

where $|C_i|$ and $|C_j|$ denote the sizes of clusters i and j before the iteration, and we assume x moves from cluster i to cluster j .

Since the sizes of the clusters as well as a and b are fixed, the last term in the above is fixed, and hence we may disregard it when analyzing $\mathbb{P}(\Delta_x(a, b) \in I)$. Since x is a Gaussian random variable, we can apply Lemma 9 to find

$$\mathbb{P}(\Delta_x(a, b) \in I) = O\left(\left(\frac{|C_i|}{|C_i| - 1} - \frac{|C_j|}{|C_j| + 1}\right)^{-1} \cdot \frac{1}{\sqrt[4]{d}} \cdot \sqrt{\frac{\epsilon}{\sigma^2}}\right).$$

It remains to bound quantity in the inner brackets from below. Since each cluster is bounded in size by n , we have

$$\frac{|C_i|}{|C_i| - 1} - \frac{|C_j|}{|C_j| + 1} \geq \frac{n}{n-1} - \frac{n}{n+1} = \frac{2n}{(n-1)(n+1)} \geq \frac{1}{n},$$

and we are done. \blacktriangleleft

As stated at the start of the analysis, analyzing a single iteration is not enough to prove Theorem 2. The following lemma extends Lemma 10 to a sequence of iterations, given a fixed point set $A \subseteq \mathcal{X}$ that moves in the sequence.

► **Lemma 11.** *Fix an active set A and starting cluster sizes $|C_i|$ for $i \in [k]$. Moreover, fix an order $\pi : A \rightarrow [|A|]$ in which the points in A move, i.e., $\pi(x) < \pi(y)$ means x moves for the first time before y moves for the first time. Let Δ denote the minimum improvement of a sequence satisfying these hypotheses over all possible configurations of $\mathcal{X} \setminus A$. Then for $\epsilon \leq 1$,*

$$\mathbb{P}(\Delta \leq \epsilon) \leq \left(\frac{2D}{\epsilon}\right)^{kd} \cdot \left(\frac{O(1) \cdot k^{|A|} \cdot d^{3/4} Dn|A|\sqrt{\epsilon}}{\sigma}\right)^{|A|}.$$

Proof. For $x \in A$, let Δ_x denote the improvement of the first move of $x \in A$. We label the points in A as $(x_1, \dots, x_{|A|})$ according to π . Let $\Delta = (\Delta_i)_{i=1}^{|A|}$.

To compute the vector Δ , we would need to know the configuration and positions of the points $P = \mathcal{X} \setminus A$, since these are required to compute the k cluster centers. However, if we had approximations to the cluster centers in every iteration corresponding to the entries of Δ , then we could compute an approximation to Δ by Lemma 7.

Since the cluster centers are convex combinations of points in $[-D/2, D/2]^d$, we know that the cluster centers at the start of S must also lie in $[-D/2, D/2]^d$. Thus, there exist grid points c'_i ($i \in [k]$) within a distance $\sqrt{d}\epsilon$ of the initial cluster centers.

Knowing these grid points, we would like to apply Lemma 8 in order to update the approximate cluster centers whenever a new point moves. We then need to know the points gained and lost by each cluster between first moves of each $x \in A$. Observe that to obtain this information, it suffices to know the configuration of the active points before the first move of each $x \in A$. Thus, we fix these configurations.

We collect the gain of each first move of a point in A , where we replace the cluster centers by these approximations, into a vector Δ' . By the reasoning above and by Lemmas 7 and 8, if there exist initial cluster centers c_i ($i \in [k]$) such that $\Delta_x \in (0, \epsilon]$ for all $x \in A$, then there exist grid points c'_i , such that $|\Delta'_x| \leq 20|A|dD\epsilon$ for all $x \in A$. (Compared to Lemma 7, we gain an extra factor of $2|A|$ due to Lemma 8.)

By this reasoning, it suffices to obtain a bound on $\mathbb{P}(\bigcap_{x \in A} |\Delta'_x| \leq 20|A|dD\epsilon)$. We can then take a union bound over these events for all $(D/\epsilon + 1)^{kd} \leq (2D/\epsilon)^{kd}$ choices of c'_i for $i \in [k]$, and a union bound over the configuration of A before the first move of each $x \in A$.

To show that $\mathbb{P}(\bigcap_{x \in A} |\Delta'_x| \leq 20|A|dD\epsilon)$ is bounded as desired, we consider the following algorithm.

1. Set $t = 1$.
2. Reveal x_t , and compute $\Delta_{x_t}(c'_{i_t}, c'_{j_t})$, where x_t moves from C_{i_t} to C_{j_t} .
3. If $|\Delta_{x_t}(c'_{i_t}, c'_{j_t})| > 20|A|dD\epsilon$, then return GOOD and halt.
4. If $t = |A|$, return BAD.
5. Update the positions of the approximate cluster centers using Lemma 8.
6. Continue executing moves in the sequence until we encounter the first move of x_{t+1} .
Observe that the information we fixed before executing this algorithm suffices to compute approximations to the cluster centers whenever a new point moves.
7. Set $t \leftarrow t + 1$ and go to step 2.

The sequence of iterations improves the potential by at most ϵ only if the above algorithm returns BAD. We now argue that

$$\mathbb{P}(\text{BAD}) \leq \left(O(1) \cdot d^{3/4} D n |A| \sqrt{\epsilon} / \sigma \right)^{|A|}.$$

Let BAD_t be the event that the above algorithm loops for at least t iterations. Then $\mathbb{P}(\text{BAD}) = \mathbb{P}(\text{BAD}_{|A|})$. Since $\mathbb{P}(\text{BAD}_t \mid \neg \text{BAD}_{t-1}) = 0$, we can immediately conclude that for all $t \in \{2, \dots, |A|\}$,

$$\mathbb{P}(\text{BAD}_t) = \mathbb{P}(\text{BAD}_t \mid \text{BAD}_{t-1}) \mathbb{P}(\text{BAD}_{t-1}).$$

By Lemma 10, we have $\mathbb{P}(\text{BAD}_t \mid \text{BAD}_{t-1}) \leq O(1) \cdot d^{3/4} D n |A| \sqrt{\epsilon} / \sigma$. Thus, $\mathbb{P}(\text{BAD}_t)$ is bounded as claimed.

Taking a union bound over all choices of the approximate grid points at the start of the sequence yields the factor $(2D/\epsilon)^{kd}$. Finally, we must take a union bound over the configuration of A before the first move of each $x \in A$, yielding a factor $k^{|A|^2}$, which concludes the proof. \blacktriangleleft

Armed with Lemma 11, we can bound the probability that there exists a sequence in which a fixed number of points moves, which improves the potential by at most ϵ .

► **Lemma 12.** *Let Δ_{\min} denote the minimum improvement of any sequence of moves in which exactly $4kd$ distinct points switch clusters. Then for $\epsilon \leq 1$,*

$$\mathbb{P}(\Delta_{\min} \leq \epsilon) \leq \left(\frac{O(1) \cdot k^{8kd+4} d^{11} D^5 n^{8+\frac{1}{d}} \epsilon}{\sigma^4} \right)^{kd}.$$

Proof. Fix an active set A of $4kd$ distinct points, an order $\pi : A \rightarrow [|A|]$ in which the points in A move, and the sizes of the two clusters at the start of the sequence.

We have by Lemma 11

$$\mathbb{P}(\Delta(S) \leq \epsilon) \leq \left(\frac{2D}{\epsilon} \right)^{kd} \left(\frac{O(1) \cdot k^{2kd} \cdot d^{7/4} D n \sqrt{\epsilon}}{\sigma} \right)^{4kd} = \left(\frac{O(1) \cdot d^7 \cdot k^{8kd} \cdot D^5 n^4 \epsilon}{\sigma^4} \right)^{kd}.$$

We conclude the proof by a union bound over the choices of A , π , and the sizes of the clusters at the start of the sequence, which yields a factor of at most $(4kd)^{4kd} \cdot n^{4kd+1}$. \blacktriangleleft

With Lemma 12, we are in a position to prove the main result of this section. The proof is essentially mechanical, following techniques used in many previous smoothed analyses [1, 3, 4, 5, 6, 13].

► **Theorem 2 (Restated).** *Let $n, k, d \in \mathbb{N}$, and assume $4kd \leq n$. Fix a set of n points $\mathcal{Y} \subseteq [0, 1]^d$, and assume that each point in \mathcal{Y} is independently perturbed by a d -dimensional Gaussian random variable with mean 0 and standard deviation σ , yielding a new set of points \mathcal{X} . Then the expected running time of the Hartigan–Wong method on \mathcal{X} is bounded by*

$$O\left(\frac{k^{12kd+5} d^{12} n^{12.5+\frac{1}{d}} \ln^{4.5}(nkd)}{\sigma^4} \right) = k^{12kd} \cdot \text{poly}(n, k, d, 1/\sigma).$$

Proof. First, we recall that the point set \mathcal{X} is contained in $[-D/2, D/2]^d$. This yields an upper bound for the value of the potential function for the initial clustering C ,

$$\Phi(C) = \sum_{i=1}^k \sum_{x \in C_i} \|x - \text{cm}(C_i)\|^2 \leq knD^2.$$

We divide the sequence of iterations executed by the Hartigan–Wong method into contiguous disjoint blocks during which exactly $4kd$ distinct points move. By Lemma 12, we know that the probability that any such block yields a bad improvement is small.

Let T be the number of such blocks traversed by the heuristic before we reach a local optimum. Then

$$\mathbb{P}(T \geq t) \leq \mathbb{P}\left(\Delta_{\min} \leq \frac{kndD^2}{t}\right) \leq \min\left\{1, \frac{O(1) \cdot k^{8kd+5} d^{12} D^7 n^{9+\frac{1}{d}}}{\sigma^4} \cdot \frac{1}{t}\right\}.$$

This probability becomes nontrivial when

$$t > \left\lceil \frac{O(1) \cdot k^{8kd+5} d^{12} D^7 n^{9+\frac{1}{d}}}{\sigma^4} \right\rceil =: t'.$$

Observe that $t' = \Omega(kndD^2)$, justifying our use of Lemma 12 above. Thus, we find

$$\mathbb{E}(T) = \sum_{t=1}^{k^n} \mathbb{P}(T \geq t) \leq t' + t' \cdot \sum_{t=t'}^{k^n} \frac{1}{t} \leq t' + t' \cdot \int_{t'}^{k^n} \frac{1}{t} dt \leq t' + t' \cdot \ln(k^n).$$

The upper limit of k^n to the sum is simply the number of possible clusterings of n points into k sets, which is a trivial upper bound to the number of iterations. To conclude, we observe that any block in which exactly $4kd$ distinct points move has a length of at most k^{4kd} , as otherwise some clustering would show up twice. Thus, we multiply $\mathbb{E}(T)$ by k^{4kd} to obtain a bound for the smoothed complexity. Finally, we insert the value of $D = \sqrt{2n \ln(nkd)}$. ◀

5 Discussion

Theorems 1 and 2 provide some of the first rigorous theoretical results concerning the Hartigan–Wong method that have been found since Telgarsky & Vattani explored the heuristic in 2010 [14]. Of course, many interesting open questions still remain.

Worst-case construction. Theorem 1 establishes the existence of exponential-length sequences on the line, but leaves open the possibility that a local optimum may be reachable more efficiently by a different improving sequence. To be precise: given an instance of k -means clustering on the line and an initial clustering, does there always exist a sequence of iterations of the Hartigan–Wong method of length $\text{poly}(n, k)$ starting from this clustering and ending in a local optimum? Although the $d = 1$ case appears very restricted at first sight, this question seems surprisingly difficult to answer.

In addition, the construction we use in Theorem 1 requires $k = \Theta(n)$ clusters. This opens up the question whether similar worst-case constructions can be made using fewer, perhaps even $O(1)$, clusters. Note that this is not true for Lloyd’s method, since the number of iterations of Lloyd’s method is bounded by $n^{O(kd)}$ [9], which is polynomial for $k, d \in O(1)$.

Smoothed complexity. Theorem 2 entails, to our knowledge, the first step towards settling the conjecture by Telgarsky & Vattani [14] that the Hartigan–Wong method has polynomial smoothed complexity. Our result is reminiscent of the smoothed complexity bound of Lloyd’s method obtained in 2009 by Manthey & Röglin [12], which is $k^{kd} \cdot \text{poly}(n, 1/\sigma)$. In the case of Lloyd’s method, the smoothed complexity was later settled to $\text{poly}(n, k, d, 1/\sigma)$ [1].

Observe that our bound is polynomial for constant k and d , and even for $kd \log k \in O(\log n)$. While this is certainly an improvement over the trivial upper bound of k^n , it falls short of a true polynomial bound. We hope that our result can function as a first step to a $\text{poly}(n, k, d, 1/\sigma)$ smoothed complexity bound of the Hartigan–Wong method.

We remark that the exponents in the bound in Theorem 2 can be easily improved by a constant factor for $d \geq 2$. The reason is that in Lemma 9, the factor $\sqrt{\epsilon}$ emerges from the $d = 1$ case, while for $d \geq 2$ we could instead obtain ϵ . We chose to combine these cases for the sake of keeping the analysis simple, as we expect the bound in Theorem 2 would be far from optimal regardless.

Improving the smoothed bound. We do not believe that the factor of $k^{O(kd)}$ is inherent in the smoothed complexity of the Hartigan–Wong method, but is rather an artifact of our analysis. To replace this factor by a polynomial in k and d , it seems that significantly new ideas might be needed.

The factors arise from two sources in our analysis. First, we take a union bound over the configuration of the active points each time we apply Lemma 8, yielding factors of $k^{O(kd)}$. Second, we analyze sequences in which $\Theta(kd)$ points move in order to guarantee a significant potential decrease. This incurs a factor of the length of such a sequence, which is another source of a factor $k^{O(kd)}$. We do not see how to avoid such factors when taking our approach.

One avenue for resolving this problem might be to analyze shorter sequences in which a significant number of points move. Angel et al. used such an approach in their analysis of the Flip heuristic for Max-Cut. They identify in any sequence L of moves a shorter subsequence B , such that the number of unique vertices that flip in B is linear in the length of B . The major challenge is then to find sufficient independence in such a short subsequence, which in our case seems challenging, as we need to compensate for a factor ϵ^{-kd} in Lemma 11.

Since our analysis greatly resembles the earlier analysis of the Flip heuristic for Squared Euclidean Max Cut [6], it might be helpful to first improve the latter. This analysis yields a bound of $2^{O(d)} \cdot \text{poly}(n, 1/\sigma)$. If this can be improved to $\text{poly}(n, d, 1/\sigma)$, then it is likely that a similar method can improve on our analysis for the Hartigan–Wong method as well.

References

- 1 David Arthur, Bodo Manthey, and Heiko Röglin. Smoothed Analysis of the k-Means Method. *Journal of the ACM*, 58(5):19:1–19:31, October 2011. doi:10.1145/2027216.2027217.
- 2 David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, USA, January 2007. Society for Industrial and Applied Mathematics.
- 3 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst Case and Probabilistic Analysis of the 2-Opt Algorithm for the TSP. *Algorithmica*, 68(1):190–264, January 2014. doi:10.1007/s00453-013-9801-4.
- 4 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Smoothed Analysis of the 2-Opt Algorithm for the General TSP. *ACM Transactions on Algorithms*, 13(1):10:1–10:15, September 2016. doi:10.1145/2972953.
- 5 Michael Etscheid and Heiko Röglin. Smoothed Analysis of the Squared Euclidean Maximum-Cut Problem. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015*, Lecture Notes in Computer Science, pages 509–520, Berlin, Heidelberg, 2015. Springer. doi:10.1007/978-3-662-48350-3_43.
- 6 Michael Etscheid and Heiko Röglin. Smoothed Analysis of Local Search for the Maximum-Cut Problem. *ACM Transactions on Algorithms*, 13(2):25:1–25:12, March 2017. doi:10.1145/3011870.

- 7 Gurobi Optimization LLC. Gurobi Optimizer Reference Manual. Gurobi Optimization, LLC, 2023.
- 8 J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. doi:10.2307/2346830.
- 9 M Inaba, Naoki Katoh, and Hiroshi Imai. Variance-based k-clustering algorithms by Voronoi diagrams and randomization. *IEICE Transactions on Information and Systems*, E83D, June 2000.
- 10 Norman L. Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. *Continuous Univariate Distributions, Volume 2*. John Wiley & Sons, May 1995.
- 11 S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982. doi:10.1109/TIT.1982.1056489.
- 12 Bodo Manthey and Heiko Röglin. Improved Smoothed Analysis of the k-Means Method. In *Proceedings of the 2009 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 461–470. Society for Industrial and Applied Mathematics, January 2009. doi:10.1137/1.9781611973068.51.
- 13 Bodo Manthey and Rianne Veenstra. Smoothed Analysis of the 2-Opt Heuristic for the TSP: Polynomial Bounds for Gaussian Noise. In Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam, editors, *Algorithms and Computation*, Lecture Notes in Computer Science, pages 579–589, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-45030-3_54.
- 14 Matus Telgarsky and Andrea Vattani. Hartigan’s Method: K-means Clustering without Voronoi. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 820–827. JMLR Workshop and Conference Proceedings, March 2010.
- 15 Andrea Vattani. K-means Requires Exponentially Many Iterations Even in the Plane. *Discrete & Computational Geometry*, 45(4):596–616, June 2011. doi:10.1007/s00454-011-9340-1.

Homomorphism-Distinguishing Closedness for Graphs of Bounded Tree-Width

Daniel Neuen  

University of Bremen, Germany

Abstract

Two graphs are *homomorphism indistinguishable* over a graph class \mathcal{F} , denoted by $G \equiv_{\mathcal{F}} H$, if $\text{hom}(F, G) = \text{hom}(F, H)$ for all $F \in \mathcal{F}$ where $\text{hom}(F, G)$ denotes the number of homomorphisms from F to G . A classical result of Lovász shows that isomorphism between graphs is equivalent to homomorphism indistinguishability over the class of all graphs. More recently, there has been a series of works giving natural algebraic and/or logical characterizations for homomorphism indistinguishability over certain restricted graph classes.

A class of graphs \mathcal{F} is *homomorphism-distinguishing closed* if, for every $F \notin \mathcal{F}$, there are graphs G and H such that $G \equiv_{\mathcal{F}} H$ and $\text{hom}(F, G) \neq \text{hom}(F, H)$. Roberson conjectured that every class closed under taking minors and disjoint unions is homomorphism-distinguishing closed which implies that every such class defines a distinct equivalence relation between graphs. In this work, we confirm this conjecture for the classes \mathcal{T}_k , $k \geq 1$, containing all graphs of tree-width at most k .

As an application of this result, we also characterize which subgraph counts are detected by the k -dimensional Weisfeiler-Leman algorithm. This answers an open question from [Arvind et al., J. Comput. Syst. Sci., 2020].

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph theory; Mathematics of computing \rightarrow Combinatorial algorithms; Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases homomorphism indistinguishability, tree-width, Weisfeiler-Leman algorithm, subgraph counts

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.53

Related Version *arXiv*: <https://arxiv.org/abs/2304.07011>

Acknowledgements I want to thank Tim Seppelt for pointing me to [24, Lemma 4].

1 Introduction

In 1967, Lovász [16] proved that two graphs G and H are isomorphic if and only if $\text{hom}(F, G) = \text{hom}(F, H)$ for every graph F where $\text{hom}(F, G)$ denotes the number of homomorphisms from F to G . A natural follow-up question is to ask whether it is necessary to take the class of all graphs F to obtain the above result, and which kind of other equivalence relations can be obtained by restricting F to come from a proper subclass of all graphs. For a graph class \mathcal{F} , we say that two graphs G and H are \mathcal{F} -*equivalent*, denoted by $G \equiv_{\mathcal{F}} H$, if $\text{hom}(F, G) = \text{hom}(F, H)$ for all $F \in \mathcal{F}$. Hence, Lovász's [16] result says that $\equiv_{\mathcal{A}}$ is identical to the isomorphism relation where \mathcal{A} denotes the class of all graphs.

In recent years, there has been a series of works giving natural algebraic and/or logical characterizations for homomorphism indistinguishability over certain restricted classes of graphs. For example, this includes graphs of bounded tree-width [8], graphs of bounded path-width [13], graphs of bounded tree-depth [12, 13] and the class of planar graphs [17]. In particular, those results imply that the equivalence relations $\equiv_{\mathcal{F}}$ obtained from the mentioned graph classes \mathcal{F} do not correspond to isomorphism, and moreover, these equivalence relations are pairwise distinct.



© Daniel Neuen;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 53; pp. 53:1–53:12



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



In [21], Roberson initiated a more systematic study of the question which types of graph classes \mathcal{F} lead to different equivalence relations $\equiv_{\mathcal{F}}$. A class of graphs \mathcal{F} is called *homomorphism-distinguishing closed* if, for every $F \notin \mathcal{F}$, there are graphs G and H such that $G \equiv_{\mathcal{F}} H$ and $\text{hom}(F, G) \neq \text{hom}(F, H)$.

► **Conjecture 1** (Roberson [21]). *Let \mathcal{F} be a graph class closed under taking disjoint unions and minors. Then \mathcal{F} is homomorphism-distinguishing closed.*

In particular, this conjecture implies that every graph class closed under taking disjoint unions and minors defines a distinct equivalence relation $\equiv_{\mathcal{F}}$. Note that not every graph class is homomorphism-distinguishing closed. For example, the class \mathcal{D}_2 of 2-degenerate graphs (which is not closed under taking minors) is not homomorphism-distinguishing closed since the corresponding equivalence relation defines the isomorphism relation between graphs [8].

For $k \geq 1$ let \mathcal{T}_k denote the class of all graphs of tree-width at most k . Roberson [21] showed that \mathcal{T}_k is homomorphism-distinguishing closed for $k \in \{1, 2\}$. In this work, we generalize this to all $k \geq 1$.

► **Theorem 2.** *The class \mathcal{T}_k is homomorphism-distinguishing closed for all $k \geq 1$.*

For the proof, we rely on known characterizations of homomorphism indistinguishability over the class \mathcal{T}_k [6, 8, 13] and existing constructions of non-isomorphic pairs of graphs that are difficult to distinguish (see, e.g., [2, 5, 21]).

We remark that, since the first publication of the result, it has already been used in [22] to analyse the Lasserre semidefinite programming hierarchy for graph isomorphism via a characterization in terms of homomorphism counts. Also, the results have been used in [9] which in particular uses a similar strategy to prove that the class \mathcal{TD}_q of all graphs of tree-depth at most q is homomorphism-distinguishing closed for all $q \geq 1$.

As an application of this result, we are able to characterize which subgraph counts are detected by the Weisfeiler-Leman algorithm (see also [1]). The Weisfeiler-Leman algorithm (WL) is a standard heuristic in the context of graph isomorphism testing (see, e.g., [2]) which recently also gained attention in a machine learning context [19, 20, 26, 28]. For $k \geq 1$, the k -dimensional Weisfeiler-Leman algorithm (k -WL) computes an isomorphism-invariant coloring of the k -tuples of vertices of a graph G . If the color patterns computed for two graphs G and H do not match, the graphs are non-isomorphic. In this case, we say that k -WL *distinguishes* G and H . It is known that two graphs G and H are distinguished by k -WL if and only if $G \not\equiv_{\mathcal{T}_k} H$, i.e., indistinguishability by k -WL can be characterized by homomorphism indistinguishability over the class of graphs of tree-width at most k [6, 8, 13].

In [10], Fürer initiated research on the question of which subgraph counts are detected by k -WL. Let F and G be two graphs. We write $\text{sub}(F, G)$ to denote the number of subgraphs of G isomorphic to F . We say the function $\text{sub}(F, \cdot)$ is *k -WL invariant* if $\text{sub}(F, G) = \text{sub}(F, H)$ for all graphs G, H that are indistinguishable by k -WL. For example, Fürer [10] shows that $\text{sub}(C_\ell, \cdot)$ is 2-WL invariant for all $\ell \leq 6$ (where C_ℓ denotes the cycle on ℓ vertices), but $\text{sub}(K_4, \cdot)$ is not 2-WL invariant. In [1], Arvind, Fuhlbrück, Köbler and Verbitsky further extended this line of research by showing $\text{sub}(F, \cdot)$ is k -WL invariant for all graphs F that have hereditary tree-width at most k . For a graph F we define its *hereditary tree-width*, denoted by $\text{hdtw}(F)$, to be the maximum tree-width of a homomorphic image of F . Arvind et al. [1] also provide some isolated negative results, but could not obtain a complete classification of which subgraph counts are detected by k -WL even for the special case $k = 2$.

Building on Theorem 2, we provide a complete classification of which subgraph counts are detected by k -WL for all $k \geq 1$. This answers an open question from [1].

► **Theorem 3.** *Let F be a graph and $k \geq 1$. Then $\text{sub}(F, \cdot)$ is k -WL invariant if and only if $\text{hdtw}(F) \leq k$.*

Observe that the backward direction is already proved in [1], i.e., the main contribution of this work is to show that for every graph F with $\text{hdtw}(F) > k$, the k -WL algorithm fails to detect subgraph counts from F .

For the proof, we use a well-known result [4] that allows us to formulate subgraph counts as a linear combination of certain homomorphism counts, and then combine Theorem 2 with an auxiliary lemma from [24].

We stress that Theorem 3 is also relevant in a machine-learning context. Indeed, it is known that the expressive power of graph neural networks (GNNs), which are a common tool for processing graph-structured data, is closely related to the expressive power of k -WL (see, e.g., [19, 20]). On the other hand, counting small subgraph patterns, also called network motifs [18], is a common technique in the study of large networks (see, e.g. [7, 14, 23, 27] for the use of network motifs in computational biology). Hence, it is natural to ask which subgraph counts can be detected by certain GNNs. This question has been studied in [3], but similar to [10, 1] only limited results have been obtained. Exploiting the connections between GNNs and k -WL (see, e.g., [19, 20]), Theorem 3 can provide a much more complete picture of which subgraph counts can be detected by GNNs. In fact, in a recent work, Lanzinger and Barceló [15] extend Theorem 3 to so-called knowledge graphs which are typically considered by GNNs.

We also remark that another extension of Theorem 3 has been obtained by Göbel, Goldberg and Roth [11] who determine the WL-dimension of counting the number of answers to an existential conjunctive query.

2 Preliminaries

A *graph* is a pair $G = (V, E)$ with vertex set $V = V(G)$ and edge relation $E = E(G)$. In this paper all graphs are finite, simple (no loops or multiple edges), and undirected. We denote edges by $vw \in E(G)$ where $v, w \in V(G)$. The *neighborhood* of $v \in V(G)$ is denoted by $N_G(v)$. Moreover, we write $E_G(v)$ to denote the set of edges incident to v . If the graph is clear from context, we usually omit the index G and simply write $N(v)$ and $E(v)$. For $A \subseteq V(G)$ we denote by $G[A]$ the *induced subgraph* of G on A . Also, we denote by $G \setminus A$ the induced subgraph on the complement of A , that is $G \setminus A := G[V(G) \setminus A]$.

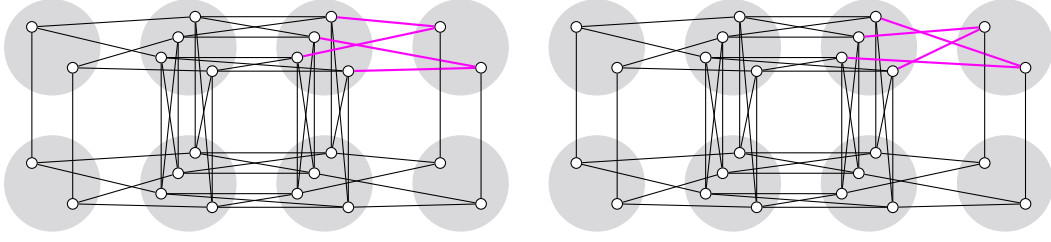
An *isomorphism* from a graph G to another graph H is a bijective mapping $\varphi: V(G) \rightarrow V(H)$ which preserves the edge relation, that is, $vw \in E(G)$ if and only if $\varphi(v)\varphi(w) \in E(H)$ for all $v, w \in V(G)$. Two graphs G and H are *isomorphic* ($G \cong H$) if there is an isomorphism from G to H . We write $\varphi: G \cong H$ to denote that φ is an isomorphism from G to H .

Let F and G be two graphs. A *homomorphism* from F to G is a mapping $\varphi: V(F) \rightarrow V(G)$ such that $\varphi(v)\varphi(w) \in E(G)$ for all $vw \in E(F)$. We write $\text{hom}(F, G)$ to denote the number of homomorphisms from F to G .

Let G be a graph. A graph H is a *minor* of G if H can be obtained from G by deleting vertices and edges of G as well as contracting edges of G . More formally, let $\mathcal{B} = \{B_1, \dots, B_h\}$ be a partition of $V(G)$ such that $G[B_i]$ is connected for all $i \in [h]$. We define G/\mathcal{B} to be the graph with vertex set $V(G/\mathcal{B}) := \mathcal{B}$ and

$$E(G/\mathcal{B}) := \{BB' \mid \exists v \in B, v' \in B': vv' \in E(G)\}.$$

A graph H is a minor of G if there is a partition $\mathcal{B} = \{B_1, \dots, B_h\}$ of connected subsets $B_i \subseteq V(G)$ such that H is isomorphic to a subgraph of G/\mathcal{B} . A graph G *excludes* H as a *minor* if H is not a minor of G .



■ **Figure 1** The figure shows the graphs $\text{CFI}(G)$ and $\text{CFI}^x(G)$ where G is the 2×4 grid. The sets $M_{G, \emptyset}(v)$ and $M_{G, \{u_0\}}(v)$ are highlighted in gray. The vertex u_0 is located in the top-right corner of the grid. The marked edges show the difference between the two graphs.

3 Homomorphism Indistinguishability and Oddomorphisms

Toward the proof of Theorem 2, we need to cover several tools introduced in [21].

▶ **Definition 4** (Roberson [21]). *Let F and G be graphs and suppose φ is a homomorphism from F to G . We say a vertex $a \in V(F)$ is odd (with respect to φ) if $|N_F(a) \cap \varphi^{-1}(v)|$ is odd for every $v \in N_G(\varphi(a))$. Similarly, we say a vertex $a \in V(F)$ is even with respect to φ if $|N_F(a) \cap \varphi^{-1}(v)|$ is even for every $v \in N_G(\varphi(a))$.*

An oddomorphism from F to G is a homomorphism φ from F to G such that

- (I) every vertex $a \in V(F)$ is odd or even (with respect to φ), and
- (II) $\varphi^{-1}(v)$ contains an odd number of odd vertices for every $v \in V(G)$.

A weak oddomorphism from F to G is a homomorphism φ from F to G such that there is a subgraph F' of F for which $\varphi|_{V(F')}$ is an oddomorphism from F' to G .

Next, we introduce a construction for pairs of similar graphs from a base graph G that has also been used in [21]. Actually, variants of this construction have already been used in several earlier works (see, e.g., [2, 5]).

Let G be a graph and let $U \subseteq V(G)$. For $v \in V(G)$ we define $\delta_{v,U} := |\{v\} \cap U|$. We define the graph $\text{CFI}(G, U)$ (the name refers to the authors of [2] where a very similar construction was first used in a related context) with vertex set

$$V(\text{CFI}(G, U)) := \{(v, S) \mid v \in V(G), S \subseteq E(v), |S| \equiv \delta_{v,U} \pmod{2}\}$$

and edge set

$$E(\text{CFI}(G, U)) := \{(v, S)(u, T) \mid uv \in E(G), uv \notin S \Delta T\}$$

(here, $S \Delta T$ denotes the symmetric difference of S and T , i.e., $S \Delta T := (S \setminus T) \cup (T \setminus S)$). For $v \in V(G)$ we also write $M_{G,U}(v) := \{(v, S) \mid S \subseteq E(v), |S| \equiv \delta_{v,U} \pmod{2}\}$ for the vertices in $\text{CFI}(G, U)$ associated with v .

The following lemma is well-known (see, e.g., [2, 21])

▶ **Lemma 5.** *Let G be a connected graph and let $U, U' \subseteq V(G)$. Then $\text{CFI}(G, U) \cong \text{CFI}(G, U')$ if and only if $|U| \equiv |U'| \pmod{2}$.*

We define $\text{CFI}(G) := \text{CFI}(G, \emptyset)$ and $\text{CFI}^x(G) := \text{CFI}(G, \{u_0\})$ for some $u_0 \in V(G)$. A visualization can also be found in Figure 1.

▶ **Theorem 6** (Roberson [21, Theorem 3.13]). *Let F, G be graphs and suppose G is connected. Then $\text{hom}(F, \text{CFI}(G)) \geq \text{hom}(F, \text{CFI}^x(G))$. Moreover, $\text{hom}(F, \text{CFI}(G)) > \text{hom}(F, \text{CFI}^x(G))$ if and only if there exists a weak oddomorphism from F to G .*

We require two additional tools from [21] stated below.

► **Lemma 7** ([21, Lemma 5.6]). *Let F and G be graphs such that there is a weak oddomorphism from F to G . Also suppose G' is a minor of G . Then there is a minor F' of F such that there is an oddomorphism from F' to G' .*

► **Lemma 8** ([21, Theorem 6.2]). *Let \mathcal{F} be a class of graphs such that*

- (1) *if $F \in \mathcal{F}$ and there is a weak oddomorphism from F to G , then $G \in \mathcal{F}$, and*
- (2) *\mathcal{F} is closed under disjoint unions and restrictions to connected components.*

Then \mathcal{F} is homomorphism-distinguishing closed.

4 Graphs of Bounded Tree-Width

In this section, we present the proof of Theorem 2. We rely on game characterizations for graphs of bounded tree-width as well as homomorphism indistinguishability over graphs of tree-width at most k .

4.1 Games

First, we cover the cops-and-robber game that characterizes tree-width of graphs. Fix some integer $k \geq 1$. For a graph G , we define the cops-and-robber game $\text{CopRob}_k(G)$ as follows:

- The game has two players called *Cops* and *Robber*.
- The game proceeds in rounds, each of which is associated with a pair of positions (\bar{v}, u) with $\bar{v} \in (V(G))^k$ and $u \in V(G)$.
- To determine the initial position, the Cops first choose a tuple $\bar{v} = (v_1, \dots, v_k) \in (V(G))^k$ and then the Robber chooses some vertex $u \in V(G) \setminus \{v_1, \dots, v_k\}$ (if no such u exists, the Cops win the play). The initial position of the game is then set to (\bar{v}, u) .
- Each round consists of the following steps. Suppose the current position of the game is $(\bar{v}, u) = ((v_1, \dots, v_k), u)$.
 - (C) The Cops choose some $i \in [k]$ and $v' \in V(G)$.
 - (R) The Robber chooses a vertex $u' \in V(G)$ such that there exists a path from u to u' in $G \setminus \{v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_k\}$. After that, the game moves to position $((v_1, \dots, v_{i-1}, v', v_{i+1}, \dots, v_k), u')$.

If $u \in \{v_1, \dots, v_k\}$ the Cops win. If there is no position of the play such that the Cops win, then the Robber wins.

We say that the Cops (and the Robber, respectively) *win* $\text{CopRob}_k(G)$ if the Cops (and the Robber, respectively) have a winning strategy for the game. We also say that k *cops can catch a robber on G* if the Cops have a winning strategy in this game.

► **Theorem 9** ([25]). *A graph G has tree-width at most k if and only if $k + 1$ cops can catch a robber on G .*

Next, we discuss a game-theoretic characterization of two graphs being indistinguishable via homomorphism counts from graphs of tree-width at most k .

Let $k \geq 1$. For graphs G and H on the same number of vertices, we define the *bijjective k -pebble game* $\text{BP}_k(G, H)$ as follows:

- The game has two players called *Spoiler* and *Duplicator*.
- The game proceeds in rounds, each of which is associated with a pair of positions (\bar{v}, \bar{w}) with $\bar{v} \in (V(G))^k$ and $\bar{w} \in (V(H))^k$.

- To determine the initial position, Duplicator plays a bijection $f: (V(G))^k \rightarrow (V(H))^k$ and Spoiler chooses some $\bar{v} \in (V(G))^k$. The initial position of the game is then set to $(\bar{v}, f(\bar{v}))$.
- Each round consists of the following steps. Suppose the current position of the game is $(\bar{v}, \bar{w}) = ((v_1, \dots, v_k), (w_1, \dots, w_k))$.
 - (S) Spoiler chooses some $i \in [k]$.
 - (D) Duplicator picks a bijection $f: V(G) \rightarrow V(H)$.
 - (S) Spoiler chooses $v \in V(G)$ and sets $w := f(v)$. Then the game moves to position $(\bar{v}[i/v], \bar{w}[i/w])$ where $\bar{v}[i/v] := (v_1, \dots, v_{i-1}, v, v_{i+1}, \dots, v_k)$ is the tuple obtained from \bar{v} by replacing the i -th entry by v .

If mapping each v_i to w_i does not define an isomorphism of the induced subgraphs of G and H , Spoiler wins the play. More precisely, Spoiler wins if there are $i, j \in [k]$ such that $v_i = v_j \not\leftrightarrow w_i = w_j$ or $v_i v_j \in E(G) \not\leftrightarrow w_i w_j \in E(H)$. If there is no position of the play such that Spoiler wins, then Duplicator wins.

We say that Spoiler (and Duplicator, respectively) *wins* $\text{BP}_k(G, H)$ if Spoiler (and Duplicator, respectively) has a winning strategy for the game. Also, for a position (\bar{v}, \bar{w}) with $\bar{v} \in (V(G))^k$ and $\bar{w} \in (V(H))^k$, we say that Spoiler (and Duplicator, respectively) *wins* $\text{BP}_k(G, H)$ *from position* (\bar{v}, \bar{w}) if Spoiler (and Duplicator, respectively) has a winning strategy for the game started at position (\bar{v}, \bar{w}) .

The following theorem follows from [2] and [6, 8, 13].

► **Theorem 10.** *Suppose $k \geq 1$. Let G and H be two graphs. Then $\text{hom}(F, G) = \text{hom}(F, H)$ for every $F \in \mathcal{T}_k$ if and only if Duplicator wins the game $\text{BP}_{k+1}(G, H)$.*

4.2 Indistinguishable Graphs

The main step in the proof of Theorem 2 is to show that $\text{CFI}(G)$ and $\text{CFI}^\times(G)$ can not be distinguished via homomorphism counts from graphs of tree-width at most k for all connected graphs G of tree-width strictly greater than k . The proof follows similar arguments from [5] used to prove a closely related statement. Toward this end, the next lemma provides certain useful isomorphisms between CFI-graphs.

► **Lemma 11.** *Let G be a connected graph and suppose $u, v \in V(G)$. Let P be a path from u to v . Then there is an isomorphism $\varphi: \text{CFI}(G, \{u\}) \cong \text{CFI}(G, \{v\})$ such that*

- (1) $\varphi(M_{G, \{u\}}(w)) = M_{G, \{v\}}(w)$ for all $w \in V(G)$, and
- (2) $\varphi(w, S) = (w, S)$ for all $w \in V(G) \setminus V(P)$ and $(w, S) \in M_{G, \{u\}}(w)$.

Proof. Let $E(P)$ denote the set of edges on the path P . Clearly,

- $|E(P) \cap E(u)| = 1$ and $|E(P) \cap E(v)| = 1$,
- $|E(P) \cap E(w)| = 2$ for all $w \in V(P) \setminus \{u, v\}$, and
- $|E(P) \cap E(w)| = 0$ for all $w \in V(G) \setminus V(P)$.

We define $\varphi(w, S) := (w, S \Delta (E(P) \cap E(w)))$ for all $(w, S) \in \text{CFI}(G, \{u\})$. It is easy to check that $\varphi: \text{CFI}(G, \{u\}) \cong \text{CFI}(G, \{v\})$ and the desired properties are satisfied. ◀

The next lemma forms the key technical step in the proof of Theorem 2.

► **Lemma 12.** *Let G be a connected graph of tree-width $\text{tw}(G) \geq k$. Then Duplicator wins the k -bijjective pebble game played on $\text{CFI}(G)$ and $\text{CFI}^\times(G)$.*

Proof. Let us fix some vertex $u_0 \in V(G)$ so that $\text{CFI}^*(G) = \text{CFI}(G, \{u_0\})$. Since $\text{tw}(G) \geq k$, the Robber has a winning strategy in the cops-and-robber game $\text{CopRob}_k(G)$ by Theorem 9. We translate the winning strategy for the Robber in $\text{CopRob}_k(G)$ into a winning strategy for Duplicator in the k -bijective pebble game played on $\text{CFI}(G)$ and $\text{CFI}^*(G)$.

We first construct the bijection f for the initialization round. Suppose $\bar{x} = (x_1, \dots, x_k) \in (V(\text{CFI}(G)))^k$. We define $A(\bar{x}) := (v_1, \dots, v_k)$ where $v_i \in V(G)$ is the unique vertex such that $x_i \in M_{G, \emptyset}(v_i)$.

Now let u be the vertex chosen by the Robber if the Cops initially place themselves on $A(\bar{x})$. Let P be a shortest path from u to u_0 (recall that G is connected), and let φ denote the isomorphism from $\text{CFI}(G, \{u\})$ to $\text{CFI}(G, \{u_0\})$ constructed in Lemma 11. We set $f(\bar{x}) := (\varphi(x_1), \dots, \varphi(x_k))$. It is easy to see that this gives a bijection f (we use the same isomorphism φ for all tuples \bar{x} having the same associated tuple $A(\bar{x})$).

Now, throughout the game, Duplicator maintains the following invariant. Let (\bar{x}, \bar{y}) denote the current position. Then there is a vertex $u \in V(G)$ and an isomorphism $\varphi: \text{CFI}(G, \{u\}) \cong \text{CFI}(G, \{u_0\})$ such that

- $\varphi(M_{G, \{u\}}(w)) = M_{G, \{u_0\}}(w)$ for all $w \in V(G)$,
- $\varphi(\bar{x}) = \bar{y}$,
- u does not appear in the tuple $A(\bar{x})$, and
- the Robber wins from the position $(A(\bar{x}), u)$, i.e., if the Cops are placed on $A(\bar{x})$ and the Robber is on u .

Note that this condition is satisfied by construction after the initialization round.

Also observe that Duplicator never loses the game in such a position. Indeed, the mapping φ restricts to an isomorphism between $\text{CFI}(G, \emptyset) - M_{G, \emptyset}(u) = \text{CFI}(G, \{u\}) - M_{G, \{u\}}(u)$ and $\text{CFI}(G, \{u_0\}) - M_{G, \{u_0\}}(u)$. Hence, since no vertex associated with u is pebbled in either graph, the pair (\bar{x}, \bar{y}) induces a local isomorphism.

So it remains to show that Duplicator can maintain the above invariant in each round of the k -bijective pebble game. Suppose (\bar{x}, \bar{y}) is the current position. Also let $(A(\bar{x}), u)$ be the associated position in the cops-and-robber game. Suppose that $A(\bar{x}) = (v_1, \dots, v_k)$.

Let $i \in [k]$ denote the index chosen by Spoiler. We describe the bijection f chosen by Duplicator. Let $v \in V(G)$. Let u' be the vertex the Robber moves to if the Cops choose i and v (i.e., the i -th cop changes its position to v) in the position $(A(\bar{x}), u)$. Let P denote a path from u to u' that avoids $\{v_1, \dots, v_k\} \setminus \{v_i\}$. Let ψ denote the isomorphism from $\text{CFI}(G, \{u'\})$ to $\text{CFI}(G, \{u\})$ constructed in Lemma 11. We set $f(x) := \varphi(\psi(x))$ for all $x \in M_{G, \emptyset}(v)$.

It is easy to see that f is a bijection. Let x denote the vertex chosen by Spoiler and let $y := f(x)$. Let $\bar{x}' := \bar{x}[i/x]$ and $\bar{y}' := \bar{y}[i/y]$, i.e., the pair (\bar{x}', \bar{y}') is the new position of the game. Also, we set $\varphi' := \psi \circ \varphi$ (i.e., $\varphi'(z) = \varphi(\psi(z))$) where ψ denotes the isomorphism from $\text{CFI}(G, \{u'\})$ to $\text{CFI}(G, \{u\})$ used in the definition of $f(x)$.

Clearly, $\varphi'(M_{G, \{u'\}}(w)) = M_{G, \{u_0\}}(w)$ for all $w \in V(G)$, since the corresponding conditions are satisfied for the mappings ψ and φ . We have $\varphi'(x) = y$ by definition. All the other entries of \bar{x}' are fixed by the mapping ψ (see Lemma 11, Part (2)) which overall implies that $\varphi'(\bar{x}') = \bar{y}'$. Also, u' does not appear in the tuple $A(\bar{x}')$ by construction, and the Robber wins from the position $(A(\bar{x}'), u')$.

So overall, this means that Duplicator can maintain the above invariant which provides the desired winning strategy. \blacktriangleleft

With this, we are almost ready to prove Theorem 2. The next corollary states the key consequence of Lemma 12 that allows us to apply Lemma 8.

► **Corollary 13.** *Let $k \geq 1$ and let F be a graph of tree-width $\text{tw}(F) \leq k$. Also let G be a graph and suppose there is a weak oddomorphism from F to G . Then $\text{tw}(G) \leq k$.*

Proof. Suppose towards a contradiction that $\text{tw}(G) > k$. Then there is a connected subgraph G' of G such that $\text{tw}(G') > k$. By Lemma 7, we conclude that there is a minor F' of F such that there is an oddomorphism from F' to G' . In particular, $\text{tw}(F') \leq \text{tw}(F) \leq k$. By Theorem 6, we conclude that $\text{hom}(F', \text{CFI}(G')) > \text{hom}(F', \text{CFI}^*(G'))$. Using Theorem 10 it follows that Spoiler wins the $(k+1)$ -bijjective pebble game $\text{BP}_{k+1}(\text{CFI}(G'), \text{CFI}^*(G'))$. But this contradicts Lemma 12 since $\text{tw}(G') \geq k+1$. ◀

Proof of Theorem 2. Let $k \geq 1$ be fixed. By Corollary 13, the class \mathcal{T}_k satisfies Condition 1 from Lemma 8. Also, the class \mathcal{T}_k clearly satisfies Condition 2 from Lemma 8. So \mathcal{T}_k is homomorphism-distinguishing closed by Lemma 8. ◀

5 Weisfeiler-Leman and Subgraph Counts

In this section, we prove Theorem 3. Towards this end, we first need to formally introduce the WL algorithm.

5.1 The Weisfeiler-Leman Algorithm

Let $\chi_1, \chi_2: V^k \rightarrow C$ be colorings of the k -tuples of vertices, where C is some finite set of colors. We say χ_1 *refines* χ_2 , denoted $\chi_1 \preceq \chi_2$, if $\chi_1(\bar{v}) = \chi_1(\bar{w})$ implies $\chi_2(\bar{v}) = \chi_2(\bar{w})$ for all $\bar{v}, \bar{w} \in V^k$. The colorings χ_1 and χ_2 are *equivalent*, denoted $\chi_1 \equiv \chi_2$, if $\chi_1 \preceq \chi_2$ and $\chi_2 \preceq \chi_1$.

We describe the *k-dimensional Weisfeiler-Leman algorithm* (k -WL) for all $k \geq 1$. For an input graph G let $\chi_{(0)}^{k,G}: (V(G))^k \rightarrow C$ be the coloring where each tuple is colored with the isomorphism type of its underlying ordered subgraph. More precisely, $\chi_{(0)}^{k,G}(v_1, \dots, v_k) = \chi_{(0)}^{k,G}(v'_1, \dots, v'_k)$ if and only if, for all $i, j \in [k]$, it holds that $v_i = v_j \Leftrightarrow v'_i = v'_j$ and $v_i v_j \in E(G) \Leftrightarrow v'_i v'_j \in E(G)$.

We then recursively define the coloring $\chi_{(i+1)}^{k,G}$ obtained after $i+1$ rounds of the algorithm (for $i \geq 0$). For $k \geq 2$ and $\bar{v} = (v_1, \dots, v_k) \in (V(G))^k$ we define

$$\chi_{(i+1)}^{k,G}(\bar{v}) := \left(\chi_{(i)}^{k,G}(\bar{v}), \mathcal{M}_i(\bar{v}) \right)$$

where

$$\mathcal{M}_i(\bar{v}) := \left\{ \left(\chi_{(i)}^{k,G}(\bar{v}[1/w]), \dots, \chi_{(i)}^{k,G}(\bar{v}[k/w]) \right) \mid w \in V(G) \right\}$$

and $\bar{v}[i/w] := (v_1, \dots, v_{i-1}, w, v_{i+1}, \dots, v_k)$ is the tuple obtained from \bar{v} by replacing the i -th entry by w . For $k=1$, the definition is similar, but we only iterate over neighbors of v_1 , i.e.,

$$\mathcal{M}_i(v_1) := \left\{ \chi_{(i)}^{k,G}(w) \mid w \in N_G(v_1) \right\}.$$

There is a minimal $i_\infty \geq 0$ such that $\chi_{(i_\infty)}^{k,G} \equiv \chi_{(i_\infty+1)}^{k,G}$ and for this i_∞ we define $\chi^{k,G} := \chi_{(i_\infty)}^{k,G}$.

Let G and H be two graphs. We say that k -WL *distinguishes* G and H if there exists a color c such that

$$\left| \left\{ \bar{v} \in (V(G))^k \mid \chi^{k,G}(\bar{v}) = c \right\} \right| \neq \left| \left\{ \bar{w} \in (V(H))^k \mid \chi^{k,H}(\bar{w}) = c \right\} \right|.$$

We write $G \simeq_k H$ if k -WL does not distinguish G and H .

Recall that \mathcal{T}_k denotes the class of graphs of tree-width at most k . The following characterization follows from [6, 8, 13] (see also Theorem 10).

► **Theorem 14.** *Suppose $k \geq 1$. Let G and H be two graphs. Then $G \simeq_k H$ if and only if $G \equiv_{\mathcal{T}_k} H$.*

Recall that we write $\text{sub}(F, G)$ to denote the number of subgraphs of G isomorphic to F . We write $\text{sub}(F, \cdot)$ to denote the function that maps each graph G to the corresponding subgraph count $\text{sub}(F, G)$.

► **Definition 15.** *Let F be a graph. The function $\text{sub}(F, \cdot)$ is k -WL invariant if*

$$\text{sub}(F, G) = \text{sub}(F, H) \tag{1}$$

for all graphs G, H such that $G \simeq_k H$.

5.2 Subgraph Counts

Using the framework from [4], it is possible to describe the subgraph count $\text{sub}(F, G)$ as a linear combination

$$\text{sub}(F, G) = \sum_{i \in [\ell]} \alpha_i \cdot \text{hom}(F_i, G)$$

for certain graphs F_1, \dots, F_ℓ and coefficients $\alpha_1, \dots, \alpha_\ell \in \mathbb{R}$ that only depend on F . More precisely, the graphs F_1, \dots, F_ℓ are exactly the homomorphic images of F .

► **Definition 16.** *Let F and H be two graphs. We say that H is a homomorphic image of F if there is a surjective homomorphism $\varphi: V(F) \rightarrow V(H)$ such that*

$$E(H) = \{\varphi(v)\varphi(w) \mid vw \in E(F)\}.$$

We write $\text{spasm}(F)$ to denote the set of homomorphic images of F . The hereditary tree-width of F , denoted by $\text{hdtw}(F)$, is the maximum tree-width of a graph in $\text{spasm}(F)$, i.e.,

$$\text{hdtw}(F) := \max_{H \in \text{spasm}(F)} \text{tw}(H).$$

In the following, we assume that $\text{spasm}(F)$ contains only one representative from each isomorphism class, i.e., for every homomorphic image H of F there is exactly one graph $H' \in \text{spasm}(F)$ that is isomorphic to H . In particular, the set $\text{spasm}(F)$ is finite.

The backward direction of Theorem 3 has already been proved in [1].

► **Lemma 17** ([1, Corollary 4.3]). *Let F be a graph such that $\text{hdtw}(F) \leq k$. Then $\text{sub}(F, \cdot)$ is k -WL invariant.*

For the sake of completeness, we still include the simple proof.

Proof. Let F be a graph such that $\text{hdtw}(F) \leq k$ and let $\mathcal{L} := \text{spasm}(F)$. By [4] there is a unique function $\alpha: \mathcal{L} \rightarrow \mathbb{R} \setminus \{0\}$ such that

$$\text{sub}(F, G) = \sum_{L \in \mathcal{L}} \alpha(L) \cdot \text{hom}(L, G)$$

for all graphs G .

Now let G, H be two graphs such that $G \simeq_k H$. Then $\text{hom}(L, G) = \text{hom}(L, H)$ for all graph $L \in \mathcal{T}_k$ by Theorem 14. Since $\text{hdtw}(F) \leq k$, we get that $\mathcal{L} \subseteq \mathcal{T}_k$. So, in particular, $\text{hom}(L, G) = \text{hom}(L, H)$ for all graph $L \in \mathcal{L}$. It follows that $\text{sub}(F, G) = \text{sub}(F, H)$. ◀

For the other direction, we combine Theorem 2 and the following lemma from [24].

► **Lemma 18** ([24, Lemma 4]). *Let \mathcal{F} be a class of graphs that is homomorphism-distinguishing closed. Let \mathcal{L} be a finite set of pairwise non-isomorphic graphs and $\alpha: \mathcal{L} \rightarrow \mathbb{R} \setminus \{0\}$. Also suppose that for all graphs G, H it holds that*

$$G \equiv_{\mathcal{F}} H \implies \sum_{L \in \mathcal{L}} \alpha(L) \cdot \text{hom}(L, G) = \sum_{L \in \mathcal{L}} \alpha(L) \cdot \text{hom}(L, H). \quad (2)$$

Then $\mathcal{L} \subseteq \mathcal{F}$.

► **Lemma 19.** *Let F be a graph such that $\text{sub}(F, \cdot)$ is k -WL invariant. Then $\text{hdtw}(F) \leq k$.*

Proof. Let \mathcal{F} denote the class of graphs of tree-width at most k . By Theorem 2 the class \mathcal{F} is homomorphism-distinguishing closed. Let $\mathcal{L} := \text{spasm}(F)$. By [4] there is a unique function $\alpha: \mathcal{L} \rightarrow \mathbb{R} \setminus \{0\}$ such that

$$\text{sub}(F, G) = \sum_{L \in \mathcal{L}} \alpha(L) \cdot \text{hom}(L, G)$$

for all graphs G . Since $\text{sub}(F, \cdot)$ is k -WL invariant it follows that Equation (2) is satisfied for all graphs G, H using Theorem 14. So $\text{spasm}(F) = \mathcal{L} \subseteq \mathcal{F}$ by Lemma 18. This implies that $\text{hdtw}(F) \leq k$. ◀

Proof of Theorem 3. The theorem follows directly from Lemmas 17 and 19. ◀

6 Conclusion

We proved that for every $k \geq 1$ the class \mathcal{T}_k of all graphs of tree-width at most k is homomorphism-distinguishing closed. As a consequence, we could answer an open question from [1] and precisely classify the subgraph counts detected by k -WL.

Still, Conjecture 1 remains wide open. As an intermediate step, it may be interesting to consider minor- and union-closed classes of bounded tree-width. More precisely, let \mathcal{F} be a graph class closed under taking disjoint unions and minors, and there is some $k \geq 1$ such that every $F \in \mathcal{F}$ has tree-width at most k . Can we show that \mathcal{F} is homomorphism-distinguishing closed? Towards this end, it may also be interesting to obtain a direct proof of Corollary 13 that does not rely on the characterization from Theorem 10.

References

- 1 Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On Weisfeiler-Leman invariance: Subgraph counts and related graph properties. *J. Comput. Syst. Sci.*, 113:42–59, 2020. doi:10.1016/j.jcss.2020.04.003.
- 2 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Comb.*, 12(4):389–410, 1992. doi:10.1007/BF01305232.
- 3 Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/75877cb75154206c4e65e76b88a12712-Abstract.html>.

- 4 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.
- 5 Anuj Dawar and David Richerby. The power of counting logics on restricted classes of finite structures. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2007. doi:10.1007/978-3-540-74915-8_10.
- 6 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász meets Weisfeiler and Leman. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.40.
- 7 Banu Dost, Tomer Shlomi, Nitin Gupta, Eytan Ruppín, Vineet Bafna, and Roded Sharan. QNet: a tool for querying protein interaction networks. *J. Comput. Biol.*, 15(7):913–925, 2008. doi:10.1089/cmb.2007.0172.
- 8 Zdeněk Dvořák. On recognizing graphs by numbers of homomorphisms. *J. Graph Theory*, 64(4):330–342, 2010. doi:10.1002/jgt.20461.
- 9 Eva Fluck, Tim Seppelt, and Gian Luca Spitzer. Going deep and going wide: Counting logic and homomorphism indistinguishability over graphs of bounded treedepth and treewidth. *CoRR*, abs/2308.06044, 2023. doi:10.48550/arXiv.2308.06044.
- 10 Martin Fürer. On the combinatorial power of the Weisfeiler-Lehman algorithm. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 260–271, 2017. doi:10.1007/978-3-319-57586-5_22.
- 11 Andreas Göbel, Leslie Ann Goldberg, and Marc Roth. The Weisfeiler-Leman dimension of existential conjunctive queries. *CoRR*, abs/2310.19006, 2023. doi:10.48550/arXiv.2310.19006.
- 12 Martin Grohe. Counting bounded tree depth homomorphisms. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 507–520. ACM, 2020. doi:10.1145/3373718.3394739.
- 13 Martin Grohe, Gaurav Rattan, and Tim Seppelt. Homomorphism tensors and linear equations. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 70:1–70:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.70.
- 14 Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 52(2):114–132, 2008. doi:10.1007/s00453-007-9008-7.
- 15 Matthias Lanzinger and Pablo Barceló. On the power of the Weisfeiler-Leman test for graph motif parameters. *CoRR*, abs/2309.17053, 2023. doi:10.48550/arXiv.2309.17053.
- 16 László Lovász. Operations with structures. *Acta Math. Acad. Sci. Hungar.*, 18:321–328, 1967. doi:10.1007/BF02280291.
- 17 Laura Mancinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 661–672. IEEE, 2020. doi:10.1109/FOCS46700.2020.00067.

- 18 Ron Milo, Shai S. Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri B. Chklovskii, and Uri Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.
- 19 Christopher Morris, Yaron Lipman, Haggai Maron, Bastian Rieck, Nils M. Kriege, Martin Grohe, Matthias Fey, and Karsten M. Borgwardt. Weisfeiler and Leman go machine learning: The story so far. *CoRR*, abs/2112.09992, 2021. arXiv:2112.09992.
- 20 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33014602.
- 21 David E. Roberson. Oddomorphisms and homomorphism indistinguishability over graphs of bounded degree. *CoRR*, abs/2206.10321, 2022. doi:10.48550/arXiv.2206.10321.
- 22 David E. Roberson and Tim Seppelt. Lasserre hierarchy for graph isomorphism and homomorphism indistinguishability. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 101:1–101:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ICALP.2023.101.
- 23 Jacob Scott, Trey Ideker, Richard M. Karp, and Roded Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *J. Comput. Biol.*, 13(2):133–144, 2006. doi:10.1089/cmb.2006.13.133.
- 24 Tim Seppelt. Logical equivalences, homomorphism indistinguishability, and forbidden minors. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPICs*, pages 82:1–82:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.MFCS.2023.82.
- 25 Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B*, 58(1):22–33, 1993. doi:10.1006/jctb.1993.1027.
- 26 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011. doi:10.5555/1953048.2078187.
- 27 Tomer Shlomi, Daniel Segal, Eytan Ruppin, and Roded Sharan. Qpath: a method for querying pathways in a protein-protein interaction network. *BMC Bioinformatics*, 7:199, 2006. doi:10.1186/1471-2105-7-199.
- 28 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: <https://openreview.net/forum?id=ryGs6iA5Km>.

Positionality in Σ_2^0 and a Completeness Result

Pierre Ohlmann  

Institute of Informatics, University of Warsaw, Poland

Michał Skrzypczak  

Institute of Informatics, University of Warsaw, Poland

Abstract

We study the existence of positional strategies for the protagonist in infinite duration games over arbitrary game graphs. We prove that prefix-independent objectives in Σ_2^0 which are positional and admit a (strongly) neutral letter are exactly those that are recognised by history-deterministic monotone co-Büchi automata over countable ordinals. This generalises a criterion proposed by [Kopczyński, ICALP 2006] and gives an alternative proof of closure under union for these objectives, which was known from [Ohlmann, TheoretCS 2023].

We then give two applications of our result. First, we prove that the mean-payoff objective is positional over arbitrary game graphs. Second, we establish the following completeness result: for any objective W which is prefix-independent, admits a (weakly) neutral letter, and is positional over finite game graphs, there is an objective W' which is equivalent to W over finite game graphs and positional over arbitrary game graphs.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases infinite duration games, positionality, Borel class Σ_2^0 , history determinism

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.54

Funding *Pierre Ohlmann:* Author supported by the European Research Council (grant agreement No 948057 – BOBR).

Michał Skrzypczak: Author supported by the National Science Centre, Poland (grant no. 2021/41/B/ST6/03914).

Acknowledgements We thank Antonio Casares and Lorenzo Clemente for discussions on and around the topic.

1 Introduction

1.1 Context

Games. We study infinite duration games on graphs. In such a game, two players, Eve and Adam, alternate forever in moving a token along the edges of a directed, possibly infinite graph (called *arena*), whose edges are labelled with elements of some set C . An *objective* $W \subseteq C^\omega$ is specified in advance; Eve wins the game if the label of the produced infinite path belongs to W . A *strategy* in such a game is called *positional* if it depends only on the current vertex occupied by the token, regardless of the history of the play.

We are interested in *positional objectives*: those for which existence of a winning strategy for Eve entails existence of a winning positional strategy for Eve, on a arbitrary arena. Sometimes we also consider a weaker property: an objective is *positional over finite arenas* if the above implication holds on any finite arena.



© Pierre Ohlmann and Michał Skrzypczak;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;

Article No. 54; pp. 54:1–54:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Early results. Although the notion of positionality is already present in Shapley’s seminal work [29], the first positionality result for infinite duration games was established by Ehrenfeucht and Mycielsky [10], and it concerns the mean-payoff objective

$$\text{Mean-Payoff}_{\leq 0} = \left\{ w_0 w_1 \cdots \in \mathbb{Z}^\omega \mid \limsup_k \frac{1}{k} \sum_{i=0}^{k-1} w_i \leq 0 \right\},$$

over finite arenas. Nowadays, many proofs are known that establish positionality of mean-payoff games over finite arenas.

Later, and in a different context, Emerson and Jutla [11] as well as Mostowski [23] independently established positionality of the parity objective

$$\text{Parity}_d = \left\{ p_0 p_1 \cdots \in \{0, 1, \dots, d\}^\omega \mid \limsup_k p_k \text{ is even} \right\}$$

over arbitrary arenas. This result was used to give a direct proof of the possibility of complementing automata over infinite trees, which is the key step in modern proofs of Rabin’s theorem on decidability of S2S [27]. By now, several proofs are known for positionality of parity games, some of which apply to arbitrary arenas.

Both parity games and mean-payoff games have been the object of considerable attention over the past three decades; we refer to [12] for a thorough exposition. By symmetry, these games are positional not only for Eve but also for the opponent, a property we call *bi-positionality*. Parity and mean-payoff objectives, as well as the vast majority of objectives that are considered in this context, are *prefix-independent*, that is, invariant under adding or removing finite prefixes.

Bi-positionality. Many efforts were devoted to understanding positionality in the early 2000’s. These culminated in Gimbert and Zielonka’s work [15] establishing a general characterisation of bi-positional objectives over finite arenas, from which it follows that an objective is bi-positional over finite arenas if and only if it is the case for 1-player games. On the other hand, Colcombet and Niwiński [8] established that bi-positionality over arbitrary arenas is very restrictive: any prefix-independent objective which is bi-positional over arbitrary arenas can be recast as a parity objective.

Together, these two results give a good understanding of bi-positional objectives, both over finite and arbitrary arenas.

Positionality for Eve. In contrast, less is known about those objectives which are positional for Eve, regardless of the opponent (this is sometimes called *half-positionality*). This is somewhat surprising, considering that positionality is more in-line with the primary application in synthesis of reactive systems, where the opponent, who models an antagonistic environment, need not have structured strategies. The thesis of Kopczyński [19] proposes a number of results on positionality, but no characterisation. Kopczyński proposed two classes of prefix-independent objectives, *concave objectives* and *monotone objectives*, which are positional respectively over finite and over arbitrary arenas. Both classes are closed under unions, which motivated the following conjecture.

► **Conjecture 1** (Kopczyński’s conjecture [19, 18]). *Prefix-independent positional objectives are closed under unions.*

This conjecture was disproved by Kozachinskiy in the case of finite arenas [20], however, it remains open for arbitrary ones (even in the case of countable unions instead of unions).

Neutral letters. Many of the considered objectives contain a *neutral letter*, that is an element $\varepsilon \in C$ such that W is invariant under removing arbitrary many occurrences of the letter ε from any infinite word. For instance, $\varepsilon = 0$ is a neutral letter of the parity objective Parity_d . There are two variants of this definition, *strongly neutral letter* and *weakly neutral letter*, which are formally introduced in the preliminaries. It is unknown whether adding a neutral letter to a given objective may affect its positionality [19, 25].

Neutral letters are typically used when one wants to modify a given game arena, by allowing players to make some additional decisions. This requires to create intermediate edges in such a way that their labels do not affect the overall outcome of the play.

Borel classes. To stratify the complexity of the considered objectives we use the Borel hierarchy [17]. This follows the classical approach to Gale-Stewart games [13], where the determinacy theorem was gradually proved for more and more complex Borel classes: Σ_2^0 in [31] and Σ_3^0 in [9]. This finally led to Martin’s celebrated result on all Borel objectives [22].

To apply this technique, we assume for the rest of the paper that C is at most countable. Thus, C^ω is a Polish topological space, with open sets of the form $L \cdot C^\omega$ where $L \subseteq C^*$ is arbitrary. Closed sets are those whose complement is open. The class Σ_2^0 contains all sets which can be obtained as a countable union of some closed sets.

Recent developments. A step forward in the study of positionality (for Eve) was recently made by Ohlmann [25] who established that an objective admitting a (strongly) neutral letter is positional over arbitrary arenas if and only if it admits well-ordered monotone universal graphs. Note that this characterisation concerns only positionality over arbitrary arenas. This allowed Ohlmann to prove closure of prefix-independent positional objectives (over arbitrary arenas) admitting a (strongly) neutral letter under finite lexicographic products, and, further assuming membership in Σ_2^0 , under countable unions¹.

Bouyer, Casares, Randour, and Vandenhove [2] also used universal graphs to characterise positionality for objectives recognised by deterministic Büchi automata. They observed that for such an objective W finiteness of the arena does not impact positionality: W is positional over arbitrary arenas if and only if it is positional over finite ones.

Going further, Casares and Ohlmann [6, 4] recently proved a characterisation of positionality for all ω -regular objectives. As a by-product, it follows that Conjecture 1 holds for ω -regular objectives², and that again finiteness of the arena does not impact positionality.

1.2 Contributions

Positionality in Σ_2^0 . As mentioned above, Kopczyński introduced the class of *monotonic objectives*, defined as those of the form $C^\omega \setminus L^\omega$, where L is a language recognised by a finite linearly-ordered automaton with certain monotonicity properties on transitions. He then proved that monotonic objectives are positional over arbitrary arenas. Such objectives are prefix-independent and belong to Σ_2^0 ; our first contribution is to extend Kopczyński’s result to a complete characterisation (up to neutral letters) of positional objectives in Σ_2^0 .

¹ In [25], an assumption called “non-healing” is used. This assumption is in fact implied by membership in Σ_2^0 .

² In fact, Casares proved a strengthening of the conjecture when only one objective is required to be prefix-independent.

54:4 Positionality in Σ_2^0 and a Completeness Result

► **Theorem 2.** *Let $W \subseteq C^\omega$ be a prefix-independent Σ_2^0 objective admitting a strongly neutral letter. Then W is positional over arbitrary arenas if and only if it is recognised by a countable history-deterministic well-founded monotone co-Büchi automaton.*

The proof of Theorem 2 is based on Ohlmann’s *structuration* technique which is the key ingredient to the proof of [25]. As an easy by-product of the above characterisation, we reobtain the result that Kopczynski’s conjecture holds for countable unions of Σ_2^0 objectives (assuming that the given objectives all have strongly neutral letters).

► **Corollary 3.** *If W_0, W_1, \dots are all positional prefix-independent Σ_2^0 objectives, each admitting a strongly neutral letter, then the union $\bigcup_{i \in \mathbb{N}} W_i$ is also positional.*

From finite to arbitrary arenas. The most important natural example of an objective which is positional over finite arenas but not over infinite ones is Mean-Payoff $_{\leq 0}$, as defined above. As a straightforward consequence of their positionality [3, Theorem 3], it holds that over finite arenas, Mean-Payoff $_{\leq 0}$ coincides with the energy condition

$$\text{Bounded} = \left\{ w_0 w_1 \cdots \in \mathbb{Z}^\omega \mid \sup_k \sum_{i=0}^{k-1} w_i \text{ is finite} \right\},$$

which turns out to be positional even over arbitrary arenas [25].

Applying Corollary 3, we establish that with strict threshold, the mean-payoff objective

$$\text{Mean-Payoff}_{< 0} = \left\{ w_0 w_1 \cdots \in \mathbb{Z}^\omega \mid \limsup_k \frac{1}{k} \sum_{i=0}^{k-1} w_i < 0 \right\}$$

is in fact positional over arbitrary arenas.

Now say that two prefix-independent objectives are *finitely equivalent*, written $W \equiv W'$, if they are won by Eve over the same finite arenas. As observed above, Mean-Payoff $_{\leq 0} \equiv$ Bounded, which is positional over arbitrary arenas. Likewise, its complement

$$\mathbb{Z}^\omega \setminus \text{Mean-Payoff}_{\leq 0} = \left\{ w_0 w_1 \cdots \in \mathbb{Z}^\omega \mid \limsup_k \frac{1}{k} \sum_{i=0}^{k-1} w_i \geq 0 \right\}$$

is, up to changing each weight $w \in \mathbb{Z}$ by the opposite one $-w \in \mathbb{Z}$, isomorphic to

$$\left\{ w_0 w_1 \cdots \in \mathbb{Z}^\omega \mid \liminf_k \frac{1}{k} \sum_{i=0}^{k-1} w_i < 0 \right\}.$$

The latter condition is finitely equivalent to Mean-Payoff $_{< 0}$ (where the liminf is replaced with a limsup), which, as explained above, turns out to be positional over arbitrary arenas.

Thus, both Mean-Payoff $_{\leq 0}$ and its complement are finitely equivalent to objectives that are positional over arbitrary arenas. This brings us to our main contribution, which generalises the above observation to any prefix-independent objective admitting a (weakly) neutral letter which is positional over finite arenas.

► **Theorem 4.** *Let $W \subseteq C^\omega$ be a prefix-independent objective which is positional over finite arenas and admits a weakly neutral letter. Then there exists an objective $W' \equiv W$ which is positional over arbitrary arenas.*

Structure of the paper. Section 2 introduces all necessary notions, including Ohlmann’s structurations results. Section 3 proves our characterisation result Theorem 2 and its consequence Corollary 3, and provides a few examples. Then we proceed in Section 4 with establishing positionality of Mean-Payoff $_{<0}$ over arbitrary arenas, and proving Theorem 4.

2 Preliminaries

Graphs. We fix a set of letters C , which we assume to be at most countable. A C -graph G is comprised of a (potentially infinite) set of *vertices* $V(G)$ together with a set of *edges* $E(G) \subseteq V(G) \times C \times V(G)$. An edge $e = (v, c, v') \in E(G)$ is written $v \xrightarrow{c} v'$, with c being the *label* of this edge. We say that e is *outgoing* from v , that it is *incoming* to v' , and that it is *adjacent* to both v and to v' . We assume that each vertex $v \in V(G)$ has at least one outgoing edge (we call this condition being *sinkless*, with a *sink* understood as a vertex with no outgoing edge).

We say that G is *finite* (resp. *countable*) if both $V(G)$ and $E(G)$ are finite (resp. countable). The *size* of a graph is defined to be $|G| = |V(G)|$.

A (finite) *path* is a (finite) sequence of edges with matching endpoints, meaning of the form $v_0 \xrightarrow{c_0} v_1, v_1 \xrightarrow{c_1} v_2, \dots$, which we conveniently write as $v_0 \xrightarrow{c_0} v_1 \xrightarrow{c_1} \dots$. We say that π is a *path from v_0 in G* , and that vertices v_0, v_1, v_2, \dots appearing on the path are *reachable* from v_0 . We use $G[v_0]$ to denote the restriction of G to vertices reachable from v_0 . The *label* of a path π is the sequence $c_0 c_1 \dots$ of labels of its edges; it belongs to C^ω if π is infinite and to C^* otherwise. We sometimes write $v \xrightarrow{w}$ to say that w labels an infinite path from v , or $v \xrightarrow{w} v'$ to say that w labels a finite path from v to v' . We write $L(G, v_0) \subseteq C^\omega$ for the set of labels of all infinite paths from v_0 in G , and $L(G) \subseteq C^\omega$ for the set of labels of all infinite paths in G , that is the union of $L(G, v_0)$ over all $v_0 \in V(G)$.

A *graph morphism* from G to G' is a map $\phi: V(G) \rightarrow V(G')$ such that for every edge $v \xrightarrow{c} v' \in E(G)$, it holds that $\phi(v) \xrightarrow{c} \phi(v') \in E(G')$. We write $G \xrightarrow{\phi} G'$. We sometimes say that G *embeds* in G' or that G' *embeds* G , and we write $G \rightarrow G'$, to say that there exists a morphism from G to G' . Note that $G \rightarrow G'$ implies $L(G) \subseteq L(G')$.

A graph G is v_0 -*rooted* if it has a distinguished vertex $v_0 \in V(G)$ called the *root*. A *tree* T is a t_0 -rooted graph such that all vertices in T admit a unique finite path from the root t_0 .

Games. A C -*arena* is given by a C -graph A together with a partition of its vertices $V(A) = V_{\text{Eve}} \sqcup V_{\text{Adam}}$ into those controlled by Eve V_{Eve} and those controlled by Adam V_{Adam} . A *strategy* (for Eve) (S, π) in an arena A is a graph S together with a surjective morphism $\pi: S \rightarrow A$ satisfying that for every vertex $v \in V_{\text{Adam}}$, every outgoing edge $v \xrightarrow{c} v' \in E(A)$, and every $s \in \pi^{-1}(v)$, there is an outgoing edge $s \xrightarrow{c} s' \in E(S)$ with $\pi(s') = v'$. Recall that under our assumptions every vertex needs to have at least one outgoing edge, thus for every $v \in V_{\text{Eve}}$ and every $s \in \pi^{-1}(v)$ there must be at least one outgoing edge from s in S .

The example arenas in this work are drawn following a standard notation, where circles (resp. squares) denote vertices controlled by Eve (resp. Adam). Vertices with a single outgoing edge are denoted by a simple dot, it does not matter who controls them.

A strategy is *positional* if π is injective. In this case, we can assume that $V(S) = V(A)$ and $E(S) \subseteq E(A)$, with π being identity.

An *objective* is a set $W \subseteq C^\omega$ of infinite sequences of elements of C . In this paper, we will always work with *prefix-independent* objectives, meaning objectives which satisfy $cW = W$ for all $c \in C$; this allows us to simplify many of the definitions. A graph G *satisfies* an objective W if $L(G) \subseteq W$. A *game* is given by a C -arena A together with an objective W .

54:6 Positionality in Σ_2^0 and a Completeness Result

It is *winning* (for Eve) if there is a strategy (S, π) such that S satisfies W . In this case, we also say that Eve *wins* the game (A, W) with the strategy (S, π) . We say that an objective W is *positional* (over finite arenas or over arbitrary arenas) if for any (finite or arbitrary) arena A , if Eve wins the game (A, W) then she wins (A, W) with a positional strategy.

Neutral letters. A letter $\varepsilon \in C$ is said to be *weakly neutral* for an objective $W \subseteq C^\omega$ if for any word $w \in C^\omega$ decomposed into $w = w_0 w_1 \dots$ with non-empty words $w_i \in C^+$,

$$w \in W \iff \varepsilon w_0 \varepsilon w_1 \varepsilon \dots \in W.$$

A weakly neutral letter $\varepsilon \in C$ is *strongly neutral* if in the above, the w_i can be chosen empty, and moreover, $\varepsilon^\omega \in W$.

A few examples: for the parity objective, the priority 0 is strongly neutral; for Bounded, the weight 0 is strongly neutral; for Mean-Payoff $_{\leq 0}$, the letter 0 is only weakly neutral (because $1^\omega \notin \text{Mean-Payoff}_{\leq 0}$ however $010010001 \dots \in \text{Mean-Payoff}_{\leq 0}$), and likewise for Mean-Payoff $_{< 0}$ because $0^\omega \notin \text{Mean-Payoff}_{< 0}$.

Monotone and universal graphs. An *ordered graph* is a graph G equipped with a total order \geq on its set of vertices $V(G)$. We say that it is *monotone* if

$$v \geq u \xrightarrow{c} u' \geq v' \text{ in } G \quad \text{implies} \quad v \xrightarrow{c} v' \in E(G).$$

Such a graph is *well founded* if the order \geq on $V(G)$ is well founded.

We will use a variant of universality called (uniform) *almost-universality* (for trees), which is convenient when working with prefix-independent objectives. A C -graph U is *almost W -universal*, if U satisfies W , and for any tree T satisfying W , there is a vertex $t \in V(T)$ such that $T[t] \rightarrow U$. We will rely on the following inductive result from [25].

► **Theorem 5** (Follows from Theorem 3.2 and Lemma 4.5 in [25]). *Let $W \subseteq C^\omega$ be a prefix-independent objective such that there is a graph which is almost W -universal. Then W is positional over arbitrary arenas.*

Structuration results. The following results were proved in Ohlmann's PhD thesis (Theorems 3.1 and 3.2 in [24]); the two incomparable variants stem from two different techniques.

► **Lemma 6** (Finite structuration). *Let W be a prefix-independent objective which is positional over finite arenas and admits a weakly neutral letter, and let G be a finite graph satisfying W . Then there is a monotone graph G' satisfying W such that $G \rightarrow G'$.*

► **Lemma 7** (Infinite structuration). *Let W be a prefix-independent objective which is positional over arbitrary arenas and admits a strongly neutral letter, and let G be any graph satisfying W . Then there is a well-founded monotone graph G' satisfying W such that $G \rightarrow G'$.*

Note that in both results, we may assume that $|G'| \leq |G|$, simply by restricting to the image of G . Details of the proof of Lemma 7 can be found in [25, Theorem 3]; Lemma 6 appears only in Ohlmann's PhD thesis [24].

Automata. A *co-Büchi automaton* over C is a q_0 -rooted $C \times \{\mathcal{N}, \mathcal{F}\}$ -graph A . In this context, vertices $V(A)$ are called *states*, edges $E(A)$ are called *transitions*, and the root q_0 is called the *initial state*. Moreover, transitions of the form $q \xrightarrow{(c, \mathcal{N})} q'$ are called *normal transitions* and simply denoted $q \xrightarrow{c} q'$, while transitions of the form $q \xrightarrow{(c, \mathcal{F})} q'$ are called

co-Büchi transitions and denoted $q \xrightarrow{c} q'$. For simplicity, we assume automata to be *complete* (for any state q and any letter c , there is at least one outgoing transition labelled c from q) and *reachable* (for any state q there is some path from q_0 to q in A).

A path $q_0 \xrightarrow{(c_0, a_0)} q_1 \xrightarrow{(c_1, a_1)} \dots$ in A is *accepting* if it contains only finitely many co-Büchi transitions, meaning that only finitely many of a_i equal \mathcal{F} . If $q \in V(A)$ is a state then define the *language* $L(A, q) \subseteq C^\omega$ of a co-Büchi automaton *from a state* $q \in V(A)$ as the set of infinite words which label accepting paths from q in A . The *language* of A denoted $L(A)$ is $L(A, q_0)$. Note that in this paper, automata are not assumed to be finite.

We say that an automaton is *monotone* if it is monotone as a $C \times \{\mathcal{N}, \mathcal{F}\}$ -graph. Likewise, morphisms between automata are just morphisms of the corresponding $C \times \{\mathcal{N}, \mathcal{F}\}$ -graphs that moreover preserve the initial state. Note that $A \rightarrow A'$ implies $L(A) \subseteq L(A')$. A co-Büchi automaton is *deterministic* if for each state $q \in V(A)$ and each letter $c \in C$ there is exactly one transition labelled by c outgoing from q .

A *resolver* for an automaton A is a deterministic automaton R with a morphism $R \rightarrow A$. Note that the existence of this morphism implies that $L(R) \subseteq L(A)$. Such a resolver is *sound* if additionally $L(R) \supseteq L(A)$ (and thus $L(R) = L(A)$). A co-Büchi automaton is *history-deterministic* if there exists a sound resolver R . Our definition of history-determinism is slightly non-standard, but it fits well with our overall use of morphisms and of possibly infinite automata. This point of view was also adopted by Colcombet (see [7, Definition 13]). For more details on history-determinism of co-Büchi automata, we refer to [21, 1, 28].

We often make use of the following simple lemma, which follows directly from the definitions and the fact that composing morphisms results in a morphism.

► **Lemma 8.** *Let A, A' be automata such that $A \rightarrow A'$, A is history-deterministic, and $L(A) = L(A')$. Then A' is history-deterministic.*

Say that an automaton A is *saturated* if it has all possible co-Büchi transitions: $V(A) \times (C \times \{\mathcal{F}\}) \times V(A) \subseteq E(A)$. The *saturation* of an automaton A is obtained from A by adding all possible co-Büchi transitions. Similar techniques of saturating co-Büchi automata have been previously used to study their structure [21, 16, 28].

Note that languages of saturated automata are always prefix-independent. The lemma below states that co-Büchi transitions are somewhat irrelevant in history-deterministic automata recognising prefix-independent languages.

► **Lemma 9.** *Let A be a history-deterministic automaton recognising a prefix-independent language and let A' be its saturation. Then $L(A) = L(A')$ and A' is history-deterministic. Moreover, $L(A') = L(A', q)$ for any state q of A' .*

Proof. Clearly $A \rightarrow A'$ thus $L(A) \subseteq L(A')$; it suffices to prove $L(A') \subseteq L(A)$ and conclude by Lemma 8. Let $w_0 w_1 \dots \in L(A')$ and let $q_0 \xrightarrow{(w_0, a_0)} q_1 \xrightarrow{(w_1, a_1)} \dots$ be an accepting path for w in A' . Then for some i , $q_i \xrightarrow{(w_i, a_i)} q_{i+1} \xrightarrow{(w_{i+1}, a_{i+1})} \dots$ is comprised only of normal transitions. Thus, this suffix of the path does not use edges added during the saturation process, which means this suffix is an accepting path in A . We conclude that $w_i w_{i+1} \dots \in L(A)$ and thus $w \in L(A)$ by prefix-independence.

The claim that $L(A', q)$ is independent on q follows directly from prefix-independence and the fact that A' is saturated. ◀

3 Positional prefix-independent Σ_2^0 objectives

3.1 A characterisation

Recall that Σ_2^0 objectives are countable unions of closed objectives; for the purpose of this paper it is convenient to observe that these are exactly those objectives recognised by (countable) deterministic co-Büchi automata (see for instance [30]).

The goal of the section is to prove Theorem 2 which we now restate for convenience.

► **Theorem 2.** *Let $W \subseteq C^\omega$ be a prefix-independent Σ_2^0 objective admitting a strongly neutral letter. Then W is positional over arbitrary arenas if and only if it is recognised by a countable history-deterministic well-founded monotone co-Büchi automaton.*

Before moving on to the proof, we proceed with a quick technical statement that allows us to put automata in a slightly more convenient form.

► **Lemma 10.** *Let A be a history-deterministic automaton recognising a non-empty prefix-independent language. There exists a history-deterministic automaton A' with $L(A') = L(A)$ and such that from every state $q' \in V(A')$, there is an infinite path comprised only of normal transitions. Moreover, if A is countable, well founded, and monotone, then so is A' .*

Proof. Let $V \subseteq V(A)$ be the set of states $q \in V(A)$ from which there is an infinite path of normal transitions. Note that $V \neq \emptyset$ since $L(A)$ is non-empty. First, since every path from $V(A) \setminus V$ visits at least one co-Büchi transition, we turn all normal transitions adjacent to states in $V(A) \setminus V$ into co-Büchi ones; this does not affect $L(A)$ or history-determinism. Next, we saturate A and restrict it to V . Call A' the resulting automaton; if $q_0 \notin V$ then we pick the initial state q'_0 of A' arbitrarily in V . It is clear that restricting A to some subset of states, changing the initial state, as well as saturating, are operations that preserve being countable, well founded, and monotone.

We claim that $L(A) = L(A')$. The inclusion $L(A') \subseteq L(A)$ follows from the proof of Lemma 9 so we focus on the converse: let $w = w_0w_1 \cdots \in L(A)$ and take an accepting path π for w . Then there is a suffix of π which remains in V and therefore defines a path in A' ; we conclude thanks to prefix-independence of $L(A')$.

It remains to see that A' is history-deterministic. For this, we observe that any transition adjacent to states in $V(A) \setminus V$ is a co-Büchi transition; therefore the map $\phi : V(A) \rightarrow V(A') = V$ which is identity on V and sends $V(A) \setminus V$ to the initial state of A' defines a morphism $A \rightarrow A'$. We conclude by Lemma 8. ◀

To prove Theorem 2, we separate both directions so as to provide more precise hypotheses.

► **Lemma 11.** *Let W be a prefix-independent Σ_2^0 objective admitting a strongly neutral letter. Then W is recognised by a countable history-deterministic monotone well-founded automaton.*

Proof. If $W = \emptyset$ then the saturated automaton with a single state and no normal transitions gives the wanted result; therefore we assume W to be non-empty. Let A be a history-deterministic co-Büchi automaton recognising W with initial state q_0 ; thanks to Lemma 10 we assume that every state in A participates in an infinite path of normal transitions. Let G be the C -graph obtained from A by removing all the co-Büchi transitions. The fact that G is sinkless (and therefore, G is indeed a graph) follows from the assumption on A . Since W is prefix-independent, it holds that G satisfies W .

Apply the infinite structuration result (Lemma 7, which requires the strongly neutral letter) to G to obtain a well-founded monotone graph G' satisfying W and such that $G \xrightarrow{\phi} G'$. Note that we may restrict $V(G')$ to the image of ϕ . Due to the fact that C is countable, this guarantees that G' is countable.

Now let A' be the co-Büchi automaton obtained from G' by turning every edge into a normal transition, setting the initial state to be $q'_0 = \phi(q_0)$, and saturating. Note that A' is countable monotone and well-founded; we claim that A' is history-deterministic and recognises W , as required.

Let $w \in L(A')$. Then $w = ww'$ where $w' \in L(G') \subseteq W$. It follows from prefix-independence that $w \in W$. Conversely, let $w_0w_1 \dots \in W$ as witnessed by an accepting path $\pi = q_0 \xrightarrow{(w_0, a_0)} q_1 \xrightarrow{(w_1, a_1)} \dots$ from q_0 in A . This path has only finitely many co-Büchi transitions.

Then consider the path $\pi' = \phi(q_0) \xrightarrow{w_0} \phi(q_1) \xrightarrow{w_1} \dots$ in A' , where we use co-Büchi transitions only when necessary, meaning when there is no normal transition $\phi(q_i) \xrightarrow{w_i} \phi(q_{i+1})$ in A' . Since π visits only finitely many co-Büchi transitions, it is eventually a path in G , and thus since ϕ is a morphism, π' is eventually a path in G' , and hence it sees only finitely many co-Büchi transitions in A' . Hence $L(A') = W$.

It remains to show that A' is history-deterministic. But since A' is saturated and $G \rightarrow G'$ we have $A \rightarrow A'$ and thus Lemma 8 concludes. \blacktriangleleft

For the converse direction, we do not require a neutral letter.

► **Lemma 12.** *If W is a prefix-independent objective recognised by a countable history-deterministic monotone well-founded co-Büchi automaton then W is positional over arbitrary arenas.*

Proof. As previously, if W is empty then it is trivially positional, so we assume that W is non-empty, and we take an automaton A satisfying the hypotheses above and apply Lemma 10 so that every state participates in an infinite path of normal transitions. Let U be the C -graph obtained from A by removing all co-Büchi transitions and turning normal transitions into edges; thanks to Lemma 10, U is sinkless so it is indeed a graph. We prove that U is almost W -universal for trees. Let T be a tree satisfying W and let t_0 be its root.

Since A is history-deterministic, there is a mapping $\phi : V(T) \rightarrow V(A)$ such that for each edge $t \xrightarrow{c} t' \in E(T)$, there is a transition $\phi(t) \xrightarrow{(c, a)} \phi(t')$ in A with some $a \in \{\mathcal{N}, \mathcal{F}\}$, and such that for all infinite paths $t_0 \xrightarrow{w_0} t_1 \xrightarrow{w_1} \dots$ in T , there are only finitely many co-Büchi transitions on the path $\phi(t_0) \xrightarrow{(w_0, a_0)} \phi(t_1) \xrightarrow{(w_1, a_1)} \dots$ in A .

▷ **Claim 13.** There is a vertex $t'_0 \in V(T)$ such that for all infinite paths $t'_0 \xrightarrow{w_0} t'_1 \xrightarrow{w_1} \dots$ from t'_0 in T , there is no co-Büchi transition on the path $\phi(t_0) \xrightarrow{w_0} \phi(t_1) \xrightarrow{w_1} \dots$ in A .

Proof. Assume towards contradiction that no such vertex exists. Then starting from the root t_0 , we build an infinite path $t_0 \xrightarrow{w_0} t_1 \xrightarrow{w_1} \dots$ in T such that $\phi(t_0) \xrightarrow{w_0} \phi(t_1) \xrightarrow{w_1} \dots$ has infinitely many co-Büchi transitions in A . Indeed, assuming the path built up to t_i , we simply pick $t_i \xrightarrow{w_i} t_{i+1}$ such that there is a co-Büchi transition in A on the corresponding path $\phi(t_i) \xrightarrow{w_i} \phi(t_{i+1})$. Thus, we constructed a path contradicting the observation below: this path has infinitely many co-Büchi transitions in A . \triangleleft

There remains to observe that ϕ maps $T[t'_0]$ to U , and thus U is almost W -universal for trees. We conclude by applying Lemma 5. \blacktriangleleft

3.2 A few examples

Kopczyński-monotonic objectives. In our terminology, Kopczyński’s monotonic objectives correspond to the prefix-independent languages that are recognised by finite monotone co-Büchi automata. Note that such automata are of course well-founded, but also they are history-deterministic (even determinisable by pruning): one should always follow a transition to a maximal state. Therefore our result proves that such objectives are positional over arbitrary arenas. A very easy example is the co-Büchi objective

$$\text{co-Büchi} = \{w \in \{\mathcal{N}, \mathcal{F}\}^\omega \mid w \text{ has finitely many occurrences of } \mathcal{F}\},$$

which is recognised by a (monotone) automaton with a single state. Some more advanced examples are given in Figure 1.



■ **Figure 1** Two finite monotone co-Büchi automata recognising prefix-independent languages. For clarity, the co-Büchi transitions are not depicted but connect every pair of states; likewise, edges following from monotonicity (such as the dashed ones for example), are omitted. The automaton on the left recognises words with finitely many ab infixes. The automaton on the right recognises words with finitely many infixes in $c(a^*cb^*)^+c$.

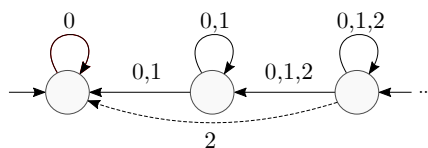
Finite support. The finite support objective is defined over ω by

$$\text{Finite} = \{w \in \omega^\omega \mid \text{finitely many distinct letters appear in } w\}$$

Consider the automaton A over $V(A) = \omega$ with

$$v \xrightarrow{w} v' \in E(A) \iff w, v' \leq v,$$

co-Büchi transitions everywhere, and initial state 0 (see Figure 2).



■ **Figure 2** An automaton A for objective Finite. Co-Büchi edges, as well as some edges following from monotonicity (such as the dashed one) are omitted for clarity.

It is countable, history-deterministic, well-founded, and monotone and recognises $L(A) = \text{Finite}$. Details of the proof are easy and left to the reader. Positionality of Finite can also be established by Corollary 3, as it is a countable union of the safety languages $F^\omega \subseteq \omega^\omega$, where F ranges over finite subsets of ω . As far as we are aware, this result is novel.³

³ A similar positionality result is proved in [14], but it assumes finite degree of the arena, vertex-labels (which is more restrictive), and injectivity of the colouring of the arena.

Energy objectives. Recall the energy objective

$$\text{Bounded} = \left\{ w_0 w_1 \dots \in \mathbb{Z}^\omega \mid \sup_k \sum_{i=0}^{k-1} w_i \text{ is finite} \right\},$$

which is prefix-independent and belongs to Σ_2^0 . Consider the automaton A whose set of states is ω , with the initial state 0 and with all possible co-Büchi transitions, and normal transitions of the form $v \xrightarrow{w} v'$ where $w \leq v - v'$. Note that A is well-founded and monotone, so we should prove that it is history-deterministic and recognises Bounded.

Note that any infinite path of normal edges $v_0 \xrightarrow{w_0} v_1 \xrightarrow{w_1} \dots$ in A is such that for all i , $w_i \leq v_i - v_{i+1}$, and therefore

$$\sum_{i=0}^{k-1} w_i \leq v_0 - v_k \leq v_0$$

and thus $L(A) \subseteq \text{Bounded}$.

A resolver for A works as follows: keep a counter c (initialised to zero), and along the run, from a vertex v and when reading an edge w ,

- if $v \geq w$ then take the normal transition $v \xrightarrow{w} v - w$;
- otherwise, take the co-Büchi transition $v \xrightarrow{w} c$ and increment the counter.

Eventually non-increasing objective. Over the alphabet ω , consider the objective

$$\text{ENI} = \{ w_0 w_1 \dots \in \omega^\omega \mid \text{there are finitely many } i \text{ such that } w_{i+1} > w_i \}.$$

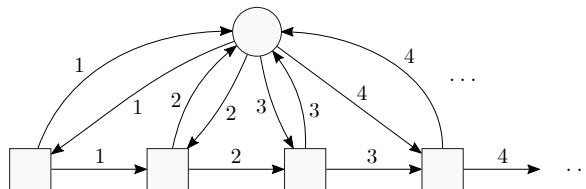
Note that since ω is well-founded, a sequence belongs to ENI if and only if it is eventually constant. Consider the automaton A over ω with the initial state 0, with all possible co-Büchi transitions, and with normal transitions $v \xrightarrow{w} v'$ if and only if $v \geq w \geq v'$. Note that A is countable, well-founded, and monotone, so we should prove that it recognises ENI and is history-deterministic.

First, note that any infinite path of normal edges $v_0 \xrightarrow{w_0} v_1 \xrightarrow{w_1} \dots$ in A is such that $v_0 \geq w_0 \geq v_1 \geq w_1 \geq \dots$, and therefore $L(A) \subseteq \text{ENI}$. A sound resolver for A simply goes to the state w when reading a letter w , using a normal transition if possible, and a co-Büchi transition otherwise. We leave the formal definition to the reader.

Eventually non-decreasing objective. In contrast, the objective

$$\text{END} = \{ w_0 w_1 \dots \in \omega^\omega \mid \text{there are finitely many } i \text{ such that } w_{i+1} < w_i \}$$

is not positional over arbitrary arenas, as witnessed by Figure 3.



■ **Figure 3** An arena over which Eve requires a non-positional strategy in order to produce a sequence which is eventually non-decreasing.

3.3 Closure under countable unions

We now move on to Corollary 3, which answers Kopczyński's conjecture in the affirmative in the case of Σ_2^0 objectives.

► **Corollary 3.** *If W_0, W_1, \dots are all positional prefix-independent Σ_2^0 objectives, each admitting a strongly neutral letter, then the union $\bigcup_{i \in \mathbb{N}} W_i$ is also positional.*

Proof. Let W_0, W_1, \dots be a family of countably many prefix-independent Σ_2^0 objectives admitting strongly neutral letters. Using Theorem 2 we get countable history-deterministic well-founded monotone co-Büchi automata A_0, A_1, \dots for the respective objectives; without loss of generality we assume that they are saturated (Lemma 9).

Then consider the automaton A obtained from the disjoint union of the A_i 's by adding all possible co-Büchi transitions, and all normal transitions from A_i to A_j with $i > j$. The initial state in A can be chosen arbitrarily. Note that A is well-founded, monotone, and countable, so we should prove that it recognises $W = \bigcup_i W_i$ and is history-deterministic.

Note that any infinite path in A which visits finitely many co-Büchi transitions eventually remains in some A_i , and thus by prefix-independence, $L(A) \subseteq W$.

It remains to prove history-determinism of A . Let R_0, R_1, \dots be resolvers for A_0, A_1, \dots witnessing that these automata are history deterministic. Consider a resolver which stores a sequence of states (r_0, r_1, \dots) , with r_i being a state of R_i . Initially these are all initial states of the respective resolvers and the transitions follow the transitions of all the resolvers synchronously. Additionally, we store a round-robin counter, which indicates one of the resolvers, following the sequence $R_0; R_0, R_1; R_0, R_1, R_2; R_0, R_1, R_2, R_3; \dots$. If we see a normal transition in the currently indicated resolver, then we also see a normal transition in R , and otherwise, we update the counter to the next resolver and see a co-Büchi transition in R .

We now prove that the above resolver is sound. For that, consider a word w which belongs to $L(A_n)$ for some n . Assume for the sake of contradiction that the path in A constructed by the above resolver reading w contains infinitely many co-Büchi transitions. It means that infinitely many times the resolver R_n reached a co-Büchi state in A_n . But this contradicts the assumption that R_n is sound. We conclude that W is positional by applying Lemma 12. ◀

4 From finite to arbitrary arenas

In this section we study the difference between positionality over finite and arbitrary arenas.

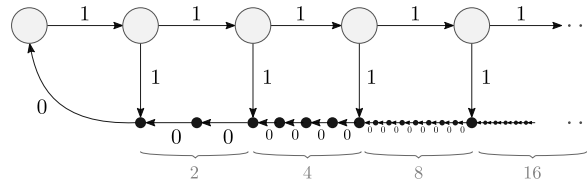
4.1 Mean-payoff games

There are, in fact, four non-isomorphic variants of the mean-payoff objective. Three of them fail to be positional over arbitrary arenas (even over bounded degree arenas), as expressed by the following facts.

► **Proposition 14.** *The mean-payoff objective $\text{Mean-Payoff}_{\leq 0}$ over $w_0 w_1 \dots \in \mathbb{Z}^\omega$ with the condition $\limsup_k \frac{1}{k} \sum_{i=0}^{k-1} w_i \leq 0$ is not positional over arbitrary arenas.*

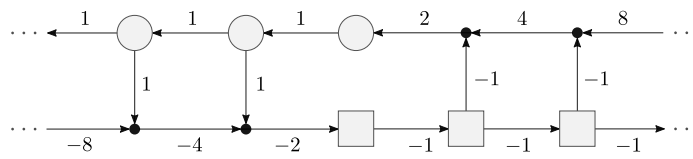
Proof. Consider the arena depicted on Figure 4. Eve can win by following bigger and bigger loops which reach arbitrarily far to the right. This strategy brings the average of the weights closer and closer to 0.

Nevertheless, each positional strategy of Eve either moves infinitely far to the right (resulting in $\lim_k \frac{1}{k} \sum_{i=0}^{k-1} w_i = 1$) or repeats some finite loop which results in a fixed positive limit $\lim_k \frac{1}{k} \sum_{i=0}^{k-1} w_i > 0$. In both cases it violates $\text{Mean-Payoff}_{\leq 0}$. ◀



■ **Figure 4** The arena used in the proof of Proposition 14.

► **Proposition 15.** Consider two lim inf variants of the mean-payoff objective over $w_0w_1 \cdots \in \mathbb{Z}^\omega$: one where we require that $\liminf_k \frac{1}{k} \sum_{i=0}^{k-1} w_i \leq 0$, and the other where that same quantity is < 0 . Both these objectives are not positional over arbitrary arenas.



■ **Figure 5** The arena used in the proof of Proposition 15.

Proof. Consider the arena depicted on Figure 5. Again, Eve has a winning strategy for both these objectives by always going sufficiently far to the left, to ensure that the average drops below for instance $-\frac{1}{2}$.

Nevertheless, each positional strategy of Eve either moves infinitely far to the left (resulting again in $\lim_k \frac{1}{k} \sum_{i=0}^{k-1} w_i = 1$), or repeats some finite loop, reaching a minimal negative weight -2^n for some $n > 0$. Now, Adam can win against this strategy by repeating a loop going to the right, in such a way to reach a weight 2^{n+1} . The label of such a path satisfies $\lim_k \frac{1}{k} \sum_{i=0}^{k-1} w_i = \frac{2^{n+1}-1}{4n+4} > 0$, violating both objectives. ◀

The remaining fourth type of a mean-payoff objective is „lim sup < 0 ”:

$$\text{Mean-Payoff}_{<0} = \left\{ w_0w_1 \cdots \in \mathbb{Z}^\omega \mid \limsup_k \frac{1}{k} \sum_{i=0}^{k-1} w_i < 0 \right\}.$$

► **Proposition 16.** The objective $\text{Mean-Payoff}_{<0}$ is positional over arbitrary arenas.

Proof. Consider the tilted boundedness objective with parameter $n \geq 1$, defined as

$$\text{Tilted-Bounded}_n = \left\{ w_0w_1 \cdots \in \mathbb{Z}^\omega \mid \sup_k \sum_{i=0}^{k-1} (w_i + 1/n) \text{ is finite} \right\}$$

Note that renaming weights by $w \mapsto nw$ maps Tilted-Bounded_n to $\text{Bounded} \cap (n\mathbb{Z})^\omega$, therefore it follows easily that Tilted-Bounded_n is positional over arbitrary arenas. Note also that for every n the objective Tilted-Bounded_n belongs to Σ_2^0 , as a union ranging over $N \in \mathbb{N}$ of closed (in other words safety) objectives $\{w_0w_1 \cdots \in \mathbb{Z}^\omega \mid \forall k \in \mathbb{N} \sum_{i=0}^{k-1} (w_i + 1/n) \leq N\}$.

▷ **Claim 17.** It holds that $\text{Mean-Payoff}_{<0} = \bigcup_{n \geq 1} \text{Tilted-Bounded}_n$.

54:14 Positionality in Σ_2^0 and a Completeness Result

Proof of Claim 17. Write $\text{mp}(w) = \limsup_k 1/k \sum_{i=0}^{k-1} w_i$. If

$$w = w_0 w_1 \cdots \in \text{Tilted-Bounded}_n$$

then there is a bound N such that for all k , $\sum_{i=0}^{k-1} (w_i + 1/n) \leq N$, therefore $1/k \sum_{i=0}^{k-1} w_i \leq N/k - 1/n$ and thus $\text{mp}(w) \leq -1/n < 0$, so $w \in \text{Mean-Payoff}_{<0}$. Conversely, if $w \in \text{Mean-Payoff}_{<0}$ and n is large enough so that $1/n \leq \text{mp}(w)$, then $w \in \text{Tilted-Bounded}_n$. \triangleleft

Now, positionality of $\text{Mean-Payoff}_{<0}$ follows from the claim together with Corollary 3, as all Tilted-Bounded_n are prefix-independent, admit a strongly neutral letter, are positional, and belong to Σ_2^0 .⁴ \blacktriangleleft

4.2 A completeness result

Equivalence over finite arenas. Recall that two prefix-independent objectives $W, W' \subseteq C^\omega$ are said to be *finitely equivalent*, written $W \equiv W'$, if for all finite C -arenas A ,

$$\text{Eve wins } (A, W) \iff \text{Eve wins } (A, W').$$

Since one may view strategies as games controlled by Adam, we obtain the following motivating result.

► **Lemma 18.** *If $W \equiv W'$ and W is positional over finite arenas then so is W' .*

Proof. Let A be a finite C -arena such that Eve wins (A, W') . Then Eve wins (A, W) , so she wins with a positional strategy S . Looking at S as a finite C -arena controlled by Adam yields that Eve wins (S, W') , thus S satisfies W' . \blacktriangleleft

We now move on to the proof of our completeness result.

► **Theorem 4.** *Let $W \subseteq C^\omega$ be a prefix-independent objective which is positional over finite arenas and admits a weakly neutral letter. Then there exists an objective $W' \equiv W$ which is positional over arbitrary arenas.*

We start with the following observation, which is a standard topological argument based on König's lemma. Note that the assumption of finiteness of G is essential here.

► **Lemma 19.** *Let G be a finite C -graph and $v \in G$. Then $L(G, v)$ is a closed subset of C^ω .*

We may now give the crucial definition. Given a prefix-independent objective $W \subseteq C^\omega$, we define its finitary substitute to be

$$W_{\text{fin}} = \{w \in C^\omega \mid w \text{ labels a path in some finite graph } G \text{ which satisfies } W\}.$$

Note that $W_{\text{fin}} \subseteq W$. Now observe that

$$W = \bigcup_{\substack{G \text{ finite graph} \\ G \text{ satisfies } W}} L(G) = \bigcup_{\substack{G \text{ finite graph} \\ G \text{ satisfies } W \\ v \in V(G)}} L(G, v),$$

and since there are (up to isomorphism) only countably many finite graphs, it follows from Lemma 19 that $W_{\text{fin}} \in \Sigma_2^0$.

⁴ We thank Lorenzo Clemente for suggesting to use closure under union. A direct proof (constructing a universal graph) is available in the unpublished preprint [26].

► **Lemma 20.** *Let $W \subseteq C^\omega$ be a prefix-independent objective which is positional over finite arenas. Then $W_{\text{fin}} \equiv W$.*

Proof. Let A be a finite C -arena. Since $W_{\text{fin}} \subseteq W$, it is clear that if Eve wins (A, W_{fin}) then she wins (A, W) . Conversely, assume Eve wins (A, W) . Then she has a positional strategy S in A which is winning for W . Since S is a finite graph, it is also winning for W_{fin} and therefore Eve wins (A, W_{fin}) . ◀

We should make the following sanity check.

► **Lemma 21.** *If W is prefix-independent, then W_{fin} is as well.*

Proof. Take a letter $c \in C$, we aim to show that $cW_{\text{fin}} = W_{\text{fin}}$. Let $w \in cW_{\text{fin}}$, and let G be a finite graph satisfying W such that cw labels a path from $v \in V[G]$ in G . Then w labels a path from a c -successor of v in G , thus $w \in W_{\text{fin}}$.

Conversely, let $w \in W_{\text{fin}}$, and let G be a finite graph satisfying W such that w labels a path from $v \in V[G]$ in G . Let G' be the graph obtained from G by adding a fresh vertex v' with a unique outgoing c -edge towards v . Since W is prefix-independent, G' satisfies W . Since cw labels a path from v' in G' , it follows that $cw \in W_{\text{fin}}$. ◀

We are now ready to prove Theorem 4.

Proof of Theorem 4. Let W be a prefix-independent objective which is positional over finite arenas and admits a weakly neutral letter ε . We show that W_{fin} is positional over arbitrary arenas. Since Lemma 20 implies that $W_{\text{fin}} \equiv W$, this concludes the proof of Theorem 4.

Thanks to Lemma 6, any finite graph H satisfying W can be embedded into a monotone finite graph G which also satisfies W ; note that $L(H) \subseteq L(G)$. Therefore

$$W_{\text{fin}} = \bigcup_{\substack{H \text{ finite graph} \\ H \text{ satisfies } W}} L(H) = \bigcup_{\substack{G \text{ finite monotone graph} \\ G \text{ satisfies } W}} L(G).$$

Let G_0, G_1, \dots be an enumeration (up to isomorphism) of all finite monotone graphs satisfying W . Then consider the automaton A obtained from the disjoint union of the G_i 's by adding all normal transitions from G_i to G_j for $i > j$, and saturating with co-Büchi transitions. The initial state q_0 is chosen to be $\max V(G_0)$, the maximal state in G_0 . Note that A is countable, monotone, and well founded, so there remains to prove that $L(A) = W_{\text{fin}}$ and that A is history-deterministic.

Clearly for any monotone graph G satisfying W , it holds that $L(G) \subseteq L(A)$, and thus $W_{\text{fin}} \subseteq L(A)$. Conversely, let $w \in L(A)$, and consider an accepting path π for W . Then eventually, π visits only normal edges, and therefore eventually, π remains in some G_i . Thus $w = ww'$ with $w' \in L(G_i) \subseteq W_{\text{fin}}$, we conclude by prefix-independence of W_{fin} (Lemma 21).

To prove that A is history-deterministic we now build a resolver: intuitively, we deterministically try to read in G_0 , then if we fail, go to G_1 , then G_2 and so on. The fact that reading in each G_i can be done deterministically follows from monotonicity: for each $v \in V(G_i)$ and each $c \in C$, the set $\{v' \in V(G_i) \mid v \xrightarrow{c} v' \in E(G_i)\}$ of c -successors of v is downward closed. We let $\delta_i(v, c)$ denote the maximal c -successor of v in G_i if it exists, and $\delta_i(v, c) = \perp$ if v does not have a c -successor. It is easy to see that in a monotone graph G , $v \leq v'$ implies $L(G, v) \subseteq L(G, v')$; in words, more continuations are available from bigger states.

Now we define the resolver A by $V(R) = V(A)$, $r_0 = q_0 = \max V(G_0)$, and for any $q, q' \in V(A)$ and $c \in C$,

$$\begin{aligned} q \xrightarrow{c} q' \in E(A) &\iff \exists i, q, q' \in V(G_i) \text{ and } q' = \delta_i(q) \neq \perp \\ q \xrightarrow{c} q' \in E(A) &\iff \exists i, q \in V(G_i) \text{ and } \delta_i(q, c) = \perp \text{ and } q' = \max V(G_{i+1}). \end{aligned}$$

Clearly, R is deterministic and $R \rightarrow A$ so R is a resolver; it remains to prove soundness. Take $w \in L(A)$ and let i such that $w \in L(G_i)$. Let π be the unique path from $r_0 = \max V(G_0)$ in R labelled by w . We claim that π remains in $\bigcup_{j \leq i} V(G_j)$ and thus it can only visit at most i co-Büchi transitions, so it is accepting. Assume for contradiction that π reaches $V(G_{i+1})$.

Then it is of the form $\pi = \pi_0 \pi_1 \dots \pi_i \pi'$ where each π_j is a path from $\max(V(G_j))$ in G_j and π' starts from $\max(G_{i+1})$. Let w_0, w_1, \dots, w_i and w' be the words labelling the paths, so that $w = w_0 w_1 \dots w_i w'$. Denote $q = \max(V(G_i))$. Then w_i is not a label of a finite path from q in G_i , therefore $w_i w' \notin L(G_i, q) = L(G_i)$. At the same time $w \in L(G_i)$ thus $q \xrightarrow{w_0 \dots w_{i-1}} q' \xrightarrow{w_i w'}$ for some $q' \in V(G_i)$. But then $w_i w' \in L(G_i, q') \subseteq L(G_i, q)$, a contradiction. ◀

5 Conclusion

We gave a characterisation of prefix-independent Σ_2^0 objectives which are positional over arbitrary arenas as being those recognised by countable history-deterministic well-founded monotone co-Büchi automata. We moreover deduced that this class is closed by unions. We proved that, with a proper definition, mean-payoff games are positional over arbitrary arenas. Finally, we showed that any prefix-independent objective which is positional over finite arenas is finitely equivalent to an objective which is positional over arbitrary arenas.

Open questions. There are many open questions on positionality. Regarding Σ_2^0 objectives, the remaining step would be to lift the prefix-independence assumptions; this requires some new techniques as the proofs presented here do not immediately adapt to this case. Another open question is whether the 1-to-2 player lift holds in Σ_2^0 : is there a Σ_2^0 objective which is positional on arenas controlled by Eve, but not on two player arenas?

As mentioned in the introduction, Casares [4] obtained a characterisation of positional ω -regular objectives, while we characterised (prefix-independent) Σ_2^0 positional objectives. A common generalisation, which we see as a far reaching open question would be to characterise positionality within Δ_3^0 ; hopefully establishing closure under union for this class.

Another interesting direction would be to understand finite memory for prefix-independent Σ_2^0 objectives; useful tools (such as structuration results) are already available [5]. A related (but independent) path is to develop a better understanding of (non-prefix-independent) closed objectives, which so far has remained elusive.

References

- 1 Udi Boker, Orna Kupferman, and Michał Skrzypczak. How deterministic are good-for-games automata? In *FSTTCS*, volume 93 of *LIPICs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.FSTTCS.2017.18.
- 2 Patricia Bouyer, Antonio Casares, Mickael Randour, and Pierre Vandenhove. Half-positional objectives recognized by deterministic büchi automata. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPICs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CONCUR.2022.20.
- 3 Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal Methods Syst. Des.*, 38(2):97–118, 2011. doi:10.1007/s10703-010-0105-x.
- 4 Antonio Casares. *Structural properties of ω -automata and strategy complexity in infinite duration games*. PhD thesis, Université de Bordeaux, 2023.
- 5 Antonio Casares and Pierre Ohlmann. Characterising memory in infinite games. *CoRR*, abs/2209.12044, 2022. doi:10.48550/arXiv.2209.12044.

- 6 Antonio Casares and Pierre Ohlmann. Half-positional ω -regular languages. *CoRR*, abs/2401.15384, 2024. [arXiv:2401.15384](https://arxiv.org/abs/2401.15384), doi:10.48550/arXiv.2401.15384.
- 7 Thomas Colcombet. Forms of determinism for automata (invited talk). In *STACS*, volume 14 of *LIPICs*, pages 1–23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICs.STACS.2012.1.
- 8 Thomas Colcombet and Damian Niwiński. On the positional determinacy of edge-labeled games. *Theor. Comput. Sci.*, 352(1-3):190–196, 2006. doi:10.1016/j.tcs.2005.10.046.
- 9 Morton Davis. Infinite games of perfect information. In *Advances in game theory*, pages 85–101. Princeton Univ. Press, Princeton, N.J., 1964.
- 10 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 109(8):109–113, 1979. doi:10.1007/BF01768705.
- 11 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.
- 12 Nathanaël Fijalkow, Nathalie Bertrand, Patricia Bouyer-Decitre, Romain Brenguier, Arnaud Carayol, John Fearnley, Hugo Gimbert, Florian Horn, Rasmus Ibsen-Jensen, Nicolas Markey, Benjamin Monmege, Petr Novotný, Mickael Randour, Ocan Sankur, Sylvain Schmitz, Olivier Serre, and Mateusz Skomra. *Games on Graphs*. Online, 2023.
- 13 David Gale and Frank M. Stewart. Infinite games with perfect information. In *Contributions to the theory of games*, volume 2 of *Annals of Mathematics Studies*, no. 28, pages 245–266. Princeton University Press, 1953.
- 14 Hugo Gimbert. Parity and exploration games on infinite graphs. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings*, volume 3210 of *Lecture Notes in Computer Science*, pages 56–70. Springer, 2004. doi:10.1007/978-3-540-30124-0_8.
- 15 Hugo Gimbert and Wiesław Zielonka. Games where you can play optimally without any memory. In *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2005. doi:10.1007/11539452_33.
- 16 Simon Iosti and Denis Kuperberg. Eventually safe languages. In Piotrek Hofman and Michał Skrzypczak, editors, *Developments in Language Theory - 23rd International Conference, DLT 2019, Warsaw, Poland, August 5-9, 2019, Proceedings*, volume 11647 of *Lecture Notes in Computer Science*, pages 192–205. Springer, 2019. doi:10.1007/978-3-030-24886-4_14.
- 17 Alexander Kechris. *Classical descriptive set theory*. Springer-Verlag, New York, 1995.
- 18 Eryk Kopczyński. Half-positional determinacy of infinite games. In *ICALP*, volume 4052 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2006. doi:10.1007/11787006_29.
- 19 Eryk Kopczyński. *Half-positional determinacy of infinite games*. PhD thesis, University of Warsaw, 2009. URL: <https://www.mimuw.edu.pl/~erykk/papers/hpwc.pdf>.
- 20 Alexander Kozachinskiy. Energy games over totally ordered groups. *CoRR*, abs/2205.04508, 2022. doi:10.48550/arXiv.2205.04508.
- 21 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *ICALP*, volume 9135 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2015. doi:10.1007/978-3-662-47666-6_24.
- 22 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.
- 23 Andrzej W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdansk, 1991.
- 24 Pierre Ohlmann. *Monotone graphs for solving parity and mean-payoff games*. PhD thesis, Université de Paris, 2021.
- 25 Pierre Ohlmann. Characterizing Positionality in Games of Infinite Duration over Infinite Graphs. *TheoretCS*, Volume 2, January 2023. doi:10.46298/theoretics.23.3.



54:18 Positionality in Σ_2^0 and a Completeness Result

- 26 Pierre Ohlmann. Positionality of mean-payoff games on infinite graphs, 2023. arXiv:2305.00347.
- 27 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969. URL: <http://www.jstor.org/stable/1995086>.
- 28 Bader Abu Radi and Orna Kupferman. Minimization and canonization of GFG transition-based automata. *Log. Methods Comput. Sci.*, 18(3), 2022. doi:10.46298/lmcs-18(3:16)2022.
- 29 Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi:10.1073/pnas.39.10.1095.
- 30 Michał Skrzypczak. Topological extension of parity automata. *Information and Computation*, 228:16–27, 2013.
- 31 Philip Wolfe. The strict determinateness of certain infinite games. *Pacific Journal of Mathematics*, 5:841–847, 1955.

Tree-Layout Based Graph Classes: Proper Chordal Graphs

Christophe Paul   

CNRS, Université Montpellier, France

Evangelos Protopapas  

CNRS, Université Montpellier, France

Abstract

Many important graph classes are characterized by means of layouts (a vertex ordering) excluding some patterns. For example, a graph $G = (V, E)$ is a proper interval graph if and only if G has a layout \mathbf{L} such that for every triple of vertices such that $x \prec_{\mathbf{L}} y \prec_{\mathbf{L}} z$, if $xz \in E$ and $yz \in E$. Such a triple x, y, z is called an *indifference triple*. In this paper, we investigate the concept of excluding a set of patterns in *tree-layouts* rather than layouts. A tree-layout $\mathbf{T}_G = (T, r, \rho_G)$ of a graph $G = (V, E)$ is a tree T rooted at some node r and equipped with a one-to-one mapping ρ_G between V and the nodes of T such that for every edge $xy \in E$, either x is an ancestor of y , denoted $x \prec_{\mathbf{T}_G} y$, or y is an ancestor of x . Excluding patterns in a tree-layout is now defined using the ancestor relation. This leads to an unexplored territory of graph classes. In this paper, we initiate the study of such graph classes with the class of *proper chordal graphs* defined by excluding indifference triples in tree-layouts. Our results combine characterization, compact and canonical representation as well as polynomial time algorithms for the recognition and the graph isomorphism of proper chordal graphs. For this, one of the key ingredients is the introduction of the concept of FPQ-hierarchy generalizing the celebrated PQ-tree data-structure.

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph algorithms; Mathematics of computing \rightarrow Combinatorial algorithms; Mathematics of computing \rightarrow Graph theory; Theory of computation \rightarrow Data structures design and analysis; Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases Graph classes, Graph representation, Graph isomorphism

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.55

Related Version *Full Version*: <https://arxiv.org/abs/2211.07550>

Funding *Christophe Paul*: Research supported by ANR-DFG project UTMA ANR-20-CE92-0027.

Evangelos Protopapas: Research supported by ANR-DFG project UTMA ANR-20-CE92-0027.

1 Introduction

Context. A graph class \mathcal{C} is hereditary if for every graph $G \in \mathcal{C}$ and every induced subgraph H of G , which we denote $H \subseteq_i G$, we have that $H \in \mathcal{C}$. A *minimal forbidden subgraph* for \mathcal{C} is a graph $F \notin \mathcal{C}$ such that for every induced subgraph $H \subseteq_i F$, $H \in \mathcal{C}$. Clearly, a hereditary graph class \mathcal{C} is characterized by its set of minimal forbidden subgraphs. Let \mathcal{F} be a set of graphs that are pairwise not induced subgraphs of one another. We say that a graph G is an *\mathcal{F} -free graph*, if it does not contain any graph of \mathcal{F} as an induced subgraph. If $\mathcal{F} = \{H\}$, then we simply say that G is *H -free* if $H \not\subseteq_i G$. Important graph classes are characterized by a finite set \mathcal{F} of minimal forbidden subgraphs. A popular example is the class of cographs [32, 45]. A graph G is a *cograph* if either G is the single vertex graph, or it is the disjoint union of two cographs, or its complement is a cograph. It is well known that G is a cograph if and only if it is a P_4 -free graph [32, 11]. As witnessed by *chordal graphs* [26, 3], not every hereditary graph family is characterized by excluding a finite set of



© Christophe Paul and Evangelos Protopapas;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 55; pp. 55:1–55:18



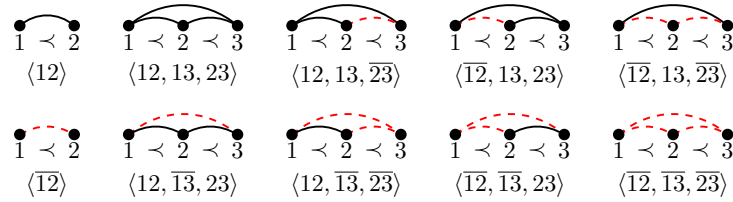
Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



minimal forbidden subgraphs: a graph is chordal if it does not contain a chordless cycle of length at least 4 as an induced subgraph.

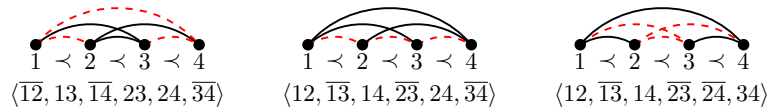
To circumvent this issue, Skrien [44] and Damaschke [13] proposed to embed graphs in some additional structure such as vertex orderings, also called *layouts*. An *ordered graph* is then defined as a pair (G, \prec_G) such that \prec_G is a total ordering of the vertex set V of the graph $G = (V, E)$. We say that an ordered graph (H, \prec_H) is a *pattern* of the ordered graph (G, \prec_G) , which we denote by $(H, \prec_H) \subseteq_p (G, \prec_G)$, if $H \subseteq_i G$ and for every pair of vertices x and y of H , $x \prec_G y$ if and only if $x \prec_H y$. A graph G excludes the pattern (H, \prec_H) , if there exists a layout \prec_G of G such that $(H, \prec_H) \not\subseteq_p (G, \prec_G)$. More generally, a graph class \mathcal{C} excludes a set \mathcal{P} of patterns if for every graph $G \in \mathcal{C}$, there exists a layout \prec_G such that for every pattern $(H, \prec_H) \in \mathcal{P}$, $(H, \prec_H) \not\subseteq_p (G, \prec_G)$. We let $\mathcal{L}(\mathcal{P})$ denote the class of graphs excluding a pattern from \mathcal{P} . Hereafter, a small size pattern (H, \prec_H) will be encoded by listing its set of (ordered) edges and non-edges. There are two patterns on two vertices and eight patterns on three vertices (see Figure 1).



■ **Figure 1** The 10 patterns on at most 3 vertices. $\mathcal{L}(\overline{12}, 13, 23)$ is the class of chordal graphs.

Interestingly, it is known that chordal graphs are characterized by excluding a unique pattern: $\mathcal{P}_{\text{chordal}} = \{\overline{12}, 13, 23\}$, see Figure 1 [13, 15]. This characterization relies on the fact that a graph is chordal if and only if it admits a *simplicial elimination ordering*¹ [14, 43]. Ginn [20] proved that for every pattern (H, \prec_H) such that H is neither the complete graph nor the edge-less graph, characterizing the graph family $\mathcal{L}((H, \prec_H))$ requires an infinite family of forbidden induced subgraphs. Observe however that excluding a unique pattern is important for that result: cographs are characterized as P_4 -free graphs (see discussion above) but needs a set $\mathcal{P}_{\text{cograph}}$ of several excluded patterns (see Figure 2) to be characterized [13]:

$$\mathcal{P}_{\text{cograph}} = \{ \langle \overline{12}, \overline{13}, 23 \rangle, \langle \overline{12}, 13, \overline{23} \rangle, \langle \overline{12}, 13, \overline{14}, 23, 24, \overline{34} \rangle, \langle \overline{12}, \overline{13}, 14, \overline{23}, 24, \overline{34} \rangle, \langle \overline{12}, \overline{13}, 14, \overline{23}, \overline{24}, 34 \rangle \}$$



■ **Figure 2** The three size 4 forbidden patterns of cographs.

In [16], Duffus et al. investigate the computational complexity of the recognition problem of $\mathcal{L}((H, \prec_H))$ for a fixed ordered graph (H, \prec_H) . They conjectured that if H is neither the complete graph nor the edge-less graph, then recognizing $\mathcal{L}((H, \prec_H))$ is NP-complete if H or

¹ A vertex is *simplicial* if its neighbourhood induces a clique. A simplicial elimination ordering can be defined by a layout \prec_G of $G = (V, E)$ such that every vertex x is simplicial in the subgraph $G[\{y \in V \mid y \prec_G x\}]$.

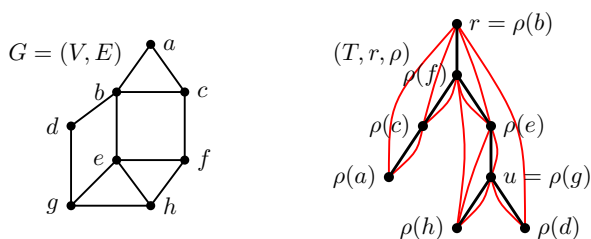
its complement is 2-connected. Hell et al. [28] have recently shown that if \mathcal{P} only contains patterns of size at most 3, then $\mathcal{L}(\mathcal{P})$ can be recognized in polynomial time using a 2-SAT approach. Besides chordal graphs (see discussion above), these graph classes comprise very important graph classes, among others:

- *Interval graphs* [27, 2, 22] exclude $\mathcal{P}_{\text{int}} = \{\langle \overline{12}, 13, \overline{23} \rangle, \langle \overline{12}, 13, 23 \rangle\}$ [40]: A graph is an interval graph if it is the intersection graph of a set of intervals on the real line. The existence of a \mathcal{P}_{int} -layout for interval graphs follows from the fact that a graph is an interval graph if and only if it is chordal and co-comparability.
- *Proper interval graphs* [41, 42] exclude $\mathcal{P}_{\text{proper}} = \mathcal{P}_{\text{int}} \cup \{\langle 12, 13, \overline{23} \rangle\}$. A graph is a proper interval graph if it is the intersection graph of a set of proper intervals on the real line (no interval is a subset of another one). This characterization of proper interval graphs as $\mathcal{L}(\mathcal{P}_{\text{proper}})$ follows from the existence of the so-called *indifference orderings* that are exactly the layouts excluding $\mathcal{P}_{\text{proper}}$ [41, 42].
- *Trivially perfect graphs* [21] exclude $\mathcal{P}_{\text{trivPer}} = \mathcal{P}_{\text{chordal}} \cup \{\langle 12, \overline{13}, 23 \rangle\}$. A graph G is a trivially perfect graph if and only if it is $\{P_4, C_4\}$ -free, or equivalently G is the comparability graph of a rooted tree T (two vertices are adjacent if one is the ancestor of the other). A $\mathcal{P}_{\text{trivPer}}$ -free layout is obtained from a depth first search ordering of T . Moreover every layout of P_4 and C_4 contains one of the patterns of $\mathcal{P}_{\text{trivPer}}$ (see [17]).

For more examples, the reader should refer to [13, 17]. Feuilloley and Habib [17] list all the graph classes that can be obtained by excluding a set of patterns each of size at most 3.

From layouts to tree-layouts. A layout \prec_G of a graph $G = (V, E)$ on n vertices can be viewed as an embedding of G into a path P on n vertices rooted at one of its extremities. Under this view point, it becomes natural to consider graph embeddings in graphs that are more general than rooted paths. Recently, Guzman-Pro et al. [23] have studied embeddings in a cyclic ordering. In this paper, we consider embedding the vertices of a graph in a rooted tree, yielding the notion of *tree-layout*.

► **Definition 1.** Let $G = (V, E)$ be a graph on n vertices. A tree-layout of G is a triple $\mathbf{T}_G = (T, r, \rho_G)$ where T is a tree on a set V_T of n nodes rooted at r and $\rho_G : V \rightarrow V_T$ is a bijection such that for every edge $xy \in E$, either x is an ancestor of y , denoted by $x \prec_{\mathbf{T}} y$, or y is an ancestor of x .



■ **Figure 3** A tree layout (T, r, ρ) of a graph $G = (V, E)$.

Let $\mathbf{T}_G = (T, r, \rho_G)$ be a *tree-layout* of a graph G . We observe that, from Definition 1, T is not a *Trémaux tree* since it is not necessarily a spanning tree of G (see [39] and [6] for similar concepts). It is easy to see that if T is a path, then \mathbf{T}_G defines a layout of G . So from now on, we shall define a *layout* as a triple $\mathbf{L}_H = (P, r, \rho_H)$, where P is a path that fulfils the conditions of Definition 1. An ordered graph then becomes a pair (H, \mathbf{L}_H) where $\mathbf{L}_H = (P, r, \rho_H)$ is a layout of G . Excluding a pattern (H, \mathbf{L}_H) in a tree-layout

$\mathbf{T}_G = (T, r, \rho_G)$ of a graph G is defined similarly as excluding a pattern in a layout, but now using the ancestor relation \prec_T . For a set \mathcal{P} of patterns, we can also define the class $\mathfrak{T}(\mathcal{P})$ of graphs admitting a tree-layout that excludes every pattern $P \in \mathcal{P}$. If $\mathcal{P} = \{(H, \mathbf{L}_H)\}$, we simply write $\mathfrak{T}((H, \mathbf{L}_H))$. Observe that, as a layout is a tree-layout, for a fixed set \mathcal{P} of patterns, we always have $\mathfrak{L}(\mathcal{P}) \subseteq \mathfrak{T}(\mathcal{P})$. As an introductory example, let us consider the pattern $\langle \overline{12} \rangle$. The following observation directly follows from the definitions of a tree-layout and trivially perfect graphs.

► **Observation 2.** *The class $\mathfrak{L}(\langle \overline{12} \rangle)$ is the set of complete graphs while the class $\mathfrak{T}(\langle \overline{12} \rangle)$ is the set of trivially perfect graphs.*

So the class of trivially perfect graphs can be viewed as the tree-like version of the class of complete graphs. This view point motivates the systematic study of the unexplored territory composed by graph classes defined by excluding a set of patterns in a tree-layout.

Our contributions. As a first case study, we consider the patterns characterizing interval graphs and proper interval graphs. We first show that if we consider the interval graphs patterns \mathcal{P}_{int} , the same phenomena as for $\{\langle \overline{12} \rangle\}$ holds, leading to a novel (up to our knowledge) characterization of chordal graphs as being exactly $\mathfrak{T}(\mathcal{P}_{\text{int}})$ (see Theorem 3).

As already discussed, proper interval graphs are obtained by restricting interval graphs to the intersection of a set of *proper* intervals (no interval is a subinterval of another). It is known that $K_{1,3}$ is an interval graph but not a proper one. Following this line, Gavril [19], in his seminal paper characterizing chordal graphs as the intersection graphs of a subset of subtrees of a tree, considered the class of intersection graphs of a set of *proper* subtrees of a tree (no subtree is contained in another). Using an easy reduction, Gavril proved that this again yields a characterization of chordal graphs. So this left open the question of proposing a natural definition for proper chordal graphs, a class of graphs that should be sandwiched between proper interval graphs and chordal graphs but incomparable to interval graphs. In a recent paper, Chaplick [9] investigated this question and considered the class of intersection graphs of non-crossing paths in a tree.

Our main contribution is to propose a natural definition of *proper chordal graphs* by means of forbidden patterns on tree-layouts: a graph is proper chordal if it belongs to $\mathfrak{T}(\mathcal{P}_{\text{proper}})$, the class of graphs admitting a $\mathcal{P}_{\text{proper}}$ -free tree-layout, hereafter called *indifference tree-layout*. Recall that $\mathcal{P}_{\text{proper}}$ are the patterns characterizing proper interval graphs on layouts. It can be observed that proper chordal graphs and intersection graphs of non-crossing paths in a tree are incomparable graph classes. Table 1 resumes the discussion above.

Forbidden patterns	Layouts	Tree-layouts
$\langle \overline{12} \rangle$	Cliques	Trivially perfect graphs
$\langle 12, 13, \overline{23} \rangle, \langle \overline{12}, 13, 23 \rangle, \langle \overline{12}, 13, \overline{23} \rangle$	Proper interval graphs	Proper chordal graphs
$\langle \overline{12}, 13, \overline{23} \rangle, \langle \overline{12}, 13, 23 \rangle$	Interval graphs	Chordal graphs

■ **Table 1** Graph classes obtained by excluding $\langle \overline{12} \rangle$, $\mathcal{P}_{\text{proper}}$ and \mathcal{P}_{int} .

We then provide a thorough study of the combinatorial and algorithmic aspects of proper chordal graphs. Our first result (see Theorem 6) is a characterization of indifference tree-layouts (and henceforth of proper chordal graphs). As discussed in Section 3, (proper) interval graphs may have multiple (proper) interval models. For a given graph, these interval models are all captured in a canonical representation encoded by the celebrated PQ-tree data-structure [7]. We show that the set of indifference tree-layouts can also be represented

in a canonical and compact way by means of a tree data-structure generalizing PQ-trees, that we call FPQ-hierarchies, see Theorem 11. These structural results have very interesting algorithmic implications. First, we can design a polynomial time recognition algorithm for proper chordal graphs, see Theorem 14. Second, we show that the isomorphism problem restricted to proper chordal graphs is polynomial time solvable, see Theorem 16. Interestingly, this problem is GI-complete on (strongly) chordal graphs [38, 46]. So considering proper chordal graphs allows us to push the tractability further towards its limit. We believe that beyond proper chordal graphs, the concept of FPQ-hierarchy is interesting on its own as it is strongly related to the concept of (weakly) partitive families [12, 10] and thereby the theory of modular decomposition [37, 25].

2 Preliminaries

2.1 Notations and definitions

Graphs. In this paper, every graph is finite, loopless, and without multiple edges. A graph is a pair $G = (V, E)$ where V is its vertex set and $E \subseteq V^2$ is the set of edges. For two vertices $x, y \in V$, we let xy denote the edge $e = \{x, y\}$. We say that the vertices x and y are incident with the edge xy . The neighbourhood of a vertex x is the set of vertices $N(x) = \{y \in V \mid xy \in E\}$. The closed neighbourhood of a vertex x is $N[x] = N(x) \cup \{x\}$. Let S be a subset of vertices of V . The graph resulting from the removal of S is denoted by $G - S$. If $S = \{x\}$ is a singleton we write $G - x$ instead of $G - \{x\}$. The subgraph of G induced by S is $G[S] = G - (V \setminus S)$. We say that S is a *separator* of G if $G - S$ contains more connected components than G . We say that S separates $X \subseteq V(G)$ from $Y \subseteq V(G)$ if X and Y are subsets of distinct connected components of $G - S$. If x is a vertex such that $x \notin S$ and $S \subseteq N(x)$, then we say that x is *S -universal*.

Rooted trees. A rooted tree is a pair (T, r) where r is a distinguished node² of the tree T . We say that a node u is an *ancestor* of the node v (and that u is a *descendant* of v) if u belongs to the unique path of T from r to v . If u is an ancestor of v , then we write $u \prec_{(T,r)} v$. For a node u of T , the set $A_{(T,r)}(u)$ contains every *ancestor* of u , that is every node v of T such that $v \prec_{(T,r)} u$. Likewise, the set $D_{(T,r)}(u)$ contains every *descendant* of u , that is every node v of T such that $u \prec_{(T,r)} v$. We may also write $A_{(T,r)}[u] = A_{(T,r)}(u) \cup \{u\}$ and $D_{(T,r)}[u] = D_{(T,r)}(u) \cup \{u\}$. The least common ancestor of two nodes u and v is denoted $\text{lca}_{(T,r)}(u, v)$. For a node u , we define T_u as the subtree of (T, r) rooted at u and containing the descendants of u . We let $\mathcal{L}(T, r)$ denote the set of leaves of (T, r) and for a node u , $\mathcal{L}_{(T,r)}(u)$ is the set of leaves of (T, r) that are descendants of the node u .

Ordered trees. An ordered tree is a rooted tree (T, r) such that the children of every internal node are totally ordered. For an internal node v , we let denote $\sigma_{(T,r)}^v$ the permutation of its children. An ordered tree is non-trivial if it contains at least one internal node. An ordered tree (T, r) defines a permutation $\sigma_{(T,r)}$ of its leaf set $\mathcal{L}(T, r)$ as follows. For every pair of leaves $x, y \in \mathcal{L}(T, r)$, let u_x and u_y be the children of $v = \text{lca}_{(T,r)}(x, y)$ respectively being an ancestor of x and of y . Then $x \prec_{\sigma_{(T,r)}} y$ if and only if $u_x \prec_{\sigma_{T,r}^v} u_y$.

² To avoid confusion, we reserve the term *vertex* for graphs and *node* for trees.

Tree-layouts of graphs. Let $\mathbf{T} = (T, r, \rho)$ be a tree-layout of a graph $G = (V, E)$ (see Definition 1). Let x and y be two vertices of G . We note $x \prec_{\mathbf{T}} y$ if $\rho(x)$ is an ancestor of $\rho(y)$ in (T, r) and use $A_{\mathbf{T}}(x)$, $D_{\mathbf{T}}(x)$ to respectively denote the ancestors and descendants of x . The notations $\mathcal{L}(\mathbf{T})$, \mathbf{T}_x , $\mathcal{L}_{\mathbf{T}}(x)$ are derived from the notations defined above.

2.2 A novel characterization of chordal graphs

Gavril [19] characterized chordal graphs as the intersection graphs of a family of subtrees of a tree. Given a chordal graph $G = (V, E)$, a *tree-intersection model* of $G = (V, E)$ is defined as a triple $\mathbf{M}_G^{\mathbf{T}} = (T, \mathcal{T}, \tau_G)$ where T is a tree, \mathcal{T} is a family of subtrees of T and $\tau_G : V \rightarrow \mathcal{T}$ is a bijection such that $xy \in E$ if and only if $\tau_G(x)$ intersects $\tau_G(y)$. Hereafter, we denote by $T^x \in \mathcal{T}$ the subtree of T such that $T^x = \tau_G(x)$. Likewise, an intersection model of an interval graph G is $\mathbf{M}_G^{\mathbf{I}} = (P, \mathcal{P}, \tau_G)$ where P is a path, \mathcal{P} is a family of subpaths of P . The interval, or subpath, $\tau_G(x) \in \mathcal{P}$ will be denoted P^x . So, chordal graphs clearly appear as the tree-like version of interval graphs. We prove that this similarity can also be observed when characterizing these graph classes by means of forbidden patterns. Recall that the class of interval graphs is $\mathfrak{L}(\mathcal{P}_{\text{int}})$ where $\mathcal{P}_{\text{int}} = \{\langle \overline{12}, 13, 23 \rangle, \langle \overline{12}, 13, \overline{23} \rangle\}$.

► **Theorem 3.** *The class of chordal graphs is $\mathfrak{T}(\mathcal{P}_{\text{int}})$.*

2.3 Proper interval graphs and proper chordal graphs

Proper interval graphs. A graph $G = (V, E)$ is a *proper interval graph* if it is an interval graph admitting an interval model $\mathbf{M}_G^{\mathbf{I}}$ such that for every pair of intervals none is a subinterval of another [41, 42]. In terms of a pattern characterization, we have seen that the class of proper interval graphs is $\mathfrak{L}(\mathcal{P}_{\text{proper}})$ where $\mathcal{P}_{\text{proper}} = \{\langle \overline{12}, 13, 23 \rangle, \langle \overline{12}, 13, \overline{23} \rangle, \langle 12, 13, \overline{23} \rangle\}$ [42, 13]. Hereafter a layout that is $\mathcal{P}_{\text{proper}}$ -free is called an *indifference layout*. Indifference layouts have several characterizations, see Theorem 4 below.

► **Theorem 4.** [35, 41] *Let \mathbf{L}_G be a layout of a graph G . The following properties are equivalent.*

1. \mathbf{L}_G is an indifference layout;
2. for every vertex v , $N[v]$ is consecutive in \mathbf{L}_G ;
3. every maximal clique is consecutive in \mathbf{L}_G ;
4. for every pair of vertices x and y with $x \prec_{\mathbf{L}_G} y$, $N(y) \cap A_{\mathbf{L}_G}(x) \subseteq N(x) \cap A_{\mathbf{L}_G}(x)$ and $N(x) \cap D_{\mathbf{L}_G}(y) \subseteq N(y) \cap D_{\mathbf{L}_G}(y)$.

Proper chordal graphs. To understand what are the *tree-like* proper interval graphs, we propose the following definition.

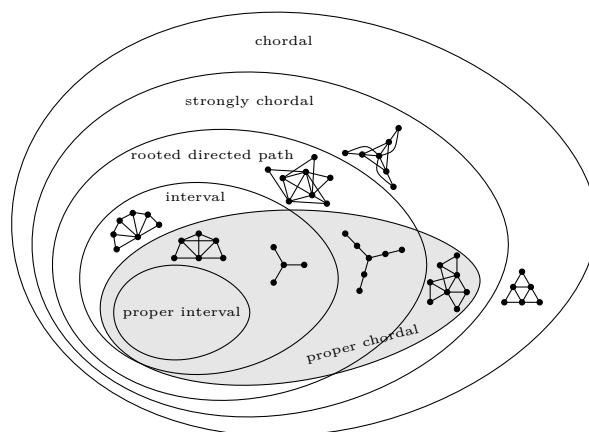
► **Definition 5.** *A graph $G = (V, E)$ is a proper chordal graph if $G \in \mathfrak{T}(\mathcal{P}_{\text{proper}})$.*

Hereafter, a tree-layout \mathbf{T}_G of a graph G that is $\mathcal{P}_{\text{proper}}$ -free will be called an *indifference tree-layout*. We first prove that Theorem 4 generalizes to indifference tree-layouts.

► **Theorem 6.** *Let $\mathbf{T}_G = (T, r, \rho_G)$ be a tree-layout of a graph G . The following properties are equivalent.*

1. \mathbf{T}_G is an indifference tree-layout;
2. for every vertex x , the vertices of $N[x]$ induces a connected subtree of T ;
3. for every maximal clique K , the vertices of K appear consecutively on a path from r in T ;
4. for every pair of vertices x and y such that $x \prec_{\mathbf{T}_G} y$, $N(y) \cap A_{\mathbf{T}_G}(x) \subseteq N(x) \cap A_{\mathbf{T}_G}(x)$ and $N(x) \cap D_{\mathbf{T}_G}(y) \subseteq N(y) \cap D_{\mathbf{T}_G}(y)$.

Clearly, as an indifference layout is an indifference tree-layout, every proper interval graph is a proper chordal graph. Also, as $\mathcal{P}_{\text{chordal}} \subset \mathcal{P}_{\text{proper}}$, proper chordal graphs are chordal graphs. However this inclusion is strict as k -suns, for $k \geq 3$, are not proper chordal. More generally, Figure 4 positions proper chordal graphs with respect to important subclasses of chordal graphs, see [8] for definitions of these classes.



■ **Figure 4** Relationship between proper chordal graphs and subclasses of chordal graphs.

3 FPQ-trees and FPQ-hierarchies

Let \mathcal{P} be a set of patterns. In general, a graph $G \in \mathcal{L}(\mathcal{P})$ admits several \mathcal{P} -free layouts. A basic example is the complete graph K_ℓ on ℓ vertices, which is a proper interval graph. It is easy to observe that every layout of K_ℓ is an indifference layout (i.e. a $\mathcal{P}_{\text{proper}}$ -free layout), but also a \mathcal{P}_{int} -free layout and a $\mathcal{P}_{\text{chordal}}$ -free layout. Let us discuss in more details the case of proper interval graphs and interval graphs.

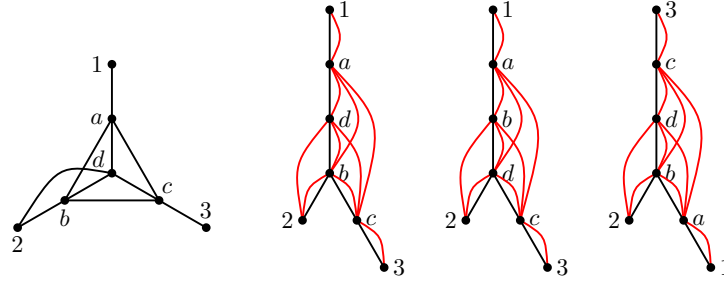
Two vertices x and y of a graph G are *true-twins* if $N[x] = N[y]$. It is easy to see that the true-twin relation is an equivalence relation. If G contains some true-twins, then, by Theorem 4, the vertices of any equivalence class occurs consecutively (and in arbitrary order) in an indifference layout. It follows that for proper interval graphs, the set of indifference layouts depends on the true-twin equivalence classes. Indeed, a proper interval graph G without any pair of true-twins has a unique (up to reversal) indifference layout.

In the case of interval graphs, the set of intersection models, and hence of \mathcal{P}_{int} -free layouts, is structured by means of *modules* [18], in a similar way to the true-twin equivalence classes for $\mathcal{P}_{\text{proper}}$ -free layouts. A subset M of vertices of a graph G is a *module* if for every $x \notin M$, either $M \subseteq N(x)$ or $M \subseteq \overline{N}(x)$. Observe that a true-twin equivalence class is a module. A graph may have exponentially many modules. For example, every subset of vertices of the complete graph is a module. Hsu [29] proved that interval graphs having a unique intersection model are those without any trivial module.

The set of modules of a graph forms a so-called *partitive family* [10] and can thereby be represented through a linear size tree, called the *modular decomposition tree* (see [25] for a survey on modular decomposition). To recognize interval graphs in linear time, Booth and Lueker [7] introduced the concept of PQ-trees which is closely related to the modular decomposition tree or more generally to the theory of (weakly-)partitive families [10, 12]. Basically, a PQ-tree on a set X is a labelled ordered tree having X as its leaf set. Since every ordered tree defines a permutation of its leaf set, by defining an equivalence relation based

on the labels of the node, every PQ-tree can be associated to a set of permutations of X . In the context of interval graphs, X is the set of maximal cliques and a PQ-tree represents the set of so-called *consecutive orderings of the maximal cliques* characterizing interval graphs.

As shown by Figure 5, a proper chordal graph can also have several indifference tree-layouts. In order to represent the set of $\mathcal{P}_{\text{proper}}$ -tree-layouts of a given graph, we will define a structure called *FPQ-hierarchies*, based on FPQ-trees [34], a variant of PQ-trees.



■ **Figure 5** A graph G with three possible indifference tree-layouts, two of them rooted at vertex 1, the third one at vertex 3.

3.1 FPQ-trees

An *FPQ-tree* on the ground set X is a labelled, ordered tree T such that its leaf set $\mathcal{L}(T)$ is mapped to X . The internal nodes of T are of three types, F-nodes, P-nodes, and Q-nodes. If $|X| = 1$, then T is the tree defined by a leaf and a Q-node as the root. Otherwise, F-nodes and Q-nodes have at least two children while P-nodes have at least three children.

Let T and T' be two FPQ-trees. We say that T and T' are isomorphic if they are isomorphic as labelled trees. We say that T and T' are *equivalent*, denoted $T \equiv_{\text{FPQ}} T'$, if one can be turned into a labelled tree isomorphic to the other by a series of the following two operations: *permute*(u) which permutes in any possible way the children of a P-node u ; and *reverse*(u) which reverses the ordering of the children of a Q-node u . It follows that the equivalence class of an FPQ-tree T on X defines a set $\mathfrak{S}_{\text{FPQ}}(T)$ of permutations of X .

Let \mathfrak{S} be a subset of permutations of X . A subset $I \subseteq X$ is a *factor* of \mathfrak{S} if in every permutation of \mathfrak{S} , the elements of I occur consecutively. It is well known that the set of factors of a set of permutations form a so called *weakly-partitive family* [10, 12]. As a consequence, we have the following property, which was also proved in [36].

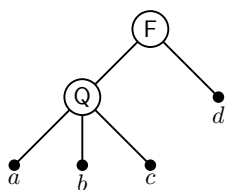
► **Lemma 7.** [10, 12, 36] *Let \mathfrak{S}_T be the subset of permutations of a non-empty set X associated to a PQ-tree T . Then a subset $I \subseteq X$ is a factor of \mathfrak{S}_T if and only if there exists an internal node u of T such that*

- *either $I = \mathcal{L}_T(u)$;*
- *or u is a Q-node and there exists a set of children v_1, \dots, v_s of u that are consecutive in $<_{T,u}$ and such that $I = \bigcup_{1 \leq i \leq s} \mathcal{L}_T(v_i)$.*

We observe that $u = \text{lca}_T(I)$.

Given a set $\mathcal{S} \subseteq 2^X$ of subsets of the ground set X , we let $\text{Convex}(\mathcal{S})$ denote the set of permutations of X such that for every $S \in \mathcal{S}$, S is a factor of $\text{Convex}(\mathcal{S})$. For a PQ-tree T , the set of permutation $\mathfrak{S}_{\text{PQ}}(T)$ is defined similarly as for FPQ-trees.

► **Lemma 8.** [24] *Let X be a non-empty set and let $\mathcal{S} \subseteq 2^X$. In linear time in $|\mathcal{S}|$, we can compute a PQ-tree T on X such that $\mathfrak{S}_{\text{PQ}}(T) = \text{Convex}(\mathcal{S})$ or decide that $\text{Convex}(\mathcal{S}) = \emptyset$.*



■ **Figure 6** An FPQ-tree T with $\mathfrak{S}_{\text{FPQ}}(T) = \{abcd, cbad\}$. The set of non-trivial common factors of \mathfrak{S}_T is $\mathcal{I} = \{\{a, b, c\}, \{a, b\}, \{b, c\}\}$. The permutations $dabc$ and $dcb a$ also belong to $\text{Convex}(\mathcal{I})$.

A set $\mathcal{N} \subseteq 2^X$ of subsets of X is *nested* if for every $Y, Z \in \mathcal{N}$, either $Y \subseteq Z$ or $Z \subseteq Y$. Let $\mathcal{C} = \langle \mathcal{N}_1, \dots, \mathcal{N}_k \rangle$ be a collection of nested sets $\mathcal{N}_i \subseteq 2^X$ ($1 \leq i \leq k$). Observe that a subset $Y \subseteq X$ may occur in several nested sets of \mathcal{C} . We set $\mathcal{S} = \bigcup_{1 \leq i \leq k} \mathcal{N}_i$. We say that a permutation $\sigma \in \text{Convex}(\mathcal{S})$ is \mathcal{C} -*nested* if for every $1 \leq i \leq k$ and every pair of sets $Y, Z \in \mathcal{N}_i$ such that $Z \subset Y$, then $Y \setminus Z \prec_\sigma Z$. We let $\text{Nested-Convex}(\mathcal{C}, \mathcal{S})$ denote the subset of permutations of $\text{Convex}(\mathcal{S})$ that are \mathcal{C} -nested.

► **Lemma 9.** *Let $\mathcal{C} = \langle \mathcal{N}_1, \dots, \mathcal{N}_k \rangle$ be a collection of nested sets such that for every $1 \leq i \leq k$, $\mathcal{N}_i \subset 2^X$. If $\text{Nested-Convex}(\mathcal{C}, \mathcal{S}) \neq \emptyset$, with $\mathcal{S} = \bigcup_{1 \leq i \leq k} \mathcal{N}_i$, then there exists an FPQ-tree T on X such that $\mathfrak{S}_T = \text{Nested-Convex}(\mathcal{C}, \mathcal{S})$. Moreover, such an FPQ-tree, when it exists, can be computed in polynomial time.*

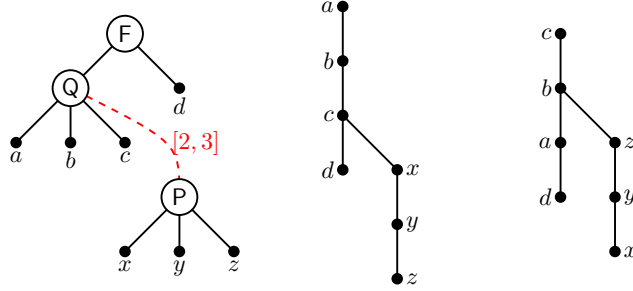
3.2 FPQ-hierarchies

A *hierarchy of ordered trees* H is defined on a set $\mathcal{T} = \{T_0, T_1, \dots, T_p\}$ of non-trivial (i.e. with at least two vertices) ordered trees arranged in an edge-labelled tree, called the *skeleton tree* S_H . More formally, for $0 < i \leq p$, the root r_i of T_i is attached, through a *skeleton edge* e_i , to an internal node f_i of some tree T_j with $j < i$. Suppose that $e_i = r_i f_i$ is the skeleton edge linking the root r_i of T_i to a node f_i of T_j having c children. Then the label of e_i is a pair of integers $I(e_i) = (a_i, b_i) \in [c] \times [c]$ with $a_i \leq b_i$. The contraction of the trees of \mathcal{T} in a single node each, results in the skeleton tree S_H .

From a hierarchy of ordered trees H , we define a rooted tree T_H whose node set is $\bigcup_{0 \leq i \leq p} \mathcal{L}(T_i)$ and that is built as follows. The root of T_H is ℓ_0 the first leaf of $\mathcal{L}(T_0)$ in σ_{T_0} . For every $0 \leq i \leq p$, the permutation σ_{T_i} of $\mathcal{L}(T_i)$, defined by T_i , is a path of T_H . Finally, for $1 \leq i \leq p$, let ℓ_i be the first leaf of $\mathcal{L}(T_i)$ in σ_{T_i} . Suppose T_i is connected in H to T_j through the skeleton edge $e_i = r_i f_i$ with label $I(e_i) = (a_i, b_i)$. Let u_j the b_i -th child of f_i in T_j and ℓ be the leaf of T_j that is a descendant of u_j and largest in σ_{T_j} . Then set ℓ as the parent of ℓ_i in T_H . See Figure 7 for an example.

An *FPQ-hierarchy* is a hierarchy of FPQ-trees with an additional constraint on the labels of the skeleton edges. Let $e_i = r_i f_i$ be the skeleton from the root r_i of T_i to the node f_i of T_j with $j \leq i$. If f_j is a P-node with c children, then $I(e_i) = (1, c)$. As in the case of FPQ-trees, we say two FPQ-hierarchies H and H' are isomorphic if they are isomorphic as labeled ordered trees. That is the types of the nodes, the skeleton edges and their labels are preserved. We say that H and H' are equivalent, denoted $H \approx_{\text{FPQ}} H'$, if one can be turned into an FPQ-hierarchy isomorphic to the other by a series of $\text{permute}(u)$ and $\text{reverse}(u)$ operations (with u being respectively a P-node and a Q-node) to modify relative ordering of the tree-children of u . Suppose that u is a Q-node with c children incident to a skeleton edge e . Then applying $\text{reverse}(u)$ transforms $I(e) = (a, b)$ into the new label $I^c(e) = (c + 1 - b, c + 1 - a)$. It follows that the equivalence class of an FPQ-hierarchy H on the set $\mathcal{T} = \{T_0, T_1, \dots, T_p\}$ of FPQ-trees defines a set $\mathfrak{T}_{\text{FPQ}}(H)$ of rooted trees on $\bigcup_{0 \leq i \leq p} \mathcal{L}(T_i)$. Observe that since

reversing a Q-node modifies the labels of the incident skeleton edges, two rooted trees of $\mathfrak{T}_{\text{FPQ}}(H)$ may not be isomorphic (see Figure 7).



■ **Figure 7** An FPQ-hierarchy H . The set $\mathfrak{T}_{\text{FPQ}}(H)$ contains 12 rooted trees, two of which are depicted. Observe that from the left to the right tree, the ordering on the leaves of the Q-node is reversed and that the ordering on the leaves of the P-nodes are different. In both trees however, the path containing $\{x, y, z\}$ is attached below the leaves $\{b, c\}$ since these leaves form the interval $[2, 3]$ of the Q-node and this interval is the label of the unique skeleton edge.

4 Compact representation of the set of indifference tree-layouts

In this section, we show how, given a proper chordal graph G and a vertex $x \in V$, an FPQ-hierarchy H can be constructed to represent the set of indifference tree-layouts rooted at a vertex x (if such an indifference tree-layout exists). To that aim, we first provide a characterization of indifference tree-layout alternative to Theorem 6. This characterization naturally leads us to define the notion of *block* that, for a fixed vertex x of a proper chordal graph, drives the combinatorics of the set of indifference tree-layouts rooted at x .

Let S be a non-empty vertex subset of a connected graph $G = (V, E)$ and let C be a connected component of $G - S$. We say that $x \in C$ is *S-maximal* if for every vertex $y \in C$, $N(y) \cap S \subseteq N(x) \cap S$. Observe that if C contains two distinct *S*-maximal vertices x and y , then $N(x) \cap S = N(y) \cap S$.

► **Definition 10.** *Let S be a subset of vertices of a graph $G = (V, E)$ and let C be a connected component of $G - S$. A maximal subset of vertices $X \subseteq C$ is an *S*-block, if every vertex of X is *S*-maximal and $(N(S) \cap V(C))$ -universal.*

We let the reader observe that a connected component C of $G - S$ may not contain an *S*-block. However, if C contains an *S*-block, then it is uniquely defined. In the following paragraph we summarize the approach taken towards showing Theorem 11.

It can be shown that indifference tree-layouts are characterized as those such that for every vertex x distinct from $\rho_G^{-1}(r)$, x is $A_{\mathbf{T}_G}(x)$ -maximal and $(N(A_{\mathbf{T}_G}(x)) \cap D_{\mathbf{T}_G}[x])$ -universal. This implies that every vertex x distinct from the root of an indifference tree-layout can be associated with a non-empty $A_{\mathbf{T}_G}(x)$ -block containing x . Hereafter, we let $B_{\mathbf{T}_G}(x)$ denote that block. If $x = \rho_G^{-1}(r)$, then we set $B_{\mathbf{T}_G}(x) = \{x\}$. Then, we prove that for every vertex $x \in V$, the vertices of the block $B_{\mathbf{T}_G}(x)$ appear consecutively on a path rooted at x and induces a clique in G . Moreover, we show that the set of $\mathcal{B}(\mathbf{T}_G)$ containing the inclusion-maximal blocks of \mathbf{T}_G , partitions the vertex set of G . It follows that we can define the *block tree* of \mathbf{T}_G , which we denote $\mathbf{T}_G|_{\mathcal{B}(\mathbf{T}_G)}$, by contracting every block of $\mathcal{B}(\mathbf{T}_G)$ into a single node. We get that if a connected proper chordal graph admits two indifference tree-layouts rooted at the same vertex, then the corresponding block trees are isomorphic,

5.1 Recognition

Given a graph $G = (V, E)$, the recognition algorithm test for every vertex $x \in V$, if G has an indifference tree-layout rooted at x . We proceed in two steps. First, we compute the block tree $\mathbf{B}_G^{\text{tree}}(x)$ of G rooted at x that would correspond to the skeleton tree of the FPQ-hierarchy $\mathbf{H}_G(x)$ if G has an indifference tree-layout rooted at x . Then, in the second step, instead of computing $\mathbf{H}_G(x)$, we verify that $\mathbf{B}_G^{\text{tree}}(x)$ can indeed be turned into an indifference tree-layout of G . If eventually we can construct an indifference tree-layout, then G is proper chordal. If G is proper chordal and has an indifference tree-layout rooted at x , then the algorithm will succeed.

Computing the blocks and the block tree. We assume that the input graph $G = (V, E)$ is proper chordal and consider a vertex $x \in V$ that is the root of some indifference tree-layout of G . As discussed above, the first step aims at computing the skeleton tree of $\mathbf{H}_G(x)$ (see Theorem 11). That skeleton tree is the block tree $\mathbf{B}_G^{\text{tree}}(x)$ that can be obtained from any indifference tree-layout rooted at x by contracting the blocks of $\mathcal{B}_G(x)$ into a single node each. To compute $\mathbf{B}_G^{\text{tree}}(x)$, we perform a search on G starting at x (see Algorithm 1 below). At every step of the search, if the set of searched vertices is S , then the algorithm either identifies, in some connected component C of $G - S$, a new block S -block of $\mathcal{B}_G(x)$ and connects it to the current block tree, or (if C does not contain an S -block) stops and declares that there is no block tree rooted at x .

■ **Algorithm 1** Block tree computation.

Input: A graph $G = (V, E)$ and a vertex $x \in V$.
Output: The block tree $\mathbf{B}_G^{\text{tree}}(x)$, if G has an indifference tree-layout rooted at x .

```

1 set  $S \leftarrow \{x\}$ ,  $\mathcal{B} \leftarrow \{\{x\}\}$  and  $\mathbf{B}^{\text{tree}} \leftarrow (\mathcal{B}, \emptyset)$ ;
2 while  $S \neq V$  do
3   let  $C$  be a connected component of  $G - S$ ;
4   if  $C$  contains a  $S$ -block  $X$  then
5      $S \leftarrow S \cup X$  and  $\mathcal{B} \leftarrow \mathcal{B} \cup \{X\}$ ;
6     let  $B \in \mathcal{B}$  such that  $N(X) \cap B \neq \emptyset$  and that is the deepest;
7     add an edge in  $\mathbf{B}^{\text{tree}}$  between  $B$  and  $X$ ;
8   else
9     stop and return  $G$  has no indifference tree-layout rooted at  $x$ 
10  end
11 end
12 return  $\mathbf{B}^{\text{tree}}$ ;
```

► **Lemma 12.** *Let x be a vertex of a graph $G = (V, E)$. If G is proper chordal and has an indifference tree-layout rooted at x , then Algorithm 1 returns the block tree $\mathbf{B}_G^{\text{tree}}(x)$ that is the skeleton tree of the FPQ-hierarchy $\mathbf{H}_G(x)$.*

Before describing the second step of the algorithm, let us discuss some properties of \mathbf{B}^{tree} returned by Algorithm 1 when \mathcal{B} is a partition of V . An *extension* of \mathbf{B}^{tree} is any tree $T_{\mathbf{B}^{\text{tree}}}$ obtained by substituting every node $B \in \mathcal{B}$ by an arbitrary permutation σ_B of the vertices of B . In this construction, if B is the parent of B' in \mathbf{B}^{tree} , x is the last vertex of B in σ_B that has a neighbor in B' and x' is the first vertex of B' in $\sigma_{B'}$, then the parent of x' in $T_{\mathbf{B}^{\text{tree}}}$ is a vertex of B that appears after x in σ .

► **Observation 13.** Let x be a vertex of a graph $G = (V, E)$. If \mathbf{B}^{tree} is returned by Algorithm 1, then every extension $T_{\mathbf{B}^{\text{tree}}}$ of \mathbf{B}^{tree} is a tree-layout of G . And moreover, if $B, B',$ and B'' are three blocks of \mathcal{B} such that $B \prec_{\mathbf{B}^{\text{tree}}} B' \prec_{\mathbf{B}^{\text{tree}}} B''$, then for every $y \in B, y' \in B', y'' \in B'', yy'' \in E$ implies that $y'y \in E$ and $y'y'' \in E$.

Nested sets. From Observation 13, an extension of \mathbf{B}^{tree} is not yet an indifference tree-layout of G . However, if G is a proper chordal graph that has an indifference tree-layout $\mathbf{T}_G = (T, r, \rho_G)$ rooted at x , then, by Lemma 12, $\mathbf{B}^{\text{tree}} = \mathbf{B}_G^{\text{tree}}(x)$. It then follows from the proof of Theorem 11 that \mathbf{T}_G is an extension of \mathbf{B}^{tree} . The second step of the algorithm consists in testing if \mathbf{B}^{tree} has an extension that is an indifference tree-layout.

By Lemma 12, we can assume that Algorithm 1 has returned $\mathcal{B}_G(x)$ and $\mathbf{B}_G^{\text{tree}}(x)$. To every block B of $\mathcal{B}_G(x)$, we assign a collection of nested subsets of 2^B which we denote $\mathcal{C}_B = \langle \mathcal{N}_1, \dots, \mathcal{N}_k \rangle$ (see Algorithm 2 for a definition of \mathcal{C}_B). The task of the second step of the recognition algorithm is to verify that for every block B , $\text{Nested-Convex}(\mathcal{C}_B, \mathcal{S}_B) \neq \emptyset$ where $\mathcal{S}_B = \cup_{1 \leq i \leq k} \mathcal{N}_i$.

To define the collection \mathcal{C}_B , we need some notations. Let \mathcal{A}_B denote the subset of $\mathcal{B}_G(x)$ such that if $B' \in \mathcal{A}_B$, then B' is an ancestor of B in $\mathbf{B}_G^{\text{tree}}(x)$. Then we set $A_B = \cup_{B' \in \mathcal{A}_B} B'$ and denote C_B the connected component of $G - A_B$ containing B .

■ **Algorithm 2** Proper chordal graph recognition.

Input: A graph $G = (V, E)$;
Output: Decide if G is a proper chordal graph.

```

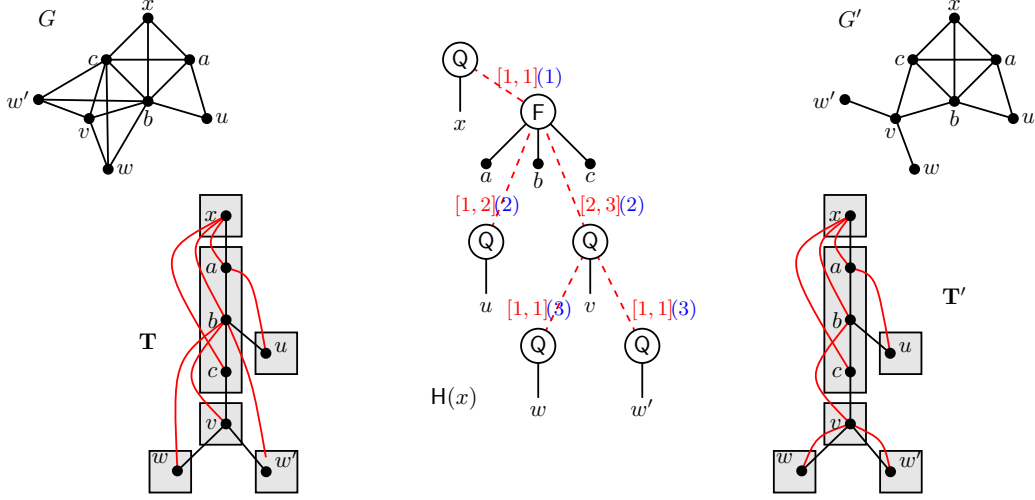
1  foreach  $x \in V$  do
2  |   if Algorithm 1 applied on  $G$  and  $x$  returns  $\mathbf{B}_G^{\text{tree}}(x)$  then
3  |   |   foreach block  $B \in \mathcal{B}_G(x)$  do
4  |   |   |   let  $C_1, \dots, C_k$  be the connected components of  $G[C_B] - B$ ;
5  |   |   |   foreach  $C_i$  in  $C_1, \dots, C_k$  do  $\mathcal{N}_i \leftarrow \{X \subseteq B \mid \exists y \in C_i, X = N(y) \cap B\}$ ;
6  |   |   |   set  $\mathcal{C}_B = \langle \mathcal{N}_1, \dots, \mathcal{N}_k \rangle$  and  $\mathcal{S}_B = \bigcup_{i \in [k]} \mathcal{N}_i$ ;
7  |   |   |   end
8  |   |   |   if  $\forall B \in \mathcal{B}(x), \mathcal{C}_B$  is a collection of nested sets st.  $\text{Nested-Convex}(\mathcal{C}_B, \mathcal{S}_B) \neq \emptyset$  then
9  |   |   |   |   stop and return  $G$  is a proper chordal graph;
10  |   |   |   end
11  |   |   end
12  |   end
13  return  $G$  is not a proper chordal graph;
```

► **Theorem 14.** We can decide in polynomial time whether a graph $G = (V, E)$ is proper chordal. Moreover, if G is proper chordal, an indifference tree-layout of G can be constructed in polynomial time.

5.2 Isomorphism

We observe that, because an FPQ-hierarchy does not carry enough information to reconstruct the original graph, two non-isomorphic proper chordal graphs G and G' may share an FPQ-hierarchy (Figure 9) satisfying the conditions of Theorem 11. More precisely, given an FPQ-hierarchy of a proper chordal graph G that satisfies the conditions of Theorem 11, one can reconstruct an indifference tree-layout \mathbf{T} of G . But \mathbf{T} is not sufficient to test the adjacency between a pair of vertices. Indeed, for a given vertex y , we cannot retrieve $N(y) \cap A_{\mathbf{T}}(y)$ from $\mathbf{H}_G(x)$ since only the intersection of $N(y)$ with the parent block is present

in $H_G(x)$. In the example of Figure 9, the vertices w and w' are also adjacent to c and b , which do not belong to their parent block.



■ **Figure 9** Two proper chordal graphs G and G' with their respective indifference tree-layouts \mathbf{T} and \mathbf{T}' . We observe that G and G' are not isomorphic, but their respective skeleton trees $T_G(x)$ and $T_{G'}(x)$ are. Moreover, $H(x)$ is an FPQ-hierarchy such that $\mathfrak{T}_{\text{FPQ}}(H(x))$ contains all the indifference tree-layouts rooted at x of G and of G' . We obtain $H^*(x)$ for G by adding to $H(x)$ the blue labels on the skeleton edges.

Let $H_G(x)$ be the FPQ-hierarchy satisfying the conditions of Theorem 11, we define the *indifference FPQ-hierarchy*, denoted $H_G^*(x)$, obtained from $H_G(x)$ by adding to every skeleton edge e , a label $\hat{A}(e)$. Suppose that e is incident to the root of the FPQ-tree of the block B , then we set $\hat{A}(e) = |N(B) \cap A_{\mathbf{T}}(B)|$. We say that two indifference FPQ-hierarchies H_1^* and H_2^* are equivalent, denoted $H_1^* \approx_{\text{FPQ}}^* H_2^*$, if $H_1 \approx_{\text{FPQ}} H_2$ and for every pair of mapped skeleton edges e_1 and e_2 we have $\hat{A}(e_1) = \hat{A}(e_2)$.

Let $\mathcal{S}_1 \in 2^{X_1}$ be a set of subsets of X_1 and $\mathcal{S}_2 \in 2^{X_2}$ be a set of subsets of X_2 . We say that \mathcal{S}_1 and \mathcal{S}_2 are *isomorphic* if there exists a bijection $f : X_1 \rightarrow X_2$ such that $S_1 \in \mathcal{S}_1$ if and only if $S_2 = \{f(x) \mid x \in S_1\} \in \mathcal{S}_2$. For $S_1 \subseteq X_1$, we denote by $f(S_1) = \{f(y) \mid y \in S_1\}$.

► **Lemma 15.** *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two (connected) proper chordal graphs. Let $H_1^*(x_1)$ be an indifference FPQ-hierarchy of G_1 and $H_2^*(x_2)$ be an indifference FPQ-hierarchy of G_2 . Then $H_1^*(x_1) \approx_{\text{FPQ}}^* H_2^*(x_2)$ if and only if G_1 and G_2 are isomorphic with x_1 mapped to x_2 .*

From Lemma 15, testing graph isomorphism on proper chordal graphs reduces to testing the equivalence between two indifference FPQ-hierarchies. To that aim, we use a similar approach to the one developed for testing interval graph isomorphism [36]. That is, we adapt the standard unordered tree isomorphism algorithm that assigns to every unordered tree a canonical *isomorphism code* [47, 1]. Testing isomorphism then amounts to testing equality between two isomorphism codes.

We proceed with a detailed description of the isomorphism test. Let H^* be an indifference FPQ-hierarchy of a proper chordal graph $G = (V, E)$. Intuitively, the isomorphism code of H^* is a string obtained by concatenating information about the root node of H^* and the isomorphism codes of the sub-hierarchies rooted at its children. To guarantee the canonicity of the isomorphism code of H^* , some of the codes of these sub-hierarchies need to be sorted

lexicographically. To that aim, we use the following convention:

$$L <_{\text{lex}} F <_{\text{lex}} P <_{\text{lex}} Q <_{\text{lex}} 0 <_{\text{lex}} 1 \dots <_{\text{lex}} n <_{\text{lex}} \dots,$$

Moreover the separating symbols (such as brackets, commas...) used in the isomorphism code for the sake of readability are irrelevant for the sort.

Before formally describing the isomorphism code of H^* , let us remind that, in an indifference FPQ-hierarchy, we can classify the children of any node t in two categories: we call a node t' a *skeleton child* of t if the tree edge $e = tt'$ is a skeleton edge of H^* , otherwise we call it a *block child* of t . We observe that the block children of a node t belong with t to the FPQ-tree of some block of $\mathcal{B}_G(x)$. It follows from the definition of an FPQ-tree, that the block children of a given node t are ordered and depending on the type of t , these nodes can be reordered. On the contrary, the skeleton children of a node t are not ordered.

For every node t of H^* , we define a code, denoted $\text{code}(t)$. We will define the isomorphism code of H^* as $\text{code}(H^*) = \text{code}(r)$, where r is the root node of H^* . We let b_1, \dots, b_k denote the block children of node t (if any, and ordered from 1 to k) and s_1, \dots, s_ℓ denote the skeleton children of t (if any). For a node t , the set of *eligible permutations* of the indices $[1, k]$ of its children depends on $\text{type}(t)$:

- if $\text{type}(t) = F$, then the identity permutation is the unique eligible permutation;
- if $\text{type}(t) = P$, then every permutation is eligible;
- if $\text{type}(t) = Q$, then the identity or its reverse permutation are the two eligible permutations.

The code of t , denoted $\text{code}(t)$, is obtained by minimizing with respect to $<_{\text{lex}}$ over all eligible permutations β of t :

$$\text{code}(t, \beta) = \begin{cases} \text{size}(t) \circ \text{type}(t) \circ \\ \text{code}(b_{\beta(1)}) \circ \dots \circ \text{code}(b_{\beta(k)}) \circ \\ \text{label}(s_{\pi_\beta(1)}, \beta) \circ \text{code}(s_{\pi_\beta(1)}) \circ \dots \circ \text{label}(s_{\pi_\beta(\ell)}, \beta) \circ \text{code}(s_{\pi_\beta(\ell)}) \end{cases}$$

where:

- $\text{type}(t) \in \{L, F, P, Q\}$, indicates whether t a leaf (L), a F-node, a P-node, or a Q-node.
- $\text{size}(t) \in \mathbb{N}$, stores the number of nodes in the sub-hierarchy rooted at t (including t).
- $\text{label}(s, \beta)$, with s being a skeleton child s of t and β being a permutation of $[1, k]$. Let e be the skeleton edge of H^* between s and t . If $I(e) = [a, b]$, we set $I^c(e) = [k+1-b, k+1-a]$. Then, we set $\text{label}(s) = \langle I^\beta(e), A(e) \rangle$, where

$$I^\beta(e) = \begin{cases} I(e), & \beta \text{ is the identity permutation} \\ I^c(e), & \text{otherwise.} \end{cases}$$

- π_β is, for some permutation β of $[1, k]$, a permutation of $[1, \ell]$ that minimizes, with respect to $<_{\text{lex}}$:

$$\text{label}(s_{\pi_\beta(1)}, \beta) \circ \text{code}(s_{\pi_\beta(1)}) \circ \dots \circ \text{label}(s_{\pi_\beta(\ell)}, \beta) \circ \text{code}(s_{\pi_\beta(\ell)}).$$

Using the previous definitions, we can show that if H_1^* and H_2^* are indifference FPQ-hierarchies of the graphs G_1 and G_2 respectively, then $H_1^* \approx_{\text{FPQ}}^* H_2^*$ if and only if $\text{code}(H_1^*) = \text{code}(H_2^*)$.

► **Theorem 16.** *Let G_1 and G_2 be two proper chordal graphs. One can test in polynomial time if $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic graphs.*

Proof. The algorithm is working as follows. First, compute a tree-layout \mathbf{T}_1 of G_1 and the indifference FPQ-hierarchy H_1^* such that $\mathbf{T}_1 \in \mathfrak{T}_{\text{FPQ}}(H_1^*)$. This can be done in polynomial time by Theorem 14. Then for every vertex $x_2 \in V_2$, we test if there exists an indifference tree-layout \mathbf{T}_2 rooted at x_2 ; compute the corresponding indifference FPQ-hierarchy H_2^* and test whether $H_1^* \approx_{\text{FPQ}}^* H_2^*$. Testing equivalence between FPQ-hierarchies can be done by computing and comparing the isomorphism codes of H_1^* and H_2^* . Moreover, this latter task can be achieved in polynomial time. By Lemma 15, if one of these tests is positive, then we can conclude that G_1 and G_2 are isomorphic graphs. ◀

6 Conclusion

A rough analysis of the complexity of the algorithms would lead to a $O(n^4)$ -time complexity for the proper chordal graph recognition problem and a for the isomorphism test. We let open the question of deriving a faster algorithms. Our results demonstrate that proper chordal graphs form a rich class of graphs. First, its relative position with respect to important graph subclasses of chordal graphs and the fact that the isomorphism problem belongs to \mathbf{P} for proper chordal graphs shows that they form a non-trivial potential island of tractability for many other algorithmic problems. In this line, we leave open the status of Hamiltonian cycle, which is polynomial time solvable in proper interval graphs [4, 30] and interval graphs [31, 5], but NP-complete on strongly chordal graphs [38]. We were only able to resolve the special case of split proper chordal graphs. An intriguing algorithmic question is whether proper chordal graphs can be recognized in linear time. Second, the canonical representation we obtained of the set of indifference tree-layouts rooted at some vertex witnesses the rich combinatorial structure of proper chordal graphs. We believe that this structure has to be further explored and could be important for the efficient resolution of more computational problems. For example, as proper chordal graphs form a hereditary class of graphs, one could wonder if the standard graph modification problems (vertex deletion, edge completion or deletion, and etc.), which are NP-complete by [33], can be resolved in FPT time. The structure of proper chordal graphs is not yet fully understood. The first natural question on this aspect is to provide a forbidden induced subgraph characterization. This will involve infinite families of forbidden subgraphs. Furthermore understanding what makes a vertex the root of an indifference tree-layout is certainly a key ingredient for a fast recognition algorithm. We would like to stress that a promising line of research is to consider further tree-layout based graph classes. For this, following the work of Damaschke [13], Hell et al. [28] and Feuilloley and Habib [17] on layouts, we need to investigate in a more systematic way various patterns to exclude, including rooted tree patterns.

References

- 1 A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- 2 S. Benzer. On the topology of the genetic fine structure. *Proceedings of the National Academy of Science*, 45(11):1607–1620, 1957. doi:10.1073/pnas.45.11.1607.
- 3 C. Berge. Färbung von Graphen deren sämtliche beziehungsweise deren ungerade Kreise starr sind (Zusammenfassung). *Wissenschaftliche Zeitschrift, Martin Luther Univ. Halle-Wittenberg, Math.-Naturwiss., Reihe*, pages 114–115, 1961.
- 4 A.A. Bertossi. Finding Hamiltonian circuits in proper interval graphs. *Information Processing Letters*, 17:97–101, 1983. doi:10.1016/0020-0190(83)90078-9.
- 5 A.A. Bertossi and M.A. Bonuccelli. Hamiltonian circuits in interval graph generalizations. *Information Processing Letters*, 23:195–200, 1986. doi:10.1016/0020-0190(86)90135-3.

- 6 D. Bienstock. On embedding graphs in trees. *Journal of Combinatorial Theory, Series B*, 49(1):103–136, 1990. doi:10.1016/0095-8956(90)90066-9.
- 7 K.S. Booth and G.S. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using pq-tree algorithm. *J. Comput. Syst. Sci.*, 13:335–379, 1976.
- 8 A. Brandstädt, V.B. Le, and J. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 1999.
- 9 S. Chaplick. Intersection graphs of non-crossing paths. In *International Workshop on Graph Theoretical Concepts in Computer Science (WG)*, volume 11789 of *Lecture Notes in Computer Science*, pages 311–324, 2019. doi:10.1007/978-3-030-30786-8_24.
- 10 M. Chein, M. Habib, and M.-C. Maurer. Partitive hypergraphs. *Discrete Mathematics*, 37:35–50, 1981. doi:10.1016/0012-365X(81)90138-2.
- 11 D.G. Corneil, H. Lerchs, and L.K. Stewart-Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(1):163–174, 1981. doi:10.1016/0166-218X(81)90013-5.
- 12 W.H. Cunningham and J. Edmonds. A combinatorial decomposition theory. *Canadian Journal of Mathematics*, 32(3):734–765, 1980. doi:10.4153/CJM-1980-057-7.
- 13 Peter Damaschke. Forbidden ordered subgraphs. *Topics in Combinatorics and Graph Theory*, pages 219–229, 1990. doi:10.1007/978-3-642-46908-4_25.
- 14 G. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961.
- 15 P. Duchet. Classical perfect graphs: An introduction with emphasis on triangulated and interval graphs. In C. Berge and V. Chvátal, editors, *Topics on Perfect Graphs*, volume 88 of *North-Holland Mathematics Studies*, pages 67–96. North-Holland, 1984. doi:10.1016/S0304-0208(08)72924-4.
- 16 Dwight Duffus, Mark Ginn, and Vojtěch Rödl. On the computational complexity of ordered subgraph recognition. *Random Structure and Algorithms*, 7(3):223–268, 1995. doi:10.1002/rsa.3240070304.
- 17 Laurent Feuilloley and Michel Habib. Graph classes and forbidden patterns on three vertices. *SIAM Journal on Discrete Mathematics*, 35(1):55–90, 2021. doi:10.1137/19M1280399.
- 18 T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1):25–66, 1967. doi:10.1007/BF02020961.
- 19 F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory Series B*, 16:47–56, 1974. doi:10.1016/0095-8956(74)90094-X.
- 20 M. Ginn. Forbidden ordered subgraph vs. forbidden subgraph characterizations of graph classes. *Journal of Graph Theory*, 30(71-76), 1999. doi:10.1002/(SICI)1097-0118(199902)30:2<71::AID-JGT1>3.0.CO;2-G.
- 21 M.C. Golumbic. Trivially perfect graphs. *Discrete Mathematics*, 24:105–107, 1978. doi:10.1016/0012-365X(78)90178-4.
- 22 M.C. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, 1980. doi:10.1016/C2013-0-10739-8.
- 23 S. Guzmán-Pro, P. Hell, and C. Hernández-Cruz. Describing hereditary properties by forbidden circular orderings. *Applied Mathematics and Computation*, 438:127555, 2023. doi:10.1016/j.amc.2022.127555.
- 24 M. Habib, R.M. McConnell, C. Paul, and L. Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234:59–84, 2000. doi:10.1016/S0304-3975(97)00241-7.
- 25 M. Habib and C. Paul. A survey on algorithmic aspects of modular decomposition. *Computer Science Review*, 4:41–59, 2010. doi:10.1016/j.cosrev.2010.01.001.
- 26 A. Hajnal and J. Surányi. Über die Auflösung von Graphen in vollständige Teilgraphen. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae. Sectio Mathematica*, 1:113–121, 1958.
- 27 G. Hajös. Über eine Art von Graphen. *Internationale Mathematische Nachrichten*, 11, 1957. Problem 65.

- 28 Pavol Hell, Bojan Mohar, and Arash Rafiey. Ordering without forbidden patterns. In *Annual European Symposium on Algorithms (ESA)*, volume 8737 of *Lecture Notes in Computer Science*, pages 554–565, 2014. doi:10.1007/978-3-662-44777-2_46.
- 29 W.-L. Hsu. $o(n.m)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM Journal on Computing*, 24(3):411–439, 1995. doi:10.1137/s0097539793260726.
- 30 L. Ibarra. A simple algorithm to find hamiltonian cycles in proper interval graphs. *Information Processing Letters*, 109:1105–1108, 2009. doi:10.1016/j.ipl.2009.07.010.
- 31 J.M. Keil. Finding Hamiltonian circuits in interval graphs. *Information Processing Letters*, 20:201–206, 1985. doi:10.1016/0020-0190(85)90050-X.
- 32 H. Lerchs. On cliques and kernels. Technical report, Departement of Computer Science, University of Toronto, 1971.
- 33 J. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.
- 34 G. Liotta, I. Rutter, and A. Tappini. Simultaneous FPQ-ordering and hybrid planarity testing. *Theoretical Computer Science*, 874:59–79, 2021. doi:10.1016/j.tcs.2021.05.012.
- 35 P. Looges and S. Olariu. Optimal greedy algorithms for indifference graphs. *Computers and Mathematics with Applications*, 25(7):15–25, 1993. doi:10.1016/0898-1221(93)90308-I.
- 36 G.S. Lueker and K.S. Booth. A linear time algorithm for deciding interval graphs isomorphism. *Journal of ACM*, 26(2):183–195, 1979. doi:10.1145/322123.322125.
- 37 R.H. Möhring and F.J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete Mathematics*, 19:257–356, 1984.
- 38 H. Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics*, 156:291–298, 1996. doi:10.1016/0012-365X(95)00057-4.
- 39 J. Nešetřil and P. Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.
- 40 S. Olariu. An optimal greedy heuristic to color interval graphs. *Information Processing Letters*, 37:21–25, 1991. doi:10.1016/0020-0190(91)90245-D.
- 41 F.S. Roberts. *Representations of indifference relations*. PhD thesis, Stanford University, 1968.
- 42 F.S. Roberts. Indifference graphs. In F. Harrary, editor, *Proof Techniques in Graph Theory*, pages 139–146, 1969.
- 43 D. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(597-609), 1970. doi:10.1016/0022-247X(70)90282-9.
- 44 Dale J. Skrien. A relationship between triangulated graphs, comparability graphs, proper interval graphs, proper circular-arc graphs, and nested interval graphs. *Journal of Graph Theory*, 6:309–316, 1982. doi:10.1002/jgt.3190060307.
- 45 D.P. Sumner. Graphs indecomposable with respect to the X -join. *Discrete Mathematics*, 6:281–298, 1973. doi:10.1016/0012-365X(73)90100-3.
- 46 R. Uehara, S. Toda, and T. Nagoya. Graph isomorphism completeness for chordal bipartite graphs and strongly chordal graphs. *Discrete Applied Mathematics*, 145:479–482, 2005. doi:10.1016/j.dam.2004.06.008.
- 47 G. Valiente. *Algorithms on trees and graphs*. Texts in Computer Science. Springer, 2002. doi:10.1007/978-3-030-81885-2.

Randomized Query Composition and Product Distributions

Swagato Sanyal   

Department of Computer Science and Engineering,
Indian Institute of Technology Kharagpur, India

Abstract

Let R_ϵ denote randomized query complexity for error probability ϵ , and $R := R_{1/3}$. In this work we investigate whether a perfect composition theorem $R(f \circ g^n) = \Omega(R(f) \cdot R(g))$ holds for a relation $f \subseteq \{0, 1\}^n \times \mathcal{S}$ and a total inner function $g : \{0, 1\}^m \rightarrow \{0, 1\}$.

Composition theorems of the form $R(f \circ g^n) = \Omega(R(f) \cdot M(g))$ are known for various measures M . Such measures include the sabotage complexity defined by Ben-David and Kothari (ICALP 2015), the max-conflict complexity defined by Gavinsky, Lee, Santha and Sanyal (ICALP 2019), and the linearized complexity measure defined by Ben-David, Blais, Göös and Maystre (FOCS 2022). The above measures are asymptotically non-decreasing in the above order. However, for total Boolean functions no asymptotic separation is known between any two of them.

Let D^{prod} denote the maximum distributional query complexity with respect to any product (over variables) distribution. In this work we show that for any total Boolean function g , the sabotage complexity $RS(g) = \tilde{\Omega}(D^{\text{prod}}(g))$. This gives the composition theorem $R(f \circ g^n) = \tilde{\Omega}(R(f) \cdot D^{\text{prod}}(g))$. In light of the minimax theorem which states that $R(g)$ is the maximum distributional complexity of g over any distribution, our result makes progress towards answering the composition question.

We prove our result by means of a complexity measure R_ϵ^{prod} that we define for total Boolean functions. Informally, $R_\epsilon^{\text{prod}}(g)$ is the minimum complexity of any randomized decision tree with unlabelled leaves with the property that, for every product distribution μ over the inputs, the average bias of its leaves is at least $((1 - \epsilon) - \epsilon)/2 = 1/2 - \epsilon$. It follows by standard arguments that $R_{1/3}^{\text{prod}}(g) = \Omega(D^{\text{prod}}(g))$. We show that $R_{1/3}^{\text{prod}}$ is equivalent to the sabotage complexity up to a logarithmic factor.

Ben-David and Kothari asked whether $RS(g) = \Theta(R(g))$ for total functions g . We generalize their question and ask if for any error ϵ , $R_\epsilon^{\text{prod}}(g) = \tilde{\Theta}(R_\epsilon(g))$. We observe that the work by Ben-David, Blais, Göös and Maystre (FOCS 2022) implies that for a perfect composition theorem $R_{1/3}(f \circ g^n) = \Omega(R_{1/3}(f) \cdot R_{1/3}(g))$ to hold for any relation f and total function g , a necessary condition is that $R_{1/3}(g) = O(\frac{1}{\epsilon} \cdot R_{\frac{1}{2}-\epsilon}(g))$ holds for any total function g . We show that $R_\epsilon^{\text{prod}}(g)$ admits a similar error-reduction $R_{1/3}^{\text{prod}}(g) = \tilde{O}(\frac{1}{\epsilon} \cdot R_{\frac{1}{2}-\epsilon}^{\text{prod}}(g))$. Note that from the definition of R_ϵ^{prod} it is not immediately clear that R_ϵ^{prod} admits any error-reduction at all.

We ask if our bound $RS(g) = \tilde{\Omega}(D^{\text{prod}}(g))$ is tight. We answer this question in the negative, by showing that for the NAND tree function, sabotage complexity is polynomially larger than D^{prod} . Our proof yields an alternative and different derivation of the tight lower bound on the bounded error randomized query complexity of the NAND tree function (originally proved by Santha in 1985), which may be of independent interest. Our result shows that sometimes, $R_{1/3}^{\text{prod}}$ and sabotage complexity may be useful in producing an asymptotically larger lower bound on $R(f \circ g^n)$ than $\tilde{\Omega}(R(f) \cdot D^{\text{prod}}(g))$. In addition, this gives an explicit polynomial separation between R and D^{prod} which, to our knowledge, was not known prior to our work.

2012 ACM Subject Classification Theory of computation \rightarrow Oracles and decision trees; Theory of computation \rightarrow Communication complexity; Mathematics of computing \rightarrow Graph theory

Keywords and phrases Randomized query complexity, Decision Tree, Boolean function complexity, Analysis of Boolean functions

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.56

Funding Supported by an ISIRD grant by Sponsored Research and Industrial Consultancy (SRIC), IIT Kharagpur.



© Swagato Sanyal;
licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 56; pp. 56:1–56:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

A decision tree or a query algorithm for a relation $f \subseteq \{0, 1\}^n \times S$ can query various bits of an input bit string $x = (x_1, \dots, x_n)$ in an adaptive fashion, with the goal of outputting an $s \in S$ such that $(x, s) \in f$. A randomized decision tree is assumed to have access to some source of randomness, and may choose a variable to query based on responses to previous queries, and the randomness. The complexity of a decision tree is the number of variables that it queries in the worst case. A decision tree that uses no randomness and for every x outputs an s such that $(x, s) \in f$ is called a deterministic decision tree computing f . The randomized query complexity of f for error ϵ , denoted by $R_\epsilon(f)$, is the least complexity of any randomized decision tree that, for every input x , outputs s such that $(x, s) \in f$ with probability (over its own randomness) at least $1 - \epsilon$. Similarly, the deterministic query complexity of f , denoted by $D(f)$, is the least complexity of any deterministic decision tree computing f . For a probability distribution μ over the domain of f , the distributional query complexity of f with respect to μ and for error ϵ , denoted by $D_\epsilon^\mu(f)$, is the least complexity of any deterministic decision tree that, for a random input x sampled from μ , fails to output an s such that $(x, s) \in f$ with probability at most ϵ . Define $R(f) := R_{1/3}(f)$ and $D^\mu(g) := D_{1/3}^\mu(g)$. See Appendix A for more details about the aforementioned notions.

For a total Boolean function $g : \{0, 1\}^m \rightarrow \{0, 1\}$, the composition $f \circ g^n$ is the relation comprising all pairs $((x_1, \dots, x_n), s) \in (\{0, 1\}^m)^n$ such that $((g(x_1), \dots, g(x_n)), s) \in f$.

It is easy to see that $D(f \circ g^n) \leq D(f) \cdot D(g)$; a decision tree for $f \circ g^n$ may be constructed simply by simulating an optimal tree of f , and serving each query that it makes by solving the corresponding copy of g using an optimal tree of g . For randomized query algorithms, a similar idea works out, albeit with some additional work to handle errors, to show that $R(f \circ g^n) = O(R(f) \cdot R(g) \cdot \log R(f))$.

Composition questions ask whether the aforementioned upper bounds on the complexity of $f \circ g^n$ are asymptotically optimal. These are fundamental questions about the structure of optimal algorithms for $f \circ g^n$, and have received considerable attention in research.

It is known from the works of Montenegro [11] and Tal [19] that $D(f \circ g^n) = D(f) \cdot D(g)$. Thus the composition question for deterministic query complexity has been completely answered. On the contrary, in spite of extensive research, a complete answer to the composition question for randomized query complexity is still lacking.

1.1 Past works on randomized query composition

Past works dealt with a more general composition question for randomized query complexity, where the inner function g is allowed to be partial. The definition of $f \circ g^n$ and the aforementioned upper bounds on $D(f \circ g^n)$ and $R(f \circ g^n)$ can be accordingly generalized; we are omitting the details in this paper. In 2015, Ben-David and Kothari [16] defined the sabotage complexity measure $RS(g)$ of a partial Boolean function g . They showed that $R(f \circ g^n) = \Omega(R(f) \cdot RS(g))$. They further showed that for total g , $RS(g) = \tilde{\Omega}(\sqrt{R(g)})$, implying $R(f \circ g^n) = \tilde{\Omega}(R(f) \cdot \sqrt{R(g)})$. In 2019, Gavinsky, Lee, Santha and Sanyal [6] introduced the max-conflict complexity $\bar{\chi}(g)$ and showed that $R(f \circ g^n) = \Omega(R(f) \cdot \bar{\chi}(g))$. They further showed that even for partial functions g , $\bar{\chi}(g) = \Omega(\sqrt{R(g)})$, implying $R(f \circ g^n) = \Omega(R(f) \cdot \sqrt{R(g)})$. Moreover, they showed that for all partial functions g , $\bar{\chi}(g) = \Omega(RS(g))$. They also demonstrated unbounded separation between $\bar{\chi}(g)$ and $RS(g)$ for a partial g . In 2022, Ben-David, Blais, Göös and Maystre [4] introduced the linearized complexity measure $L(g)$. They showed that for any partial g , $R(f \circ g^n) = \Omega(R(f) \cdot L(g))$, and that L is the largest measure M for which the statement $R(f \circ g^n) = \Omega(R(f) \cdot M(g))$ holds. They also demonstrated polynomial separation between $L(g)$ and $\bar{\chi}(g)$ for a partial g .

A different line of work has focused on proving bounds on $R(f \circ g^n)$ of the form $\Omega(M(f) \cdot R(g))$ for some complexity measure M [1, 7, 2]. In 2020 Ben-David and Blais [3] defined the noisy query complexity noisyR and showed that noisyR is the largest measure M for which the statement $R(f \circ g^n) = \Omega(M(f) \cdot R(g))$ holds. In 2023, Chakraborty et al. [5] showed that for the special case when $R(f) = \Theta(n)$, a near-perfect randomized query composition theorem $R(f \circ g^n) = \tilde{\Omega}(R(f) \cdot R(g))$ holds.

1.2 Our results

This work investigates the possibility of a perfect randomized query composition theorem $R(f \circ g^n) = \Omega(R(f) \cdot R(g))$ when g is a total function. As discussed in the preceding section, past works have introduced measures RS , $\bar{\chi}$ and L that are asymptotically non-decreasing in the above order. As discussed before, we also know that any two of them can be asymptotically separated. However, the Boolean functions that witness these separations are all partial, and to the best of our knowledge, no separation between these measures is known for total functions. Does one of these measures coincides with R for total functions?

Ben-David and Kothari asked in their paper whether $\text{RS}(g) = \Theta(R(g))$ for total g . Our first result is that for any total g , $\text{RS}(g)$ is, up to a logarithmic factor, at least the maximum distributional query complexity of g for any product (over variables) distribution. Let PROD be the set of all product distributions over the domain $\{0, 1\}^m$ of g . Define $D^{\text{prod}}(g) := \max_{\mu \in \text{PROD}} \{D^\mu(g)\}$.

► **Theorem 1.** *For any total function $g : \{0, 1\}^m \rightarrow \{0, 1\}$,*

$$\text{RS}(g) = \tilde{\Omega}(D^{\text{prod}}(g)).$$

Informally, the sabotage complexity captures the minimum number of randomized queries required to distinguish any pair of input strings on which the function values differ (see Section 2.3 for a formal definition). Theorem 1 shows that this task is at least as hard as deciding the function on every possible product distribution (potentially with a different query algorithm for each distribution).

Together with the composition theorem of Ben-David and Kothari, Theorem 1 immediately yields the following corollary.

► **Corollary 2.** *For any total function $g : \{0, 1\}^m \rightarrow \{0, 1\}$,*

$$R(f \circ g^n) = \tilde{\Omega}(R(f) \cdot D^{\text{prod}}(g)).$$

The minimax theorem (Fact 17) states that $R(g) = \max_{\mu} D^\mu(g)$, where the maximum is over all probability distributions over the domain of g . In this light Corollary 2 makes progress towards answering the randomized composition question for total inner functions. An additional motivation for our first result is that product distributions comprise a natural class of distributions that has received significant attention in Boolean function complexity research [9, 8, 10, 17].

We prove Theorem 1 by introducing a new complexity measure R_ϵ^{prod} . Informally speaking, $R_\epsilon^{\text{prod}}(g)$ is the minimum complexity of any randomized decision tree with unlabelled leaves with the property that, for every product distribution μ over the inputs, the average bias of its leaves is at least $((1 - \epsilon) - \epsilon)/2 = 1/2 - \epsilon$. Define $R^{\text{prod}}(g) := R_{1/3}^{\text{prod}}(g)$. See Section 2.2 for formal definitions. It follows by standard arguments that $R^{\text{prod}}(g) = \Omega(D^{\text{prod}}(g))$ (see Claim 11). Our next next result shows that RS is characterized by R^{prod} up to a logarithmic factor.

► **Theorem 3.** For all total functions $g : \{0, 1\}^m \rightarrow \{0, 1\}$,

1. $\text{RS}(g) = O(\text{R}^{\text{prod}}(g))$, and
2. $\text{RS}(g) = \Omega(\text{R}^{\text{prod}}(g) / \log \text{R}^{\text{prod}}(g))$.

Theorem 1 follows immediately from Theorem 3(2) and the fact that $\text{R}^{\text{prod}}(g) = \Omega(\text{D}^{\text{prod}}(g))$ (Claim 11).

Since any non-trivial product distribution is supported on all of $\{0, 1\}^m$, $\text{R}^{\text{prod}}(g)$ and $\text{D}^{\text{prod}}(g)$ are well-defined only for total functions g . The proof of Theorem 3 (that goes via Lemma 6 discussed later) makes important use of the totality of g . We hope that the measure R^{prod} , the characterization of RS presented in Theorem 3, and the insights acquired in our proof techniques, specially pertaining to ways of exploiting totality, will be useful in future research to resolve whether $\text{RS}(g) = \Theta(\text{R}(g))$ for total functions g .

In light of Theorem 3 the question whether $\text{R}(g) = \Theta(\text{RS}(g))$ for total functions g translates to the question whether $\text{R}(g) = \tilde{\Theta}(\text{R}^{\text{prod}}(g))$. We generalize this question for every error ϵ .

► **Question 4.** Is it true that for every total function $g : \{0, 1\}^m \rightarrow \{0, 1\}$ and $\epsilon : \mathbb{N} \rightarrow (0, 1/2)$, $\text{R}_{\epsilon(m)}(g) = \tilde{\Theta}(\text{R}_{\epsilon(m)}^{\text{prod}}(g))$?

From the work of Ben-David, Blais, Göös and Maystre [4] it follows that for any error parameter ϵ , the linearized complexity measure $\text{L}(g)$ of g is bounded above by $O\left(\frac{1}{\epsilon} \cdot \text{R}_{\frac{1}{2}-\epsilon}(g)\right)$. As discussed before, they also show that L is the largest measure M for which the statement $\text{R}(f \circ g) = \Omega(\text{R}(f)\text{M}(g))$ holds for all relations f and partial functions g . We thus have that for a perfect composition theorem $\text{R}(f \circ g) = \Omega(\text{R}(f)\text{R}(g))$ to hold for any relation f and any total Boolean function g , a necessary condition is that $\text{R}(g) = O\left(\frac{1}{\epsilon} \cdot \text{R}_{\frac{1}{2}-\epsilon}(g)\right)$ holds for any total Boolean function g . In light of Question 4, we may ask if $\text{R}_{\epsilon}^{\text{prod}}$ admits a similar error reduction. Our next result answers this question in the affirmative (up to a logarithmic factor).

► **Theorem 5.** For every total function $g : \{0, 1\}^m \rightarrow \{0, 1\}$ and $\epsilon : \mathbb{N} \rightarrow (0, 1/2)$,

$$\text{R}^{\text{prod}}(g) = \tilde{O}\left(\frac{1}{\epsilon(m)} \cdot \text{R}_{\frac{1}{2}-\epsilon(m)}^{\text{prod}}(g)\right).$$

We remark here that from the definition of $\text{R}_{\epsilon}^{\text{prod}}$ it is not immediately clear that it admits any error-reduction at all.

To prove Theorems 3 and 5, we define a version of sabotage complexity with errors, that we denote by RS_{ϵ} . Informally, $\text{RS}_{\epsilon}(g)$ is the minimum number of randomized queries required to distinguish every pair of inputs with different function values with probability at least $1 - \epsilon$ (see Section 2.3 for a formal definition). Let $s(g)$ denote the sensitivity of g (see Section 2.1 for a formal definition). The following lemma constitutes the technical core of our proofs of Theorems 3 and 5.

► **Lemma 6.** For all total Boolean functions $g : \{0, 1\}^n \rightarrow \{0, 1\}$, and $\epsilon : \mathbb{N} \rightarrow (0, 1/2)$,

1. $\text{R}^{\text{prod}}(g) = O\left(\frac{1}{\epsilon(n)} \cdot \text{RS}_{1-\epsilon(n)}(g) \log s(g)\right)$, and
2. $\text{RS}_{1-2\epsilon(n)}(g) \leq \text{R}_{\frac{1}{2}-\epsilon(n)}^{\text{prod}}(g)$.

Is the bound in Theorem 1 tight? Our next result gives a negative answer to this question. We show that for the NAND tree function (defined shortly), RS and D^{prod} are polynomially separated. Consider a complete binary tree of depth d . Each leaf is labelled by a distinct Boolean variable. Each internal node is a binary NAND gate. For each input, the evaluation of this Boolean formula is the output of the NAND tree function, that we denote by g_d .

► **Theorem 7.** $D^{\text{prod}}(g_d) = O(\text{RS}(g_d)^{1-\Omega(1)})$.

Saks and Wigderson [14] showed that the zero-error randomized query complexity of g_d is $\Theta(\alpha^d)$ for $\alpha = \frac{1+\sqrt{33}}{4}$. Later Santha [15] showed that $R(g_d) = \Theta(\alpha^d)$. We prove Theorem 7 in two parts. First, we show an upper bound of $O((\alpha - \Omega(1))^d)$ on $D^{\text{prod}}(g_d)$.

► **Lemma 8.** *There exists a constant $\delta > 0$ such that $D^{\text{prod}}(g_d) = O((\alpha - \delta)^d)$.*

Works by Pearls [13] and Tarsi [19] showed that there exists a constant $\eta > 0$ such that for all distributions μ where each variable is set to 1 independently with some probability p , $D^\mu(g_d) = O((\alpha - \eta)^d)$. In Lemma 8 we bound $D^\mu(g_d)$ for any product distribution μ . Our bound is quantitatively weaker than those by Pearls [13] and Tarsi [19], and we do not comment on its tightness.

Lemma 8 also gives an explicit polynomial separation between R and D^{prod} which, to our knowledge, was not known prior to our work.

Next, we prove a tight lower bound on $\text{RS}(g_d)$. As a by-product, our proof of the following lemma yields a different proof of the bound $R(g_d) = \Omega(\alpha^d)$ from the one by Santha [15], and may be of independent interest.

► **Lemma 9.** $\text{RS}(g_d) = \Omega(\alpha^d)$.

Lemma 9, together with the upper bound by Saks and Wigderson, shows that $\text{RS}(g_d) = \Theta(R(g_d))$. From the composition theorem proven by Ben-David and Kothari, we thus have that for all relations f , $R(f \circ g_d) = \Theta(R(f) \cdot R(g_d))$.

Lemmata 8 and 9 immediately imply Theorem 7.

1.3 Proof ideas

In this section, we sketch the ideas and techniques that have gone into the proofs of our results. We begin with Lemma 6, from which Theorems 3 and 5 follow. We then discuss Lemmata 8 and 9, from which Theorem 7 follows.

Lemma 6

We first discuss part 2, which is easier. Note that if a randomized algorithm \mathcal{R} decides g on each input with error probability $\frac{1}{2} - \epsilon$, then by a union bound two simultaneous runs of \mathcal{R} on $x \in g^{-1}(0), y \in g^{-1}(1)$ decide both $g(x)$ and $g(y)$ with error probability $1 - 2\epsilon$. This implies that \mathcal{R} distinguishes x and y with error probability $1 - 2\epsilon$.

Now we turn to part 1. This step needs arguments involving sensitivity and influence of Boolean functions, that are defined and discussed in Section 2.1. The first step is showing that distinguishing each pair of inputs with high confidence is equivalent to reading each sensitive bit of each input with the same confidence (Lemma 14). Using this, by a sequence of arguments involving standard error-reduction, we infer that there is a randomized tree \mathcal{R} of complexity $O\left(\frac{1}{\epsilon(n)} \cdot \text{RS}_{1-\epsilon(n)}(g) \log s(g)\right)$ that, for every input, with probability $1 - \frac{1}{s(g)}$, queries all its sensitive bits. This translates to the claim that the average influence of the restrictions of g to the leaves of \mathcal{R} is low. Poincaré inequality (Lemma 10) now lets us conclude that the average bias of those restrictions is small, yielding the lemma.

Lemma 8

Saks and Wigderson gave a zero-error recursive algorithm for g_d . Their algorithm recursively evaluates a randomly chosen child of the root. If that child evaluates to 0, the algorithm outputs 1 and terminates. Else, the algorithm recursively evaluates the other child and outputs the complement.

If the output of the function is 0, then the algorithm will be forced to evaluate both children. However, if the output is 1, then the algorithm avoids evaluating one of the children with probability $1/2$.

We observe that if the inputs are sampled from a product distributions, then firstly, the output will not always be 0; so we will always have scope to avoid evaluating one child. Secondly, we will also have both children evaluating to 0 with positive probability, in which case we are guaranteed to save evaluating one child.

We modify the algorithm by Saks and Wigderson to tap these opportunities; in each step we query the child which is more likely to evaluate to 0. Note that this requires knowledge of the distribution. We look at two successive levels of the tree and show that the above considerations bring us significant advantage over the algorithm by Saks and Wigderson.

Lemma 9

As mentioned before, here we work with the original definition of sabotage complexity. Our proof splits into the following steps.

1. We recursively define a “hard” distribution \mathcal{P}_d over pairs in $g_d^{-1}(0) \times g_d^{-1}(1)$.
2. We consider an arbitrary zero-error randomized algorithm \mathcal{R} for g_d . We now wish to give a lower bound on the number of queries it makes on expectation to distinguish a random pair sampled from \mathcal{P}_d .
3. Using \mathcal{R} , we recursively define a sequence of algorithms $\mathcal{A}_d, \mathcal{A}_{d-1}, \dots, \mathcal{A}_0$ such that for each i , \mathcal{A}_i is a zero-error algorithm for g_i .
4. We establish a recursive relation amongst the expected number of queries that g_i makes to distinguish a pair sampled from \mathcal{P}_i , for various i . We make a distinction between queries based on their answers (0 or 1). This step involves a small case analysis involving all deterministic trees with two variables.
5. We finish by solving the recursion established in the previous step.

2 Preliminaries

Refer to Appendix A for some notations and definitions that will be used throughout the paper.

2.1 Sensitivity and influence

For a total Boolean function g , a variable x_i is said to be sensitive for an input x if $g(x) \neq g(x^{\oplus i})$. The sensitivity of x with respect to g , denoted by $s(g, x)$, is the number of sensitive bits of x , i.e., $|\{i \in [n] \mid g(x) \neq g(x^{\oplus i})\}|$. The sensitivity of g , denoted by $s(g)$, is the maximum sensitivity of any input x with respect to g , i.e.,

$$s(g) = \max_{x \in \{0,1\}^m} s(g, x).$$

For a product distribution $\mu \in \text{PROD}$ given by parameters p_1, \dots, p_m , the *influence* of x_i with respect to g and μ is defined as

$$\text{Inf}_i(g) := 4p_i(1 - p_i) \Pr_{x \sim \mu} [g(x) \neq g(x^{\oplus i})],$$

and the influence of g with respect to μ is defined as

$$\text{Inf}(g) = \sum_{i=1}^m \text{Inf}_i(g).$$

The following inequality follows easily from the above definitions, linearity of expectation, and the observation that $4p_i(1 - p_i) \leq 1$ for all $p_i \in [0, 1]$.

$$\text{Inf}(g) \leq \mathbf{E}_{x \sim \mu} s(g, x). \quad (1)$$

Let $\text{Var}(g)$ denote the variance of the random variable $g(x)$ when x is drawn from μ . The Poincaré inequality bounds $\text{Var}(g)$ in terms of $\text{Inf}(g)$.

► **Lemma 10** (Poincaré inequality). *For every product distribution μ , $4\text{Var}(g) \leq \text{Inf}(g)$.*

A proof of the Poincaré inequality may be found in [12].

In the notations Inf_i , Inf and Var , the dependence on μ is suppressed. μ will be clear from the context.

2.2 Randomized query complexity for product distributions

Let μ be a product distribution, and T be a deterministic decision tree. For each leaf ℓ of T , let p_ℓ^μ be the probability that the computation of T on an input drawn from μ reaches ℓ . Let \mathbf{p}^μ denote the probability distribution $(p_\ell^\mu)_\ell$ over the leaves of T . We say that a randomized decision tree \mathcal{R} computes g with error ϵ for product distributions if for every product distribution $\mu \in \text{PROD}$,

$$\mathbf{E}_{T \sim \mathcal{R}} \mathbf{E}_{\ell \sim \mathbf{p}^\mu} [\min\{\Pr_{x \in \mu|_\ell} [g(x) = 0], \Pr_{x \in \mu|_\ell} [g(x) = 1]\}] \leq \epsilon,$$

where the inner expectation is over the leaves of T . We define $\min\{\Pr_{x \in \mu|_\ell} [f(x) = 0], \Pr_{x \in \mu|_\ell} [f(x) = 1]\}$ to be 0 if $p_\ell^\mu = 0$; the conditional distribution $\mu|_\ell$ is not defined in this case. The *randomized query complexity* of g for product distribution for error ϵ , denoted by $\mathbf{R}_\epsilon^{\text{prod}}(g)$, is the minimum query complexity of a randomized decision tree \mathcal{R} that satisfies the above condition. Define $\mathbf{R}^{\text{prod}}(g) := \mathbf{R}_{1/3}^{\text{prod}}(g)$.

Note that in the above definition, no reference has been made to the labels of the leaves of T . For the purpose of this definition, \mathcal{R} can be thought of as a probability distribution over trees whose leaves are unlabelled.

The following claim shows that $\mathbf{D}_\epsilon^{\text{prod}}(g)$ is bounded above by $\mathbf{R}_\epsilon^{\text{prod}}(g)$. A proof may be found in Section B.

▷ **Claim 11.** For every Boolean function g and parameter $\epsilon \in [0, 1/2]$, $\mathbf{D}_\epsilon^{\text{prod}}(g) \leq \mathbf{R}_\epsilon^{\text{prod}}(g)$.

2.3 Sabotage complexity

The *sabotage complexity* of a Boolean function g for error ϵ , denoted by $\mathbf{RS}_\epsilon(g)$, is defined to be the minimum query complexity of any randomized decision tree \mathcal{R} for which the following is true: For every $x = (x_1, \dots, x_m) \in g^{-1}(0)$, $y = (y_1, \dots, y_m) \in g^{-1}(1)$, with probability at least $1 - \epsilon$, a decision tree T drawn from \mathcal{R} when run on x queries an index i such that $x_i \neq y_i$ ¹. Define $\mathbf{RS}(g) := \mathbf{RS}_{1/3}(g)$.

Sabotage complexity was defined by Ben-David and Kothari [16]. They defined the measure as the minimum expected query complexity of any randomized decision tree to distinguish each pair of inputs $x \in g^{-1}(0), y \in g^{-1}(1)$. However, as the authors observed, the definition stated above is within a constant factor of the original definition in [16]. See more discussion on this in Section 5 where we work with the original definition.

¹ Note that T queries an index i such that $x_i \neq y_i$ when run on x if and only if T queries an index j such that $x_j \neq y_j$ when run on y .

The following fact can be proven by standard BPP amplification.

► **Fact 12.** $\forall \epsilon, \epsilon' \in (0, 1)$ and $\epsilon < \epsilon'$, $\text{RS}_\epsilon(g) = O\left(\text{RS}_{\epsilon'}(g) \cdot \frac{\log(1/\epsilon)}{\log(1/\epsilon')}\right)$.

Ben-David and Kothari proved that the sabotage complexity is lower bounded by many complexity measures that are studied in the context of decision trees. In particular, $\text{RS}(g)$ is lower bounded by $s(g)$.

► **Fact 13** ([16]). *For all Boolean function $g : \{0, 1\}^m \rightarrow \{0, 1\}$, $\text{RS}(g) = \Omega(s(g))$.*

3 Sabotage complexity and product distributions

In this section we first prove Lemma 6. We then use Lemma 6 to prove Theorems 3 and 5. The following lemma says that to distinguish each pair of inputs on which the function values differ with high probability, it is necessary and sufficient to query each sensitive bit of each input with high probability.

► **Lemma 14.** *Let $g : \{0, 1\}^n \rightarrow \{0, 1\}$ be a total Boolean function. Then, $\text{RS}_\epsilon(g) \leq r$ if and only if there is a randomized decision tree \mathcal{R} of query complexity at most r such that for each input x and each variable x_i sensitive for x , $\Pr_{T \sim \mathcal{R}}[T \text{ does not query } x_i \text{ when run on } x] \leq \epsilon$.*

Proof.

(If) Let \mathcal{R} be a randomized decision tree of complexity at most r such that for every input x and every variable x_i sensitive for x , $\Pr_{T \sim \mathcal{R}}[T \text{ does not query } x_i \text{ when run on } x] \leq \epsilon$.

We will show that \mathcal{R} fails to distinguish any pair $w \in g^{-1}(0), y \in g^{-1}(1)$ with probability at most ϵ . Fix such a pair w, y . let $B = \{i_1, \dots, i_k\}$ be the positions where w and y differ. Define $B_0 := \emptyset$ and for $1 \leq j \leq k$, define $B_j := \{i_1, \dots, i_j\}$. Let m be the smallest index such that $g(w^{\oplus B_m}) = 1$. Thus, variable w_m is sensitive for $w^{\oplus B_{m-1}}$ and $w^{\oplus B_m}$. Now, observe that if T does not query any variable w_{i_j} with $i_j \in B$ when run on w , then T does not query w_m when run on $w^{\oplus B_{m-1}}$. By our assumption about \mathcal{R} , the probability of this happening when T is sampled from \mathcal{R} is at most ϵ .

(Only if) Let $\text{RS}_\epsilon(g) \leq r$. Thus there exists a randomized decision tree \mathcal{R} of query complexity r that fails to distinguish each pair $w \in g^{-1}(0), y \in g^{-1}(1)$ with probability at most ϵ . Without loss of generality, assume that $x \in g^{-1}(0)$. Then $x^{\oplus i} \in g^{-1}(1)$. Since distinguishing x and $x^{\oplus i}$ is equivalent to querying x_i when run on x , the proof follows. ◀

Now we proceed to proving Lemma 6. For convenience, we use ϵ for $\epsilon(n)$ throughout the following proof.

Proof of Lemma 6.

Part 1. Let $\text{RS}_{1-\epsilon}(g) = r$. By Lemma 14, there exists a randomized query algorithm \mathcal{R} of complexity at most r such that for each input x and each variable x_i sensitive for x , $\Pr_{T \sim \mathcal{R}}[T \text{ does not query } x_i \text{ when run on } x] \leq 1 - \epsilon$. Let \mathcal{R}' be the algorithm obtained by repeating \mathcal{R} $\frac{2}{\epsilon} \ln s(g)$ times with independent randomness. Thus for each input x and each variable x_i sensitive for x , we have that $\Pr_{T \sim \mathcal{R}'}[T \text{ does not query } x_i \text{ when run on } x] \leq (1 - \epsilon)^{\frac{1}{\epsilon} \cdot 2 \ln s(g)} \leq \frac{1}{s(g)^2}$, where we have used the inequality $1 - x \leq e^{-x}$ for all $x \in (-\infty, \infty)$. Again for each input x , by a union bound over all variables x_i sensitive for x , we have that the probability that a deterministic tree sampled from \mathcal{R}' does not query all variables sensitive for x when run on x , is at most $\frac{s(g, x)}{s(g)^2} \leq \frac{1}{s(g)}$. The query complexity of \mathcal{R}' is $O\left(\frac{1}{\epsilon} \cdot \text{RS}_{1-\epsilon}(g) \log s(g)\right)$. We will show that \mathcal{R}' computes g with error $1/3$ for product distributions. This will complete the proof of this part.

To this end, fix a product distribution μ . For any deterministic decision tree T and input x of f , define

$$\mathbf{Q}(T, x) = \begin{cases} 1 & \text{if } T \text{ does not query all sensitive variables of } x \text{ when run on } x, \\ 0 & \text{otherwise.} \end{cases}$$

By the property of \mathcal{R}' , for every input x , we have that

$$\mathbf{E}_{T \sim \mathcal{R}'}[\mathbf{Q}(T, x)] = \Pr_{T \sim \mathcal{R}'}[\mathbf{Q}(T, x) = 1] \leq \frac{1}{s(g)}.$$

Since the above is true for each x , we have the following for a random input x sampled from μ .

$$\mathbf{E}_{T \sim \mathcal{R}'} \mathbf{E}_{x \sim \mu}[\mathbf{Q}(T, x)] \leq \frac{1}{s(g)}. \quad (2)$$

For each leaf ℓ of T , let p_ℓ^μ be the probability that the computation of T on an input drawn from μ reaches ℓ and \mathbf{p}^μ denote the probability distribution $(p_\ell^\mu)_\ell$ over the leaves of T . We rewrite (2) as follows.

$$\mathbf{E}_{T \sim \mathcal{R}'} \mathbf{E}_{\ell \sim \mathbf{p}^\mu} \mathbf{E}_{x \sim \mu | \ell}[\mathbf{Q}(T, x)] \leq \frac{1}{s(g)}, \quad (3)$$

treating $\mathbf{E}_{x \sim \mu | \ell}[\mathbf{Q}(T, x)]$ as 0 if $p_\ell^\mu = 0$. Now, fix an arbitrary leaf ℓ of T such that $p_\ell^\mu > 0$, and consider the Boolean function $g |_\ell$. Note that for any $x \in \ell$, if $\mathbf{Q}(T, x) = 0$, then $s(g |_\ell, x) = 0$. We thus have that

$$\mathbf{E}_{x \sim \mu | \ell}[s(g |_\ell, x)] \leq \Pr_{x \sim \mu | \ell}[\mathbf{Q}(T, x) = 1] \cdot s(g |_\ell) \leq \mathbf{E}_{x \sim \mu | \ell}[\mathbf{Q}(T, x)] \cdot s(g). \quad (4)$$

Since μ is a product distribution and ℓ is a subcube, $\mu |_\ell$ is also a product distribution. Equations (1) and (4) thus imply that

$$\text{Inf}(g |_\ell) \leq \mathbf{E}_{x \sim \mu | \ell}[\mathbf{Q}(T, x)] \cdot s(g). \quad (5)$$

Together with Poincaré inequality (Lemma 10), (5) implies that

$$\text{Var}(f |_\ell) \leq \frac{1}{4} \cdot \mathbf{E}_{x \sim \mu | \ell}[\mathbf{Q}(T, x)] \cdot s(f). \quad (6)$$

Now, for a random variable X taking value in $\{0, 1\}$, $\text{Var}(X) = 4 \Pr[X = 0] \Pr[X = 1] \geq 2 \min\{\Pr[X = 0], \Pr[X = 1]\}$ (since $\max\{\Pr[X = 0], \Pr[X = 1]\} \geq \frac{1}{2}$). Since ℓ is an arbitrary leaf, we have by Equations (6) and (3) that

$$\begin{aligned} & \mathbf{E}_{T \sim \mathcal{R}'} \mathbf{E}_{\ell \sim \mathbf{p}^\mu} [\min\{\Pr_{x \sim \mu | \ell}[g(x) = 0], \Pr_{x \sim \mu | \ell}[g(x) = 1]\}] \\ & \leq \frac{1}{2} \cdot \mathbf{E}_{T \sim \mathcal{R}'} \mathbf{E}_{\ell \sim \mathbf{p}^\mu} [\text{Var}(g |_\ell)] && \text{by the above discussion} \\ & \leq \frac{1}{8} \cdot \mathbf{E}_{T \sim \mathcal{R}'} \mathbf{E}_{\ell \sim \mathbf{p}^\mu} \mathbf{E}_{x \sim \mu | \ell}[\mathbf{Q}(T, x)] \cdot s(g) && \text{by Equation (6)} \\ & \leq \frac{1}{8} < \frac{1}{3}. && \text{by Equation (3)} \end{aligned}$$

Since μ is an arbitrary product distribution, we have that \mathcal{R}' computes g with error $1/3$ for product distributions.

56:10 Randomized Query Composition and Product Distributions

Part 2. Fix a randomized query algorithm \mathcal{R} that attains $R_{\frac{1}{2}-\epsilon}^{\text{prod}}(g)$. We will show that \mathcal{R} also attains $RS_{1-2\epsilon}(g)$. By Lemma 14 it is sufficient to show that for each input x and each variable x_i sensitive for x , $\Pr_{T \sim \mathcal{R}}[T \text{ does not query } x_i \text{ when run on } x] \leq 1 - 2\epsilon$. To this end, fix an input x and a variable x_i sensitive for x . Now consider the distribution μ that places a probability mass of $1/2$ on x and places the remaining mass of $1/2$ on $x^{\oplus i}$. Note that μ is a product distribution. Thus from the property of R we have that

$$\mathbf{E}_{T \sim \mathcal{R}} \mathbf{E}_{\ell \sim \mathbf{p}^\mu} [\min\{\Pr_{x \in \mu|_\ell} [g(x) = 0], \Pr_{x \in \mu|_\ell} [g(x) = 1]\}] \leq \frac{1}{2} - \epsilon. \quad (7)$$

Now if T does not query x_i when run on x , then T has a leaf ℓ that contains both x and x^i , $p_\ell^\mu = 1$, and for all other leaves ℓ' of T , $p_{\ell'}^\mu = 0$. Furthermore, $\min\{\Pr_{x \in \mu|_\ell} [g(x) = 0], \Pr_{x \in \mu|_\ell} [g(x) = 1]\} = 1/2$. Thus, $\mathbf{E}_{\ell \sim \mathbf{p}^\mu} [\min\{\Pr_{x \in \mu|_\ell} [g(x) = 0], \Pr_{x \in \mu|_\ell} [g(x) = 1]\}] = 1/2$. We thus have that,

$$\begin{aligned} & \mathbf{E}_{T \sim \mathcal{R}} \mathbf{E}_{\ell \sim \mathbf{p}^\mu} [\min\{\Pr_{x \in \mu|_\ell} [g(x) = 0], \Pr_{x \in \mu|_\ell} [g(x) = 1]\}] \\ & \geq \Pr_{T \sim \mathcal{R}} [T \text{ does not query } x_i \text{ when run on } x] \cdot \frac{1}{2}. \end{aligned} \quad (8)$$

Equations (7) and (8) imply that

$$\begin{aligned} & \Pr_{T \sim \mathcal{R}} [T \text{ does not query } x_i \text{ when run on } x] \cdot \frac{1}{2} \leq \frac{1}{2} - \epsilon \\ \implies & \Pr_{T \sim \mathcal{R}} [T \text{ does not query } x_i \text{ when run on } x] \leq 1 - 2\epsilon. \end{aligned}$$

This completes the proof. ◀

Now we prove Theorems 3 and 5.

Proof of Theorem 3 Part 1. Substituting $\epsilon(n) = 1/6$ in part 2 of Lemma 6 we have that

$$R^{\text{prod}}(g) \geq RS_{2/3} = \Omega(RS(g)),$$

where the second equality follows from Fact 12. ◀

Theorem 3 (1) and Fact 13 imply that

$$R^{\text{prod}}(g) = \Omega(s(g)). \quad (9)$$

Proof of Theorem 3 Part 2.

$$\begin{aligned} RS(g) &= RS_{1/3}(g) \geq RS_{2/3}(g) \\ &= \Omega(R^{\text{prod}}(g)/\log s(g)) && \text{by Lemma 6 part 1 with } \epsilon(n) = 1/3 \\ &= \Omega(R^{\text{prod}}(g)/\log R^{\text{prod}}(g)) && \text{by Equation (9)} \end{aligned}$$
◀

Proof of Theorem 5. We have

$$\begin{aligned} R^{\text{prod}}(g) &= O\left(\frac{1}{\epsilon(n)} \cdot RS_{1-\epsilon(n)}(g) \log s(g)\right) && \text{Part (1) of Lemma 6} \\ &= O\left(\frac{1}{\epsilon(n)} \cdot RS_{1-2\epsilon(n)}(g) \log s(g)\right) && \text{since } 1 - \epsilon(n) \geq 1 - 2\epsilon(n) \\ &= O\left(\frac{1}{\epsilon(n)} \cdot R_{\frac{1}{2}-\epsilon(n)}^{\text{prod}}(g) \log s(g)\right) && \text{by part (2) of Lemma 6} \\ &= O\left(\frac{1}{\epsilon(n)} \cdot R_{\frac{1}{2}-\epsilon(n)}^{\text{prod}}(g) \log R^{\text{prod}}(g)\right). && \text{by Equation (9)} \end{aligned}$$
◀

4 Separation between D^{prod} and R

In this section we prove Lemma 8. Recall that g_d denotes the NAND tree function of depth d . Snir [18] and Saks and Wigderson [14] were the first to study g_d in the context of randomized query complexity. As mentioned in Section 1, it is known from the works of Saks and Wigderson [14] and Santha [15] that $R(g_d) = \Theta(d^\alpha)$ where $\alpha = \frac{1+\sqrt{33}}{4}$.

For a distribution μ , the zero-error distributional complexity of a Boolean function g , that we denote by $\overline{D}_0^\mu(g)$, is the least expected number of queries made by any (deterministic) tree T on a random input sampled from μ . Define $\overline{D}_0^{\text{prod}}(g) := \max_{\mu \in \text{PROD}} \overline{D}_0^\mu(g)$. By Markov's inequality, it follows that $D^{\text{prod}}(g) = O(\overline{D}_0^{\text{prod}}(g))$.

Proof of Lemma 9. We will prove an upper bound on $\overline{D}_0^{\text{prod}}(g)$. By the preceding discussion, that will prove the lemma.

Let μ be any product distribution over $\{0, 1\}^n$. Define $T(d, \mu) := \overline{D}_0^\mu(g_d)$ and $T(d) := \overline{D}_0^{\text{prod}}(g_d)$. Consider the query algorithm \mathcal{A}_μ ² given in Algorithm 1.

Algorithm 1 $\mathcal{A}_\mu(x)$.

```

1 Input: Query access to  $x = (x_1, \dots, x_{2^d})$ .
2  $g \leftarrow g_d$ .
3 if  $g$  is a variable then
4   | Query  $g$ . Return the outcome of the query.
5 end
6 else
7   | Let  $g_\ell$  and  $g_r$  respectively be the left and right subtrees of  $g$ .
8   | Let  $\mu_\ell$  and  $\mu_r$  respectively be the product distributions induced on the input
   |   spaces of  $g_\ell$  and  $g_r$  by  $\mu$ .
9   |  $t \leftarrow \arg \max_{i \in \{\ell, r\}} \Pr_{y \sim \mu_i} [g_i(y) = 0]$ .
10  |  $s \leftarrow \{\ell, r\} \setminus \{t\}$ .
11  | For  $i \in \{\ell, r\}$ , let  $x^{(i)}$  be the input to  $g_i$ .
12  | if  $\mathcal{A}_{\mu_t}(x^{(t)}) = 0$  then
13  |   | return 1.
14  | end
15  | else
16  |   | return  $\overline{\mathcal{A}_{\mu_s}(x^{(s)})}$ .
17  | end
18 end

```

\mathcal{A}_μ works as follows: if $d = 1$, i.e., if g_d is a single variable, then \mathcal{A}_μ queries and returns the value of the variable. Else, \mathcal{A}_μ recursively evaluates a subtree of the root of g_d whose probability of evaluating to 0 is at least that of the other subtree.³ If the recursive call returns 0, \mathcal{A}_μ returns 1. Else, \mathcal{A}_μ recursively evaluates the other subtree of the root of g_d and returns the complement of the value returned by that recursive call. It is clear that on every input, \mathcal{A}_μ returns the correct answer with probability 1.

² Note that \mathcal{A}_μ needs the knowledge of μ .

³ By “recursively evaluates” we mean that \mathcal{A}_μ invokes $\mathcal{A}_{\mu'}$ for the distribution μ' induced by μ on the domain of the subfunction under consideration.

56:12 Randomized Query Composition and Product Distributions

Now we analyze the query complexity of \mathcal{A} . For $i \in \{\ell, r\}$, define $p_i := \Pr_{x^{(i)} \sim \mu_i}[g_d(x^{(i)}) = 0]$. WLOG assume that $p_\ell \geq p_r$. \mathcal{A}_μ on input x will recursively evaluate g_ℓ by invoking \mathcal{A}_{μ_ℓ} on input $x^{(\ell)}$. If the recursive call returns 1, then \mathcal{A} will recursively evaluate g_r by invoking \mathcal{A}_{μ_r} on input $x^{(r)}$. We thus have that,

$$T(d, \mu) = T(d-1, \mu_\ell) + (1 - p_\ell)T(d-1, \mu_r). \quad (10)$$

Let α_ℓ, α_r respectively be the probabilities that the left and right children of g_ℓ evaluate to 0. Similarly, let β_ℓ, β_r respectively be the probabilities that the left and right children of g_r evaluate to 0. Without loss of generality assume that $\alpha_\ell \geq \alpha_r, \beta_\ell \geq \beta_r$ and $\alpha_\ell \leq \beta_\ell$ (other cases are similar). By using a similar analysis as above and then upper bounding distributional query complexity for specific product distributions by the product distributional complexity we have that,

$$T(d-1, \mu_\ell) \leq T(d-2) + (1 - \alpha_\ell)T(d-2), \text{ and} \quad (11)$$

$$\begin{aligned} T(d-1, \mu_r) &\leq T(d-2) + (1 - \beta_\ell)T(d-2) \\ &\leq T(d-2) + (1 - \alpha_\ell)T(d-2). \end{aligned} \quad (12)$$

Substituting Equations (11) and (12) in (10) we have that

$$\begin{aligned} T(d) &\leq T(d, \mu) \\ &\leq (2 - \alpha_\ell)(2 - p_\ell)T(d-2). \end{aligned} \quad (13)$$

Now, we have that $p_\ell = (1 - \alpha_r)(1 - \alpha_\ell) \geq (1 - \alpha_\ell)^2$.⁴ Substituting in Equation (13) we have that

$$\begin{aligned} T(d) &\leq (2 - \alpha_\ell)(2 - (1 - \alpha_\ell)^2)T(d-2) \\ &= (2 - \alpha_\ell)(1 + 2\alpha_\ell - \alpha_\ell^2)T(d-2). \end{aligned} \quad (14)$$

The maximum value of the function $f(x) := (2 - x)(1 + 2x - x^2)$ in the domain $[0, 1]$ is $\frac{2}{27} \cdot (17 + 7\sqrt{7})$. From Equation (14) we have that

$$T(d) = O\left(\sqrt{\frac{2}{27} \cdot (17 + 7\sqrt{7})}\right)^d = O(\alpha - \delta)^d \text{ for some constant } \delta > 0. \quad \blacktriangleleft$$

5 Sabotage complexity of NAND tree

In this section, we prove Lemma 9. Recall that g_d stands for the NAND tree function of depth d . Define $g_0(b) = b$ for $b \in \{0, 1\}$.

For a randomized query algorithm \mathcal{R} that decides $g : \{0, 1\}^m \rightarrow \{0, 1\}$ with error probability 0, and for inputs x, y such that $g(x) = 0, g(y) = 1$, define the expected sabotage complexity of \mathcal{R} on the pair x, y , denoted by $\text{RS}_E(\mathcal{R}, x, y)$, to be the expected number of queries that \mathcal{R} makes until (and including) it queries an index i such that $x_i \neq y_i$ when run on x (or y). Define the expected sabotage complexity $\text{RS}_E(\mathcal{R})$ to be $\max_{x, y \in \{0, 1\}^m, g(x)=0, g(y)=1} \text{RS}_E(\mathcal{R}, x, y)$, and the expected sabotage complexity $\text{RS}_E(g)$ to be the minimum $\text{RS}_E(\mathcal{R})$ for any randomized query algorithm \mathcal{R} that decided g with error probability 0. As observed by Ben-David and Kothari, $\text{RS}_E(g) = \Theta(\text{RS}(g))$. In this section, we will work with RS_E in place of $\text{RS}(g)$.

⁴ Here we use that μ is a product distribution.

It follows by standard arguments that for every distribution \mathcal{D} on $g^{-1}(0) \times g^{-1}(1)$ there exists a zero-error randomized (even deterministic) decision tree \mathcal{R} of g such that $\mathbf{E}_{(x,y) \sim \mathcal{D}}[\text{RS}_E(|\mathcal{R}, x, y|)] \leq \text{RS}_E(g)$. To prove Lemma 9 it thus suffices to exhibit a hard distribution \mathcal{D} on $g^{-1}(0) \times g^{-1}(1)$ such that for every zero-error randomized tree \mathcal{R} of g , $\mathbf{E}_{(x,y) \sim \mathcal{D}}[\text{RS}_E(|\mathcal{R}, x, y|)]$ is large. The first step in our proof is to define a hard distribution.

A hard distribution

We define a probability distribution \mathcal{P}_d on $g_d^{-1}(0) \times g_d^{-1}(1)$ as follows. Define \mathcal{P}_0 to be the point distribution $\{(0, 1)\}$. For $d \geq 1$, \mathcal{P}_d is defined recursively by the following sampling procedure. Let $n := 2^{d-1}$.

1. Sample $(x, y) \sim \mathcal{P}_{d-1}$. Let $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$.
2. Sample $\bar{b} := (b_1, \dots, b_n)$ uniformly at random from $\{0, 1\}^n$.
3. For each $i = 1, \dots, n$, let $u_i = (u_i^{(0)}, u_i^{(1)})$, $v_i = (v_i^{(0)}, v_i^{(1)}) \in \{0, 1\}^2$ be defined as follows:
 - a. If $(x_i, y_i) = (0, 0)$, set $u_i, v_i \leftarrow (1, 1)$.
 - b. If $(x_i, y_i) = (0, 1)$, set $u_i \leftarrow (1, 1)$ and set $v_i \leftarrow (b_i, 1 - b_i)$.
 - c. If $(x_i, y_i) = (1, 0)$, set $u_i \leftarrow (b_i, 1 - b_i)$ and set $v_i \leftarrow (1, 1)$.
 - d. If $(x_i, y_i) = (1, 1)$, set $u_i, v_i \leftarrow (b_i, 1 - b_i)$.
4. Let x' be the string obtained from x by replacing each x_i by u_i . Similarly let y' be the string obtained from y by replacing each y_i by v_i .
5. Return (x', y') .

Notice that for each $i = 1, \dots, n$, $x_i = \text{NAND}(u_i^{(1)}, u_i^{(2)})$ and $y_i = \text{NAND}(v_i^{(1)}, v_i^{(2)})$. Hence, $g_d(x') = g_{d-1}(x)$ and $g_d(y') = g_{d-1}(y)$. Thus we inductively establish that \mathcal{P}_d is supported on $g_d^{-1}(0) \times g_d^{-1}(1)$. The following observation can be verified to be true by a simple case analysis.

► **Observation 15.** For each $i = 1, \dots, n$, $x_i = \overline{u_i^{(b_i)}}$ and $y_i = \overline{v_i^{(b_i)}}$. Furthermore, $u_i^{(1-b_i)} = \overline{v_i^{(1-b_i)}} = 1$.

In light of Observation 15, the sampling process above can be intuitively described as follows. We first sample (x, y) from \mathcal{P}_{d-1} . Then, for each i , we sample two two-bit strings u_i and v_i that are jointly distributed in a certain way. If $x_i = y_i$, then $u_i = v_i$. If $x_i \neq y_i$, then the values of x_i and y_i are embedded (as complements) in the b_i -th bits of u_i and v_i respectively. The $(1 - b_i)$ -th bit of u_i and v_i are set to 1. The marginals of \mathcal{P}_d can be seen to be obtained by conditioning uniform distribution on the “reluctant inputs” considered by Saks and Wigderson [14] to the events $g(x) = 0$ and $g(x) = 1$. We couple these two conditional distributions in a specific way to obtain \mathcal{P}_d .

A sequence of algorithms

Now we proceed to prove a lower bound on $\mathbf{E}_{(x,y) \sim \mathcal{P}_d}[\text{RS}_E(\mathcal{R}, x, y)]$ for any zero-error algorithm \mathcal{R} of g_d . Towards this goal, let \mathcal{R} be a zero-error randomized query algorithm for g_d . Now, using \mathcal{R} , we will define a sequence of randomized query algorithms $\mathcal{A}_d, \mathcal{A}_{d-1}, \dots, \mathcal{A}_1, \mathcal{A}_0$, where for each $t = d, d-1, \dots, 0$, \mathcal{A}_t is a zero-error randomized query algorithm for g_t . Define $\mathcal{A}_d := \mathcal{R}$. Now for $t \leq d-1$, define \mathcal{A}_t recursively as follows. Let $x = (x_1, \dots, x_{2^t})$ be the input to \mathcal{A}_t .

56:14 Randomized Query Composition and Product Distributions

1. Sample $\bar{b} = (b_1, \dots, b_{2^t})$ uniformly at random from $\{0, 1\}^{2^t}$.
2. For each $i = 1, \dots, 2^t$, define $u_i \in \{0, 1\}^{2^t}$ as in the definition of \mathcal{P}_d above. Let $x' \in \{0, 1\}^{2^{t+1}}$ be the string obtained from x by replacing each x_i by u_i .
3. Simulate \mathcal{A}_{t+1} on x' . If \mathcal{A}_{t+1} queries $u_i^{(1-b_i)}$ for some i , answer 1. If \mathcal{A}_{t+1} queries $u_i^{(b_i)}$ for some i , make a query to x_i and answer \bar{x}_i . The correctness of this simulation follows from Observation 15.
4. When \mathcal{A}_{t+1} terminates, terminate and return what \mathcal{A}_{t+1} returns.⁵

We observe that $g_t(x) = g_{t+1}(x')$. Thus we may inductively establish that for every $t = 1, \dots, d$, \mathcal{A}_t is a zero-error randomized decision tree of g_t . Moreover, observe that sampling (x, y) from \mathcal{P}_t and running \mathcal{A}_t on x (or y) amounts to sampling (x', y') from \mathcal{P}_{t+1} and running \mathcal{A}_{t+1} on x' (or y'). Furthermore, \mathcal{A}_t queries the first index i such that $x_i \neq y_i$ exactly when the simulation of \mathcal{A}_{t+1} inside it queries the first index j such that $x'_j \neq y'_j$. We will index the bits of x' as tuples (i, b) where $i \in \{1, \dots, 2^t\}$ and $b \in \{0, 1\}$. Thus $x'_{(i,b)} = u_i^{(b)}$.

The lower bound

For each $b \in \{0, 1\}$, each $t = 0, \dots, d$ and each (x, y) in the support of \mathcal{P}_t , define $Q(t, b, x, y)$ to be the number of variables with value b queried by \mathcal{A}_t when run on x until (and including) \mathcal{A}_t queries an index i such that $x_i \neq y_i$. Define $\mathbb{E}Q(t, b)$ to be the expected value of $Q(t, b, x, y)$ for a random sample (x, y) from \mathcal{P}_t , where the expectation is over both the internal randomness of \mathcal{A}_t , and the randomness of \mathcal{P}_t . Our goal is to derive a recursive relationship amongst the quantities $Q(t, b)$, and then obtain a lower bound on $Q(d, b)$. Since $\mathbf{E}[\text{RS}_E(\mathcal{R}, x, y)] = Q(d, 0) + Q(d, 1)$, the lemma will follow.

Let $0 \leq t \leq d$. For (x, y) in the support of \mathcal{P}_t and $i \in \{1, \dots, 2^t\}$ define $\mathbf{l}(t, b, i, x, y) := 1$ if $x_i = b$ and \mathcal{A}_t queries x_i when run on x not later than it queries an index on which x and y differ, and define $\mathbf{l}(t, b, i, x, y) := 0$ otherwise. We thus have that

$$Q(t, b, x, y) = \sum_{i=1}^{2^t} \mathbf{l}(t, b, i, x, y). \quad (15)$$

Consider $0 \leq t \leq d-1$, an (x, y) in the support of \mathcal{P}_t , bits $b, b' \in \{0, 1\}$ and $i \in \{1, \dots, 2^t\}$ such that $x_i = b$. We are interested in a lower bound on the quantity

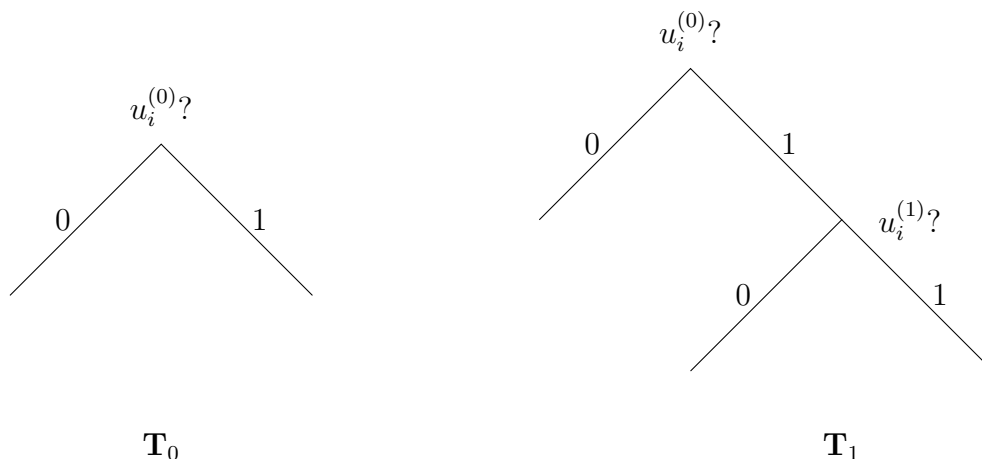
$$F(t, b, b', i, x, y) := \frac{\mathbf{E}[\mathbf{l}(t+1, b', (i, 0), x', y') + \mathbf{l}(t+1, b', (i, 1), x', y')]}{\mathbf{E}[\mathbf{l}(t, b, i, x, y)]}, \quad (16)$$

whenever the denominator is not 0. We now describe F in words. t, b, b', i, x and y are fixed. x_i is assumed to be b . The denominator is the probability that \mathcal{A}_t queries x_i not later than it queries an index where x and y differ. The numerator is the expected number of b' -valued variables in $\{u_i^{(0)}, u_i^{(1)}\}$ that is queried by the simulation of \mathcal{A}_{t+1} inside \mathcal{A}_t , not later than the simulation of \mathcal{A}_{t+1} queries an index where x' and y' differ (which, as discussed before, is exactly when \mathcal{A}_t queries an index where x and y differ). Both expectations are over the randomness of \mathcal{A}_t , which includes the sampling of $\bar{b} = (b_1, \dots, b_{2^t})$ and the randomness in \mathcal{A}_{t+1} that is simulated inside \mathcal{A}_t . Note that x' and y' are random strings, as they depend on b_1, \dots, b_{2^t} .

⁵ The return value is not important here. We are bothered only about separating x and y . The algorithms may be thought to have unlabelled leaves.

Lower bounding F

We wish to show a lower bound on $F(t, b, b', i, x, y)$. Towards this, let us fix a deterministic decision tree T in the support of \mathcal{A}_{t+1} . Furthermore, fix the values of all b_j for $j \neq i$. This fixes all the bits of the string x' except $u_i^{(0)}$ and $u_i^{(1)}$. Now, consider the expression for F where the expectations are conditioned on the above fixings, and are only over the randomness of b_i (notice that b_i determines $u_i^{(0)}, u_i^{(1)}$ and whether \mathcal{A}_t queries x_i). Under the above fixing, the action of T on the variables $u_i^{(0)}$ and $u_i^{(1)}$ before it queries an index where x' and y' differ is a deterministic decision tree on these two variables. We assume that the tree is not the empty tree (which in particular implies that T does not query an index where x' and y' differ before it queries any of $u_i^{(0)}$ and $u_i^{(1)}$). Assume further that if one of the two variables is queried and found to be 0, the other one is not queried (as their NAND is already fixed to 1, and so the value of g_d is insensitive to the value of the other variable). Under these assumptions there are only two structurally different trees on two variables. The two trees T_0 and T_1 are given below. Two other trees can be obtained by interchanging the roles of $u_i^{(0)}$ and $u_i^{(1)}$ in T_0 and T_1 . However, from the symmetry of the NAND function and our distributions, considering T_0 and T_1 suffices.



We now show how to bound F for $b = 1$ and $b' = 0$. Bounds for other combinations can be derived similarly; we list them in Table 1.

First consider tree T_0 . Assume that $x_i = b = 1$. Consider the denominator of F . A_t queries x_i if and only if T_0 queries $u_i^{(b_i)}$. T_0 queries only $u_i^{(0)}$. Thus, T_0 queries $u_i^{(b_i)}$ if and only if $b_i = 0$, which happens with probability $1/2$. Thus the denominator is $1/2$.

Now consider the numerator. Number of variables with value $b' = 0$ queried by T is 1 if $u_i^{(0)} = 0$ and 0 otherwise. $u_i^{(0)} = 0$ if and only if $b_i = 0$, which happens with probability $1/2$. Thus the numerator is $\frac{1}{2} \cdot 1 = 1/2$. Hence, in this case, $F = (1/2)/(1/2) = 1$.

Next, consider tree T_1 . In this case, x_i is guaranteed to be queried, as the tree always queries the variable whose value is 0. Thus, the denominator is 1. The numerator is also 1; exactly one of the two variables is $b' = 0$ and T_1 stops when it queries a 0. Thus, in this case too, $F = 1/1 = 1$.

We conclude that when $b = 1$ and $b' = 0$, a lower bound on F is $\min\{1, 1\} = 1$. The above analysis holds for a fixed T , as long as its restriction to $\{u_i^{(1)}, u_i^{(2)}\}$ until it queries an index where x' and y' differ, is not an empty tree. By averaging, the lower bounds in Table 1 hold for \mathcal{A}_{t+1} and a random b_1, \dots, b_{2^t} as long as with positive probability the aforementioned restricted tree is not empty.

■ **Table 1** Lower bounds on F .

b	b'	F
0	0	≥ 0
0	1	≥ 2
1	0	≥ 1
1	1	$\geq 1/2$

A recursive relation for $Q(t, b)$

Fix $0 \leq t \leq d-1$, inputs $x, y \in \{0, 1\}^{2^t}$ such that (x, y) is in the support of \mathcal{P}_t , and bit $b' \in \{0, 1\}$. Now, consider $Q(t+1, b', x', y')$. Note that x' and y' are random strings, and are determined by x, y (fixed) and b_1, \dots, b_{2^t} (random). We have that

$$Q(t+1, b', x', y') = \sum_{i=1}^{2^t} (\mathbf{l}(t+1, b', (i, 0), x', y') + \mathbf{l}(t+1, b', (i, 1), x', y')). \quad (17)$$

We split the above sum into two parts depending on x_i .

$$\begin{aligned} Q(t+1, b', x', y') &= \sum_{1 \leq i \leq 2^t, x_i=0} (\mathbf{l}(t+1, b', (i, 0), x', y') + \mathbf{l}(t+1, b', (i, 1), x', y')) \\ &\quad + \sum_{1 \leq i \leq 2^t, x_i=1} (\mathbf{l}(t+1, b', (i, 0), x', y') + \mathbf{l}(t+1, b', (i, 1), x', y')). \end{aligned} \quad (18)$$

Now, we take an expectation on both sides over b_1, \dots, b_{2^t} and the randomness of \mathcal{A}_{t+1} , and apply linearity of expectation.

$$\begin{aligned} \mathbf{E}[Q(t+1, b', x', y')] &= \sum_{1 \leq i \leq 2^t, x_i=0} \mathbf{E}[\mathbf{l}(t+1, b', (i, 0), x', y') + \mathbf{l}(t+1, b', (i, 1), x', y')] \\ &\quad + \sum_{1 \leq i \leq 2^t, x_i=1} \mathbf{E}[\mathbf{l}(t+1, b', (i, 0), x', y') + \mathbf{l}(t+1, b', (i, 1), x', y')]. \end{aligned} \quad (19)$$

Note that if $\mathbf{E}[\mathbf{l}(t, 0, i, x, y)]$ is not 0, then the summands of the first sum are $F(t, 0, b', i, x, y) \cdot \mathbf{E}[\mathbf{l}(t, 0, i, x, y)]$. A similar statement holds for the second sum. We thus have,

$$\begin{aligned} \mathbf{E}[Q(t+1, b', x', y')] &\geq \sum_{1 \leq i \leq 2^t, x_i=0, \mathbf{l}(t, 0, i, x, y) \neq 0} F(t, 0, b', i, x, y) \cdot \mathbf{E}[\mathbf{l}(t, 0, i, x, y)] \\ &\quad + \sum_{1 \leq i \leq 2^t, x_i=1, \mathbf{l}(t, 1, i, x, y) \neq 0} F(t, 1, b', i, x, y) \cdot \mathbf{E}[\mathbf{l}(t, 1, i, x, y)]. \end{aligned} \quad (20)$$

We would now like to consider $b' = 0$ and 1 separately, and plug the bounds of Table 1 into Equation (20). If $\mathbf{E}[\mathbf{l}(t, b, i, x, y)]$ is non-zero, then with positive probability, the restriction of the tree T considered earlier to variables $u_i^{(0)}, u_i^{(1)}$ is not the empty tree; thus the lower bounds of Table 1 are applicable. We thus have

$$\mathbf{E}[Q(t+1, 0, x', y')] \geq \mathbf{E}[Q(t, 1, x, y)], \text{ and} \quad (21)$$

$$\mathbf{E}[Q(t+1, 1, x', y')] \geq 2\mathbf{E}[Q(t, 0, x, y)] + \frac{1}{2}\mathbf{E}[Q(t, 1, x, y)]. \quad (22)$$

Finally, we take expectations over $(x, y) \sim \mathcal{P}_t$. As discussed before, this has the effect of inducing the distribution \mathcal{P}_{t+1} on (x', y') . We thus have

$$Q(t+1, 0) \geq Q(t, 1), \text{ and} \quad (23)$$

$$Q(t+1, 1) \geq 2Q(t, 0) + \frac{1}{2}Q(t, 1). \quad (24)$$

One can directly check by enumerating all deterministic zero-error trees for $t = 0, 1$ that $Q(0, 0), Q(0, 1), Q(1, 0)$ and $Q(1, 1)$ are all $\Omega(1)$. It thus follows from Equations (23) and (24) that $Q(t, b) = \Omega(\alpha^t)$ for $b \in \{0, 1\}$. In particular, $Q(d, 0), Q(d, 1) = \Omega(\alpha^d)$. This completes the proof of Lemma 9.

References

- 1 Anurag Anshu, Dmitry Gavinsky, Rahul Jain, Srijita Kundu, Troy Lee, Priyanka Mukhopadhyay, Miklos Santha, and Swagato Sanyal. A composition theorem for randomized query complexity. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, page 1, 2018.
- 2 Andrew Bassilakis, Andrew Drucker, Mika Göös, Lunjia Hu, Weiyun Ma, and Li-Yang Tan. The power of many samples in query complexity. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 3 Shalev Ben-David and Eric Blais. A tight composition theorem for the randomized query complexity of partial functions. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 240–246. IEEE, 2020.
- 4 Shalev Ben-David, Eric Blais, Mika Göös, and Gilbert Maystre. Randomised composition and small-bias minimax. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 624–635. IEEE, 2022.
- 5 Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar, Swagato Sanyal, and Nitin Saurabh. On the composition of randomized query complexity and approximate degree. In Nicole Megow and Adam D. Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 63:1–63:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- 6 Dmitry Gavinsky, Troy Lee, Miklos Santha, and Swagato Sanyal. A composition theorem for randomized query complexity via max-conflict complexity. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 64:1–64:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 7 Mika Göös, TS Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication versus partition number. *ACM Transactions on Computation Theory (TOCT)*, 10(1):1–20, 2018.
- 8 Prahladh Harsha, Rahul Jain, and Jaikumar Radhakrishnan. Partition bound is quadratically tight for product distributions. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 9 Rahul Jain, Hartmut Klauck, Srijita Kundu, Troy Lee, Miklos Santha, Swagato Sanyal, and Jevgēnijs Vihrovs. Quadratically tight relations for randomized query complexity. *Theory of Computing Systems*, 64(1):101–119, 2020.
- 10 Gillat Kol. Interactive compression for product distributions. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 987–998, 2016.
- 11 Ashley Montanaro. A composition theorem for decision tree complexity. *Chicago Journal Of Theoretical Computer Science*, 6:1–8, 2014.
- 12 Ryan O’Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

- 13 Judea Pearl. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *Communications of the ACM*, 25(8):559–564, 1982.
- 14 Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 29–38. IEEE, 1986.
- 15 M Santha. On the monte carlo boolean decision tree complexity of read-once formulae. In *1991 Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, pages 180–181. IEEE Computer Society, 1991.
- 16 Ben-David Shalev and Robin Kothari. Randomized query complexity of sabotaged and composed functions. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 17 Clifford Smyth. Reimer’s inequality and tardos’ conjecture. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 218–221, 2002.
- 18 Marc Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38:69–82, 1985.
- 19 Avishay Tal. Properties and applications of boolean function composition. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 441–454, 2013.

A Extended preliminaries

The notation $[n]$ denotes the set $\{1, \dots, n\}$. Throughout the paper, $g : \{0, 1\}^m \rightarrow \{0, 1\}$ will stand for a Boolean function and $x = (x_1, \dots, x_m)$ will stand for a generic input to g . For $b \in \{0, 1\}$, $f^{-1}(b) = \{x \in \{0, 1\}^n \mid f(x) = b\}$. For a subset S of $\{0, 1\}^m$, let $f|_S$ denote the restriction of f to S . A probability distribution μ over $\{0, 1\}^m$ is a function $\mu : \{0, 1\}^m \rightarrow [0, 1]$ such that $\sum_{x \in \{0, 1\}^m} \mu(x) = 1$. For a subset S of $\{0, 1\}^m$, define $\mu(S) := \sum_{x \in S} \mu(x)$. For a subset S of $\{0, 1\}^m$ such that $\mu(S) > 0$, $\mu|_S$ is the distribution obtained by conditioning μ on the event that the sample belongs to S . In other words:

$$\mu|_S(x) = \begin{cases} 0 & \text{if } x \notin S, \\ \frac{\mu(x)}{\mu(S)} & \text{if } x \in S \end{cases}$$

μ is said to be a product distribution if there exist $p_1, \dots, p_m \in [0, 1]$ such that for each $x \in \{0, 1\}^n$, $\mu(x) = \prod_{i=1}^m (x_i p_i + (1 - x_i)(1 - p_i))$. In other words, each x_i is independently equal to 1 with probability p_i and 0 with probability $1 - p_i$. Let PROD be the set of all product distributions of $\{0, 1\}^m$.

For a subset $I \in [m]$ of indices, $x^{\oplus I}$ denotes the string obtained from x by flipping the variables x_i for each $i \in I$. If $I = \{i\}$, we abuse notation and write $x^{\oplus i}$.

► **Definition 16 (Subcube).** A subset C of $\{0, 1\}^m$ is called a subcube if there exists a set $S \in [m]$ of indices and bits $\{b_i \mid i \in S\}$ such that $C = \{x \in \{0, 1\}^m \mid \forall i \in S, x_i = b_i\}$. The co-dimension of C is defined to be $|S|$.

A.1 Decision trees for Boolean functions

A decision tree for m variables is a binary tree T . Each internal node of T is labelled by a variable x_i for $i \in [m]$, and has two children that corresponds to $x_i = 0$ and $x_i = 1$. Each leaf is labelled by 0 or 1. A decision tree is evaluated on a given input $x = (x_1, \dots, x_m)$, as follows. Start at the root. In each step, if the current node is an internal node, then query its label x_i . Then navigate to that child of the current node that corresponds to the value of x_i . The computation stops when it reaches a leaf, and outputs the label of the leaf. Let $T(x)$ denotes the output of the tree at x .

The inputs x that take the tree T to leaf ℓ is exactly the ones which agree with the path from the root to ℓ for every variable queried on the path. Thus, the set of such inputs is a subcube of $\{0, 1\}^m$ of co-dimension equal to the depth of ℓ . The notation ℓ will also refer to the subcube corresponding to the leaf ℓ .

T is said to compute $g : \{0, 1\}^m \rightarrow \{0, 1\}$ if

$$\forall x \in \{0, 1\}^m, T(x) = g(x).$$

The *Deterministic Decision Tree complexity* of g , denoted by $D(g)$ is the minimum depth of a decision tree that computes f .

Let μ be a distribution over $\{0, 1\}^m$. For a given error parameter $\epsilon \in [0, 1/2]$, T computes g with error probability ϵ over μ if

$$\Pr_{x \sim \mu} [g(x) \neq T(x)] \leq \epsilon.$$

The *distributional query complexity* of g for error ϵ with respect to μ , denoted by $D_\epsilon^\mu(g)$, is the minimum depth of a decision tree that computes f with error probability ϵ over μ .

A randomized decision tree is a probability distribution \mathcal{R} over deterministic decision trees. \mathcal{R} is said to compute g with error probability ϵ if

$$\forall x \in \{0, 1\}^m, \Pr_{T \sim \mathcal{R}} [T(x) \neq g(x)] \leq \epsilon.$$

The query complexity of \mathcal{R} is the maximum depth of any decision tree in its support. The *randomized query complexity* of g for error ϵ , denoted by $R_\epsilon(g)$, is the minimum query complexity of any randomized decision tree \mathcal{R} that computes g with error ϵ . Define $R(g) := R_{1/3}(g)$. The following fact is well-known (see, for example [6] for a proof).

► **Fact 17** (Minimax theorem). $R_\epsilon(g) = \max_\mu D_\epsilon^\mu(g)$.

We define the *product distributional query complexity* of g with error ϵ , $D_\epsilon^{\text{prod}}(g)$, as follows.

$$D_\epsilon^{\text{prod}}(g) := \max_{\mu \in \text{PROD}} D_\epsilon^\mu(g).$$

B Proof of Claim 11

In this section we prove Claim 11.

Proof of Claim 11. Let \mathcal{R} be a randomized decision tree that achieves $R_\epsilon^{\text{prod}}(g)$. Fix a product distribution μ . From \mathcal{R} , we will construct a deterministic decision tree (with labelled leaves) T' that errs with probability at most ϵ with respect to μ . This will complete the proof.

To this end, consider any deterministic decision tree T (with unlabelled leaves) in the support of \mathcal{R} . We label each leaf ℓ of T as follows. Condition μ on ℓ (assume that the conditional probability is defined; otherwise label ℓ arbitrarily). If the probability of the event “ $g(x) = 1$ ” with respect to this conditional distribution is at least $1/2$, we label ℓ as 1. Else, we label ℓ as 0.

In this way we label each leaf of each deterministic decision tree in the support of \mathcal{R} . By the guarantee of \mathcal{R} , the resulting randomized decision tree (with labelled leaves) computes g on inputs from μ with error at most ϵ .

Finally, by averaging, it follows that there exists a deterministic tree T' in the support of \mathcal{R} which computes g on a random $x \sim \mu$ with error probability at most ϵ . ◁

Shortest Two Disjoint Paths in Conservative Graphs

Ildikó Schlotter  

Centre for Economic and Regional Studies, Budapest, Hungary
Budapest University of Technology and Economics, Hungary

Abstract

We consider the following problem that we call the SHORTEST TWO DISJOINT PATHS problem: given an undirected graph $G = (V, E)$ with edge weights $w : E \rightarrow \mathbb{R}$, two terminals s and t in G , find two internally vertex-disjoint paths between s and t with minimum total weight. As shown recently by Schlotter and Sebő (2022), this problem becomes NP-hard if edges can have negative weights, even if the weight function is conservative, i.e., there are no cycles in G with negative total weight. We propose a polynomial-time algorithm that solves the SHORTEST TWO DISJOINT PATHS problem for conservative weights in the case when the negative-weight edges form a constant number of trees in G .

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Shortest paths; Theory of computation \rightarrow Dynamic programming

Keywords and phrases Shortest paths, disjoint paths, conservative weights, graph algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.57

Related Version *Full Version*: <https://arxiv.org/abs/2307.12602> [12]

Funding *Ildikó Schlotter*: supported by the Hungarian Academy of Sciences under its Momentum Programme (LP2021-2) and its János Bolyai Research Scholarship.

1 Introduction

Finding disjoint paths between given terminals is a fundamental problem in algorithmic graph theory and combinatorial optimization. Besides its theoretical importance, it is also motivated by numerous applications in transportation, VLSI design, and network routing. In the DISJOINT PATHS problem, we are given k terminal pairs (s_i, t_i) for $i \in \{1, \dots, k\}$ in an undirected graph G , and the task is to find pairwise vertex-disjoint paths P_1, \dots, P_k so that P_i connects s_i with t_i for each $i \in \{1, \dots, k\}$. This problem was shown to be NP-hard by Karp [7] when k is part of the input, and remains NP-hard even on planar graphs [10]. Robertson and Seymour [11] proved that there exists an $f(k)n^3$ algorithm for DISJOINT PATHS with k terminal pairs, where n is the number of vertices in G and f some computable function; this celebrated result is among the most important achievements of graph minor theory. In the SHORTEST DISJOINT PATHS problem we additionally require that P_1, \dots, P_k have minimum total length (in terms of the number of edges). For fixed k , the complexity of this problem is one of the most important open questions in the area. Even the case for $k = 2$ had been open for a long time, until Björklund and Husfeldt [3] gave a randomized polynomial-time algorithm for it in 2019. For directed graphs the problem becomes much harder: the DIRECTED DISJOINT PATHS problem is NP-hard already for $k = 2$. The DISJOINT PATHS problem and its variants have also received considerable attention when restricted to planar graphs [6, 5, 8, 1, 4, 14, 9].

The variant of DISJOINT PATHS when $s_1 = \dots = s_k = s$ and $t_1 = \dots = t_k = t$ is considerably easier, since one can find k pairwise (openly vertex- or edge-) disjoint paths between s and t using a max-flow computation. Applying standard techniques for computing



© Ildikó Schlotter;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 57; pp. 57:1–57:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



a minimum-cost flow (see e.g. [15]), one can even find k pairwise disjoint paths between s and t with minimum total weight, given non-negative weights on the edges. Notice that if negative weights are allowed, then flow techniques break down for undirected graphs: in order to construct an appropriate flow network based on our undirected graph G , the standard technique is to direct each edge of G in both directions; however, if edges can have negative weight, then this operation creates negative cycles consisting of two arcs, an obstacle for computing a minimum-cost flow. Recently, Schlotter and Sebő [13] have shown that this issue is a manifestation of a complexity barrier: finding two openly disjoint paths with minimum total weight between two vertices in an undirected edge-weighted graph is NP-hard, even if weights are *conservative* (i.e., no cycle has negative total weight) and each edge has weight in $\{-1, 1\}$.¹ Note that negative edge weights occur in network problems due to various reasons: for example, they might arise as a result of some reduction (e.g., deciding the feasibility of certain scheduling problems with deadlines translates into finding negative-weight cycles), or as a result of data that is represented on a logarithmic scale. We remark that the SINGLE-SOURCE SHORTEST PATHS problem is the subject of active research for the case when negative edges are allowed; see Bernstein et al. [2] for an overview of the area and their state-of-the-art algorithm running in near-linear time on directed graphs.

Our contribution

We consider the following problem which concerns finding paths between two fixed terminals (as opposed to the classic SHORTEST DISJOINT PATHS problem):

SHORTEST TWO DISJOINT PATHS:

Input: An undirected graph $G = (V, E)$, a weight function $w: E \rightarrow \mathbb{R}$ that is conservative on G , and two vertices s and t in G .

Task: Find two paths P_1 and P_2 between s and t with $V(P_1) \cap V(P_2) = \{s, t\}$ that minimizes $w(P_1) + w(P_2)$.

A *solution* for an instance (G, w, s, t) of SHORTEST TWO DISJOINT PATHS is a pair of (s, t) -paths that are openly disjoint, i.e., do not share vertices other than their endpoints.

From the NP-hardness proof for SHORTEST TWO DISJOINT PATHS by Schlotter and Sebő [13] it follows that the problem remains NP-hard even if the set of negative-weight edges forms a perfect matching. Motivated by this intractability, we focus on the “opposite” case when the subgraph of G spanned by the set $E^- = \{e \in E : w(e) < 0\}$ of negative-weight edges, denoted by $G[E^-]$, has only few connected components.² Note that since w is conservative on G , the graph $G[E^-]$ is acyclic. Hence, if c denotes the number of connected components in $G[E^-]$, then $G[E^-]$ in fact consists of c trees.

We can think of our assumption that c is constant as a compromise for allowing negative-weight edges but requiring that they be confined to a small part of the graph. For a motivation, consider a network where negative-weight edges arise as some rare anomaly. Such an anomaly may occur when, in a certain part of a computer network, some information can be collected while traversing the given edge. If such information concerns, e.g., the detection of (possibly) faulty nodes or edges in the network, then it is not unreasonable to assume that these faults are concentrated to a certain part of the network, due to underlying physical causes that are responsible for the fault.

¹ In fact, Schlotter and Sebő use an equivalent formulation of the problem where, instead of finding two openly disjoint paths between s and t , the task is to find two vertex-disjoint paths between $\{s_1, s_2\}$ and $\{t_1, t_2\}$ for four vertices $s_1, s_2, t_1, t_2 \in V$.

² See Section 2 for the precise definition of a subgraph spanned by an edge set.

Ideally, one would aim for an algorithm that is fixed-parameter tractable (FPT) when parameterized by c ; however, already the case $c = 1$ turns out to be challenging. We prove the following result, which can be thought of as a first step towards an FPT algorithm:

► **Theorem 1.** *For each constant $c \in \mathbb{N}$, SHORTEST TWO DISJOINT PATHS can be solved in polynomial time on instances where the set of negative edges spans c trees in G .*

Our algorithm first applies standard flow techniques to find minimum-weight solutions among those that have a simple structure in the sense that there is no negative tree in $G[E^-]$ used by both paths. To deal with more complex solutions where there is at least one tree T in $G[E^-]$ used by both paths, we use recursion to find two openly disjoint paths from s to T , and from T to t ; to deal with the subpaths of the solution that heavily use negative edges from T , we apply an intricate dynamic programming method that is based on significant insight into the structural properties of such solutions.

Organization

We give all necessary definitions in Section 2. In Section 3 we make initial observations about optimal solutions for an instance (G, w, s, t) of SHORTEST TWO DISJOINT PATHS, and we also present a lemma of key importance that will enable us to create solutions by combining partial solutions that are easier to find (Lemma 10). We present the algorithm proving our main result, Theorem 1, in Section 4. In Section 4.1 we give a general description of our algorithm, and explain which types of solutions can be found using flow-based techniques. We proceed in Section 4.2 by establishing structural observation that we need to exploit in order to find those types of solutions where more advanced techniques are necessary. Section 4.3 contains our dynamic programming method for finding partial solutions which, together with Lemma 10, form the heart of our algorithm. We conclude with some questions for further research in Section 5. All proofs are deferred to the full version of our paper [12].

2 Notation

For a positive integer ℓ , we use $[\ell] := \{1, 2, \dots, \ell\}$.

Let a graph G be a pair (V, E) where V and E are the set of vertices and edges, respectively. For two vertices u and v in V , an edge connecting u and v is denoted by uv or vu .

For a set X of vertices (or edges), let $G - X$ denote the subgraph of G obtained by deleting the vertices (or edges, respectively) of X ; if $X = \{x\}$ then we may simply write $G - x$ instead of $G - \{x\}$. Given a set $F \subseteq E$ of edges in G , we denote by $V(F)$ the vertices incident to some edge of F . The subgraph of G spanned by F is the graph $(V(F), F)$; we denote this subgraph as $G[F]$.

A *walk* W in G is a series e_1, e_2, \dots, e_ℓ of edges in G for which there exist vertices v_0, v_1, \dots, v_ℓ in G such that $e_i = v_{i-1}v_i$ for each $i \in [\ell]$; note that both vertices and edges may appear repeatedly on a walk. We denote by $V(W)$ the set of vertices *contained by* or *appearing on* W , that is, $V(W) = \{v_0, v_1, \dots, v_\ell\}$. The *endpoints* of W are v_0 and v_ℓ , or in other words, it is a (v_0, v_ℓ) -walk, while all vertices on W that are not endpoints are *inner vertices*. If $v_0 = v_\ell$, then we say that W is a *closed walk*.

A *path* is a walk on which no vertex appears more than once. By a slight abuse of notation, we will usually treat a path as a *set* $\{e_1, e_2, \dots, e_\ell\}$ of edges for which there exist distinct vertices v_0, v_1, \dots, v_ℓ in G such that $e_i = v_{i-1}v_i$ for each $i \in [\ell]$. For any i and j with $0 \leq i \leq j \leq \ell$ we will write $P[v_i, v_j]$ for the *subpath* of P between v_i and v_j , consisting of edges e_{i+1}, \dots, e_j . Note that since we associate no direction with P , we have

$P[v_i, v_j] = P[v_j, v_i]$. Given two vertices s and t , an (s, t) -path is a path whose endpoints are s and t . Similarly, for two subsets S and T of vertices, an (S, T) -path is a path with one endpoint in S and the other endpoint in T .

We say that two paths are *vertex-* or *edge-disjoint*, if they do not share a common vertex or edge, respectively. Two paths are *openly disjoint*, if they share no common vertices apart from possibly their endpoints. Given vertices s_1, s_2, t_1 , and t_2 , we say that two $(\{s_1, s_2\}, \{t_1, t_2\})$ -paths are *permissively disjoint*, if a vertex v can only appear on both paths if either $v = s_1 = s_2$ or $v = t_1 = t_2$. Two paths properly intersect, if they share at least one edge, but neither is the subpath of the other.

A *cycle* in G is a set $\{e_1, e_2, \dots, e_\ell\}$ of distinct edges in G such that $e_1, e_2, \dots, e_{\ell-1}$ form a path in $G - e_\ell$ whose endpoints are connected by e_ℓ . A set $T \subseteq E$ of edges in G is *connected*, if for every pair of edges e and e' in T , there is a path contained in T containing both e and e' . If T is connected and *acyclic*, i.e., contains no cycle, then T is a *tree* in G . Given two vertices a and b in a tree T , we denote by $T[a, b]$ the unique path contained in T whose endpoints are a and b . For an edge $uv \in T$ and a path P within T such that $uv \notin P$, we say that v is *closer to P* in T than u , if $v \in V(T[u, p])$ for some vertex $p \in V(P)$.

Given a weight function $w: E \rightarrow \mathbb{R}$ on the edge set of G , we define the *weight* of any edge set $F \subseteq E$ as $w(F) = \sum_{e \in F} w(e)$. We extend this notion for any pair $\mathcal{F} = (F_1, F_2)$ of edge sets by letting $w(\mathcal{F}) = w(F_1) + w(F_2)$. The restriction of w to an edge set $F \subseteq E$, i.e., the function whose domain is F and has value $w(f)$ on each $f \in F$, is denoted by $w|_F$. We say that w (or, to make the dependency on G explicit, the weighted graph (G, w)) is *conservative*, if no cycle in G has negative total weight.

3 Structural Observations

Let $G = (V, E)$ be an undirected graph with a conservative weight function $w: E \rightarrow \mathbb{R}$. Let $E^- = \{e \in E : w(e) < 0\}$ denote the set of negative edges, and \mathcal{T} the set of negative trees they form. More precisely, let \mathcal{T} be the set of connected components in the subgraph $G[E^-]$; the acyclicity of each $T \in \mathcal{T}$ follows from the conservativeness of w . For any subset \mathcal{T}' of \mathcal{T} , we use the notation $E(\mathcal{T}') = \bigcup_{T \in \mathcal{T}'} E(T)$ and $V(\mathcal{T}') = \bigcup_{T \in \mathcal{T}'} V(T)$.

In Section 3.1 we gather a few useful properties of conservative weight functions. In Section 3.2 we collect observations on how an optimal solution can use different trees in \mathcal{T} . We close the section with a lemma of key importance in Section 3.3 that enables us to compose solutions by combining two path pairs without violating our requirement of disjointness.

3.1 Implications of Conservative Weights

The next two lemmas establish implications of the conservativeness of our weight function. Lemma 2 concerns closed walks, while Lemma 3 considers paths running between two vertices on some negative tree in \mathcal{T} . These lemmas will be useful in proofs where a given hypothetical solution is “edited” – by removing certain subpaths from it and replacing them with paths within some negative tree – in order to obtain a specific form without increasing its weight.

► **Lemma 2.** *If W is a closed walk that does not contain any edge with negative weight more than once, then $w(W) \geq 0$.*

► **Lemma 3.** *Let x, y, x', y' be four distinct vertices on a tree T in \mathcal{T} .*

(1) *If Q is an (x, y) -walk in G using each edge of E^- at most once, then $w(Q) \geq w(T[x, y])$.*

(2) *If Q is an (x, y) -path and Q' is an (x', y') -path vertex-disjoint from Q , then*

$$w(Q) + w(Q') \geq w(T[x, y] \setminus T[x', y']) + w(T[x', y'] \setminus T[x, y]).$$

3.2 Solution Structure on Negative Trees

We first observe a simple property of minimum-weight solutions.

► **Definition 4 (Locally cheapest path pairs).** *Let s_1, s_2, t_1, t_2 be vertices in G , and let P_1 and P_2 be two permissively disjoint $(\{s_1, s_2\}, \{t_1, t_2\})$ -paths. A path $T[u, v]$ in some $T \in \mathcal{T}$ is called a shortcut for P_1 and P_2 , if u and v both appear on the same path, either P_1 or P_2 , and there is no inner vertex or edge of $T[u, v]$ contained in $P_1 \cup P_2$. We will call P_1 and P_2 locally cheapest, if there is no shortcut for them.*

The idea behind this concept is the following. Suppose that P_1 and P_2 are permissively disjoint $(\{s_1, s_2\}, \{t_1, t_2\})$ -paths, and $T[u, v]$ is a shortcut for P_1 and P_2 . Suppose that u and v both lie on P_i (for some $i \in [2]$), and let P'_i be the path obtained by replacing $P_i[u, v]$ with $T[u, v]$; we refer to this operation as *amending* the shortcut $T[u, v]$. Then P'_i is also permissively disjoint from P_{3-i} and, since Lemma 2 implies $w(P'_i[u, v]) \geq -w(T[u, v]) > 0$, has weight less than $w(P_i)$. Hence, we have the following observation.

► **Observation 5.** *Let P_1 and P_2 be two permissively disjoint $(\{s_1, s_2\}, \{t_1, t_2\})$ -paths admitting a shortcut $T[z, z']$. Suppose that z and z' are on the path, say, P_1 . Let P'_1 be the path obtained by amending $T[z, z']$ on P_1 . Then P'_1 and P_2 are permissively disjoint $(\{s_1, s_2\}, \{t_1, t_2\})$ -paths and $w(P'_1) < w(P_1)$.*

► **Corollary 6.** *Any minimum-weight solution for (G, w, s, t) is a pair of locally cheapest paths.*

For convenience, for any (s, t) -path P and vertices $u, v \in V(P)$ we say that u precedes v on P , or equivalently, v follows u on P , if u lies on $P[s, v]$. When defining a vertex as the “first” (or “last”) vertex with some property on P or on a subpath P' of P then, unless otherwise stated, we mean the vertex on P or on P' that is closest to s (or farthest from s , respectively) that has the given property.

The following lemma shows that if two paths in a minimum-weight solution both use negative trees T and T' for some $T, T' \in \mathcal{T}$ then, roughly speaking, they must traverse T and T' in the same order; otherwise one would be able to replace the subpaths of the solution running between T and T' by two paths, one within T and one within T' , of smaller weight.

► **Lemma 7.** *Let P_1 and P_2 be two openly disjoint (s, t) -paths of minimum total weight, and let T and T' be distinct trees in \mathcal{T} . Suppose that v_1, v_2, v'_1 and v'_2 are vertices such that $v_i \in V(T) \cap V(P_i)$ and $v'_i \in V(T') \cap V(P_i)$ for $i \in [2]$, with v_1 preceding v'_1 on P_1 . Then v_2 precedes v'_2 on P_2 .*

The following lemma is a consequence of Corollary 6 and Lemma 7, and considers a situation when one of the paths in an optimal solution visits a negative tree $T \in \mathcal{T}$ at least twice, and visits some $T' \in \mathcal{T} \setminus \{T\}$ in between.

► **Lemma 8.** *Let P_1 and P_2 be two openly disjoint (s, t) -paths of minimum total weight, and let T and T' be distinct trees in \mathcal{T} . Suppose that v_1, v'_2 , and v_3 are vertices appearing in this order on P_1 when traversed from s to t , and suppose $v_1, v_3 \in V(T)$ while $v'_2 \in V(T')$. Then*

- $V(P_2) \cap V(T) \neq \emptyset$;
- $V(P_2) \cap V(T') = \emptyset$;
- no vertex of $V(P_1) \cap V(T')$ precedes v_1 or follows v_3 on P_1 .

We say that two paths P_1 and P_2 are *in contact* at T , if there is a tree $T \in \mathcal{T}$ and two distinct vertices v_1 and v_2 in T such that v_1 lies on P_1 , and v_2 lies on P_2 . Lemmas 7 and 8 imply the following fact that will enable us to use recursion in our algorithm to find solutions that consist of two paths in contact.

► **Lemma 9.** *Let P_1 and P_2 be two openly disjoint (s, t) -paths of minimum total weight, and assume that they are in contact at some tree $T \in \mathcal{T}$. For $i \in [2]$, let a_i and b_i denote the first and last vertices of P_i on T when traversed from s to t . Then we can partition $\mathcal{T} \setminus \{T\}$ into $(\mathcal{T}_s, \mathcal{T}_0, \mathcal{T}_t)$ such that for each $T' \in \mathcal{T} \setminus \{T\}$ and $i \in [2]$ it holds that*

- (i) *if $P_i[s, a_i]$ contains a vertex of T' , then $T' \in \mathcal{T}_s$*
- (ii) *if $P_i[b_i, t]$ contains a vertex of T' , then $T' \in \mathcal{T}_t$*
- (iii) *if $P_i[a_i, b_i]$ contains a vertex of T' , then $T' \in \mathcal{T}_0$.*

Given a solution (P_1, P_2) whose paths are in contact at some tree $T \in \mathcal{T}$, a partition of $\mathcal{T} \setminus \{T\}$ is T -valid with respect to (P_1, P_2) , if it satisfies the conditions of Lemma 9.

3.3 Combining Path Pairs

The following lemma will be a crucial ingredient in our algorithm, as it enables us to combine “partial solutions” without violating the requirement of vertex-disjointness.

► **Lemma 10.** *Let p_1, p_2, q_1, q_2 be vertices in G , and let $T \in \mathcal{T}$ contain vertices v_1 and v_2 with $v_1 \neq v_2$. Let P_1 and P_2 be two permissively disjoint $(\{p_1, p_2\}, \{v_1, v_2\})$ -paths in G , and let Q_1 and Q_2 be two permissively disjoint $(\{v_1, v_2\}, \{q_1, q_2\})$ -paths in G that are locally cheapest. Assume also that we can partition \mathcal{T} into two sets \mathcal{T}_1 and \mathcal{T}_2 with $T \in \mathcal{T}_2$ such that*

- (i) *$V(T) \cap V(P_1 \cup P_2) = \{v_1, v_2\}$, and*
- (ii) *$P_1 \cup P_2$ contains no edge of $E(\mathcal{T}_2)$, and $Q_1 \cup Q_2$ contains no edge of $E(\mathcal{T}_1)$.*

Then we can find in linear time two permissively disjoint $(\{p_1, p_2\}, \{q_1, q_2\})$ -paths S_1 and S_2 in G such that $w(S_1) + w(S_2) \leq w(P_1) + w(P_2) + w(Q_1) + w(Q_2)$.

We remark that Lemma 10 heavily relies on the condition that v_1 and v_2 are both on T : to construct the desired paths S_1 and S_2 , we not only use the path pairs (P_1, P_2) and (Q_1, Q_2) but, if necessary, remove certain subpaths from them and stitch together the remainder with a path running within T .

4 Polynomial-Time Algorithm for Constant $|\mathcal{T}|$

This section contains the algorithm proving our main result, Theorem 1. Let (G, w, s, t) be our instance of SHORTEST TWO DISJOINT PATHS with input graph $G = (V, E)$, and assume that the set E^- of negative edges spans c trees in G for some constant c . We present a polynomial-time algorithm that computes a solution for (G, w, s, t) with minimum total weight, or correctly concludes that no solution exists for (G, w, s, t) . The running time of our algorithm is $O(n^{2c+9})$ where $n = |V|$, so in the language of parameterized complexity, our algorithm is in XP with respect to the parameter c .

In Section 4.1 we present the main ideas and definitions necessary for our algorithm, and provide its high-level description together with some further details. We will distinguish between so-called *separable* and *non-separable* solutions. Finding an optimal and separable solution will be relatively easy, requiring extensive guessing but only standard techniques for computing minimum-cost flows. By contrast, finding an optimal but non-separable solution is much more difficult. Therefore, in Section 4.2 we collect useful properties of optimal, non-separable solutions. These observations form the basis for an important subroutine necessary for finding optimal, non-separable solutions; this subroutine is presented in Section 4.3.

4.1 The Algorithm

We distinguish between two types of solutions for our instance (G, w, s, t) of SHORTEST TWO DISJOINT PATHS.

► **Definition 11 (Separable solution).** Let (P_1, P_2) be a solution for (G, w, s, t) . We say that P_1 and P_2 are separable, if either

- they are not in contact, i.e., there is no tree $T \in \mathcal{T}$ that shares distinct vertices with both P_1 and P_2 , or
- there is a unique tree $T \in \mathcal{T}$ such that P_1 and P_2 are in contact at T , but the intersection of T with both P_1 and P_2 is a path, possibly containing only a single vertex;

otherwise they are non-separable.

In Section 4.1.1 we show how to find an optimal, separable solution, whenever such a solution exists for (G, w, s, t) . Section 4.1.2 deals with the case when we need to find an optimal, non-separable solution. Our algorithm for the latter case is more involved, and relies on a subroutine that is based on dynamic programming and is developed throughout Sections 4.2 and 4.3. The existence of this subroutine is stated in Corollary 33.

4.1.1 Finding Separable Solutions

Suppose that (P_1, P_2) is a minimum-weight solution for (G, w, s, t) that is separable. The following definition establishes conditions when we are able to apply a simple strategy for finding a minimum-weight solution using well-known flow techniques.

► **Definition 12 (Strongly separable solution).** Let (P_1, P_2) be a solution for (G, w, s, t) . We say that P_1 and P_2 are strongly separable, if they are separable and they are either not in contact at any tree of \mathcal{T} , or they are contact at a tree of \mathcal{T} that contains s or t .

Suppose that P_1 and P_2 are either not in contact, or they are in contact at a tree of \mathcal{T} that contains s or t . Let $\mathcal{T}^{\neq s, t}$ denote the set of all trees in \mathcal{T} that contain neither s nor t . For each $T \in \mathcal{T}^{\neq s, t}$ that shares a vertex with P_i for some $i \in [2]$, we define a_T as the first vertex on P_i (when traversed from s to t) that is contained in T ; note that T cannot share vertices with both P_1 and P_2 as they are strongly separable, so P_i is uniquely defined.

Our approach is the following: we guess the vertex a_T for each $T \in \mathcal{T}^{\neq s, t}$, and then compute a minimum-cost flow in an appropriately defined network. More precisely, for each possible choice of vertices $Z = \{z_T \in V(T) : T \in \mathcal{T}^{\neq s, t}\}$, we build a network N_Z as follows.

► **Definition 13 (Flow network N_Z for strongly separable solutions).** Given a set $Z \subseteq V$ such that $Z \cap V(T) = \{z_T\}$ for each $T \in \mathcal{T}^{\neq s, t}$, we create N_Z as follows. We direct each non-negative edge in G in both directions. Then for each $T \in \mathcal{T}^{\neq s, t}$, we direct the edges of T away from z_T .³ If some $T \in \mathcal{T}$ contains s , then we direct all edges of T away from s ; similarly, if some $T \in \mathcal{T}$ contains t , then we direct all edges of T towards t . We assign a capacity of 1 to each arc and to each vertex⁴ in the network except for s and t , and we retain the cost function w (meaning that we define $w(\vec{e})$ as $w(e)$ for any arc \vec{e} obtained by directing some edge e). We let s and t be the source and the sink in N_Z , respectively.

³ Directing a tree T away from a vertex $z \in V(T)$ means that an edge uv in T becomes an arc (u, v) if and only if $T[z, u]$ has fewer edges than $T[z, v]$; directing T towards z is defined analogously.

⁴ The standard network flow model can be adjusted by well-known techniques to allow for vertex capacities.

► **Lemma 14.** *If there exists a strongly separable solution for (G, w, s, t) with weight k , then there exists a flow of value 2 having cost k in the network N_Z for some choice of $Z \subseteq V$ containing exactly one vertex from each tree in $\mathcal{T}^{\neq s, t}$. Conversely, a flow of value 2 and cost k in the constructed network N_Z for some Z yields a solution for (G, w, s, t) with weight at most k .*

Next, we show how to deal with the case when a minimum-weight solution (P_1, P_2) is separable, but not strongly separable; then there is a unique tree $T \in \mathcal{T}$ at which P_1 and P_2 are in contact. In such a case, we can simply delete an edge from T in a way that paths P_1 and P_2 cease to be in contact in the resulting instance. This way, we can reduce our problem to the case when there is a strongly separable optimal solution; note, however, that the number of trees spanned by the negative edges (our parameter c) increases by 1.

► **Lemma 15.** *If there exists a separable, but not strongly separable solution for (G, w, s, t) with weight k , then there exists an edge $e \in E^-$ such that setting $G' = G - e$ and $E' = E \setminus \{e\}$, the instance $(G', w|_{E'}, s, t)$ admits a strongly separable solution with weight at most k . Conversely, a solution for $(G', w|_{E'}, s, t)$ is also a solution for (G, w, s, t) with the same weight.*

Thanks to Lemmas 14 and 15, if there exists a minimum-weight solution for (G, w, s, t) that is separable, then we can find some minimum-weight solution using standard algorithms for computing minimum-cost flows. In Section 4.1.2 we explain how we can find a minimum-weight non-separable solution for (G, w, s, t) .

4.1.2 Finding Non-separable Solutions

To find a minimum-weight solution for (G, w, s, t) that is not separable, we need a more involved approach. We now provide a high-level presentation of our algorithm for finding a non-separable solution of minimum weight. We remark that Algorithm STDP contains a pseudocode; however, we believe that it is best to read the following description first.

Step 1. We guess certain properties of a minimum-weight non-separable solution (P_1, P_2) :

First, we guess a tree $T \in \mathcal{T}$ that shares distinct vertices both with P_1 and P_2 , i.e., a tree at which P_1 and P_2 are in contact. Second, we guess a partition $(\mathcal{T}_s, \mathcal{T}_0, \mathcal{T}_t)$ of $\mathcal{T} \setminus \{T\}$ that is T -valid with respect to (P_1, P_2) .

Step 2. If $\mathcal{T}_s \neq \emptyset$, then we guess the first vertex of P_1 and of P_2 contained in $V(T)$, denoted by a_1 and a_2 , respectively. We use recursion to compute two permissively disjoint $(s, \{a_1, a_2\})$ -paths using only the negative trees in \mathcal{T}_s , and to compute two permissively disjoint $(\{a_1, a_2\}, t)$ -paths using only the negative trees in $\mathcal{T} \setminus \mathcal{T}_s$. Observe that by $\mathcal{T}_s \neq \emptyset$ and $T \in \mathcal{T} \setminus \mathcal{T}_s$, we search for these paths in graphs that contain only a strict subset of the negative trees in \mathcal{T} . Thus, our parameter c strictly decreases in both constructed sub-instances. We combine the obtained pairs of paths into a solution by using Lemma 10.

We proceed in a similar fashion when $\mathcal{T}_t \neq \emptyset$.

Step 3. If $\mathcal{T}_s = \mathcal{T}_t = \emptyset$, then for both $i \in [2]$ we guess the first and last vertex of P_i contained in $V(T)$, denoted by a_i and b_i , respectively. We apply standard flow techniques to compute two $(s, \{a_1, a_2\})$ -paths and two $(\{b_1, b_2\}, t)$ -paths with no inner vertices in $V(T)$ that are pairwise permissively disjoint. Then, we apply the polynomial-time algorithm we devise for computing a pair of permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths in G . This algorithm is the cornerstone of our method, and is based on important structural observations that allow for efficient dynamic programming. We combine the obtained pairs of paths into a solution by applying Lemma 10.

Step 4. We output a solution of minimum weight among all solutions found in Steps 2 and 3.

Let us now provide more details about these steps; see also Algorithm STDP.

Step 1: Initial guesses on \mathcal{T}

There are $c = |\mathcal{T}|$ possibilities to choose T from \mathcal{T} , and there are 3^{c-1} further possibilities to partition $\mathcal{T} \setminus \{T\}$ into $(\mathcal{T}_s, \mathcal{T}_0, \mathcal{T}_t)$, yielding $c3^{c-1}$ possibilities in total for our guesses.

Step 2: Applying recursion

To apply recursion in Step 2 when $\mathcal{T}_s \neq \emptyset$, we guess two distinct vertices a_1 and a_2 in T , and define the following sub-instances.

- **Definition 16 (Sub-instances for $\mathcal{T}_s \neq \emptyset$).** For some $\emptyset \neq \mathcal{T}_s \subseteq \mathcal{T}$ and distinct vertices a_1 and a_2 , we create a graph G_s by adding a new vertex a^* , and connecting it to both a_1 and a_2 with an edge of weight $w_{a^*} = |w(T[a_1, a_2])|/2$. We create instances I_s^1 and I_s^2 as follows:
- to get I_s^1 , we delete $E(\mathcal{T} \setminus \mathcal{T}_s)$ and $V(\mathcal{T} \setminus \mathcal{T}_s) \setminus \{a_1, a_2\}$ from G_s , and designate s and a^* as our two terminals;
 - to get I_s^2 , we delete $V(\mathcal{T}_s)$ from G_s , and designate a^* and t as our two terminals.

We use recursion to solve SHORTEST TWO DISJOINT PATHS on sub-instances I_s^1 and I_s^2 . Note that conservativeness is maintained for both I_s^1 and I_s^2 , due to our choice of w_{a^*} and statement (1) of Lemma 3. If both sub-instances admit solutions, we obtain two permissively disjoint $(s, \{a_1, a_2\})$ -paths Q_1^{\nearrow} and Q_2^{\nearrow} from our solution to I_s^1 by deleting the vertex a^* . Similarly, we obtain two permissively disjoint $(\{a_1, a_2\}, t)$ -paths Q_1^{\searrow} and Q_2^{\searrow} from our solution to I_s^2 by deleting the vertex a^* . Next we use Lemma 10 to create a solution for our original instance (G, w, s, t) using paths $Q_1^{\nearrow}, Q_2^{\nearrow}, Q_1^{\searrow}$, and Q_2^{\searrow} .

If $\mathcal{T}_t \neq \emptyset$, then we proceed similarly: after guessing two distinct vertices b_1 and b_2 in T , we use a construction analogous to Definition 16.

- **Definition 17 (Sub-instances for $\mathcal{T}_t \neq \emptyset$).** For some $\emptyset \neq \mathcal{T}_t \subseteq \mathcal{T}$ and distinct vertices b_1 and b_2 , we create a graph G_t by adding a new vertex b^* , and connecting it to both b_1 and b_2 with an edge of weight $w_{b^*} = |w(T[b_1, b_2])|/2$. We create instances I_t^1 and I_t^2 as follows:
- to get I_t^1 , we delete $V(\mathcal{T}_t)$ from G_t , and designate s and b^* as our two terminals;
 - to get I_t^2 , we delete $E(\mathcal{T} \setminus \mathcal{T}_t)$ and $V(\mathcal{T} \setminus \mathcal{T}_t) \setminus \{b_1, b_2\}$ from G_t and designate b^* and t as our two terminals.

We use recursion to solve SHORTEST TWO DISJOINT PATHS on sub-instances I_t^1 and I_t^2 ; again, conservativeness is ensured for I_t^1 and I_t^2 by our choice of w_{b^*} and statement (1) of Lemma 3. If both sub-instances admit solutions, we obtain two permissively disjoint $(s, \{b_1, b_2\})$ -paths Q_1^{\nearrow} and Q_2^{\nearrow} from our solution to I_t^1 by deleting the vertex b^* . Similarly, we obtain two permissively disjoint $(\{b_1, b_2\}, t)$ -paths Q_1^{\searrow} and Q_2^{\searrow} from our solution to I_t^2 by deleting the vertex b^* . Again, we use Lemma 10 to create a solution for our original instance (G, w, s, t) using paths $Q_1^{\nearrow}, Q_2^{\nearrow}, Q_1^{\searrow}$, and Q_2^{\searrow} .

We state the correctness of Step 2 in the following lemma:

- **Lemma 18.** Suppose that (P_1, P_2) is a minimum-weight solution for (G, w, s, t) such that
- P_1 and P_2 are in contact at some $T \in \mathcal{T}$,
 - a_i and b_i are the first and last vertices of P_i contained in T , respectively, for $i \in [2]$, and
 - $(\mathcal{T}_s, \mathcal{T}_0, \mathcal{T}_t)$ is a T -valid partition w.r.t. (P_1, P_2) .

Then instances I_s^1 and I_s^2 admit solutions whose total weight (summed over all four paths) is $w(P_1) + w(P_2) + 4w_{a^*}$. Furthermore, given a solution \mathcal{S}_i for I_s^i for both $i \in [2]$, we can compute in linear time a solution for (G, w, s, t) of weight at most $w(\mathcal{S}_1) + w(\mathcal{S}_2) - 4w_{a^*}$. The same holds when substituting I_s^1, I_s^2 , and w_{a^*} with I_t^1, I_t^2 , and w_{b^*} in these claims.

■ **Algorithm STDP** Solving SHORTEST TWO DISJOINT PATHS with conservative weights.

Input: An instance (G, w, s, t) where w is conservative on G .

Output: A solution for (G, w, s, t) with minimum weight, or \emptyset if no solution exist.

```

1: Let  $\mathcal{S} = \emptyset$ .
2: for all  $E' \subseteq E$  such that  $E \setminus E' \subseteq E^-$  and  $|E \setminus E'| \leq 1$  do      ▷ Separable solutions.
3:   Create the instance  $(G[E'], w|_{E'}, s, t)$ .
4:   Let  $\mathcal{T}^{\neq s, t} = \{T : T \text{ is a maximal tree in } G[E' \cap E^-] \text{ with } s, t \notin V(T)\}$ .
5:   for all  $Z \subseteq V$  such that  $|Z \cap V(T)| = 1$  for each  $T \in \mathcal{T}^{\neq s, t}$  do
6:     Create the network  $N_Z$  from instance  $(G[E'], w|_{E'}, s, t)$ .      ▷ Use Def. 13.
7:     if  $\exists$  a flow  $f$  of value 2 in  $N_Z$  then
8:       Compute a minimum-cost flow  $f$  of value 2 in  $N_Z$ .
9:       Construct a solution  $(S_1, S_2)$  from  $f$  using Lemma 14.
10:       $\mathcal{S} \leftarrow (S_1, S_2)$ .
11: for all  $T \in \mathcal{T}$  do      ▷ Non-separable solutions.
12:   for all partitions  $(\mathcal{T}_s, \mathcal{T}_0, \mathcal{T}_t)$  of  $\mathcal{T} \setminus \{T\}$  do
13:     if  $\mathcal{T}_s \neq \emptyset$  then
14:       for all  $a_1, a_2 \in V(T)$  with  $a_1 \neq a_2$  do
15:         Create sub-instances  $I_s^1$  and  $I_s^2$ .      ▷ Use Def. 16.
16:         Compute  $(Q_1^\rightarrow, Q_2^\rightarrow) = \text{STDP}(I_s^1)$ .
17:         Compute  $(Q_1^\leftarrow, Q_2^\leftarrow) = \text{STDP}(I_s^2)$ .
18:         if  $(Q_1^\rightarrow, Q_2^\rightarrow) \neq \emptyset$  and  $(Q_1^\leftarrow, Q_2^\leftarrow) \neq \emptyset$  then
19:           Create a solution  $(S_1, S_2)$  from  $(Q_1^\rightarrow, Q_2^\rightarrow)$  and  $(Q_1^\leftarrow, Q_2^\leftarrow)$  using Lemma 18.
20:            $\mathcal{S} \leftarrow (S_1, S_2)$ .
21:       else if  $\mathcal{T}_t \neq \emptyset$  then
22:         for all  $b_1, b_2 \in V(T)$  with  $b_1 \neq b_2$  do
23:           Create sub-instances  $I_t^1$  and  $I_t^2$ .      ▷ Use Def. 17.
24:           Compute  $(Q_1^\rightarrow, Q_2^\rightarrow) = \text{STDP}(I_t^1)$ .
25:           Compute  $(Q_1^\leftarrow, Q_2^\leftarrow) = \text{STDP}(I_t^2)$ .
26:           if  $(Q_1^\rightarrow, Q_2^\rightarrow) \neq \emptyset$  and  $(Q_1^\leftarrow, Q_2^\leftarrow) \neq \emptyset$  then
27:             Create a solution  $(S_1, S_2)$  from  $(Q_1^\rightarrow, Q_2^\rightarrow)$  and  $(Q_1^\leftarrow, Q_2^\leftarrow)$  using Lemma 18.
28:              $\mathcal{S} \leftarrow (S_1, S_2)$ .
29:       else      ▷  $\mathcal{T}_s = \mathcal{T}_t = \emptyset$ .
30:         for all  $a_1, a_2, b_1, b_2 \in V(T)$  that constitute a reasonable guess do
31:           if  $\exists$  a flow  $f$  of value 4 in  $N_{(a_1, b_1, a_2, b_2)}$  then      ▷ Use Def. 19.
32:             if  $\exists$  two permissively disjoint  $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths in  $G$  then
33:               Compute a minimum-cost flow  $f$  of value 4 in  $N_{(a_1, b_1, a_2, b_2)}$ .
34:               Compute permissively disjoint  $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths  $Q_1$  and  $Q_2$ .
35:               ▷ Use Corollary 33 in Section 4.3
36:               Construct a solution  $(S_1, S_2)$  from  $f$ ,  $Q_1$ , and  $Q_2$  using Lemma 20.
37:                $\mathcal{S} \leftarrow (S_1, S_2)$ .
38: if  $\mathcal{S} = \emptyset$  then return  $\emptyset$ .
39: else Let  $S^*$  be the cheapest pair among those in  $\mathcal{S}$ , and return  $S^*$ .

```

Step 3: Applying flow techniques and dynamic programming

We now describe Step 3 in more detail, which concerns the case when $\mathcal{T}_s = \mathcal{T}_t = \emptyset$.

First, we guess vertices a_1, b_1, a_2 , and b_2 ; the intended meaning of these vertices is that a_i and b_i are the first and last vertices of P_i contained in $V(T)$, for both $i \in [2]$. We only consider guesses that are *reasonable*, meaning that they satisfy the following conditions:

- if $s \in V(T)$, then $s = a_1 = a_2$, otherwise $a_1 \neq a_2$;
- if $t \in V(T)$, then $t = b_1 = b_2$, otherwise $b_1 \neq b_2$;
- $T[a_1, b_1]$ and $T[a_2, b_2]$ share at least one edge.

Then we compute four paths from $\{s, t\}$ to $\{a_1, b_1, a_2, b_2\}$ in the graph $G - E(T) - (V(T) \setminus \{a_1, a_2, b_1, b_2\})$ with minimum weight such that two paths have s as an endpoint, the other two have t as an endpoint, and no other vertex appears on more than one path. To this end, we define the following network and compute a minimum-cost flow of value 4 in it.

► **Definition 19 (Flow network $N_{(a_1, b_1, a_2, b_2)}$ for non-separable solutions.)** *Given vertices a_1, b_1, a_2 , and b_2 , we create $N_{(a_1, b_1, a_2, b_2)}$ as follows. First, we delete all edges in E^- and all vertices in $V(T) \setminus \{a_1, a_2, b_1, b_2\}$ from G , and direct each edge in G in both directions. We then add new vertices s^* and t^* , along with arcs (s^*, s) and (s^*, t) of capacity 2, and arcs (a_i, t^*) and (b_i, t^*) for $i = 1, 2$ with capacity 1.⁵ We assign capacity 1 to all other arcs, and also to each vertex of $V(G) \setminus \{s, t\}$. All newly added arcs will have cost 0, otherwise we retain the cost function w . We let s^* and t^* be the source and the sink in $N_{(a_1, b_1, a_2, b_2)}$, respectively.*

Next, we compute two $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths Q_1 and Q_2 in G with minimum total weight that are permissively disjoint. A polynomial-time computation for this problem, building on structural observations from Section 4.2, is provided in Section 4.3 (see Corollary 33). Finally, we apply Lemma 10 (in fact, twice) to obtain a solution to our instance (G, w, s, t) based on a minimum-cost flow of value 4 in $N_{(a_1, b_1, a_2, b_2)}$ and paths Q_1 and Q_2 . We finish this section with Lemma 20, stating the correctness of Step 3.

- **Lemma 20.** *Suppose that (P_1, P_2) is a minimum-weight solution for (G, w, s, t) such that*
- P_1 and P_2 are non-separable, and in contact at some $T \in \mathcal{T}$,
 - a_i and b_i are the first and last vertices of P_i contained in T , respectively, for $i \in [2]$, and
 - $(\emptyset, \mathcal{T} \setminus \{T\}, \emptyset)$ is a T -valid partition w.r.t. (P_1, P_2) .

Let Q_1 and Q_2 be two permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths in G with minimum total weight. Then the following holds:

If w^ is the minimum cost of a flow of value 4 in the network $N_{(a_1, b_1, a_2, b_2)}$, then $w^* + w(Q_1) + w(Q_2) \leq w(P_1) + w(P_2)$. Conversely, given a flow of value 4 in the network $N_{(a_1, b_1, a_2, b_2)}$ with cost w^* , together with paths Q_1 and Q_2 , we can find a solution for (G, w, s, t) with cost at most $w^* + w(Q_1) + w(Q_2)$ in linear time.*

4.2 Properties of a Non-separable Solution

Let us now turn our attention to the subroutine lying at the heart of our algorithm for SHORTEST TWO DISJOINT PATHS: an algorithm that, given two source terminals and two sink terminals on some tree $T \in \mathcal{T}$, computes two permissively disjoint paths from the two source terminals to the two sink terminals, with minimum total weight. It is straightforward to see that any non-separable solution whose paths are in contact at T contains such a pair

⁵ In the degenerate case when $s = a_1 = a_2$ or $t = b_1 = b_2$ this yields two parallel arcs from s or t to t^* .

of paths. Therefore, as described in Section 4.1.2, finding such paths is a necessary step to computing an optimal, non-separable solution for our instance (G, w, s, t) . This section contains observations about the properties of such paths.

Let us now formalize our setting. Let a_1, a_2, b_1 , and b_2 be vertices on a fixed tree $T \in \mathcal{T}$ such that $T[a_1, b_1]$ and $T[a_2, b_2]$ intersect in a path X (with at least one edge), with one component of $T \setminus X$ containing a_1 and a_2 , and the other containing b_1 and b_2 . Let the vertices on X be x_1, \dots, x_r with x_1 being the closest to a_1 and a_2 . We will use the notation $A_i = T[a_i, x_1]$ and $B_i = T[b_i, x_r]$ for each $i \in [2]$. For each $i \in [r]$, let T_i be the maximal subtree of T containing x_i but no other vertex of X . We also define $T_{(i,j)} = \bigcup_{i \leq h \leq j} T_h$ for some i and j with $1 \leq i \leq j \leq r$.

For convenience, for any path Q that has $a_i \in \{a_1, a_2\}$ as its endpoint, we will say that Q starts at a_i and ends at its other endpoint. Accordingly, for vertices $u, v \in V(Q)$ we say that u precedes v on Q , or equivalently, v follows u on Q , if u lies on $Q[a_i, v]$. When defining a vertex as the “first” (or “last”) vertex with some property on Q or on a subpath Q' of Q then, unless otherwise stated, we mean the vertex on Q or on Q' that is closest to a_i (or farthest from a_i , respectively) that has the given property.

Using Lemma 2 and (an extended version of) Lemma 3, we can establish the following properties of a non-separable minimum-weight solution.

► **Definition 21 (X-monotone path).** A path Q starting at a_1 or a_2 is X -monotone if for any vertices u_1 and u_2 on Q such that $u_1 \in V(T_{j_1})$ and $u_2 \in V(T_{j_2})$ for some $j_1 < j_2$ it holds that u_1 precedes u_2 on Q .

► **Definition 22 (Plain path).** A path Q is plain, if whenever Q contains some $x_i \in V(X)$, then the vertices of Q in T_i induce a path in T_i . In other words, if vertices $x_j \in V(X)$ and $u \in V(T_j)$ both appear on Q , then $T[u, x_j] \subseteq Q$.

► **Lemma 23.** Let Q_1 and Q_2 be two permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths in G with minimum total weight. Then both Q_1 and Q_2 are X -monotone and plain.

The following observation summarizes our understanding on how an optimal solution uses paths A_1, A_2, B_1 , and B_2 .

► **Lemma 24.** If Q_1 and Q_2 are two permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths that are locally cheapest and also plain, then one of them contains A_1 or A_2 , and one of them contains B_1 or B_2 .

4.3 Computing Partial Solutions

In this section we design a dynamic programming algorithm that computes two permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths of minimum total weight (we keep all definitions introduced in Section 4.2, including our assumptions on vertices a_1, b_1, a_2 , and b_2). In Section 4.2 we have established that two permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths of minimum total weight are necessarily X -monotone, plain, and they form a locally cheapest pair. A natural approach would be to require these same properties from a partial solution that we aim to compute. However, it turns out that the property of X -monotonicity is quite hard to ensure when building subpaths of a solution. The following relaxed version of monotonicity can be satisfied much easier, and still suffices for our purposes:

► **Definition 25 (Quasi-monotone path).** A path P starting at a_1 or a_2 is quasi-monotone, if the following holds: if $x_i \in V(P)$ for some $i \in [r]$, then all vertices in $\bigcup_{h \in [i-1]} V(T_h) \cap V(P)$ precede x_i on P , and all vertices in $\bigcup_{h \in [r] \setminus [i]} V(T_h) \cap V(P)$ follow x_i on P .

► **Definition 26 (Well-formed path pair).** Two paths P_1 and P_2 form a well-formed pair, if they are locally cheapest, and both are plain and quasi-monotone.

We are now ready to define partial solutions, the central notion that our dynamic programming algorithm relies on.

► **Definition 27 (Partial solution).** Given vertices $u \in V(T_i)$ and $v \in V(T_j)$ for some $i \leq j$ and a set $\tau \subseteq \mathcal{T} \setminus \{T\}$, two paths Q_1 and Q_2 form a partial solution (Q_1, Q_2) for (u, v, τ) , if

- (a) Q_1 and Q_2 are permissively disjoint $(\{a_1, a_2\}, \{u, v\})$ -paths;
- (b) Q_1 and Q_2 are a well-formed pair;
- (c) Q_1 ends with the subpath $T[x_i, u]$;
- (d) $V(T_{(i+1, r)}) \cap V(Q_2) \subseteq \{v\}$;
- (e) if $Q_1 \cup Q_2$ contains a vertex of some $T' \in \mathcal{T} \setminus \{T\}$, then $T' \in \tau$;
- (f) there exists no tree $T' \in \mathcal{T} \setminus \{T\}$ such that Q_1 and Q_2 are in contact at T' .

We will say that the vertices of $V(\mathcal{T} \setminus (\tau \cup \{T\}))$ are *forbidden* for (τ, T) ; then condition (e) asks for $Q_1 \cup Q_2$ not to contain vertices forbidden for (τ, T) .

Before turning our attention to the problem of computing partial solutions, let us first show how partial solutions enable us to find two permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths.

► **Lemma 28.** Paths P_1 and P_2 are permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths of minimum weight in G if and only if they form a partial solution for $(b_h, b_{3-h}, \mathcal{T} \setminus \{T\})$ of minimum weight for some $h \in [2]$.

We now present our approach for computing partial solutions using dynamic programming.

Computing partial solutions: high-level view

For each $u \in V(T_i)$ and $v \in V(T_j)$ for some $i \leq j$, and each $\tau \subseteq \mathcal{T} \setminus \{T\}$, we are going to compute a partial solution for (u, v, τ) of minimum weight, denoted by $F(u, v, \tau)$, using dynamic programming; if there exists no partial solution for (u, v, τ) , we set $F(u, v, \tau) = \emptyset$.

To apply dynamic programming, we fix an ordering \prec over $V(T)$ fulfilling the condition that for each $i' < i$, $u \in V(T_i)$ and $u' \in V(T_{i'})$ we have $u' \prec u$. We compute the values $F(u, v, \tau)$ based on the ordering \prec in the sense that $F(u', v', \tau')$ is computed before $F(u, v, \tau)$ whenever $u' \prec u$. This computation is performed by Algorithm PARTSOL which determines a partial solution $F(u, v, \tau)$ based on partial solutions already computed.

To compute $F(u, v, \tau)$ in a recursive manner, we use an observation that either the partial solution has a fairly simple structure, or it strictly contains a partial solution for (u', v', τ') for some vertices u' and v' with $u' \in V(T_{i'})$ and $i' < i$, and some set $\tau' \subseteq \tau$. We can thus try all possible values for u', v' and τ' , and use the partial solution (Q'_1, Q'_2) we have already computed and stored in $F(u', v', \tau')$. To obtain a partial solution for (u, v, τ) based on Q'_1 and Q'_2 , we append paths to Q'_1 and to Q'_2 so that they fulfill the requirements of Definition 27 – most importantly, that Q_1 ends with $T[x_i, u]$, that Q_2 ends at v , and that $Q_1 \cup Q_2$ contains no vertex of $V(\mathcal{T} \setminus (\tau \cup \{T\}))$. To this end, we create a path $P_1 = Q'_2 \cup T[v', u]$ and a path $P_2 = Q'_1 \cup R$ where R is a shortest (u', v) -path in a certain auxiliary graph. Essentially, we use the tree T for getting from v' to u , and we use the “remainder” of the graph for getting from u' to v ; note that we need to avoid the forbidden vertices and ensure condition (f) as well. The precise definition of the auxiliary subgraph of G that we use for this purpose is provided in Definition 29. If the obtained path pair (P_1, P_2) is indeed a partial solution for (u, v, τ) , then we store it. After trying all possible values for u', v' , and τ' , we select a partial solution that has minimum weight among those we computed.

► **Definition 29 (Auxiliary graph).** For some $T \in \mathcal{T}$, let $P \subseteq T$ be a path within T , let u and v be two vertices on T , and let $\tau \subseteq \mathcal{T} \setminus \{T\}$. Then the auxiliary graph $G\langle P, u, v, \tau \rangle$ denotes the graph defined as

$$G\langle P, u, v, \tau \rangle = G - \left(\bigcup \{V(T_h) : V(T_h) \cap V(P) = \emptyset\} \cup V(P) \setminus \{u, v\} \cup V(\mathcal{T} \setminus (\tau \cup \{T\})) \right).$$

In other words, we obtain $G\langle P, u, v, \tau \rangle$ from G by deleting all trees T_h that do not intersect P , and deleting P itself as well, while taking care not to delete u or v , and additionally deleting all vertices forbidden for (τ, T) .

Working towards explaining the main ideas behind Algorithm PARTSOL, we start with two simple observations. The first one, stated by Lemma 30 below, essentially says that a path in a partial solution that uses a subtree T_h of T for some $h \in [r]$ should also go through the vertex x_h whenever possible, that is, unless the other path uses x_h .

► **Lemma 30.** Let Q_1 and Q_2 be two permissively disjoint, locally cheapest $(\{a_1, a_2\}, \{x_i, v\})$ -paths for some $v \in V(T_j)$ where $1 \leq i \leq j \leq r$. Let $z \in V(T_h)$ for some $h \leq j$ such that $h < j$ or $x_h \in V(T[z, v])$. If $z \in V(Q_1 \cup Q_2)$, then $x_h \in V(Q_1 \cup Q_2)$.

As a consequence of Lemma 30, applied with a_1 taking the role of z and x_1 taking the role of x_h , we get that every partial solution for some (u, v, τ) must contain x_1 ; using that both paths in a partial solution must be plain, we get the following fact.

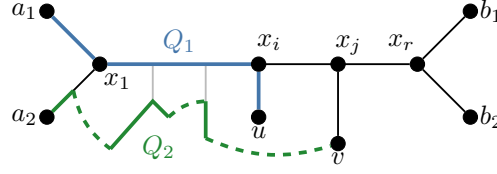
► **Observation 31.** Let (Q_1, Q_2) be a partial solution for (u, v, τ) for vertices $u \in V(T_i)$ and $v \in V(T_j)$ for some $i \leq j$ and for $\tau \subseteq \mathcal{T} \setminus \{T\}$. Then either Q_1 or Q_2 contains A_1 or A_2 .

Let us now give some insight on Algorithm PARTSOL that computes a minimum-weight partial solution (Q_1, Q_2) for (u, v, τ) , if it exists, for vertices $u \in V(T_i)$ and $v \in V(T_j)$ with $i \leq j$ and trees $\tau \subseteq \mathcal{T} \setminus \{T\}$. It distinguishes between two cases based on whether Q_2 contains a vertex of $T[x_1, x_i]$ or not; see Figure 1 for an illustration. In both cases it constructs candidates for a partial solution, and then chooses one among these with minimum weight.

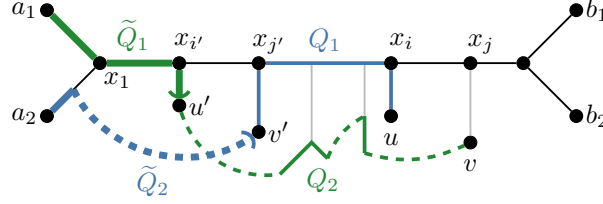
Case A: Q_2 does not contain any vertices from $T[x_1, x_i]$. In this case, due to Observation 31, we know that Q_1 contains A_h for some $h \in [2]$; let us fix this value of h . Since Q_1 and Q_2 are locally cheapest and Q_1 ends with $T[x_i, u]$, we also know that Q_1 must contain $T[x_1, x_i]$. Therefore, we obtain $Q_1 = A_h \cup T[x_1, u]$. In this case, we can also prove that Q_2 is a shortest (a_{3-h}, v) -path in the auxiliary graph $G\langle A_h \cup T[x_1, u], a_{3-h}, v, \tau \rangle$. Hence, Algorithm PARTSOL computes such a path R and constructs the pair $(A_h \cup T[x_1, u], R)$ as a candidate for a partial solution for (u, v, τ) .

Case B: Q_2 contains a vertex from $T[x_1, x_i]$. In this case, let $x_{i'}$ be the vertex on $T[x_1, x_{i-1}]$ closest to x_i that appears on Q_2 , and let u' be the last vertex of Q_2 in $T_{i'}$; since Q_2 is plain, we know $T[x_{i'}, u'] \subseteq Q_2$. Let $x_{j'}$ denote the vertex on $T[x_{i'}, x_i]$ closest to $x_{i'}$ that appears on Q_1 ; then $i' < j' \leq i$. As Q_1 and Q_2 are locally cheapest, $T[x_{j'}, x_i] \subseteq Q_1$ follows. Let v' denote the first vertex of Q_1 in $T_{j'}$. Since Q_1 is plain, we know $T[v', x_{j'}] \subseteq Q_1$.

Define $\tilde{Q}_1 = Q_2 \setminus Q_2[u', v]$ and $\tilde{Q}_2 = Q_1 \setminus Q_1[v', u]$. Let also τ' denote those trees in $\mathcal{T} \setminus \{T\}$ that share a vertex with $\tilde{Q}_1 \cup \tilde{Q}_2$. We can then prove that $(\tilde{Q}_1, \tilde{Q}_2)$ is a partial solution for (u', v', τ') ; moreover, $Q_2[u', v]$ is a path in the auxiliary graph $G\langle T[v', u], u', v, \tau \setminus \tau' \rangle$. Thus, Algorithm PARTSOL takes a partial solution (Q'_1, Q'_2) for (u', v', τ') , already computed, and computes a shortest (u', v) -path R in $G\langle T[v', u], u', v, \tau \setminus \tau' \rangle$. It then creates the path pair $(Q'_2 \cup T[v', u], Q'_1 \cup R)$ as a candidate for a partial solution for (u, v, τ) .



(a) Case A. The figure assumes $Q_1 = A_1 \cup T[x_1, u]$ (so $h = 1$).



(b) Case B. The subpaths \tilde{Q}_2 and \tilde{Q}_1 of Q_1 and Q_2 , respectively, form a partial solution for (u', v', τ') and are depicted in bold, with their endings marked by a parenthesis-shaped delimiter.

■ **Figure 1** Illustration for Algorithm PARTSOL. Edges within T are depicted using solid lines, edges not in T using dashed lines. Paths Q_1 and Q_2 are shown in blue and in green, respectively (see the online version for colored figures).

■ **Algorithm PartSol** Computes a partial solution $F(u, v, \tau)$ of minimum weight for (u, v, τ) where $u \in V(T_i)$ and $v \in V(T_j)$ with $i \leq j$, and $\tau \subseteq \mathcal{T} \setminus \{T\}$.

Input: Vertices u and v where $u \in V(T_i)$ and $v \in V(T_j)$ for some $i \leq j$, and a set $\tau \subseteq \mathcal{T} \setminus \{T\}$.

Output: A partial solution $F(u, v, \tau)$ for (u, v, τ) of minimum weight, or \emptyset if not existent.

- 1: Let $\mathcal{S} = \emptyset$.
- 2: **for all** $h \in [2]$ **do**
- 3: **if** $A_h \cup T[x_1, u]$ is a path **then**
- 4: **if** v is reachable from a_{3-h} in $G\langle A_h \cup T[x_1, u], a_{3-h}, v, \tau \rangle$ **then**
- 5: Compute a shortest (a_{3-h}, v) -path R in $G\langle A_h \cup T[x_1, u], u, a_{3-h}, v, \tau \rangle$.
- 6: **if** $(A_h \cup T[x_1, u], R)$ is a partial solution for (u, v, τ) **then**
- 7: $\mathcal{S} \leftarrow (A_h \cup T[x_1, u], R)$.
- 8: **for all** $i' \in [i-1]$ and $u' \in V(T_{i'})$ **do**
- 9: **for all** $j' \in [j] \setminus [i']$ and $v' \in V(T_{j'})$ such that $T[x_{j'}, v'] \cap T[x_i, u] = \emptyset$ **do**
- 10: **for all** $\tau' \subseteq \tau$ **do**
- 11: **if** $F(u', v', \tau') = \emptyset$ **then continue;**
- 12: Let $(Q'_1, Q'_2) = F(u', v', \tau')$.
- 13: **if** v is not reachable from u' in $G\langle T[v', u], u', v, \tau \setminus \tau' \rangle$ **then continue;**
- 14: Compute a shortest (u', v) -path R in $G\langle T[v', u], u', v, \tau \setminus \tau' \rangle$.
- 15: Let $P_1 = Q'_2 \cup T[v', u]$ and $P_2 = Q'_1 \cup R$.
- 16: **if** (P_1, P_2) is a partial solution for (u, v, τ) **then**
- 17: $\mathcal{S} \leftarrow (P_1, P_2)$.
- 18: **if** $\mathcal{S} = \emptyset$ **then return** \emptyset .
- 19: **else** Let S^* be the cheapest pair among those in \mathcal{S} , and **return** $F(u, v, \tau) := S^*$.

The following lemma guarantees the correctness of Algorithm PARTSOL. Formally, we say that $F(u, v, \tau)$ is *correctly computed* if either it contains a minimum-weight partial solution for (u, v, τ) , or no partial solution for (u, v, τ) exists and $F(u, v, \tau) = \emptyset$.

► **Lemma 32.** *Let $i, j \in [r]$ with $i \leq j$, $u \in V(T_i)$, $v \in V(T_j)$ and $\tau \subseteq \mathcal{T} \setminus \{T\}$. Assuming that the values $F(u', v', \tau')$ are correctly computed for each $u' \in V(T_{i'})$ with $i' < i$, Algorithm PARTSOL correctly computes $F(u, v, \tau)$.*

Using the correctness of Algorithm PARTSOL, as established by Lemma 32, and the observation in Lemma 28 on how partial solutions can be used to find two permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths of minimum total weight, we obtain the following.

► **Corollary 33.** *For each constant $c \in \mathbb{N}$, there is a polynomial-time algorithm that finds two permissively disjoint $(\{a_1, a_2\}, \{b_1, b_2\})$ -paths of minimum total weight in G (if such paths exist), where the set of negative edges in G spans c trees.*

5 Conclusion

We have presented a polynomial-time algorithm for solving the SHORTEST TWO DISJOINT PATHS problem on undirected graphs G with conservative edge weights, assuming that the number of connected components in the subgraph $G[E^-]$ spanned by all negative-weight edges is a fixed constant c . The running time of our algorithm is $O(n^{2c+9})$ on an n -vertex graph. Is it possible to give a substantially faster algorithm for this problem? In particular, is it possible to give a fixed-parameter tractable algorithm for SHORTEST TWO DISJOINT PATHS on undirected conservative graphs when parameterized by c ?

More generally, is it possible to find in polynomial time k openly disjoint (s, t) -paths with minimum total weight for some fixed $k \geq 3$ in undirected conservative graphs with constant values of c ?

References

- 1 Isolde Adler, Stavros G. Kolliopoulos, Philipp Klaus Krause, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Irrelevant vertices for the Planar Disjoint Paths problem. *Journal of Combinatorial Theory, Series B*, 122:815–843, 2017. doi:10.1016/j.jctb.2016.10.001.
- 2 Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *FOCS 2022: Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science*, pages 600–611. IEEE Computer Society, 2022. doi:10.1109/FOCS54457.2022.00063.
- 3 Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. *SIAM Journal on Computing*, 48(6):1698–1710, 2019. doi:10.1137/18M1223034.
- 4 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. The planar directed k -vertex-disjoint paths problem is fixed-parameter tractable. In *FOCS 2013: Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, pages 197–206. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.29.
- 5 Éric Colin de Verdière and Alexander Schrijver. Shortest vertex-disjoint two-face paths in planar graphs. *ACM Trans. Algorithms*, 7(2):19:1–19:12, 2011. doi:10.1145/1921659.1921665.
- 6 Guoli Ding, A. Schrijver, and P. D. Seymour. Disjoint paths in a planar graph – a general theorem. *SIAM Journal on Discrete Mathematics*, 5(1):112–116, 1992. doi:10.1137/0405009.
- 7 Richard M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.
- 8 Yusuke Kobayashi and Christian Sommer. On shortest disjoint paths in planar graphs. *Discrete Optimization*, 7(4):234–245, 2010. doi:10.1016/j.disopt.2010.05.002.

- 9 Daniel Lokshantov, Pranabendu Misra, Michał Pilipczuk, Saket Saurabh, and Meirav Zehavi. An exponential time parameterized algorithm for planar disjoint paths. In *STOC 2020: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1307–1316. Association for Computing Machinery, 2020. doi:10.1145/3357713.3384250.
- 10 James F. Lynch. The equivalence of theorem proving and the interconnection problem. *ACM SIGDA Newsletter*, 5:31–36, 1975. doi:10.1145/1061425.1061430.
- 11 Neil Robertson and P.D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.
- 12 Ildikó Schlotter. Shortest two disjoint paths in conservative graphs, 2023. arXiv:2307.12602.
- 13 Ildikó Schlotter and András Sebő. Odd paths, cycles and T -joins: Connections and algorithms, 2022. arXiv:2211.12862.
- 14 Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal on Computing*, 23(4):780–788, 1994. doi:10.1137/S0097539792224061.
- 15 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin, 2003.

Algorithms for Computing Closest Points for Segments

Haitao Wang   

Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

Given a set P of n points and a set S of n segments in the plane, we consider the problem of computing for each segment of S its closest point in P . The previously best algorithm solves the problem in $n^{4/3}2^{O(\log^* n)}$ time [Bespamyatnikh, 2003] and a lower bound (under a somewhat restricted model) $\Omega(n^{4/3})$ has also been proved. In this paper, we present an $O(n^{4/3})$ time algorithm and thus solve the problem optimally (under the restricted model). In addition, we also present data structures for solving the online version of the problem, i.e., given a query segment (or a line as a special case), find its closest point in P . Our new results improve the previous work.

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

Keywords and phrases Closest points, Voronoi diagrams, Segment dragging queries, Hopcroft's problem, Algebraic decision tree model

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.58

Related Version *Full Version*: <https://arxiv.org/abs/2401.02636>

Funding This research was supported in part by NSF under Grant CCF-2300356.

1 Introduction

Given a set P of n points and a set S of n segments in the plane, we consider the problem of computing for each segment of S its closest point in P . We call it the *segment-closest-point* problem. Previously, Bespamyatnikh [6] gave an $n^{4/3}2^{O(\log^* n)}$ time algorithm for the problem, improving upon an $O(n^{4/3} \log^{O(1)} n)$ time result of Agarwal and Procopiuc. The problem can be viewed as a generalization of Hopcroft's problem [1, 10, 13, 20, 30], which is to determine whether any point of a given set of n points lies on any of the given n lines. Erickson [21] proved an $\Omega(n^{4/3})$ time lower bound for Hopcroft's problem under a somewhat restricted *partition model*. This implies the same lower bound on the segment-closest-point problem. For Hopcroft's problem, Chan and Zheng [10] recently gave an $O(n^{4/3})$ time algorithm, which matches the lower bound and thus is optimal.

In this paper, with some new observations on the problem as well as the techniques from Chan and Zheng [10] (more specifically, the Γ -algorithm framework for bounding algebraic decision tree complexities), we present a new algorithm that solves the segment-closest-point problem in $O(n^{4/3})$ time and thus is optimal under Erickson's partition model [21]. It should be noted that our result is not a direct application of Chan and Zheng's techniques [10], but rather many new observations and techniques are needed. For example, one subroutine in our problem is the following *outside-hull segment queries*: Given a segment outside the convex hull of P , find its closest point in P . Bespamyatnikh and Snoeyink [7] built a data structure in $O(n)$ space and $O(n \log n)$ time such that each query can be answered in $O(\log n)$ time. Unfortunately, their query algorithm does not fit the Γ -algorithm framework of Chan and Zheng [10]. To resolve the issue, we develop another algorithm for the problem based on new observations. Our approach is simpler, and more importantly, it fits into the Γ -algorithm framework of Chan and Zheng [10]. The result may be interesting in its own right.



© Haitao Wang;

licensed under Creative Commons License CC-BY 4.0

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).

Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov;

Article No. 58; pp. 58:1–58:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



We also consider the online version of the problem, called *the segment query problem*: Preprocess P so that given a query segment, its closest point in P can be found efficiently. For the special case where the query segment is outside the convex hull of P , one can use the data structure of Bespamyatnikh and Snoeyink [7] mentioned above. For simplicity, we use $(T_1(n), T_2(n), T_3(n))$ to denote the complexity of a data structure if its preprocessing time, space, and query time are on the order of $T_1(n)$, $T_2(n)$, and $T_3(n)$, respectively. Using this notation, the complexity of the above data structure of Bespamyatnikh and Snoeyink [7] is $O(n \log n, n, \log n)$. The general problem, however, is much more challenging. Goswami, Das, and Nandy [25]’s method yields a result of complexity $O(n^2, n^2, \log^2 n)$. We present a new data structure of complexity $O(nm(n/m)^\delta, nm \log(n/m), \sqrt{n/m} \log(n/m))$, any m with $1 \leq m \leq n \log^2 \log n / \log^4 n$ and any $\delta > 0$. Note that for the large space case (i.e., when $m = n \log^2 \log n / \log^4 n$), the complexity of our data structure is $O(n^2 / \log^{4-\delta} n, n^2 \log^3 \log n / \log^4 n, \log^2 n)$, which improves the above result of [25] on the preprocessing time and space by a factor of roughly $\log^4 n$. We also give a faster randomized data structure of complexity $O(nm \log(n/m), nm \log(n/m), \sqrt{n/m})$ for any m with $1 \leq m \leq n / \log^4 n$, where the preprocessing time is expected and the query time holds with high probability. In addition, using Chan’s randomized techniques [8] and Chan and Zheng’s recent randomized result on triangle range counting [10], one can obtain a randomized data structure of complexity $O(n^{4/3}, n^{4/3}, n^{1/3})$. Note that this data structure immediately leads to a randomized algorithm of $O(n^{4/3})$ expected time for the segment-closest-point problem. As such, for solving the segment-closest-point problem, our main effort is to derive an $O(n^{4/3})$ deterministic time algorithm. Note that this is aligned with the motivation of proposing the Γ -algorithm framework in [10], whose goal was to obtain an $O(n^{4/3})$ deterministic time algorithm for Hopcroft’s problem although a much simpler randomized algorithm of $O(n^{4/3})$ expected time was already presented.

If each query segment is a line, we call it the *line query problem*, which has been extensively studied. Previous work includes Cole and Yap [17]’s and Lee and Ching [28]’s data structures of complexity $O(n^2, n^2, \log n)$, Mitra and Chaudhuri [31]’s work of complexity $O(n \log n, n, n^{0.695})$, Mukhopadhyay [32]’s result of complexity $O(n^{1+\delta}, n \log n, n^{1/2+\delta})$ for any $\delta > 0$. As observed by Lee and Ching [28], the problem can be reduced to vertical ray-shooting in the dual plane, i.e., finding the first line hit by a query vertical ray among a given set of n lines. Using the ray-shooting algorithms, the best deterministic result is $O(n^{1.5}, n, \sqrt{n} \log n)$ [35] while the best randomized result is $O(n \log n, n, \sqrt{n})$ [11]; refer to [2, 4, 10, 16] for other (less efficient) work on ray-shootings. We build a new deterministic data structure of complexity $O(nm(n/m)^\delta, nm \log(n/m), \sqrt{n/m})$, for any $1 \leq m \leq n / \log^2 n$. We also have another faster randomized result of complexity $O(nm \log(n/m), nm \log(n/m), \sqrt{n/m})$, for any m with $1 \leq m \leq n / \log^2 n$, where the preprocessing time is expected while the query time holds with high probability. Our results improve all previous work except the randomized result of Chan and Zheng [11]. For example, if $m = 1$, our data structure is the only deterministic one whose query time is $O(\sqrt{n})$ with near linear space; if $m = n / \log^2 n$, our result achieves $O(\log n)$ query time while the preprocessing is subquadratic, better than those by Cole and Yap [17] and Lee and Ching [28].

Other related work. If all segments are pairwise disjoint, then the segment-closest-point problem was solved in $O(n \log^2 n)$ time by Bespamyatnikh [6], improving over the $O(n \log^3 n)$ time algorithm of Bespamyatnikh and Snoeyink [7].

If every segment of S is a single point, then the problem can be easily solved in $O(n \log n)$ time using the Voronoi diagram of P . Also, for any segment $s \in S$, if the point of s closest to P is an endpoint of s , then finding the closest point of s in P can be done using the Voronoi

diagram of P . Hence, the remaining issue is to find the first point of P hit by s if we drag s along the directions perpendicularly to s . If all segments of S have the same slope, then the problem can be solved in $O(n \log n)$ time using the segment dragging query data structure of Chazelle [12], which can answer each query in $O(\log n)$ time after $O(n)$ space and $O(n \log n)$ time preprocessing. However, the algorithm [12] does not work if the query segments have arbitrary slopes. As such, the challenge of the problem is to solve the dragging queries for all segments of S when their slopes are not the same.

The *segment-farthest-point* problem has also been studied, where one wants to find for each segment of S its farthest point in P . The problem appears much easier. For the line query problem (i.e., given a query line, find its farthest point in P), Daescu et al. [18] gave a data structure of complexity $O(n \log n, n, \log n)$. Using this result, they also proposed a data structure of complexity $O(n \log n, n \log n, \log^2 n)$ for the segment query problem. Using this segment query data structure, the segment-farthest-point can be solved in $O(n \log^2 n)$ time.

Outline. The rest of the paper is organized as follows. In Section 2, we introduce some notation and concepts. In Section 3, we present our $O(n^{4/3})$ deterministic time algorithm for the segment-closest-point problem. We actually solve a more general problem where the number of points is not equal to the number of segments, referred to as the *asymmetric case*, and our algorithm runs in $O(n^{2/3}m^{2/3} + n \log n + m \log^2 n)$ time with n as the number of points and m as the number of segments. For the line case of the problem where all segments are lines, a simpler algorithm is presented in the full paper and the algorithm also runs in $O(n^{4/3})$ time (and $O(n^{2/3}m^{2/3} + (n + m) \log n)$ time for the asymmetric case). The online query problem is sketched in Section 4 with details in the full paper. Due to the space limit, many lemma proofs are omitted but can be found in the full paper.

2 Preliminaries

For two closed subsets A and B in the plane, let $d(A, B)$ denote the minimum distance between any point of A and any point of B . The point p of A closest to B , i.e., $d(p, B) = d(A, B)$, is called the *closest point* of B in A . For any two points a and b in the plane, we use \overline{ab} to denote the segment with a and b as its two endpoints.

For any point p in the plane, we use $x(p)$ and $y(p)$ to denote its x - and y -coordinates, respectively. For a point p and a region A in the plane, we say that p is *to the left* of A if $x(p) \leq x(q)$ for all points $q \in A$, and p is *strictly to the left* of A if $x(p) < x(q)$ for all points $q \in A$; the concepts (*strictly*) *to the right* is defined symmetrically.

For a set Q of points in the plane, we usually use $\text{VD}(Q)$ to denote the Voronoi diagram of Q and use $\text{CH}(Q)$ to denote the convex hull of Q ; we also use $Q(A)$ to denote the subset of Q in A , i.e., $Q(A) = Q \cap A$, for any region A in the plane.

Cuttings. Let H be a set of n lines in the plane. Let H_A denote the subset of lines of H that intersect the interior of A (we also say that these lines *cross* A), for a compact region A in the plane. A *cutting* is a collection Ξ of closed cells (each of which is a triangle) with disjoint interiors, which together cover the entire plane [13, 30]. The *size* of Ξ is the number of cells in Ξ . For a parameter r with $1 \leq r \leq n$, a $(1/r)$ -*cutting* for H is a cutting Ξ satisfying $|H_\sigma| \leq n/r$ for every cell $\sigma \in \Xi$.

A cutting Ξ' *c-refines* another cutting Ξ if every cell of Ξ' is contained in a single cell of Ξ and every cell of Ξ contains at most c cells of Ξ' . A *hierarchical* $(1/r)$ -*cutting* for H (with two constants c and ρ) is a sequence of cuttings $\Xi_0, \Xi_1, \dots, \Xi_k$ with the following properties.

Ξ_0 is the entire plane. For each $1 \leq i \leq k$, Ξ_i is a $(1/\rho^i)$ -cutting of size $O(\rho^{2i})$ which c -refines Ξ_{i-1} . In order to make Ξ_k a $(1/r)$ -cutting, we set $k = \lceil \log_\rho r \rceil$. Hence, the size of the last cutting Ξ_k is $O(r^2)$. If a cell $\sigma \in \Xi_{i-1}$ contains a cell $\sigma' \in \Xi_i$, we say that σ is the *parent* of σ' and σ' is a *child* of σ . As such, one could view Ξ as a tree in which each node corresponds to a cell $\sigma \in \Xi_i$, $0 \leq i \leq k$.

For any $1 \leq r \leq n$, a hierarchical $(1/r)$ -cutting of size $O(r^2)$ for H (together with H_σ for every cell σ of Ξ_i for all $i = 0, 1, \dots, k$) can be computed in $O(nr)$ time [13]. Also, it is easy to check that $\sum_{i=0}^k \sum_{\sigma \in \Xi_i} |H_\sigma| = O(nr)$.

3 The segment-closest-point problem

In this section, we consider the segment-closest-point problem. Let P be a set of n points and S a set of m segments in the plane. The problem is to compute for each segment of S its closest point in P . We make a general position assumption that no segment of S is vertical (for a vertical segment, its closest point can be easily found, e.g., by building a segment dragging query data structure [12] along with the Voronoi diagram of P).

We start with a review of an algorithm of Bespamyatnikh [6], which will be needed in our new approach.

3.1 A review of Bespamyatnikh's algorithm [6]

As we will deal with subproblems in which the number of lines is not equal to the number of segments, we let m denote the number of segments in S and n the number of points in P . As such, the size of our original problem (S, P) is (m, n) .

Let H be the set of the supporting lines of the segments of S . For a parameter r with $1 \leq r \leq \min\{m, \sqrt{n}\}$, compute a hierarchical $(1/r)$ -cutting $\Xi_0, \Xi_1, \dots, \Xi_k$ for H . For each cell $\sigma \in \Xi_i$, $0 \leq i \leq k$, let $P(\sigma) = P \cap \sigma$, i.e., the subset of the points of P in σ ; let $S(\sigma)$ denote the subset of the segments of S intersecting σ . We further partition each cell of Ξ_k into triangles so that each triangle contains at most n/r^2 points of P and the number of new triangles in Ξ_k is still bounded by $O(r^2)$. For convenience, we consider the new triangles as new cells of Ξ_k (we still define $P(\sigma)$ and $S(\sigma)$ for each new cell σ in the same way as above; so now $|P(\sigma)| \leq n/r^2$ and $|S(\sigma)| \leq m/r$ hold for each cell $\sigma \in \Xi_k$).

For each cell $\sigma \in \Xi_k$, form a subproblem $(S(\sigma), P(\sigma))$ of size $(m/r, n/r^2)$, i.e., find for each segment s of $S(\sigma)$ its closest point in $P(\sigma)$. After the subproblem is solved, to find the closest point of s in P , it suffices to find its closest point in $P \setminus P(\sigma)$. To this end, observe that $P \setminus P(\sigma)$ is exactly the union of $P(\sigma'')$ for all cells σ'' such that σ'' is a child of an ancestor σ' of σ and $s \notin S(\sigma')$. As such, for each of such cells σ'' , find the closest point of s in $P(\sigma'')$. For this, since $s \notin S(\sigma'')$, s is outside σ'' and thus is outside the convex hull of $P(\sigma'')$. Hence, finding the closest point of s in $P(\sigma'')$ is an outside-hull segment query and thus the data structure of Bespamyatnikh and Snoeyink [7] (referred to as *the BS data structure* in the rest of the paper) is used, which takes $O(|P(\sigma'')|)$ space and $O(|P(\sigma'')| \log |P(\sigma'')|)$ time preprocessing and can answer each query in $O(\log |P(\sigma'')|)$ time. More precisely, the processing can be done in $O(|P(\sigma'')|)$ time if the Voronoi diagram of $P(\sigma'')$ is known.

For the time analysis, let $T(m, n)$ denote the time of the algorithm for solving a problem of size (m, n) . Then, solving all subproblems takes $O(r^2) \cdot T(m/r, n/r^2)$ time as there are $O(r^2)$ subproblems of size $(m/r, n/r^2)$. Constructing the hierarchical cutting as well as computing $S(\sigma)$ for all cells σ in all cuttings Ξ_i , $0 \leq i \leq k$, takes $O(mr)$ time [13]. Computing $P(\sigma)$ for all cells σ can be done in $O(n \log r)$ time. Preprocessing for constructing the BS data structure for $P(\sigma)$ for all cells σ can be done in $O(n \log n \log r)$ time as $\sum_{\sigma \in \Xi_i} |P(\sigma)| = n$

for each $0 \leq i \leq k$, and $k = O(\log r)$. We can further reduce the time to $O(n(\log r + \log n))$ as follows. We build the BS data structure for cells of the cuttings in a bottom-up manner, i.e., processing cells of Ξ_k first and then Ξ_{k-1} and so on. After the preprocessing for $P(\sigma)$ for a cell $\sigma \in \Xi_k$, which takes $O(|P(\sigma)| \log(n/r^2))$ time since $|P(\sigma)| \leq n/r^2$, the Voronoi diagram of $P(\sigma)$ is available. After the preprocessing for all cells σ of Ξ_k is done, for each cell σ' of Ξ_{k-1} , to construct the Voronoi diagram of $P(\sigma')$, merge the Voronoi diagrams of $P(\sigma)$ for all children σ of σ' . To this end, as σ' has $O(1)$ children, the merge can be done in $O(|P(\sigma')|)$ time by using the algorithm of Kirkpatrick [27], and thus the preprocessing for $P(\sigma')$ takes only linear time. In this way, the total preprocessing time for all cells in all cuttings Ξ_i , $0 \leq i \leq k$, is bounded by $O(n(\log r + \log(n/r^2)))$ time, i.e., the time spent on cells of Ξ_k is $O(n \log(n/r^2))$ and the time on other cuttings is $O(n \log r)$ in total. Note that $\log r + \log(n/r^2) = \log(n/r)$. As for the outside-hull segment queries, according to the properties of the hierarchical cutting, $\sum_{i=0}^k \sum_{\sigma \in \Xi_i} |S(\sigma)| = O(mr)$. Hence, the total number of outside-hull segment queries on the BS data structure is $O(mr)$ and thus the total query time is $O(mr \log n)$. In summary, the following recurrence is obtained for any $1 \leq r \leq \min\{m, \sqrt{n}\}$:

$$T(m, n) = O(n \log(n/r) + mr \log n) + O(r^2) \cdot T(m/r, n/r^2). \quad (1)$$

Using the duality, Bespamyatnikh [6] gave a second algorithm (we will not review this algorithm here because it is not relevant to our new approach) and obtained the following recurrence for any $1 \leq r \leq \min\{n, \sqrt{m}\}$:

$$T(m, n) = O(nr \log n + m \log r \log n) + O(r^2) \cdot T(m/r^2, n/r). \quad (2)$$

Setting $m = n$ and applying (2) and (1) in succession (using the same r) obtain $T(n, n) = O(nr \log n) + O(r^4) \cdot T(n/r^3, n/r^3)$. Setting $r = n^{1/3}/\log n$ leads to

$$T(n, n) = O(n^{4/3}) + O((n/\log^3 n)^{4/3}) \cdot T(\log^3 n, \log^3 n). \quad (3)$$

The recurrence solves to $T(n, n) = n^{4/3} 2^{O(\log^* n)}$, which is the time bound obtained in [6].

3.2 Our new algorithm

In this section, we improve the algorithm to $O(n^{4/3})$ time.

By applying recurrence (3) three times we obtain the following:

$$T(n, n) = O(n^{4/3}) + O((n/b)^{4/3}) \cdot T(b, b), \quad (4)$$

where $b = (\log \log \log n)^3$.

Using the property that b is tiny, we show in the following that after $O(n)$ time preprocessing, we can solve each subproblem $T(b, b)$ in $O(b^{4/3})$ time (for convenience, by slightly abusing the notation, we also use $T(m, n)$ to denote a subproblem of size (m, n)). Plugging the result into (4), we obtain $T(n, n) = O(n^{4/3})$.

More precisely, we show that after $O(2^{\text{poly}(b)})$ time preprocessing, where $\text{poly}(\cdot)$ is a polynomial function, we can solve each $T(b, b)$ using $O(b^{4/3})$ comparisons, or alternatively, $T(b, b)$ can be solved by an algebraic decision tree of height $O(b^{4/3})$. As $b = (\log \log \log n)^3$, $2^{\text{poly}(b)}$ is bounded by $O(n)$. To turn this into an algorithm under the standard real-RAM model, we explicitly construct the algebraic decision tree for the above algorithm (we may also consider this step as part of preprocessing for solving $T(b, b)$), which can again be done in $O(2^{\text{poly}(b)})$ time. As such, that after $O(n)$ time preprocessing, we can solve each $T(b, b)$ in $O(b^{4/3})$ time. In the following, for notational convenience, we will use n to denote b , and our goal is to prove the following lemma.

► **Lemma 1.** *After $O(2^{\text{poly}(n)})$ time preprocessing, $T(n, n)$ can be solved using $O(n^{4/3})$ comparisons.*

We apply recurrence (1) by setting $m = n$ and $r = n^{1/3}$, and obtain the following

$$T(n, n) = O(n \log n + n^{4/3} \log n) + O(n^{2/3}) \cdot T(n^{2/3}, n^{1/3}). \quad (5)$$

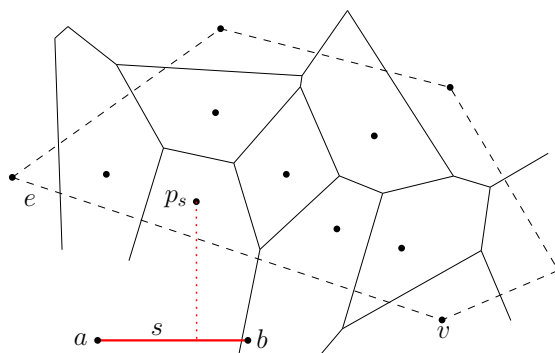
Recall that the term $n^{4/3} \log n$ is due to that there are $O(n^{4/3})$ outside-hull segment queries. To show that $T(n, n)$ can be solved by $O(n^{4/3})$ comparisons, there are two challenges: (1) solve all outside-hull segment queries using $O(n^{4/3})$ comparisons; (2) solve each subproblem $T(n^{2/3}, n^{1/3})$ using $O(n^{2/3})$ comparisons.

Γ -algorithm framework. To tackle these challenges, we use a Γ -algorithm framework for bounding decision tree complexities proposed by Chan and Zheng [10]. We briefly review it here (see Section 4.1 [10] for the details). Roughly speaking, this framework is an algorithm that only counts the number of comparisons (called Γ -comparisons in [10]) for determining whether a point belongs to a semialgebraic set of $O(1)$ degree in a constant-dimensional space. Solving our segment-closest-point problem is equivalent to locating the cell C^* containing a point p^* parameterized by the input of our problem (i.e., the segments of S and the points of P) in an arrangement \mathcal{A} of the boundaries of $\text{poly}(n)$ semialgebraic sets in $O(n)$ -dimensional space. This arrangement can be built in $O(2^{\text{poly}(n)})$ time without examining the values of the input and thus does not require any comparisons. In particular, the number of cells of \mathcal{A} is bounded by $n^{O(n)}$. As a Γ -algorithm progresses, it maintains a set Π of cells of \mathcal{A} . Initially, Π consisting of all cells of \mathcal{A} . During the course of the algorithm, Π can only shrink but always contains the cell C^* . At the end of the algorithm, C^* will be found. Define the potential $\Phi = \log |\Pi|$. As \mathcal{A} has $n^{O(n)}$ cells, initially $\Phi = O(n \log n)$. For any operation or subroutine of the algorithm, we use $\Delta\Phi$ to denote the change of Φ . As Φ only decreases during the algorithm, $\Delta\Phi \leq 0$ always holds and the sum of $-\Delta\Phi$ during the entire algorithm is $O(n \log n)$. This implies that we may afford an expensive operation/subroutine during the algorithm as long as it decreases Φ a lot.

Two algorithmic tools are developed in [10] under the framework: *basic search lemma* (Lemma 4.1 [10]) and *search lemma* (Lemma A.1 [10]). Roughly speaking, given r predicates (each predicate is a test of whether $\gamma(x)$ is true for the input vector x), suppose it is promised that at least one of them is true for all inputs in the active cells; then the basic search lemma can find a predicate that is true by making $O(1 - r \cdot \Delta\Phi)$ comparisons. Given a binary tree (or a more general DAG of $O(1)$ degree) such that each node v is associated with a predicate γ_v , suppose for each internal node v , γ_v implies γ_u for a child u of v for all inputs in the active cells. Then, the search lemma can find a leaf v such that γ_v is true by making $O(1 - \Delta\Phi)$ comparisons.

An application of both lemmas particularly discussed in [10] is to find a predecessor of a query number among a sorted list of input numbers. In our algorithm, as will be seen later, the subproblem that needs the Γ -algorithm framework is also finding predecessors among sorted lists and thus both the basic search lemma and the search lemma are applicable.

In the following two subsections, we will tackle the above two challenges, respectively. By slightly abusing the notation, let P be a set of n points and S a set of n segments for the problem in recurrence (5).



■ **Figure 1** Illustrating an outside-hull segment query.

3.3 Solving outside-hull segment queries

Recall that we have used the BS data structure to answer the outside-hull segment queries. Unfortunately the algorithm does not fit the Γ -algorithm framework. Indeed, the BS data structure is a binary tree. However, each node of the tree represents a convex hull of a subset of points and it is not associated with a predicate that we can use to apply the Γ -algorithm framework (e.g., the search lemma as discussed above).

In the following, we first present a new algorithm for solving the outside-hull segment queries. Our algorithm, whose performance matches that of the BS data structure, is simpler, and thus may be of independent interest; more importantly, it leads to an algorithm that fits the Γ -algorithm framework to provide an $O(n^{4/3})$ upper bound.

Let Q be a set of n' points. The problem is to preprocess Q so that given any query segment s outside the convex hull $\text{CH}(Q)$ of Q , the closest point of s in Q can be computed efficiently. Recall that in our original problem (i.e., the recurrence (5)) Q is a subset of P and the sum of n' for all subsets of P that we need to build the outside-hull query data structures is $O(n \log n)$. We make this an observation below, which will be referred to later.

► **Observation 2.** *The size of the subsets of P that we need to build the outside-hull query data structures is $O(n \log n)$, i.e., $\sum n' = O(n \log n)$.*

In the preprocessing, we compute the Voronoi diagram $\text{VD}(Q)$ of Q , from which we can obtain the convex hull $\text{CH}(Q)$ in linear time. For each edge e of $\text{CH}(Q)$, we determine the subset Q_e of points of Q whose Voronoi cells intersect e in order along e . This order is exactly the order of the perpendicular projections of the points of Q_e onto e [7].

Consider a query segment s that is outside $\text{CH}(Q)$. Let p_s be the first point of Q hit by s if we drag s along the direction perpendicularly to s and towards $\text{CH}(Q)$; see Fig. 1. For ease of exposition, we assume that p_s is unique. Our goal is to compute p_s in the case where the point of s closest to Q is not an endpoint of s since the other case can be easily solved by using $\text{VD}(Q)$. Henceforth, we assume that the point of s closest to Q is not an endpoint of s , implying that p_s is the point of Q closest to s . Without loss of generality, we assume that s is horizontal and s is below $\text{CH}(Q)$. Let a and b be the left and right endpoints of s , respectively (see Fig. 1).

We first find the lowest vertex v of $\text{CH}(Q)$, which can be done in $O(\log n')$ time by doing binary search on $\text{CH}(Q)$. If $x(a) \leq x(v) \leq x(b)$, then v is p_s and we are done with the query. Otherwise, without loss of generality, we assume that $x(b) < x(v)$. By binary search on $\text{CH}(Q)$, we find the edge e in the lower hull of $\text{CH}(Q)$ that intersects the vertical line through b . Since $x(a) \leq x(b) < x(v)$, e must have a negative slope (see Fig. 1). Then, as discussed in [7], p_s must be in Q_e . To find p_s efficiently, we first make some observations (which were not discovered in the previous work).

Suppose p_1, p_2, \dots, p_m are the points of Q_e , sorted following the order of their Voronoi cells in $\text{VD}(Q)$ intersecting e from left to right. We define two special indices i^* and j^* of Q_e with respect to a and b , respectively.

► **Definition 3.** Define j^* as the largest index of the point of Q_e that is to the left of b . Define i^* as the smallest index of the point of Q_e such that p_j is to the right of a for all $j \geq i^*$.

Note that j^* must exist as p_s is in Q_e and is to the left of b . We have the following lemma.

► **Lemma 4.** If i^* does not exist or $i^* > j^*$, then p_s cannot be the closest point of s in Q .

By Lemma 4, if i^* does not exist or if $i^* > j^*$, then we can simply stop the query algorithm. In the following, we assume that i^* exists and $i^* \leq j^*$. Let $Q_e[i^*, j^*]$ denote the subset of points of Q_e whose indices are between i^* and j^* inclusively. The following lemma implies that we can use the supporting line of s to search p_s .

► **Lemma 5.** Suppose p_s is the closest point of s in Q . Then, p_s is the point of $Q_e[i^*, j^*]$ closest to the supporting line of s (i.e., the line containing s).

Based on Lemma 5, we have the following three steps to compute p_s : (1) compute j^* ; (2) compute i^* ; (3) find the point of $Q_e[i^*, j^*]$ closest to the supporting line ℓ_s of s .

The following Lemma 6, which is for outside-hull segment queries, is a by-product of our above observations. Its complexity is the same as that in [7]. However, we feel that our new query algorithm is simpler and thus this result may be interesting in its own right.

► **Lemma 6.** Given a set Q of n' points in the plane, we can build a data structure of $O(n')$ space in $O(n' \log n')$ time such that each outside-hull query can be answered in $O(\log n')$ time. The preprocessing time is $O(n')$ if the Voronoi diagram of Q is known.

The query algorithm of Lemma 6 actually does not fit the Γ -algorithm framework. Instead, following the above observations we will give another query algorithm that fits the Γ -algorithm framework. We now give a new algorithm that fits the Γ -algorithm framework. The new algorithm requires slightly more preprocessing than Lemma 6. But for our purpose, we are satisfied with $O(n^{4/3})$ preprocessing time. We have different preprocessing for each of the three steps of the query algorithm, as follows.

The first step: computing j^* . For computing j^* , we will use the *basic search lemma* (i.e., Lemma 4.1) in [10]. In order to apply the lemma, we perform the following preprocessing.

Recall that $Q_e = \{p_1, p_2, \dots, p_m\}$ is ordered by their Voronoi cells intersecting e . We partition the sequence into r contiguous subsequences of size roughly m/r each. Let Q_e^i denote the i -th subsequence, with $1 \leq i \leq r$. For each $i \in [1, r]$, we compute and explicitly maintain the convex hull $\text{CH}(i)$ of all points in the union of the subsequences Q_e^j , $j = i, i+1, \dots, r$. Next, for each subsequence Q_e^i , we further partition it into r contiguous sequences of size roughly $|Q_e^i|/r$ and process it in the same way as above. We do this recursively until the subsequence has no more than r points. In this way, we obtain a tree T with m leaves such that each node has r children. For each node v , we use $\text{CH}(v)$ to denote the convex hull that is computed above corresponding to v (e.g., if v is the child of the root corresponding to Q_e^i , then $\text{CH}(v)$ is $\text{CH}(i)$ defined above). The total time for constructing T can be easily bounded by $O(mr \log m \log_r m)$ as the height of T is $O(\log_r m)$.

Now to compute j^* , we search the tree T : starting from the root, for each node v , we apply the basic search lemma on all r children of v . Indeed, this is possible due to the following. Consider the root v . For each i with $1 \leq i \leq r$, let x_i denote the x -coordinate

of the leftmost point of the union of the subsequences Q_e^j , $j = i, i + 1, \dots, r$; note that x_i is also the leftmost vertex of $\text{CH}(i)$. It is not difficult to see that $x_1 \leq x_2 \leq \dots \leq x_r$. Observe that p_{j^*} is in Q_e^i if and only if $x_i \leq x(b) < x_{i+1}$. Therefore, we find the index i such that $x_i \leq x(b) < x_{i+1}$ and then proceed to the child of v corresponding to Q_e^i . This property satisfies the condition of the basic search lemma (essentially, we are looking for the predecessor of b in the sequence x_1, x_2, \dots, x_r and this is somewhat similar to the insertion sort algorithm of Theorem 4.1 [10], which uses the basic search lemma). By the basic search lemma, finding the index i can be done using $O(1 - r\Delta\Phi)$ comparisons provided that the x -coordinates x_1, x_2, \dots, x_r are available to us (we will discuss how to compute them later). We then follow the same idea recursively until we reach a leaf. In this way, the total number of comparisons for computing j^* is $O(\log_r m - r\Delta\Phi)$.

By setting $r = m^\epsilon$ for a small constant ϵ , the preprocessing time is $O(m^{1+\epsilon} \log m)$ and computing j^* can be done using $O(1 - m^\epsilon \Delta\Phi)$ comparisons. Recall that there are $O(n^{4/3})$ queries in our original problem (i.e., recurrence (5)) and the total time for $-\Delta\Phi$ during the entire algorithm is $O(n \log n)$. Also, since m is the number of points of Q whose Voronoi cells intersecting the edge e of $\text{CH}(Q)$, the sum of m for all outside-hull segment query data structures for all edges of $\text{CH}(Q)$ is $|Q|$, which is n' . By Observation 2, the sum of n' for all data structures in our original problem is $O(n \log n)$. Hence, the total preprocessing time for our original problem is $O(n^{1+\epsilon} \log^{2+\epsilon} n)$, which is bounded by $O(n^{4/3})$ if we set ϵ to a small constant (e.g., $\epsilon = 1/4$). As such, with a preprocessing step of $O(n^{4/3})$ time, we can compute j^* for all queries using a total of $O(n^{4/3})$ comparisons.

The above complexity analysis for computing j^* is based on the assumption that the leftmost point of $\text{CH}(v)$ for each node v of T is known. To find these points during the queries, we take advantage of the property that all queries are offline, i.e., we know all query segments before we start the queries. Notice that although there are $O(n^{4/3})$ queries, the number of distinct query segments is n , i.e., those in S (a segment may be queried on different subsets of P). Let s be the current query segment and p be the leftmost point of a convex hull $\text{CH}(v)$ with respect to s (i.e., by assuming s is horizontal). Let ρ_1 be the ray from p going vertically upwards. Let ρ_2 be another ray from p going through the clockwise neighbor of p on CH_v , i.e., ρ_2 contains the clockwise edge of CH_v incident to v . Observe that for another query segment s' , p is still the leftmost point of CH_v with respect to s' as long as the direction perpendicular to s' is within the angle from ρ_1 clockwise to ρ_2 . Based on this observation, before we start any query, we sort the perpendicular directions of all segments of S along with the directions of all edges of all convex hulls of all nodes of the trees T for all outside-hull segment query data structures in our original problem (i.e., the recurrence (5)). As analyzed above, the total size of convex hulls of all trees T is $O(n^{1+\epsilon} \log^{2+\epsilon} n)$. Hence, the sorting can be done in $O(n^{1+\epsilon} \log^{3+\epsilon} n)$ time. Let L be the sorted list. We solve the queries for segments following their order in L . Let s and s' be two consecutive segments of S in L . After we solve all queries for s , the directions between s and s' in L correspond to those nodes of the trees T whose leftmost points need to get updated, and we then update the leftmost points of those nodes before we solve queries for s' . The total time we update the tree nodes for all queries is proportional to the total size of all trees, which is $O(n^{1+\epsilon} \log^{2+\epsilon} n)$.

In summary, after $O(n^{4/3})$ time preprocessing, computing j^* for all $O(n^{4/3})$ outside-hull segment queries can be done using $O(n^{4/3})$ comparisons.

The second step: computing i^* . For computing i^* , the idea is similar and we only sketch it. In the preprocessing, we build the same tree T as above for the first step. One change is that we add the first point p of the subsequence Q_e^i to the end of Q_e^{i-1} , i.e., p appears in both Q_e^i and Q_e^{i-1} . This does not change the complexities asymptotically.

For each query, to compute i^* , consider the root v . Observe that i^* is in Q_e^i if and only if $x_i < x(a) \leq x_{i+1}$ (x_i and x_{i+1} are defined in the same way as before). As such, we can apply the basic search lemma to find i^* in $O(1 - m^\epsilon \Delta\Phi)$ comparisons. We can use the same approach as above to update the leftmost points of convex hulls of nodes of the trees T (i.e., computing a sorted list L and process the queries of the segments following their order in L).

In summary, after $O(n^{4/3})$ time preprocessing, computing i^* for all $O(n^{4/3})$ outside-hull segment queries can be done using $O(n^{4/3})$ comparisons.

The third step. The third step is to find the point p_s of $Q_e[i^*, j^*]$ closest to $\ell(s)$, where $\ell(s)$ is the supporting line of s . We first discuss the preprocessing step on Q_e .

We build a balanced binary search tree T_e whose leaves corresponding to the points of $Q_e = \{p_1, p_2, \dots, p_m\}$ in their index order as discussed before. For each node v of T_e , we use $Q_e(v)$ to denote the set of points in the leaves of the subtree rooted at v . For each node v of T_e , we explicitly store the convex hull of $Q_e(v)$ at v . Further, for each leaf v , which stores a point p_i of Q_e , for each ancestor u of v , we compute the convex hull $\text{CH}_r(v, u)$ of all points p_i, p_{i+1}, \dots, p_j , where p_j is the point in the rightmost leaf of the subtree at u . We do this in a bottom-up manner starting from v following the path from v to u . More specifically, suppose we are currently at a node w , which is v initially. Suppose we have the convex hull $\text{CH}_r(v, w)$. We proceed on the parent w' of w as follows. If w is the right child of w' , then $\text{CH}_r(v, w')$ is $\text{CH}_r(v, w)$ and thus we do nothing. Otherwise, we merge $\text{CH}_r(v, w)$ with the convex hull of $Q_e(w'')$ at w'' , where w'' is the right child of w' . Since points of $\text{CH}_r(v, w)$ are separated from points of $Q_e(w'')$ by a line perpendicular to e [7], we can merge the two hulls by computing their common tangents in $O(\log m)$ time [33]. We use a persistent tree to maintain the convex hulls (e.g., by a path-copying method) [19, 34] so that after the merge we still keep $\text{CH}_r(v, w)$. In this way, we have computed $\text{CH}_r(v, w')$ and we then proceed on the parent of w' . We do this until we reach the root. As such, the total time and extra space for computing the convex hulls for a leaf v is $O(\log^2 m)$, and the total time and space for doing this for all leaves is $O(m \log^2 m)$. Symmetrically, for each leaf v , which stores a point p_i of Q_e , for each ancestor u of v , we compute the convex hull $\text{CH}_l(v, u)$ of all points p_h, p_{h+1}, \dots, p_i , whether p_h is the point in the leftmost leaf of the subtree at u . Computing the convex hulls $\text{CH}_l(v, u)$ for all ancestors u for all leaves v can be done in $O(m \log^2 m)$ in a similar way as above. In addition, we construct a lowest common ancestor (LCA) data structure on the tree T_e in $O(m)$ time so that the LCA of any two query nodes of T_e can be found in $O(1)$ time [5, 26]. The total preprocessing time for constructing the tree T_e as above is $O(m \log^2 m)$. Recall that the sum of m for all outside-hull segment query data structures is $O(n \log n)$. Hence, the total preprocessing time of all data structures is $O(n \log^3 n)$.

Now consider the third step of the query algorithm. Suppose i^* and j^* are known. The problem is to compute the point p_s of $Q_e[i^*, j^*]$ closest to the supporting line $\ell(s)$ of s . Let u and v be the two leaves of T_e storing the two points p_{i^*} and p_{j^*} , respectively. Let w be the lowest common ancestor of u and v . Let u' and v' be the left and right children of w , respectively. It is not difficult to see that the convex hull of $\text{CH}_r(u, u')$ and $\text{CH}_l(v, v')$ is the convex hull of $Q_e[i^*, j^*]$. As such, to find p_s , it suffices to compute the vertex of $\text{CH}_r(u, u')$ closest to $\ell(s)$ and the vertex of $\text{CH}_l(v, v')$ closest to $\ell(s)$, and among the two points, return the one closer to $\ell(s)$ as p_s . To implement the algorithm, finding w can be done in $O(1)$ time using the LCA data structure [5, 26]. To find the closest vertex of $\text{CH}_r(u, u')$ to $\ell(s)$, recall that the preprocessing computes a balanced binary search tree (maintained by a persistent tree), denoted by $T_r(u, u')$, for maintaining $\text{CH}_r(u, u')$. We apply a *search lemma* of Chan and Zheng (Lemma A.1 [10]) on the tree $T_r(u, u')$. Indeed, the problem is equivalent to

finding the predecessor of the slope of $\ell(s)$ among the slopes of the edges of $\text{CH}_r(u, u')$. Using the search lemma, we can find the vertex of $\text{CH}_r(u, u')$ closest to $\ell(s)$ using $O(1 - \Delta\Phi)$ comparisons. Similarly, the vertex of $\text{CH}_l(v, v')$ closest to $\ell(s)$ can be found using $O(1 - \Delta\Phi)$ comparisons. In this way, p_s can be computed using $O(1 - \Delta\Phi)$ comparisons.

In summary, with $O(n \log^3 n)$ time preprocessing, the third step of the query algorithm for all $O(n^{4/3})$ queries can be done using a total of $O(n^{4/3})$ comparisons (recall that the sum of $-\Delta\Phi$ in the entire algorithm is $O(n \log n)$).

Summary. Combining the three steps discussed above, all $O(n^{4/3})$ outside-hull segment queries can be solved using $O(n^{4/3})$ comparisons. Recall that the above only discussed the query on the data structure for a single edge e of the convex hull of Q . As the first procedure of the query, we need to find the vertex of $\text{CH}(Q)$ closest to the supporting line of s . For this, we can maintain the convex hull $\text{CH}(Q)$ by a balanced binary search tree and apply the search lemma of Chan and Zheng (Lemma A.1 [10]) in the same way as discussed above. As such, this procedure for all queries uses $O(n^{4/3})$ comparisons. The second procedure of the query is to find the edge of $\text{CH}(Q)$ intersecting the line through one of the endpoints of s and perpendicular to s . This operation is essentially to find a predecessor of the above endpoint of s on the vertices of the lower hull of $\text{CH}(Q)$. Therefore, we can also apply the search lemma of Chan and Zheng, and thus this procedure for all queries also uses $O(n^{4/3})$ comparisons. As such, we can solve all $O(n^{4/3})$ outside-hull segment queries using $O(n^{4/3})$ comparisons, or alternatively, we have an algebraic decision tree of height $O(n^{4/3})$ that can solve all $O(n^{4/3})$ queries.

3.4 Solving the subproblems $T(n^{2/3}, n^{1/3})$

We now tackle the second challenge, i.e., solve each subproblem $T(n^{2/3}, n^{1/3})$ in recurrence (5) using $O(n^{2/3})$ comparisons, or solve all $O(n^{2/3})$ subproblems $T(n^{2/3}, n^{1/3})$ in (5) using $O(n^{4/3})$ comparisons.

Recall that P is the set of n points and S is the set of n segments for the original problem in recurrence (5). If the closest point of a segment $s \in S$ to P is an endpoint of s , then finding the closest point of s in P can be done using the Voronoi diagram of P . Hence, it suffices to find the first point of P hit by s if we drag s along the directions perpendicularly to s . There are two such directions, but in the following discussion we will only consider dragging s along the upward direction perpendicularly to s (recall that s is not vertical due to our general position assumption) and let p_s be the first point of P hit by s , since the algorithm for the downward direction is similar. As such, the goal is to compute p_s for each segment $s \in S$.

For notational convenience, let $m = n^{1/3}$ and thus we want to solve $T(m^2, m)$ using $O(m^2)$ comparisons. More specifically, we are given m points and m^2 segments; the problem is to compute for each segment s the point p_s (with respect to the m points, i.e., the first point hit by s if we drag s along the upward direction perpendicular to s). Our goal is to solve all $O(m^2)$ segment dragging queries using $O(m^2)$ comparisons after certain preprocessing. In what follows, we begin with the preprocessing algorithm.

Preprocessing. For two sets $A = \{a_1, a_2, \dots, a_m\}$ and $B = \{b_1, b_2, \dots, b_m\}$ of m points each, we say that they have the same *order type* if for each i , the index order of the points of A sorted around a_i is the same as that of the points of B sorted around b_i (equivalently, in the dual plane, the index order of the dual lines intersecting the dual line of a_i is the same as that of the dual lines intersecting the dual line of b_i); the concept has been used elsewhere,

e.g., [3, 10, 23]. Because constructing the arrangement of a set of m lines can be computed in $O(m^2)$ time [15], we can decide whether two sets A and B have the same order type in $O(m^2)$ time, e.g., simply follow the incremental line arrangement construction algorithm [15]. We actually build an algebraic decision tree T_D so that each node of T_D corresponds to a comparison of the algorithm. As such, the height of T_D is $O(m^2)$ and T_D has $2^{O(m^2)}$ leaves, each of which corresponds to an order type (note that the number of distinct order types is at most m^{6m} [24], but here using $2^{O(m^2)}$ as an upper bound suffices for our purpose).

Let Q be a set of m points whose order type corresponds to a leaf v of T_D . Let K_Q denote the set of the slopes of all lines through pairs of points of Q . Note that $|K_Q| = O(m^2)$. We sort the slopes of K_Q . Consider two consecutive slopes k_1 and k_2 of the sorted K_Q . In the dual plane, for any vertical line ℓ whose x -coordinate is between k_1 and k_2 , ℓ intersects the dual lines of all points of Q in the same order (because k_1 and k_2 respectively are x -coordinates of two consecutive vertices of the arrangement of dual lines). This implies the following in the primal plane. Consider any two lines ℓ_1 and ℓ_2 whose slopes are between k_1 and k_2 such that all points of Q are above ℓ_i for each $i = 1, 2$. Then, the order of the lines of Q by their distances to ℓ_1 is the same as their order by the distances to ℓ_2 . However, if we project all points of Q onto ℓ_1 and ℓ_2 , the orders of their projections along the two lines may not be the same. To solve our problem, we need a stronger property that the above projection orders are also the same. To this end, we further refine the order type as follows.

For each pair of points q_i and q_j of Q , we add the slope of the line perpendicular to the line through p and q to K_Q . As such, the size of K_Q is still $O(m^2)$. Although K_Q has $O(m^2)$ values, all these values are defined by the m points of Q . Using this property, K_Q can be sorted using $O(m^2)$ comparisons [10, 22].

For two sets $Q = \{q_1, q_2, \dots, q_m\}$ and $Q' = \{q'_1, q'_2, \dots, q'_m\}$ of m points each with the same order type, we say that they have the same *refined order type* if the order of K_Q is the same as that of $K_{Q'}$, i.e., the slope of the line through q_i and q_j (resp., the slope of the line perpendicular to the line through q_i and q_j) is in the k -th position of the sorted list of K_Q if and only if the slope of the line through q'_i and q'_j (resp., the slope of the line perpendicular to the line through q'_i and q'_j) is in the k -th position of the sorted list of $K_{Q'}$. We further enhance the decision tree T_D by attaching a new decision tree at each leaf v of T_D for sorting K_Q (recall that K_Q can be sorted using $O(m^2)$ comparisons, i.e., there is an algebraic decision tree of height $O(m^2)$ that can sort K_Q), where Q is a set of m points whose order type corresponds to v . We still use T_D to refer to the new tree. The height of T_D is still $O(m^2)$.

We perform the following preprocessing work for each leaf v of T_D . Let Q be a set of m points that has the refined order type of v . We associate Q with v , compute and sort K_Q , and store the sorted list using a balanced binary search tree. Let k_1 and k_2 be two consecutive slopes in the sorted list of K_Q . Consider a line ℓ whose slope is in (k_1, k_2) such that ℓ is below all points of Q . We project all points perpendicularly onto ℓ . According to the definition of K_Q , the order of the projections is fixed for all such lines ℓ whose slopes are in (k_1, k_2) . Without loss of generality, we assume that ℓ is horizontal. Let q_1, q_2, \dots, q_m denote the points of Q ordered by their projections on ℓ from left to right and we maintain the sorted list in a balanced binary search tree. For each pair (i, j) with $1 \leq i \leq j \leq m$, let $Q[i, j] = \{q_i, q_{i+1}, \dots, q_j\}$; we sort all points of $Q[i, j]$ by their distances to ℓ and store the sorted list in a balanced binary search tree. As such, the time we spent on the preprocessing at v is $O(m^5 \log m)$.

Since T_D is a decision tree of height $O(m^2)$, the number of leaves of T_D is $2^{O(m^2)}$. Therefore, the total preprocessing time for all leaves of T_D is $m^5 \log m \cdot 2^{O(m^2)}$. As T_D can be built in $O(2^{\text{poly}(m)})$ time, the total preprocessing time is bounded by $O(2^{\text{poly}(m)})$.

Solving a subproblem $T(m^2, m)$. Consider a subproblem $T(m^2, m)$ with a set P' of m points and a set S' of m^2 segments. We arbitrarily assign indices to points of P' as $\{p_1, p_2, \dots, p_m\}$. By using the decision tree T_D , we first find the leaf v of T_D that corresponds to the refined order type of P' , which can be done using $O(m^2)$ comparisons as the height of T_D is $O(m^2)$. Let $Q = \{q_1, q_2, \dots, q_m\}$ be the set of m points associated with v . Below we find for each segment $s \in S'$ its point p_s in P' . Let ℓ denote the supporting line of s .

We first find two consecutive slopes k_1 and k_2 in $K_{P'}$ such that the slope of ℓ is in $[k_1, k_2)$. Note that we do not explicitly have the sorted list of $K_{P'}$, but recall that we have the sorted list of K_Q stored at v . Since P' and Q have the same refined order type, a slope defined by two points p_i and p_j is in the k -th position of $K_{P'}$ if and only if the slope defined by two points q_i and q_j is in the k -th position of K_Q . Hence, we can search K_Q instead; however, whenever we need to use a slope whose definition involves a point $q_i \in Q$, we use p_i instead. In this way, we could find k_1 and k_2 using $O(\log m)$ comparisons. Further, since we have the balanced binary search tree storing K_Q , we can apply the search lemma of Chan and Zheng [10] as discussed above to find k_1 and k_2 using only $O(1 - \Delta\Phi)$ comparisons.

Without loss of generality, we assume that s is horizontal. Let a and b denote the left and right endpoints of s , respectively. Suppose we project all points of P' perpendicularly onto ℓ . Let $p_{\pi(1)}, p_{\pi(2)}, \dots, p_{\pi(m)}$ be the sorted list following their projections along ℓ from left to right, where $\pi(i)$ is the index of the i -th point in this order. We wish to find the index i such that a is between $p_{\pi(i-1)}$ and $p_{\pi(i)}$ as well as the index j such that b is between $p_{\pi(j)}$ and $p_{\pi(j+1)}$. To this end, we do the following. Since P' and Q have the same refined order type, if we project all points of Q perpendicularly onto ℓ , then $q_{\pi(1)}, q_{\pi(2)}, \dots, q_{\pi(m)}$ is the sorted list following their projections along ℓ with the same permutation $\pi(\cdot)$. Hence, to find the index i , we can query a in the sorted list $q_{\pi(1)}, q_{\pi(2)}, \dots, q_{\pi(m)}$, which is maintained at v due to our preprocessing, but again, whenever we need to use a point $q_{\pi(k)}$, we use $p_{\pi(k)}$ instead. Using the search lemma of Chan and Zheng as discussed before, we can find i using $O(1 - \Delta\Phi)$ comparisons. Similarly, the index j can be found using $O(1 - \Delta\Phi)$ comparisons.

Let $P'_\ell[i, j] = \{p_{\pi(i)}, p_{\pi(i+1)}, \dots, p_{\pi(j)}\}$. By the definitions of i and j , the point p_s we are looking for is the point of $P'_\ell[i, j]$ closest to the line ℓ . To find p_s , we do the following. Let ℓ' be a line parallel to ℓ but is below all points of P' and Q . Let $P'_{\ell'}[i, j]$ denote the sorted list of $P'_\ell[i, j]$ ordered by their distances from ℓ' . Then, p_s can be found by binary search on $P'_{\ell'}[i, j]$. Since P' and Q have the same refined order type, we can instead do binary search on $Q_{\ell'}[i, j]$, whose order is consistent with that of $Q[i, j]$, which is maintained at v due to the preprocessing. As such we can search $Q[i, j]$, but again whenever the algorithm wants to use a point $q_k \in Q[i, j]$, we will use p_k instead to perform a comparison. Using the search lemma of Chan and Zheng, we can find p_s using $O(1 - \Delta\Phi)$ comparisons.

The above shows that p_s can be found using $O(1 - \Delta\Phi)$ comparisons. Therefore, doing this for all $O(m^2)$ segments can be done using $O(m^2 - \Delta\Phi)$ comparisons.

In summary, with $O(2^{\text{poly}(n)})$ time preprocessing, we can solve each subproblem $T(n^{2/3}, n^{1/3})$ using $O(n^{2/3})$ comparisons without considering the term $-\Delta\Phi$, whose total sum in the entire algorithm of recurrence (5) is $O(n \log n)$.

3.5 Wrapping things up

The above proves Lemma 1, and thus $T(n, n)$ in (5) can be bounded by $O(n^{4/3})$ after $O(2^{\text{poly}(n)})$ time preprocessing as discussed before. Equivalently, $T(b, b)$ in (4) can be bounded by $O(b^{4/3})$ after $O(2^{\text{poly}(b)})$ time preprocessing. Notice that the preprocessing work is done only once and for all subproblems $T(b, b)$ in (4). Since $b = (\log \log \log n)^3$, we have $2^{\text{poly}(b)} = O(n)$. As such, $T(n, n)$ in (4) solves to $O(n^{4/3})$ and we have the following.

► **Theorem 7.** *Given a set of n points and a set of n segments in the plane, we can find for each segment its closest point in $O(n^{4/3})$ time.*

The following solves the asymmetric case of the problem (see the full paper for details).

► **Corollary 8.** *Given a set of n points and a set of m segments in the plane, we can find for each segment its closest point in $O(n^{2/3}m^{2/3} + n \log n + m \log^2 n)$ time.*

4 The online query problem

Let P be a set of n points in the plane. We wish to build a data structure so that the point of P closest to a query segment can be computed efficiently.

4.1 The line query problem

We first consider the special case where the query segment is a line ℓ . The main idea is to adapt the simplex range searching data structures [9, 29, 30] (which works in any fixed dimensional space; but for our purpose it suffices to only consider half-plane range counting queries in the plane). Each of these half-plane range counting query data structures [9, 29, 30] defines canonical subsets of P and usually only maintains the cardinalities of them. To solve our problem, roughly speaking, the change is that we compute and maintain the convex hulls of these canonical subsets, which increases the space by a factor proportional to the height of the underlying trees (which is $O(\log n)$ for the data structures in [9, 30] and is $O(\log \log n)$ for the one in [29]). To answer a query, we follow the similar algorithms as half-plane range counting queries on these data structures. The difference is that for certain canonical subsets, we do binary search on their convex hulls to find their closest vertices to the query line, which does not intersect these convex hulls (in the half-plane range counting query algorithms only the cardinalities of these canonical subsets are added to a total count). This increases the query time by a logarithmic factor comparing to the original half-plane range counting query algorithms. We manage to reduce the additional logarithmic factor using fractional cascading [14] on the data structures of [9, 30] because each node in the underlying trees of these data structures has $O(1)$ children. Some extra efforts are also needed to achieve the claimed performance. Finally, the trade-off is obtained by combining these results with cuttings in the dual space.

In the rest of this subsection, we present a randomized result based on Chan's partition tree [9] while the deterministic results are given in the full paper.

A randomized result based on Chan's partition tree [9]. We first review Chan's partition tree [9]. Chan's partition tree T for the point set P is a tree structure by recursively subdividing the plane into triangles. Each node v of T is associated with a triangle $\Delta(v)$, which is the entire plane if v is the root. If v is an internal node, it has $O(1)$ children, whose associated triangles form a disjoint partition of $\Delta(v)$. Let $P(v) = P \cap \Delta(v)$, i.e., the subset of points of P in $\Delta(v)$. For each internal node v , the cardinality $|P(v)|$ is stored at v . If v is a leaf, then $|P(v)| = O(1)$ and $P(v)$ is explicitly stored at v . The height of T is $O(\log n)$ and the space of T is $O(n)$. Let $\alpha(T)$ denote the maximum number of triangles $\Delta(v)$ among all nodes v of T crossed by any line in the plane. Given P , Chan's randomized algorithm can compute T in $O(n \log n)$ expected time such that $\alpha(T) = O(\sqrt{n})$ holds with high probability.

To solve our problem, we modify the tree T as follows. For each node v , we compute the convex hull $\text{CH}(v)$ of $P(v)$ and store $\text{CH}(v)$ at v . This increases the space to $O(n \log n)$, but the preprocessing time is still bounded by $O(n \log n)$.

Given a query line ℓ , our goal is to compute the point of P closest to ℓ . We only discuss how to find the closest point of ℓ among all points of P below ℓ since the other case is similar. Starting from the root of T , consider a node v . We assume that ℓ crosses $\Delta(v)$, which is true initially when v is the root. For each child u of v , we do the following. If ℓ crosses $\Delta(u)$, then we proceed on u recursively. Otherwise, if $\Delta(u)$ is below ℓ , we do binary search on the convex hull $\text{CH}(u)$ to find in $O(\log n)$ time the closest point to ℓ among the vertices of $\text{CH}(u)$ and keep the point as a candidate. Since each internal node of T has $O(1)$ children, the algorithm eventually finds $O(\alpha(T))$ candidate points and among them we finally return the one closest to ℓ as our solution. The total time of the algorithm is $O(\alpha(T) \cdot \log n)$.

To further reduce the query time, we observe that all nodes v whose triangles $\Delta(v)$ are crossed by ℓ form a subtree T_ℓ of T containing the root. This is because if the triangle $\Delta(v)$ of a node v is crossed by ℓ , then the triangle $\Delta(u)$ is also crossed by ℓ for any ancestor u of v . In light of the observation, we can further reduce the query algorithm time to $O(\alpha(T) + \log n)$ by constructing a fractional cascading structure [14] on the convex hulls of all nodes of T so that if a tangent to the convex hull at a node v is known, then the tangents of the same slope to the convex hulls of the children of v can be found in constant time. The total time for constructing the fractional cascading structure is linear in the total size of all convex hulls, which is $O(n \log n)$. With the fractional cascading structure, we only need to perform binary search on the convex hull at the root and then spend only $O(1)$ time on each node of T_ℓ and each of their children. As such, the query time becomes $O(\alpha(T) + \log n)$, which is bounded by $O(\sqrt{n})$ with high probability.

► **Lemma 9.** *Given a set P of n points in the plane, we can build a data structure of $O(n \log n)$ space in $O(n \log n)$ expected time such that for any query line its closest point in P can be computed in $O(\sqrt{n})$ time with high probability.*

4.2 The segment query problem

To answer the general segment queries, the main idea is essentially the same as the line case with one change: whenever we compute the convex hull for a canonical subset of P (e.g., the subset $P(v)$ for a node v in a partition tree) for outside-hull line queries, we instead build the BS data structure [7] for outside-hull segment queries. Because the fractional cascading does not help anymore, the query time in general has an additional logarithmic factor, with the exception that when using Chan's partition tree [9] we still manage to bound the query time by $O(\sqrt{n})$ due to some nice properties of the partition tree.

In the rest of this subsection, we present a randomized result based on Chan's partition tree [9] while the deterministic results are given in the full paper.

The randomized result. For our randomized result using Chan's partition tree [9] (by modifying the one in Section 4.1), for each node v of the partition tree T , we construct the BS data structure for $P(v)$. The total space is still $O(n \log n)$. For the preprocessing time, constructing the BS data structure can be done in linear time if we know the Voronoi diagram of $P(v)$. For this, as discussed in Section 3.1, we can process all nodes of T in a bottom-up manner and using the linear-time Voronoi diagram merge algorithm of Kirkpatrick [27]. As such, constructing the BS data structures for all nodes of T can be done in $O(n \log n)$ time in total. Therefore, the total preprocessing time is still $O(n \log n)$ expected time.

The query algorithm follows the same scheme as before but instead use the BS algorithm to answer outside-hull segment queries. The total query time becomes $O(\sqrt{n} \log n)$ with high probability. In fact, due to certain properties of Chan's partition tree, the time is bounded by $O(\sqrt{n})$, as shown in the following lemma (similar idea was used elsewhere, e.g., [11]).

► **Lemma 10.** *The query time is bounded by $O(\sqrt{n})$ with high probability.*

As such, we obtain the following result.

► **Lemma 11.** *Given a set P of n segments in the plane, we can build a data structure of $O(n \log n)$ space in $O(n \log n)$ expected time such that for any query segment its closest point in P can be computed in $O(\sqrt{n})$ time with high probability.*

As discussed in Section 1, another randomized solution of complexity $O(n^{4/3}, n^{4/3}, n^{1/3})$ can be obtained using Chan’s randomized techniques [8] and Chan and Zheng’s recent randomized result on triangle range counting [10]. Refer to the full paper for details. We thank an anonymous reviewer for suggesting the idea.

References

- 1 Pankaj K. Agarwal. Partitioning arrangements of lines II: Applications. *Discrete and Computational Geometry*, 5:533–573, 1990.
- 2 Pankaj K. Agarwal and Micha Sharir. Applications of a new space-partitioning technique. *Discrete and Computational Geometry*, 9:11–38, 1993.
- 3 Boris Aronov, Mark de Berg, Jean Cardinal, Esther Ezra, John Iacono, and Micha Sharir. Subquadratic algorithms for some 3Sum-Hard geometric problems in the algebraic decision tree model. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC)*, pages 3:1–3:15, 2021.
- 4 Reuven Bar-Yehuda and Sergio Fogel. Variations on ray shootings. *Algorithmica*, 11:133–145, 1994.
- 5 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pages 88–94, 2000.
- 6 Sergei Bespamyatnikh. Computing closest points for segments. *International Journal of Computational Geometry and Application*, 13:419–438, 2003.
- 7 Sergei Bespamyatnikh and Jack Snoeyink. Queries with segments in Voronoi diagrams. *Computational Geometry: Theory and Applications*, 16:23–33, 2000.
- 8 Timothy M. Chan. Geometric applications of a randomized optimization technique. *Discrete and Computational Geometry*, 22:547–567, 1999.
- 9 Timothy M. Chan. Optimal partition trees. *Discrete and Computational Geometry*, 47:661–690, 2012.
- 10 Timothy M. Chan and Da Wei Zheng. Hopcroft’s problem, log-star shaving, 2D fractional cascading, and decision trees. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 190–210, 2022.
- 11 Timothy M. Chan and Da Wei Zheng. Simplex range searching revisited: How to shave logs in multi-level data structures. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1493–1511, 2023.
- 12 Bernard Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3:205–221, 1988.
- 13 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete and Computational Geometry*, 9:145–158, 1993.
- 14 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.
- 15 Bernard Chazelle, Leonidas J. Guibas, and D.T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.
- 16 Siu Wing Cheng and Ravi Janardan. Algorithms for ray-shooting and intersection searching. *Journal of Algorithms*, 13:670–692, 1992.
- 17 Richard Cole and Chee-Keng Yap. Geometric retrieval problems. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 112–121, 1983.

- 18 Ovidiu Daescu, Ningfang Mi, Chan-Su Shin, and Alexander Wolff. Farthest-point queries with geometric and combinatorial constraints. *Computational Geometry: Theory and Applications*, 33:174–185, 2006.
- 19 James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38:86–124, 1989.
- 20 Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. The complexity and construction of many faces in arrangement of lines and of segments. *Discrete and Computational Geometry*, 5:161–196, 1990.
- 21 Jeff Erickson. New lower bounds for Hopcroft’s problem. *Discrete and Computational Geometry*, 16:389–418, 1996.
- 22 Michael L. Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1:355–361, 1976.
- 23 Jacob E. Goodman and Richard Pollack. Multidimensional sorting. *SIAM Journal on Computing*, 12:484–507, 1983.
- 24 Jacob E. Goodman and Richard Pollack. Upper bounds for configurations and polytopes in R^d . *Discrete and Computational Geometry*, pages 219–227, 1986.
- 25 Partha P. Goswami, Sandip Das, and Subhas C. Nandy. Triangular range counting query in 2D and its application in finding k nearest neighbors of a line segment. *Computational Geometry: Theory and Applications*, 29:163–175, 2004.
- 26 Dov Harel and Robert E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13:338–355, 1984.
- 27 David G. Kirkpatrick. Efficient computation of continuous skeletons. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 18–27, 1979.
- 28 D. T. Lee and Y. T. Ching. The power of geometric duality revisited. *Information Processing Letters*, 21:117–122, 1985.
- 29 Jiří Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8:315–334, 1992.
- 30 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete and Computational Geometry*, 10:157–182, 1993.
- 31 Pinaki Mitra and Bidyut B. Chaudhuri. Efficiently computing the closest point to a query line. *Pattern Recognition Letters*, 19:1027–1035, 1998.
- 32 Asish Mukhopadhyay. Using simplicial partitions to determine a closest point to a query line. *Pattern Recognition Letters*, 24:1915–1920, 2003.
- 33 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23:166–204, 1981.
- 34 Neil Sarnak and Robert E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29:669–679, 1986.
- 35 Haitao Wang. Algorithms for subpath convex hull queries and ray-shooting among segments. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG)*, pages 69:1–69:14, 2020.

Lower Bounds for Set-Blocked Clauses Proofs

Emre Yolcu   

Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We study propositional proof systems with inference rules that formalize restricted versions of the ability to make assumptions that hold without loss of generality, commonly used informally to shorten proofs. Each system we study is built on resolution. They are called BC^- , RAT^- , SBC^- , and GER^- , denoting respectively blocked clauses, resolution asymmetric tautologies, set-blocked clauses, and generalized extended resolution – all “without new variables.” They may be viewed as weak versions of extended resolution (ER) since they are defined by first generalizing the extension rule and then taking away the ability to introduce new variables. Except for SBC^- , they are known to be strictly between resolution and extended resolution.

Several separations between these systems were proved earlier by exploiting the fact that they effectively simulate ER. We answer the questions left open: We prove exponential lower bounds for SBC^- proofs of a binary encoding of the pigeonhole principle, which separates ER from SBC^- . Using this new separation, we prove that both RAT^- and GER^- are exponentially separated from SBC^- . This completes the picture of their relative strengths.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases proof complexity, separations, resolution, extended resolution, blocked clauses

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.59

Related Version *arXiv Version*: <https://arxiv.org/abs/2401.11266>

Funding This material is based upon work supported by the National Science Foundation under grant CCF-2015445.

Acknowledgements I thank Sam Buss, Marijn Heule, Jakob Nordström, and Ryan O’Donnell for useful discussions. I thank Jeremy Avigad, Ryan O’Donnell, and Bernardo Subercaseaux for feedback on an earlier version of the paper. Finally, I thank the STACS reviewers for their highly detailed comments and suggestions.

1 Introduction

When writing proofs informally, it is sometimes convenient to make assumptions that hold “without loss of generality.” For instance, if we are proving a statement about real numbers x and y , we might assume without loss of generality that $x \geq y$ and continue the proof under this additional assumption. Such an assumption requires justification, for instance by arguing that the two variables are interchangeable in the statement being proved. Assumptions of this kind are not essential to proofs but they simplify or shorten the presentation.

We study propositional proof systems¹ with inference rules that allow making such assumptions. Extended resolution [21] (equivalently, Extended Frege [5]) already simulates this kind of reasoning; however, it presumably does more, and its strength is poorly understood. We thus focus on “weak” systems built on top of resolution [1, 20] and lacking the ability to introduce any new variables, while still being able to reason without loss of generality. Each system relies on a polynomial-time verifiable syntactic condition to automatically justify the assumption being made, and the exact form of this condition determines the strength of the proof system. The systems are defined by first generalizing the extension rule and then taking away the ability to introduce new variables, so, for lack of a better term, we

¹ Throughout the rest of this paper, by “proof” we mean a proof of unsatisfiability (i.e., a refutation).

will refer to these systems collectively as “weak extended resolution systems” in this section. Although we are referring to them as weak, some variants are surprisingly strong in that they admit polynomial-size proofs of the pigeonhole principle, bit pigeonhole principle, parity principle, clique-coloring principle, and Tseitin tautologies, as well as being able to undo (with polynomial-size derivations) the effects of or-ification, xor-ification, and lifting with indexing gadgets [2]. In this paper, we study the relative strengths of several variants of those systems and answer the questions left open in previous work [23, Section 1.4].

1.1 Motivation

Our interest in weak extended resolution systems is rooted in two different areas: proof complexity and satisfiability (SAT) solving.

1.1.1 Proof complexity

Proof complexity is concerned with the sizes of proofs in propositional proof systems. The notion of a proof system as accepted in proof complexity is rather general – it covers not only the “textbook” deductive systems for propositional logic but also several systems that capture different forms of mathematical reasoning. For instance, the widely studied proof systems of cutting planes [6] and polynomial calculus [3] utilize simple forms of geometric and algebraic reasoning, respectively. Weak extended resolution systems are somewhat similar, although they do not originate from any specific branch of mathematics. Instead, they capture the pervasive technique of reasoning without loss of generality, often used to shorten proofs. Since proof complexity is concerned with proof size, the limits of the degree of brevity achievable by this form of reasoning is a natural question from the perspective of proof complexity.

Moreover, the upper bounds proved by Buss and Thapen [2] show that many of the usual “hard” combinatorial principles used for proof complexity lower bounds are easy to prove in certain weak extended resolution systems. In other words, a modest amount of the ability to reason without loss of generality lends surprising strength to even a system as weak as resolution, and the full strength of extended resolution is not required for the combinatorial principles previously mentioned. Thus, many of the existing separations between extended resolution and the commonly studied proof systems can be attributed to the fact that extended resolution can reason without loss of generality while the other systems cannot. Searching for principles that separate extended resolution from the weak extended resolution systems will help us better understand other facets of the strength of extended resolution.

1.1.2 SAT solving

Another motivation for studying the weak extended resolution systems is their potential usefulness for improvements in SAT solvers, which are practical implementations of propositional theorem provers that determine whether a given formula in conjunctive normal form is satisfiable. When a solver claims unsatisfiability, it is expected to produce a proof that can be used to verify the claim efficiently. Modern SAT solvers, which are based on conflict-driven clause learning (CDCL) [17], essentially search for resolution proofs. Consequently, the well-known exponential lower bounds against resolution (e.g., [7, 22]) imply exponential lower bounds against the runtimes of CDCL-based solvers. To overcome the limitations of resolution, SAT solvers are forced to go beyond CDCL.

Many of the current solvers employ “inprocessing” techniques [11], which support inferences of the kind that we study in this paper. These techniques are useful in practice; however, they are implemented as ad hoc additions to CDCL. Weak extended resolution

systems hold the potential for improving SAT solvers in a more principled manner (e.g., through the development of a solving paradigm that corresponds to one of those systems in a manner similar to how CDCL corresponds to resolution). In comparison, proof systems such as cutting planes, polynomial calculus, DNF resolution [13], or Frege [5] appear more difficult to take advantage of, at least in part due to their richer syntax. When dealing only with clauses, it becomes possible to achieve highly efficient constraint propagation, which is an important reason for the speed of CDCL-based solvers. Extended resolution also works only with clauses; however, there are currently no widely applicable heuristics for introducing new variables during proof search. Weak extended resolution systems are relatively strong despite using only clauses without new variables. Thus, those systems are promising for practical proof search algorithms.

Earlier works [10, 8] showed that solvers based on certain weak extended resolution systems can automatically discover small proofs of some formulas that are hard for resolution, such as the pigeonhole principle and the mutilated chessboard principle. Still, those solvers fall behind CDCL-based solvers on other classes of formulas. Moreover, there appears to be a tradeoff when choosing a system for proof search: stronger systems enable smaller proofs; however, proof search in such systems is costlier with respect to proof size. To be able to choose the ideal system for proof search (e.g., the weakest system that is strong enough for one’s purposes), it is important to understand the relative strengths of the systems in question. This paper is a step towards that goal.

1.2 Background

We briefly review some background, deferring formal definitions to Section 2. For comprehensive overviews of related work, see Buss and Thapen [2] and Yolcu and Heule [23].

The proof systems we study are based on the notion of “redundancy.” A clause is *redundant* with respect to a formula if it can be added to or removed from the formula without affecting satisfiability. Redundancy is a generalization of logical implication: if $\Gamma \models C$ then the clause C is redundant with respect to the set Γ of clauses;² however, the converse is not necessarily true. When proving unsatisfiability, deriving redundant clauses corresponds to making assumptions that hold without loss of generality.³ To ensure that proofs can be checked in polynomial time, we work with restricted versions of redundancy that rely on syntactic conditions.

Possibly the simplest interesting version is blockedness [14, 15]. We say a clause C is *blocked* with respect to a set Γ of clauses if there exists a literal $p \in C$ such that all possible resolvents of C on p against clauses from Γ are tautological (i.e., contain a literal and its negation). Kullmann [16] showed that blocked clauses are special cases of redundant clauses and thus considered an inference rule that, given a formula Γ , allows us to extend Γ with a clause that is blocked with respect to Γ . This rule, along with resolution, gives the proof system called *blocked clauses* (BC). It is apparent from the definition of a blocked clause that deleting clauses from Γ enlarges the set of clauses that are blocked with respect to Γ . With this observation, Kullmann defined a strengthening of BC called *generalized extended resolution* (GER) that allows temporary deletion of clauses from Γ . Later works [11, 12] defined more general classes of redundant clauses and proof systems based on them, called *resolution asymmetric tautologies* (RAT) and *set-blocked clauses* (SBC). Both RAT and SBC

² We use “set of clauses” and “formula” interchangeably.

³ When refuting a formula Γ , deriving a redundant clause C may be viewed as stating the following: “If there exists an assignment satisfying Γ , then there also exists an assignment satisfying both Γ and C , so without loss of generality we can assume that C holds.”

use relaxed versions of blocked clauses: RAT changes the word “tautological” in the definition of a blocked clause, and, in a sense, SBC considers possible resolvents on more than a single literal.

As defined, BC simulates extended resolution since extension clauses can be added in sequence as blocked clauses if we are allowed to introduce new variables (see [16, Section 6]). Thus, we disallow new variables to weaken the systems. A proof of Γ is *without new variables* if it contains only the variables that already occur in Γ . We denote a proof system variant that disallows new variables with the superscript “ $-$ ” (e.g., BC^- is BC without new variables). We are concerned in this paper with the strengths of the systems RAT^- , SBC^- , and GER^- , each of which generalizes BC^- in different ways.

As a technical side note, the systems we study are unusual in the following respects: They are not monotonic, since a clause redundant with respect to Γ is not necessarily redundant with respect to $\Gamma' \supseteq \Gamma$. This causes *deletion* (i.e., the ability to delete clauses in the middle of a proof) to increase the strength of those systems. It also requires one to pay attention to the order of inferences when proving upper bounds. Additionally, they are not a priori closed under restrictions, which means that extra care is required when proving lower bounds against them. More specifically, a proof system P that simulates BC^- is not closed under restrictions unless P also simulates extended resolution [2, Theorem 2.4]. It follows from earlier lower bounds [16, 2] and Section 3 in this paper that BC^- , RAT^- , SBC^- , and GER^- are not closed under restrictions.

1.3 Results

This work follows up on Yolcu and Heule [23], which proved separations between the different generalizations of BC^- by exploiting the fact that, although the systems cannot introduce new variables, they nevertheless effectively simulate [18] extended resolution. Their strategy uses so-called “guarded extension variables,” where we consider systems P and Q that both effectively simulate a strong system R , and we incorporate extension variables into formulas in a guarded way that allows P to simulate an R -proof while preventing Q from making any meaningful use of the extension variables to achieve a speedup. This allows using, as black-box, a separation of R from Q to separate P from Q . For more details about this strategy, we refer the reader to Yolcu and Heule [23, Section 1.3].

In this paper, we prove the following results, where each formula indexed by n has $n^{O(1)}$ variables and $n^{O(1)}$ clauses. Figure 1 summarizes the proof complexity landscape around BC^- after these results.

We first show exponential lower bounds for SBC^- proofs of a binary encoding of the pigeonhole principle called the “bit pigeonhole principle,” defined in Section 3. (Note that the usual unary encoding of the pigeonhole principle admits polynomial-size proofs in SBC^- [23, Lemma 7.1].)

► **Theorem 1.** *The bit pigeonhole principle BPHP_n requires SBC^- proofs of size $2^{\Omega(n)}$.*

We then show, using constructions that incorporate guarded extension variables into BPHP_n , that RAT^- and GER^- are both exponentially separated from SBC^- .

► **Theorem 2.** *There exists an infinite sequence $(\Gamma_n)_{n=1}^\infty$ of unsatisfiable formulas such that Γ_n admits RAT^- proofs of size $n^{O(1)}$ but requires SBC^- proofs of size $2^{\Omega(n)}$.*

► **Theorem 3.** *There exists an infinite sequence $(\Delta_n)_{n=1}^\infty$ of unsatisfiable formulas such that Δ_n admits GER^- proofs of size $n^{O(1)}$ but requires SBC^- proofs of size $2^{\Omega(n)}$.*

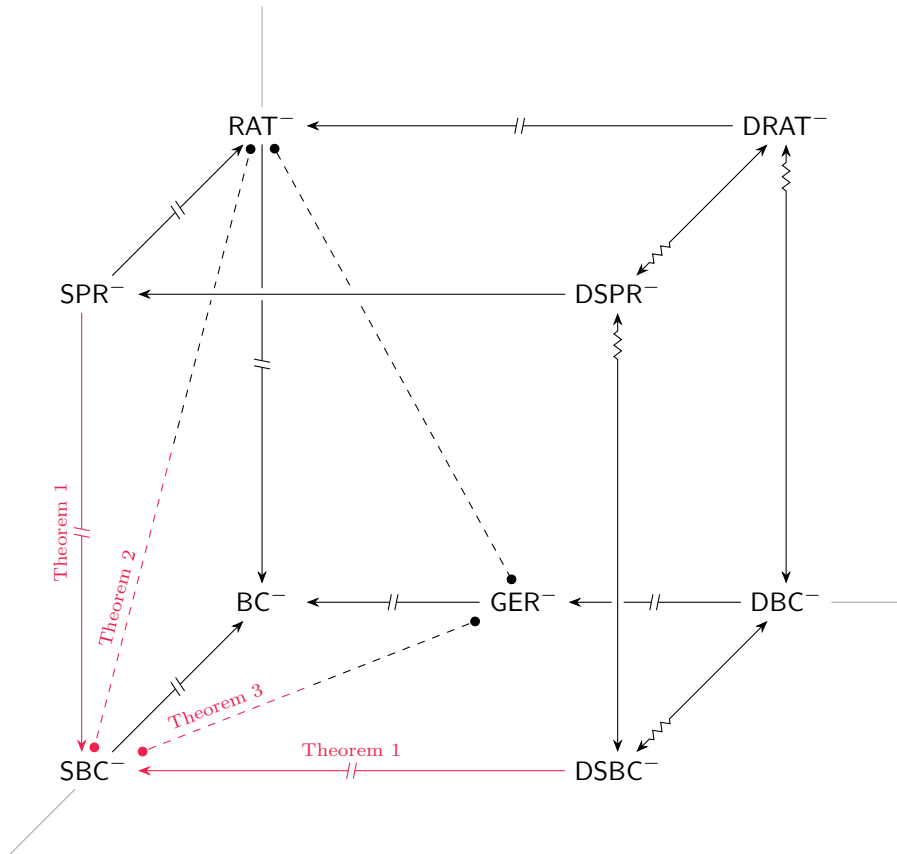


Figure 1 In the above diagram, the proof systems are placed in three-dimensional space with BC^- , the weakest system, at the origin. Moving away from the origin along each axis corresponds to a particular way of generalizing (i.e., strengthening) a proof system. The systems prefixed with “D” allow the arbitrary deletion of a clause as a proof step. For systems P and Q , we use $P \rightarrow Q$ to denote that P simulates Q ; (and $P \rightsquigarrow Q$ to indicate an “interesting” simulation, where P is not simply a generalization of Q); $P \bullet Q$ to denote that P is exponentially separated from Q (i.e., there exists an infinite sequence of formulas admitting polynomial-size proofs in P while requiring exponential-size proofs in Q); and $P \dashv\vdash Q$ to denote that P both simulates Q and is exponentially separated from Q . Arrows in red indicate the relationships that are new in this paper. To reduce clutter, some relationships that are implied by transitivity are not displayed (e.g., DBC^- simulates RAT^- and is exponentially separated from it through $DRAT^-$).

The above results, along with earlier ones, completely describe the relative strengths of the weakest generalizations of BC^- along each axis in Figure 1. For lower bounds, this pushes the frontier to the system called *set propagation redundancy* (SPR^-) [9]. We do not define SPR^- formally in this paper, although it may be thought of intuitively as combining SBC^- and RAT^- . The upper bounds proved by Buss and Thapen [2] establish SPR^- as an interesting target for proof complexity lower bounds. (Note that the binary encoding of the pigeonhole principle that we use to prove exponential lower bounds for SBC^- admits polynomial-size proofs in SPR^- [2, Theorem 4.4].)

2 Preliminaries

We assume that the reader is familiar with propositional logic, proof complexity, resolution, and extended resolution. We review some concepts to describe our notation. For notation we follow Yolcu and Heule [23] from which this section is adapted.

We denote the set of strictly positive integers by \mathbb{N}^+ . For $n \in \mathbb{N}^+$, we let $[n] := \{1, \dots, n\}$. For a sequence $S = (x_1, \dots, x_n)$, its *length* is n , which we denote by $|S|$.

2.1 Propositional logic

We use 0 and 1 to denote *False* and *True*, respectively. A *literal* is a propositional variable or its negation. A set of literals is *tautological* if it contains a pair of complementary literals x and \bar{x} . A *clause* is the disjunction of a nontautological set of literals. We use \perp to denote the empty clause. We denote by \mathbf{V} and \mathbf{L} respectively the sets of all variables and all literals. A *conjunctive normal form formula* (CNF) is a conjunction of clauses. Throughout this paper, by “formula” we mean a CNF. We identify clauses with sets of literals and formulas with sets of clauses. In the rest of this section we use C, D to denote clauses and Γ, Δ to denote formulas.

We say D is a *weakening* of C if $C \subseteq D$. We denote by $\text{var}(\Gamma)$ the set of all the variables occurring in Γ .

When we know $C \cup D$ to be nontautological, we write it as $C \vee D$. We write $C \dot{\vee} D$ to indicate a *disjoint disjunction*, where C and D have no variables in common. We take the disjunction of a clause and a formula as

$$C \vee \Delta := \{C \vee D : D \in \Delta \text{ and } C \cup D \text{ is nontautological}\}.$$

An *assignment* α is a partial function $\alpha : \mathbf{V} \rightarrow \{0, 1\}$, which also acts on literals by letting $\alpha(\bar{x}) := \alpha(x)$. We identify α with the set $\{p \in \mathbf{L} : \alpha(p) = 1\}$, consisting of all the literals it satisfies. For a set L of literals, we let $\bar{L} := \{\bar{x} : x \in L\}$. In particular, we use \bar{C} to denote the smallest assignment that falsifies all the literals in C . We say α *satisfies* C , denoted $\alpha \models C$, if there exists some $p \in C$ such that $\alpha(p) = 1$. We say α satisfies Γ if for all $C \in \Gamma$ we have $\alpha \models C$. For C that α does not satisfy, the *restriction* of C under α is $C|_\alpha := C \setminus \{p \in C : \alpha(p) = 0\}$. Extending this to formulas, the restriction of Γ under α is $\Gamma|_\alpha := \{C|_\alpha : C \in \Gamma \text{ and } \alpha \not\models C\}$.

We say Γ and Δ are *equisatisfiable*, denoted $\Gamma \equiv_{\text{sat}} \Delta$, if they are either both satisfiable or both unsatisfiable. With respect to Γ , a clause C is *redundant* if $\Gamma \setminus \{C\} \equiv_{\text{sat}} \Gamma \equiv_{\text{sat}} \Gamma \cup \{C\}$. We sometimes write $\Gamma \cup \{C\}$ as $\Gamma \wedge C$.

2.2 Proof complexity and resolution

For a proof system P and a formula Γ , we define

$$\text{size}_P(\Gamma) := \min\{|\Pi| : \Pi \text{ is a } P\text{-proof of } \Gamma\}$$

if Γ is unsatisfiable and $\text{size}_P(\Gamma) := \infty$ otherwise. A proof system P *simulates* Q if every Q -proof can be converted in polynomial time into a P -proof of the same formula. Proof systems P and Q are *equivalent* if they simulate each other. We say P is *exponentially separated* from Q if there exists some sequence $(\Gamma_n)_{n=1}^\infty$ of formulas such that $\text{size}_P(\Gamma_n) = n^{O(1)}$ while $\text{size}_Q(\Gamma_n) = 2^{\Omega(n)}$. We call such a sequence of formulas *easy* for P and *hard* for Q .

Let $C \dot{\vee} x$ and $D \dot{\vee} \bar{x}$ be clauses, where x is a variable, such that the set $C \cup D$ is nontautological. We call the clause $C \vee D$ the *resolvent* of $C \vee x$ and $D \vee \bar{x}$ on x . We define a resolution proof in a slightly different form than usual: as a sequence of formulas instead of a sequence of clauses.⁴

⁴ The resulting proof system is equivalent to the usual version of resolution.

► **Definition 4.** A resolution proof of a formula Γ is a sequence $\Pi = (\Gamma_1, \dots, \Gamma_N)$ of formulas such that $\Gamma_1 = \Gamma$, $\perp \in \Gamma_N$, and for all $i \in [N - 1]$, we have $\Gamma_{i+1} = \Gamma_i \cup \{C\}$, where C is either the resolvent of two clauses in Γ_i or a weakening of some clause in Γ_i . The size of Π is N .

We write Res to denote the resolution proof system. A well known fact is that resolution is closed under restrictions: if $(\Gamma_1, \Gamma_2, \dots, \Gamma_N)$ is a resolution proof of Γ , then for every assignment α , the sequence $(\Gamma_1|_\alpha, \Gamma_2|_\alpha, \dots, \Gamma_N|_\alpha)$ contains as a subsequence a resolution proof of $\Gamma|_\alpha$. This implies in particular the following.

► **Lemma 5.** For every formula Γ and every assignment α , $\text{size}_{\text{Res}}(\Gamma|_\alpha) \leq \text{size}_{\text{Res}}(\Gamma)$.

A *unit propagation proof* is a resolution proof where each use of the resolution rule has as at least one of its premises a clause that consists of a single literal. Unit propagation is not complete. With Γ a formula and $L = \{p_1, \dots, p_k\}$ a set of literals, we write $\Gamma \vdash_1 L$ to denote that there exists a unit propagation proof of $\Gamma \wedge \bar{p}_1 \wedge \dots \wedge \bar{p}_k$.

We next define *extended resolution* (ER), which is a strengthening of resolution.

► **Definition 6.** Let Γ be a formula and p, q be arbitrary literals. Consider a new variable x (i.e., not occurring in any one of Γ, p, q). We refer to $\{\bar{x} \vee p, \bar{x} \vee q, x \vee \bar{p} \vee \bar{q}\}$ as a set of extension clauses for Γ . In this context, we call x the extension variable.

► **Definition 7.** A formula Λ is an extension for a formula Γ if there exists a sequence $(\lambda_1, \dots, \lambda_t)$ such that $\Lambda = \bigcup_{i=1}^t \lambda_i$, and for all $i \in [t]$, we have that λ_i is a set of extension clauses for $\Gamma \cup \bigcup_{j=1}^{i-1} \lambda_j$.

► **Definition 8.** An extended resolution proof of a formula Γ is a pair (Λ, Π) , where Λ is an extension for Γ and Π is a resolution proof of $\Gamma \cup \Lambda$. The size of (Λ, Π) is $|\Lambda| + |\Pi|$.

2.3 Redundancy criteria

We recall the syntactic redundancy criteria that lead to the inference rules we study. The definitions are taken from Yolcu and Heule [23, Section 3], which in turn adapted them from previous works [16, 11, 12, 9, 2].

► **Definition 9.** A clause $C = p \dot{\vee} C'$ is a blocked clause (BC) for the literal p with respect to a formula Γ if, for every clause D of the form $\bar{p} \dot{\vee} D'$ in Γ , the set $C' \cup D'$ is tautological.

► **Definition 10.** A clause $C = p \dot{\vee} C'$ is a resolution asymmetric tautology (RAT) for the literal p with respect to a formula Γ if, for every clause D of the form $\bar{p} \dot{\vee} D'$ in Γ , we have $\Gamma \vdash_1 C' \cup D'$.

► **Definition 11.** A clause C is a set-blocked clause (SBC) for a nonempty $L \subseteq C$ with respect to a formula Γ if, for every clause $D \in \Gamma$ with $D \cap \bar{L} \neq \emptyset$ and $D \cap L = \emptyset$, the set $(C \setminus L) \cup (D \setminus \bar{L})$ is tautological.

We say C is a BC with respect to Γ if there exists a literal $p \in C$ for which C is a BC with respect to Γ , and similarly for RAT and SBC. It was shown in previous works [16, 11, 12] that BCs, RATs, and SBCs are redundant, which makes it possible to use them to define proof systems.

► **Definition 12.** A blocked clauses proof of a formula Γ is a sequence $\Pi = (\Gamma_1, \dots, \Gamma_N)$ of formulas such that $\Gamma_1 = \Gamma$, $\perp \in \Gamma_N$, and for all $i \in [N - 1]$, we have $\Gamma_{i+1} = \Gamma_i \cup \{C\}$, where C is either the resolvent of two clauses in Γ_i , a weakening of some clause in Γ_i , or a blocked clause with respect to Γ_i . The size of Π is N .

We write BC to denote the blocked clauses proof system. Replacing “blocked clause” by “resolution asymmetric tautology” in the above definition gives the RAT proof system. Replacing it by “set-blocked clause” gives the SBC proof system.⁵ RAT and SBC are two generalizations of BC, and we now define another, called *generalized extended resolution* (GER), which reduces the dependence of the validity of BC inferences on the order of clause additions (see [16, Section 1.3]). We need to introduce the concept of a blocked extension⁶ before we can proceed with the definition of GER.

► **Definition 13.** *A formula Λ is a blocked extension for a formula Γ if there exists a subset Γ' of Γ and an ordering (C_1, \dots, C_r) of all the clauses in $\Lambda \cup (\Gamma \setminus \Gamma')$ such that for all $i \in [r]$ the clause C_i is blocked with respect to $\Gamma' \cup \bigcup_{j=1}^{i-1} \{C_j\}$.*

► **Definition 14.** *A generalized extended resolution proof of a formula Γ is a pair (Λ, Π) , where Λ is a blocked extension for Γ and Π is a resolution proof of $\Gamma \cup \Lambda$. The size of (Λ, Π) is $|\Lambda| + |\Pi|$.*

Note that the definitions in this section do not prohibit BCs, RATs, or SBCs with respect to Γ from containing variables not occurring in Γ . We study the variants of BC, RAT, SBC, and GER that disallow the use of new variables. A proof of Γ is *without new variables* if all the variables occurring in the proof are in $\text{var}(\Gamma)$. In the case of GER, this constraint applies to both the blocked extension and the resolution part: a proof (Λ, Π) of Γ is without new variables if all the variables occurring in Λ or Π are in $\text{var}(\Gamma)$. We use BC^- , RAT^- , SBC^- , and GER^- to denote the variants without new variables.

3 Lower bound for the bit pigeonhole principle

Let $n = 2^k$, with $k \in \mathbb{N}^+$. For a propositional variable v , let us write $v \neq 0$ and $v \neq 1$ to denote the literals v and \bar{v} , respectively. The *bit pigeonhole principle* is the contradiction stating that each pigeon in $[n+1]$ can be assigned a distinct binary string from $\{0, 1\}^k$, where we identify strings with holes. For each pigeon $x \in [n+1]$, the variables p_1^x, \dots, p_k^x represent the bits of the string assigned to x . More formally, we write the bit pigeonhole principle as

$$\text{BPHP}_n := \bigcup_{\substack{x, y \in [n+1], x \neq y \\ (h_1, \dots, h_k) \in \{0, 1\}^k}} \left\{ \bigvee_{\ell=1}^k p_\ell^x \neq h_\ell \vee \bigvee_{\ell=1}^k p_\ell^y \neq h_\ell \right\},$$

which asserts that for all $x, y \in [n+1]$ such that $x \neq y$, the binary strings $p_1^x \dots p_k^x$ and $p_1^y \dots p_k^y$ are different.⁷ We denote by P_x the set $\{p_1^x, \dots, p_k^x, \bar{p}_1^x, \dots, \bar{p}_k^x\}$ of all the literals concerning pigeon x .

For a set L of literals, its *pigeon-width* is the number of distinct pigeons it mentions, where a pigeon $x \in [n+1]$ is *mentioned* if there exists some $\ell \in [k]$ such that some literal of the variable p_ℓ^x is in L . We write $L(x)$ to denote the set $L \cap P_x$. In other words, $L(x)$ is the largest subset of L that mentions only the pigeon x .

⁵ For an SBC proof to be polynomial-time verifiable, every step in the proof that adds a clause C as set-blocked is expected to indicate the subset $L \subseteq C$ for which C is set-blocked. With that said, we leave this requirement out of our definitions to reduce clutter.

⁶ Instead of the original definition of a blocked extension [16, Definition 6.3], we use a convenient characterization [16, Lemma 6.5], which is simpler to state, as the definition.

⁷ In our asymptotic results that use the bit pigeonhole principle, it is tacitly understood that BPHP_n could be defined for every integer $n \geq 2$ (as opposed to only powers of two) by letting BPHP_n be identical to BPHP_m , where m is the largest power of two not exceeding n .

Before proceeding with the SBC^- lower bound for the bit pigeonhole principle, we observe the result below, which will also be useful later. It is deduced in a straightforward way from the definition of a set-blocked clause, using the following fact: if a clause C is an SBC with respect to a formula Γ , then C is an SBC with respect to every subset of Γ . (Note that a similar result does not necessarily hold for RAT^- .)

► **Lemma 15.** *Without loss of generality, all of the set-blocked clause additions in an SBC^- proof are performed before any resolution or weakening steps.*

Lemma 15 allows us to reduce SBC^- lower bounds for a formula Γ to resolution lower bounds for another formula $\Gamma \cup \Sigma$, where Σ is a set of clauses derivable from Γ by a sequence of set-blocked clause additions without new variables.

A common strategy for proving resolution lower bounds is to first show that every proof contains some “complex” clause (in our case, a clause of large pigeon-width), and then argue that the existence of a small proof implies the existence of another proof where no clause is complex. The second step typically involves restricting the clauses of the proof under a suitable assignment. We work with assignments that correspond to partial matchings of pigeons to holes, as in the case of the RAT^- lower bound by Buss and Thapen [2, Section 5].

We say an assignment ρ that sets some variables of BPHP_n is a *partial matching* if ρ sets all of the bits for the pigeons it mentions in such a way that no two pigeons are in the same hole, thus representing a matching of pigeons to holes. To prove SBC^- lower bounds for BPHP_n , we will need the following pigeon-width lower bound for resolution proofs of restrictions of BPHP_n under partial matchings, which is established by a straightforward Adversary strategy in the Prover–Adversary game [19]. We define the pigeon-width of a proof as the maximum pigeon-width of any clause in the proof.

► **Lemma 16** ([2, Lemma 5.2]). *Let ρ be a partial matching of m pigeons to holes. Then every resolution proof of $(\text{BPHP}_n)|_\rho$ has pigeon-width at least $n - m$.*

We will additionally need a pigeon-width lower bound for set-blocked clauses (without new variables) with respect to BPHP_n , which follows from a simple inspection.

► **Lemma 17.** *Every set-blocked clause with respect to BPHP_n that is without new variables has pigeon-width $n + 1$.*

Proof. Let $C = L \dot{\vee} C'$ be a set-blocked clause for L with respect to BPHP_n that is without new variables. Let x be a pigeon mentioned in L . Such a pigeon exists since L is nonempty. Let y be a pigeon different from x . We claim that C mentions y .

Let $D \in \text{BPHP}_n$ be a clause that contains $\overline{L(x)} \cup C'(x)$ and mentions y . Such a clause exists since $\overline{L(x)} \cup C'(x)$ is simply a nontautological subset of P_x and, by the definition of BPHP_n , each such subset is contained in some clause in BPHP_n that mentions y . Note that every clause in BPHP_n mentions exactly two pigeons; in particular, the clause D mentions only the pigeons x and y .

Since D is a clause, it is nontautological. As a consequence, $D \cap L(x)$ is empty and $C'(x) \cup (D \setminus \overline{L})$ is nontautological. Then, since C is set-blocked for L with respect to BPHP_n , either $D \cap (L \setminus L(x))$ is nonempty or $(C' \setminus C'(x)) \cup (D \setminus \overline{L})$ is tautological. Now, neither of $L \setminus L(x)$ and $C' \setminus C'(x)$ mentions x . Since D mentions only the pigeons x and y , the pigeon y must be mentioned by L in the former case and C' in the latter. Either way, C mentions y . ◀

► **Theorem 18.** *The formula BPHP_n requires exponential-size proofs in SBC^- .*

Proof. Let Π be an SBC^- proof of BPHP_n of size N . By Lemma 15, we may view Π as a resolution proof of the formula $\text{BPHP}_n \cup \Sigma$, where Σ is a set of clauses derivable from BPHP_n by a sequence of set-blocked clause additions without new variables. We will show by the probabilistic method that if $N < 2^{n/64}$, then there exists a partial matching ρ of $n/2$ pigeons to holes such that $(\text{BPHP}_n)|_\rho$ has a resolution proof of pigeon-width strictly less than $n/2$.

Let R be a random partial matching constructed by choosing a random pigeon and assigning it to a random available hole until $n/2$ pigeons are matched to holes. We denote by r_i the random assignment performed at the i th step of this process: if pigeon x was assigned to hole (h_1, \dots, h_k) , then $r_i(p_\ell^x) = h_\ell$ for all $\ell \in [k]$.

We say a clause is *wide* if it has pigeon-width at least $n/2$. Let C be a wide clause. Let x denote the i th pigeon chosen when constructing R , with $i \leq n/4$. The probability that x is mentioned by C is at least

$$\frac{n/2 - (n/4 - 1)}{n + 1} \geq 1/4.$$

Suppose that x is mentioned by C through a literal p . When x is about to be assigned to a hole, there are at least $n/2 - (n/4 - 1) \geq n/4$ available ones that would result in r_i satisfying p . Therefore, the conditional probability that r_i satisfies C given that the assignments r_1, \dots, r_{i-1} do not satisfy C is at least $1/16$. As a result,

$$\Pr[R \not\models C] < (1 - 1/16)^{n/4} \leq 2^{-n/64}.$$

Suppose that $N < 2^{n/64}$. Let Δ be the set of all the wide clauses appearing in Π . By the union bound, $\Pr[R \not\models \Delta] < 1$. Thus, there exists a partial matching ρ of $n/2$ pigeons to holes such that $\rho \models \Delta$. Also, observe that $\rho \models \Sigma$ because we have $\Sigma \subseteq \Delta$ by Lemma 17.

Since resolution is closed under restrictions, when we restrict the proof Π under ρ we obtain a resolution proof of $(\text{BPHP}_n \cup \Sigma)|_\rho = (\text{BPHP}_n)|_\rho$ without any wide clauses, which contradicts Lemma 16. \blacktriangleleft

4 Separations using guarded extension variables

From this point on, given a formula Γ , we use (Λ, Π) to denote the minimum-size ER proof of Γ ,⁸ where Λ is the union of a sequence of $t(\Gamma) := |\Lambda|/3$ sets of extension clauses such that the i th set λ_i is of the form $\{\bar{x}_i \vee p_i, \bar{x}_i \vee q_i, x_i \vee \bar{p}_i \vee \bar{q}_i\}$. We thus reserve $\{x_1, \dots, x_{t(\Gamma)}\}$ as the set of extension variables used in Λ . We assume without loss of generality that the variables of p_i and q_i are in $\text{var}(\Gamma) \cup \{x_1, \dots, x_{i-1}\}$ for all $i \in [t(\Gamma)]$.

4.1 Separation of RAT^- from SBC^-

Let Γ be a formula and (Λ, Π) be the minimum-size ER proof of Γ as described above. Consider the transformation

$$\mathcal{G}(\Gamma) := \Gamma \cup \bigcup_{i=1}^{t(\Gamma)} [(x_i \vee \Gamma) \cup (\bar{x}_i \vee \Gamma)], \quad (1)$$

where $x_1, \dots, x_{t(\Gamma)}$ are the extension variables used in Λ .

It becomes possible to prove $\mathcal{G}(\Gamma)$ in RAT^- by simulating the ER proof of Γ using the extension variables present in the formula, resulting in the following.

⁸ We refer to *the* minimum-size proof with the assumption of having fixed some way of choosing a proof among those with minimum size.

► **Lemma 19** ([23, Lemma 5.1]). *For every formula Γ , $\text{size}_{\text{RAT}^-}(\mathcal{G}(\Gamma)) \leq \text{size}_{\text{ER}}(\Gamma)$.*

For SBC^- , the formula $\mathcal{G}(\Gamma)$ is at least as hard as Γ . We need the following definition before we prove this fact.

► **Definition 20.** *The projection of a formula Γ onto a literal p is the formula*

$$\text{proj}_p(\Gamma) := \{C \setminus \{p\} : C \in \Gamma \text{ and } p \in C\}.$$

Our main tool in proving SBC^- lower bounds against the constructions that incorporate guarded extension variables is a version of the following characterization of blocked clauses, which was already observed by Kullmann (see [16, Section 4]).

► **Lemma 21** ([23, Lemma 3.15]). *A clause $C = p \dot{\vee} C'$ is a BC for p with respect to a formula Γ if and only if the assignment $\overline{C'}$ satisfies $\text{proj}_{\overline{p}}(\Gamma)$.*

Lemma 21 suffices for proving BC^- lower bounds against $\mathcal{G}(\Gamma)$ (see [23, Lemma 5.2]); however, for SBC^- lower bounds we need the following result.

► **Lemma 22.** *If a clause $C = L \dot{\vee} C'$ is an SBC for L with respect to a formula Γ , then for every $p \in L$, the assignment $L \cup \overline{C'}$ satisfies $\text{proj}_{\overline{p}}(\Gamma)$.*

Proof. Suppose that $C = L \dot{\vee} C'$ is an SBC for L with respect to Γ , let $p \in L$, and let $D' \in \text{proj}_{\overline{p}}(\Gamma)$. Then the clause $D = \overline{p} \dot{\vee} D'$ is in Γ . Note that $D \cap \overline{L}$ is nonempty. Then, since C is an SBC for L , either $D \cap L$ is nonempty or $C' \cup (D \setminus \overline{L})$ is tautological.

Case 1: $D \cap L \neq \emptyset$. Since $\overline{p} \notin L$, the set $D' \cap L$ also is nonempty; therefore $L \models D'$.

Case 2: $C' \cup (D \setminus \overline{L})$ is tautological. Since neither of C' and $D \setminus \overline{L}$ is tautological, their union is tautological if and only if $\overline{C'} \cap (D \setminus \overline{L})$ is nonempty. This implies in particular that $\overline{C'} \cap D'$ is nonempty; therefore $\overline{C'} \models D'$. ◀

Lemma 22 implies that if the projection of a formula onto a literal p is unsatisfiable, then, with respect to the formula, no clause is set-blocked for any set that contains \overline{p} .

The intuition behind the construction of $\mathcal{G}(\Gamma)$ is as follows. We incorporate the extension variables into the formula while having Γ be the projection for each added literal. Thus, if Γ is unsatisfiable, we render the extension variables useless in set-blocked clause additions with respect to $\mathcal{G}(\Gamma)$ while still allowing RAT^- to take advantage of them. In particular, it becomes unnecessary for a set-blocked clause C with respect to $\mathcal{G}(\Gamma)$ to include any of the extension variables present in the formula. This is because every such clause C has some subset C' without any of the extension variables that is still set-blocked with respect to $\mathcal{G}(\Gamma)$. Moreover, since $\Gamma \subseteq \mathcal{G}(\Gamma)$, the clause C' is set-blocked also with respect to Γ . The alternative way to use the extension variables in $\mathcal{G}(\Gamma)$ is to derive x_i from $x_i \vee \Gamma$, but this involves proving Γ . When Γ is hard for SBC^- , we leave no way for SBC^- to make any meaningful use of the extension variables to achieve a speedup. In the end, an SBC^- proof of $\mathcal{G}(\Gamma)$ might as well ignore the extension variables present in the formula, falling back to an SBC^- proof of Γ .

► **Lemma 23.** *For every formula Γ , $\text{size}_{\text{SBC}^-}(\mathcal{G}(\Gamma)) \geq \text{size}_{\text{SBC}^-}(\Gamma)$.*

Proof. When Γ is satisfiable, the inequality holds trivially, so suppose that Γ is unsatisfiable.

Suppose that $\mathcal{G}(\Gamma)$ has an SBC^- proof of size N . By Lemma 15, we may view such a proof as a resolution proof of the formula $\mathcal{G}(\Gamma) \cup \Sigma$, where Σ is a set of clauses derivable from $\mathcal{G}(\Gamma)$ by a sequence of set-blocked clause additions without new variables. Let $X = \{x_1, \dots, x_{t(\Gamma)}\}$ denote the set of extension variables incorporated into $\mathcal{G}(\Gamma)$, and consider an assignment α defined as

$$\alpha(v) = \begin{cases} 1 & \text{if } v \in X \\ \text{undefined} & \text{otherwise.} \end{cases}$$

59:12 Lower Bounds for Set-Blocked Clauses Proofs

By Lemma 5, there exists a resolution proof of the formula $(\mathcal{G}(\Gamma) \cup \Sigma)|_\alpha = \Gamma \cup \Sigma|_\alpha$ of size at most $N - |\Sigma|$. We claim that the clauses in $\Sigma|_\alpha$ can be derived in sequence from Γ by set-blocked clause additions, which implies that there exists an SBC^- proof of Γ of size at most N .

Let $S = (C_1, \dots, C_r)$ be the ordering in which the clauses of Σ are derived from $\mathcal{G}(\Gamma)$. We will show that if we restrict each clause in S under α and remove the satisfied clauses, then the remaining sequence of clauses can be derived from Γ in the same order by set-blocked clause additions. More specifically, the goal is to prove that for all $i \in [r]$ such that α does not satisfy C_i , the clause $C_i|_\alpha$ is set-blocked with respect to $\Gamma \cup \Phi_{i-1}|_\alpha$, where

$$\Phi_{i-1} := \bigcup_{j \in [i-1]} \{C_j\}.$$

Let $i \in [r]$, and consider the clause C_i , which we write as C from this point on. Suppose that α does not satisfy C , so the variables from X can occur only negatively in C . Let $L \subseteq C$ be a subset for which C is set-blocked with respect to $\mathcal{G}(\Gamma) \cup \Phi_{i-1}$. We will prove that $C|_\alpha$ is set-blocked for $L|_\alpha$ with respect to both Γ and $\Phi_{i-1}|_\alpha$, which, by the definition of a set-blocked clause, implies that $C|_\alpha$ is set-blocked with respect to $\Gamma \cup \Phi_{i-1}|_\alpha$.

Before proceeding, observe that L cannot contain any variables from X : If some \bar{x}_i is in L , then the assignment $L \cup \overline{(C \setminus L)}$ satisfies $\text{proj}_{x_i}(\mathcal{G}(\Gamma)) = \Gamma$ by Lemma 22. Since Γ is unsatisfiable, no such assignment exists. Therefore, L cannot contain \bar{x}_i , which implies that $L|_\alpha = L$.

$C|_\alpha$ is set-blocked for L with respect to Γ : Since $\Gamma \subseteq \mathcal{G}(\Gamma)$, the clause C is set-blocked for L in particular with respect to Γ . Noting that the variables from X do not occur in Γ , we conclude that $C|_\alpha$ also is set-blocked for L with respect to Γ .

$C|_\alpha$ is set-blocked for L with respect to $\Phi_{i-1}|_\alpha$: Consider an arbitrary $D' \in \Phi_{i-1}|_\alpha$, which is the restriction under α of some clause $D \in \Phi_{i-1}$ that α does not satisfy. Suppose $D' \cap \bar{L} \neq \emptyset$ and $D' \cap L = \emptyset$. We need to show that $(C|_\alpha \setminus L) \cup (D' \setminus \bar{L})$ is tautological.

Since $D' \subseteq D$, we immediately have $D \cap \bar{L} \neq \emptyset$. Now, recall that the variables from X do not occur in L , and observe that D' is simply D with the variables from X removed. We thus have $D \cap L = \emptyset$. Then, because C is set-blocked for L with respect to Φ_{i-1} , the set $E = (C \setminus L) \cup (D \setminus \bar{L})$ must be tautological. A variable that occurs both positively and negatively in E cannot be from X , since in that case α would satisfy C or D . Therefore, the set $(C|_\alpha \setminus L) \cup (D' \setminus \bar{L})$ also is tautological. ◀

Invoking Lemmas 19 and 23 with Γ as the bit pigeonhole principle gives us the separation.

► **Theorem 24.** *The formula $\mathcal{G}(\text{BPHP}_n)$ admits polynomial-size proofs in RAT^- but requires exponential-size proofs in SBC^- .*

Proof. Buss and Thapen [2, Theorem 4.4] gave polynomial-size proofs of BPHP_n in SPR^- , which ER simulates.⁹ By Lemma 19, we have $\text{size}_{\text{RAT}^-}(\mathcal{G}(\text{BPHP}_n)) = n^{O(1)}$. Theorem 18 and Lemma 23 give $\text{size}_{\text{SBC}^-}(\mathcal{G}(\text{BPHP}_n)) = 2^{\Omega(n)}$. Thus, the bit pigeonhole principle with \mathcal{G} applied to it exponentially separates RAT^- from SBC^- . ◀

⁹ It is also possible to deduce the existence of polynomial-size ER proofs of BPHP_n from the fact that the pigeonhole principle (PHP_n) is easy for ER [4], combined with the observation that PHP_n can be derived from BPHP_n in polynomial size in ER.

4.2 Separation of GER^- from SBC^-

We proceed in a similar way to the previous section. Let Γ be a formula and (Λ, Π) be the minimum-size ER proof of Γ . Let $m \in \mathbb{N}^+$, and let

$$\{y_1, \dots, y_m, z_1, \dots, z_m\} \subseteq \mathbf{V} \setminus \text{var}(\Gamma \cup \Lambda)$$

be a set of $2m$ distinct variables. Consider

$$\begin{aligned} V_m(\Gamma) &:= \bigcup_{i=1}^{t(\Gamma)} \bigcup_{j=1}^m \{x_i \vee y_j \vee \bar{z}_j, \bar{x}_i \vee y_j \vee \bar{z}_j\}, \\ W_m(\Gamma) &:= \bigcup_{j=1}^m [\{\bar{y}_j \vee z_j\} \cup (y_j \vee \Gamma) \cup (\bar{z}_j \vee \Gamma)], \\ \mathcal{I}_m(\Gamma) &:= \Gamma \cup V_m(\Gamma) \cup W_m(\Gamma), \end{aligned} \tag{2}$$

where $x_1, \dots, x_{t(\Gamma)}$ are the extension variables used in Λ .

To prove $\mathcal{I}_m(\Gamma)$ in GER^- by simulating the ER proof of Γ , we essentially remove the clauses in $V_m(\Gamma)$, derive the extension clauses, and rederive $V_m(\Gamma)$ by a sequence of blocked clause additions.

► **Lemma 25.** *For every formula Γ and every $m \in \mathbb{N}^+$, $\text{size}_{\text{GER}^-}(\mathcal{I}_m(\Gamma)) \leq \text{size}_{\text{ER}}(\Gamma)$.*

Proof. Let (Λ, Π) be the minimum-size ER proof of Γ . We will show that the clauses in $\Lambda \cup V_m(\Gamma)$ can be derived from $\Gamma \cup W_m(\Gamma)$ in some sequence by blocked clause additions, which implies by Definition 13 that Λ is a blocked extension for $\mathcal{I}_m(\Gamma)$.

Recall that extension clauses can be derived in sequence by blocked clause additions. The formula Λ is an extension for $\Gamma \cup W_m(\Gamma)$, so we derive Λ by such a sequence. Next, from $\Gamma \cup W_m(\Gamma) \cup \Lambda$, we derive the clauses in $V_m(\Gamma)$ in any order. Let V' be a proper subset of $V_m(\Gamma)$, and let C be a clause in $V_m(\Gamma) \setminus V'$. For some $i \in [t(\Gamma)]$ and $j \in [m]$, the clause C is of the form $p \vee y_j \vee \bar{z}_j$, where p is either x_i or \bar{x}_i . With respect to $\Gamma \cup W_m(\Gamma) \cup \Lambda \cup V'$, the clause C is blocked for y_j since the only earlier occurrence of \bar{y}_j is the clause $\bar{y}_j \vee z_j$ and $\{p, \bar{z}_j, z_j\}$ is tautological. It follows by induction that we can derive $V_m(\Gamma)$ from $\Gamma \cup W_m(\Gamma) \cup \Lambda$. Thus, Λ is a blocked extension for $\mathcal{I}_m(\Gamma)$.

Noting that Π is a resolution proof of $\Gamma \cup \Lambda$ and that $\mathcal{I}_m(\Gamma)$ contains Γ as a subset, we conclude that there exists a GER^- proof of $\mathcal{I}_m(\Gamma)$ of size $|\Lambda| + |\Pi| = \text{size}_{\text{ER}}(\Gamma)$. ◀

For SBC^- , the formula $\mathcal{I}_m(\Gamma)$ stays at least as hard as Γ if fewer than 2^m set-blocked clauses are derived. As before, our goal is to render the added variables useless in set-blocked clause additions.

We will eventually choose Γ to be hard for SBC^- , which makes the literals \bar{y}_j and z_j useless in set-blocked clause additions. Moreover, the presence of the clause $\bar{y}_j \vee z_j$ ensures that if a clause is set-blocked for a set containing y_j or \bar{z}_j , then the clause is a weakening of $y_j \vee \bar{z}_j$. Such clauses are killed by assignments that set y_j and z_j to the same value.

We also need to consider the clauses that are set-blocked for sets containing the variables x_i . The projection of $\mathcal{I}_m(\Gamma)$ onto x_i or \bar{x}_i is the formula $\bigcup_{j=1}^m \{y_j \vee \bar{z}_j\}$, which has 2^m minimal satisfying assignments. Without deriving clauses that rule out all of those assignments, SBC^- proofs cannot use the variables x_i in any meaningful way. Since the variables y_j and z_j are rendered useless in set-blocked clause additions, the assignments can only be ruled out one at a time, which forces SBC^- proofs of $\mathcal{I}_m(\Gamma)$ to either derive at least 2^m clauses or ignore the variables x_i .

► **Lemma 26.** For every formula Γ and every $m \in \mathbb{N}^+$,

$$\text{size}_{\text{SBC}^-}(\mathcal{I}_m(\Gamma)) \geq \min\{2^m, \text{size}_{\text{SBC}^-}(\Gamma)\}.$$

Proof. Fix some $m \in \mathbb{N}^+$. When Γ is satisfiable, the inequality holds trivially, so suppose that Γ is unsatisfiable.

Suppose that $\mathcal{I}_m(\Gamma)$ has an SBC^- proof of size N . By Lemma 15, we may view such a proof as a resolution proof of the formula $\mathcal{I}_m(\Gamma) \cup \Sigma$, where Σ is a set of clauses derivable from $\mathcal{I}_m(\Gamma)$ by a sequence of set-blocked clause additions without new variables. We claim that if $|\Sigma| < 2^m$, then there exists an SBC^- proof of Γ of size at most N . This implies the desired lower bound.

Let $X = \{x_1, \dots, x_{t(\Gamma)}\}$ and $U = \{y_1, \dots, y_m, z_1, \dots, z_m\}$ denote the two sets of variables incorporated into $\mathcal{I}_m(\Gamma)$. Consider a clause $C = L \dot{\vee} C'$ that is set-blocked for L with respect to $\mathcal{I}_m(\Gamma)$. We start by inspecting the ways in which the variables from $X \cup U$ can occur in L :

- We first consider the variables from U . If either $\overline{y_j}$ or z_j for some $j \in [m]$ occurs in L , then, by Lemma 22, the assignment $L \cup \overline{C'}$ satisfies Γ since Γ is contained in the projections of $\mathcal{I}_m(\Gamma)$ onto the negations of these literals. Thus, since Γ is unsatisfiable, neither of the literals $\overline{y_j}$ and z_j for any $j \in [m]$ can occur in L . Moreover, if the literal y_j occurs in L , then, by Lemma 22, the assignment $L \cup \overline{C'}$ satisfies z_j since the clause $\overline{y_j} \vee z_j$ is in $\mathcal{I}_m(\Gamma)$. In that case, since z_j cannot occur in L , the literal z_j must occur in $\overline{C'}$, which implies that C contains the literal $\overline{z_j}$. Thus, if the literal y_j occurs in L , then C contains the literal $\overline{z_j}$. By a similar argument, if the literal $\overline{z_j}$ occurs in L , then C contains the literal y_j . To summarize, if some variable from U is in L , then C is a weakening of some clause in $\bigcup_{j=1}^m \{y_j \vee \overline{z_j}\}$.
- Next, we consider the variables from X . For all $j \in [m]$, define $A_j := \{y_j, \overline{z_j}\}$ (intended to be viewed as an assignment). Let $\mathbf{A} = A_1 \times \dots \times A_m$. Suppose that a literal p of some x_i is in L . Then, by Lemma 22, the assignment $L \cup \overline{C'}$ satisfies the formula

$$\text{proj}_{\overline{p}}(\mathcal{I}_m(\Gamma)) = \bigcup_{j=1}^m \{y_j \vee \overline{z_j}\}.$$

This implies that if no variable from U occurs in L , then there exists some assignment $\beta \in \mathbf{A}$ such that $\beta \subseteq \overline{C'}$. We say C is a *good* clause if some variable from X occurs in L but no variable from U occurs in L .

From this point on, suppose $|\Sigma| < 2^m$. For each good clause E in Σ , choose a single subset $F \subseteq E$ such that $\overline{F} \in \mathbf{A}$. Let Δ be the collection of those subsets. Since $|\Delta| < 2^m$, there exists some $\beta \in \mathbf{A}$ such that $\overline{\beta} \notin \Delta$. Recall that for each $j \in [m]$, the assignment β sets exactly one of the variables y_j and z_j . Let β' be the smallest assignment extending β such that $\beta'(y_j) = \beta'(z_j)$ for all $j \in [m]$.

► **Claim 27.** Let C be a clause in Σ , and let $L \subseteq C$ be a subset for which C is set-blocked with respect to $\mathcal{I}_m(\Gamma)$. If some variable from $X \cup U$ occurs in L , then β' satisfies C .

Proof. Let C be a clause in Σ set-blocked for $L \subseteq C$ with respect to $\mathcal{I}_m(\Gamma)$. Suppose that some variable from $X \cup U$ occurs in L . Then either $\text{var}(L) \cap U \neq \emptyset$ or C is a good clause.

Case 1: $\text{var}(L) \cap U \neq \emptyset$. Since C is a weakening of some clause in $\bigcup_{j=1}^m \{y_j \vee \overline{z_j}\}$ and $\beta'(y_j) = \beta'(z_j)$ for all $j \in [m]$, the assignment β' satisfies C .

Case 2: C is a good clause. Let F be a subset of C such that $F \in \Delta$. Since $\overline{\beta} \notin \Delta$, there exists some $j \in [m]$ such that either $\overline{y_j} \in \overline{\beta}$ and $z_j \in F$ or $z_j \in \overline{\beta}$ and $\overline{y_j} \in F$. We have $\beta'(z_j) = 1$ in the former case and $\beta'(y_j) = 0$ in the latter. Either way, β' satisfies F and hence it also satisfies C . ◁

Now, let α be the assignment defined as

$$\alpha(v) = \begin{cases} 1 & \text{if } v \in X \\ \beta'(v) & \text{if } v \in U \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The point of α is to set all of the variables from $X \cup U$ in such a way that kills all of the clauses in Σ that are set-blocked for sets containing those variables, leaving behind only the clauses that could also be derived in an SBC^- proof of Γ .

The rest of the argument is similar at a high level to the proof of Lemma 23, so we will be relatively brief. By Lemma 5, there exists a resolution proof of the formula $(\mathcal{I}_m(\Gamma) \cup \Sigma)|_\alpha = \Gamma \cup \Sigma|_\alpha$ of size at most $N - |\Sigma|$. We claim that the clauses in $\Sigma|_\alpha$ can be derived in sequence from Γ by set-blocked clause additions.

As before, let $S = (C_1, \dots, C_r)$ be the ordering in which the clauses of Σ are derived from $\mathcal{I}_m(\Gamma)$. We will prove that for all $i \in [r]$ such that α does not satisfy C_i , the clause $C_i|_\alpha$ is set-blocked with respect to $\Gamma \cup \Phi_{i-1}|_\alpha$, where

$$\Phi_{i-1} := \bigcup_{j \in [i-1]} \{C_j\}.$$

Let $i \in [r]$, and consider the clause C_i , which we write as C from this point on. Let $L \subseteq C$ be a subset for which C is set-blocked with respect to $\mathcal{I}_m(\Gamma) \cup \Phi_{i-1}$. Suppose that α does not satisfy C . Noting that α extends β' , by Claim 27, no variable from $X \cup U$ is in L . As a result, $L|_\alpha = L$. We will prove that $C|_\alpha$ is set-blocked for L with respect to both Γ and $\Phi_{i-1}|_\alpha$.

$C|_\alpha$ is set-blocked for L with respect to Γ : Since $\Gamma \subseteq \mathcal{I}_m(\Gamma)$, the clause C is set-blocked for L in particular with respect to Γ . No variable from $X \cup U$ occurs in Γ , so the clause $C|_\alpha$ also is set-blocked for L with respect to Γ .

$C|_\alpha$ is set-blocked for L with respect to $\Phi_{i-1}|_\alpha$: Consider an arbitrary $D' \in \Phi_{i-1}|_\alpha$, which is the restriction under α of some clause $D \in \Phi_{i-1}$ that α does not satisfy. Suppose $D' \cap \bar{L} \neq \emptyset$ and $D' \cap L = \emptyset$. We need to show that $(C|_\alpha \setminus L) \cup (D' \setminus \bar{L})$ is tautological.

We have $D \cap \bar{L} \neq \emptyset$ because $D' \subseteq D$. Recall that no variable from $X \cup U$ is in L , and observe that D' is simply D with the variables from $X \cup U$ removed. This implies $D \cap L = \emptyset$. Now, because C is set-blocked for L with respect to Φ_{i-1} , the set $E = (C \setminus L) \cup (D \setminus \bar{L})$ must be tautological. A variable that occurs both positively and negatively in E cannot be from $X \cup U$, since in that case α would satisfy C or D . Therefore, the set $(C|_\alpha \setminus L) \cup (D' \setminus \bar{L})$ also is tautological. ◀

Invoking Lemmas 25 and 26 with a suitable choice of m and with Γ as the bit pigeonhole principle gives us the separation.

► **Theorem 28.** *For every unsatisfiable formula Γ , let $m(\Gamma) := \lceil \log(\text{size}_{\text{SBC}^-}(\Gamma)) \rceil$ and define $\mathcal{K}(\Gamma) := \mathcal{I}_{m(\Gamma)}(\Gamma)$. The formula $\mathcal{K}(\text{BPHP}_n)$ admits polynomial-size proofs in GER^- but requires exponential-size proofs in SBC^- .*

Proof. Buss and Thapen [2, Theorem 4.4] gave polynomial-size proofs of BPHP_n in SPR^- , which ER simulates. By Lemma 25, we have $\text{size}_{\text{GER}^-}(\mathcal{K}(\text{BPHP}_n)) = n^{O(1)}$. Theorem 18 and Lemma 26 give $\text{size}_{\text{SBC}^-}(\mathcal{K}(\text{BPHP}_n)) = 2^{\Omega(n)}$. Thus, the bit pigeonhole principle with \mathcal{K} applied to it exponentially separates GER^- from SBC^- . ◀

References

- 1 Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, The University of Chicago, 1937.
- 2 Sam Buss and Neil Thapen. DRAT and propagation redundancy proofs without new variables. *Logical Methods in Computer Science*, 17(2):12:1–12:31, 2021. doi:10.23638/LMCS-17(2:12)2021.
- 3 Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th Symposium on Theory of Computing (STOC)*, pages 174–183. Association for Computing Machinery, 1996. doi:10.1145/237814.237860.
- 4 Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *ACM SIGACT News*, 8(4):28–32, 1976. doi:10.1145/1008335.1008338.
- 5 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979. doi:10.2307/2273702.
- 6 William Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987. doi:10.1016/0166-218X(87)90039-4.
- 7 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.
- 8 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Encoding redundancy for satisfaction-driven clause learning. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, number 11427 in Lecture Notes in Computer Science, pages 41–58. Springer, 2019. doi:10.1007/978-3-030-17462-0_3.
- 9 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Strong extension-free proof systems. *Journal of Automated Reasoning*, 64(3):533–554, 2020. doi:10.1007/s10817-019-09516-0.
- 10 Marijn J. H. Heule, Benjamin Kiesl, Martina Seidl, and Armin Biere. PRuning through satisfaction. In *Proceedings of the 13th Haifa Verification Conference (HVC)*, number 10629 in Lecture Notes in Computer Science, pages 179–194. Springer, 2017. doi:10.1007/978-3-319-70389-3_12.
- 11 Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR)*, number 7364 in Lecture Notes in Computer Science, pages 355–370. Springer, 2012. doi:10.1007/978-3-642-31365-3_28.
- 12 Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. Local redundancy in SAT: Generalizations of blocked clauses. *Logical Methods in Computer Science*, 14(4:3):1–23, 2018. doi:10.23638/LMCS-14(4:3)2018.
- 13 Jan Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170(1–2):123–140, 2001.
- 14 Oliver Kullmann. Worst-case analysis, 3-SAT decision and lower bounds: Approaches for improved SAT algorithms. In *Satisfiability Problem: Theory and Applications*, number 35 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 261–313. American Mathematical Society, 1997. doi:10.1090/dimacs/035.
- 15 Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1–2):1–72, 1999. doi:10.1016/S0304-3975(98)00017-6.
- 16 Oliver Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96–97:149–176, 1999. doi:10.1016/S0166-218X(99)00037-2.
- 17 João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999. doi:10.1109/12.769433.
- 18 Toniann Pitassi and Rahul Santhanam. Effectively polynomial simulations. In *Proceedings of the 1st Innovations in Computer Science (ICS)*, pages 370–382. Tsinghua University Press, 2010.

- 19 Pavel Pudlák. Proofs as games. *The American Mathematical Monthly*, 107(6):541–550, 2000. doi:10.2307/2589349.
- 20 John A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965. doi:10.1145/321250.321253.
- 21 Grigori S. Tseitin. On the complexity of derivation in propositional calculus. *Zapiski Nauchnykh Seminarov LOMI*, 8:234–259, 1968.
- 22 Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, 1987. doi:10.1145/7531.8928.
- 23 Emre Yolcu and Marijn J. H. Heule. Exponential separations using guarded extension variables. In *Proceedings of the 14th Innovations in Theoretical Computer Science (ITCS)*, number 251 in Leibniz International Proceedings in Informatics, pages 101:1–101:22. Schloss Dagstuhl, 2023. doi:10.4230/LIPIcs.ITCS.2023.101.

