A Linear Fixed Parameter Tractable Algorithm for Connected Pathwidth

Mamadou Moustapha Kanté
Université Clermont Auvergne, LIMOS, CNRS, Aubière, France
mamadou.kante@uca.fr

Christophe Paul
LIRMM, Université de Montpellier, CNRS, France
christophe.paul@lirmm.fr

Dimitrios M. Thilikos
LIRMM, Université de Montpellier, CNRS, France
sedthilk@thilikos.info

Abstract

The graph parameter of *pathwidth* can be seen as a measure of the topological resemblance of a graph to a path. A popular definition of pathwidth is given in terms of *node search* where we are given a system of tunnels (represented by a graph) that is contaminated by some infectious substance and we are looking for a search strategy that, at each step, either places a searcher on a vertex or removes a searcher from a vertex and where an edge is cleaned when both endpoints are simultaneously occupied by searchers. It was proved that the minimum number of searchers required for a successful cleaning strategy is equal to the pathwidth of the graph plus one. Two desired characteristics for a cleaning strategy is to be *monotone* (no recontamination occurs) and *connected* (clean territories always remain connected). Under these two demands, the number of searchers is equivalent to a variant of pathwidth called *connected pathwidth*. We prove that connected pathwidth is fixed parameter tractable, in particular we design a $2^{O(k^2)} \cdot n$ time algorithm that checks whether the connected pathwidth of G is at most k. This resolves an open question by [*Dereniowski, Osula, and Rzążewski, Finding small-width connected path-decompositions in polynomial time. Theor. Comput. Sci., 794:85–100, 2019*]. For our algorithm, we enrich the *typical sequence technique* that is able to deal with the connectivity demand. Typical sequences have been introduced in [*Bodlaender and Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms, 21(2):358–402, 1996*] for the design of linear parameterized algorithms for treewidth and pathwidth. While this technique has been later applied to other parameters, none of its advancements was able to deal with the connectivity demand, as it is a "global" demand that concerns an unbounded number of parts of the graph of unbounded size. The proposed extension is based on an encoding of the connectivity property that is quite versatile and may be adapted so to deliver linear parameterized algorithms for the connected variants of other width parameters as well. An immediate consequence of our result is a $2^{O(k^2)} \cdot n$ time algorithm for the monotone and connected version of the edge search number.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory; Theory of computation → Fixed parameter tractability

Keywords and phrases Graph decompositions, Parameterized algorithms, Typical sequences, Pathwidth, Graph searching

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.64

Related Version A full version of this extended abstract is available at [26], `https://arxiv.org/abs/2004.11937`.

Funding *Mamadou Moustapha Kanté*: DEMOGRAPH (ANR-16-CE40-0028) and ASSK (ANR-18-CE40-0025-01).
Christophe Paul: DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010).
Dimitrios M. Thilikos: DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010).

1 Introduction

Pathwidth. A *path-decomposition* of a graph $G = (V, E)$ is a sequence $\mathsf{Q} = \langle B_1, \ldots, B_q \rangle$ of vertex sets, called *bags* of Q, such that

1. $\bigcup_{i \in \{1,\ldots,q\}} B_i = V$,
2. every edge $e \in E$ is a subset of some member of Q, and
3. the *trace* of every vertex $v \in V$, that is the set $\{i \mid v \in B_i\}$, is a set of consecutive integers.

The *width* of a path-decomposition is $\max\{|B_i| - 1 \mid i \in \{1, \ldots, q\}\}$ and the *pathwidth* of a graph G, denoted by $\mathsf{pw}(G)$, is the minimum width of a path-decomposition of G.

The above definition appeared for the first time in [36]. Pathwidth can be seen as a measure of the topological resemblance of a graph to a path. Pathwidth, along with its tree-analogue *treewidth*, have been used as key combinatorial tools in the Graph Minors series of Robertson and Seymour [37] and they are omnipresent in both structural and algorithmic graph theory.

Deciding whether the pathwidth of a graph is at most k is an NP-complete problem [2]. This motivated the problem of the existence, or not, of a *parameterized algorithm* for this problem, and algorithm running in $f(k) \cdot n^{O(1)}$ time algorithm. An affirmative answer to this question was directly implied as a consequence of the algorithmic and combinatorial results of the Graph Minors series and the fact that, for every k, the class of graphs with pathwidth at most k is closed under taking of minors[1]. On the negative side, this implication was purely existential. The challenge of *constructing* an $f(k) \cdot n^{O(1)}$ time algorithm for pathwidth (as well as for treewidth) was a consequence of the classic result of Bodlaender and Kloks in [8] (see also [15, 30]). The main result in [8] implies a $2^{O(k^3)} \cdot n$ time algorithm. This was later improved to one running in $2^{O(k^2)} \cdot n$ time by Fürer in [20]).

Graph searching. In a *graph searching* game, the opponents are a group of *searchers* and an evading *fugitive*. The opponents move in turns in a graph. The objective of the searchers is to deploy a strategy of moves that leads to the capture of the fugitive. At each step of the *node searching game*, the searchers may either place a searcher at a vertex or remove a searcher from a vertex. The fugitive resides in the edges of the graph and is lucky, invisible, fast, and agile. The capture of the fugitive occurs when searchers occupy both endpoints of the edge where he currently resides. A *node searching strategy* is a sequence of moves of the searchers that can guarantee the eventual capture of the fugitive.[2] The cost of a searching strategy is the maximum number of searchers simultaneously present in the graph during the deployment of the strategy. The *node search number* of a graph G, denoted by $\mathsf{ns}(G)$, is defined as the minimum cost of a node searching strategy. Node searching was defined by Kirousis and Papadimitriou in [29] who proved that the game is equivalent to its monotone variant where search strategies are *monotone* in the sense that they prevent the fugitive from pervading again areas from where he had been expelled. This result along with the results in [27, 28, 31], imply that, for every graph G, $\mathsf{ns}(G) = \mathsf{pw}(G) + 1$.

The connectivity issue. In several applications of graph searching it is important to guarantee secure communication channels between the searchers so that they can safely exchange information. This issue was treated for the first time in the area of distributed

[1] A graph H is a *minor* of a graph G if H can be obtained by some subgraph of G by contracting edges.

[2] An equivalent setting of graph searching is to see G as a system of pipelines or corridors that is contaminated by some poisonous gas or some highly infectious substance. The searchers can be seen as cleaners that deploy a decontamination strategy [13, 19]. The fact that the fugitive is invisible, fast, lucky, and agile permits us to see him as being omnipresent in any edge that has not yet been cleaned.

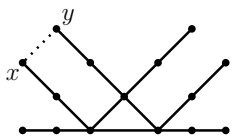

Figure 1 A graph G of connected pathwidth 2 with a subgraph of connected pathwidth 3.

computing, in particular in [4], where the authors considered the problem of capturing an intruder by mobile agents (acting for example as antivirus programs). As agents deploy their cleaning strategy, they must guarantee that, at each moment of the search, the cleaned territories remain connected, so to permit the safe exchange of information between the coordinating agents.

The systematic study of connected graph searching was initiated in [3, 5]. When, in node searching, we demand that the search strategies are monotone and connected, we define *monotone connected node search number*, denoted by $\mathsf{mcns}(G)$. The graph decomposition counterpart of this parameter was introduced by Dereniowski in [16]. He defined the *connected pathwidth* of a connected graph, denoted by $\mathsf{cpw}(G)$, by considering *connected* path-decompositions $\mathsf{Q} = \{B_1, \ldots, B_q\}$ where the following additional property is satisfied:

▶ For every $i \in \{1, \ldots, q\}$, the subgraph of G induced by $\bigcup_{h\in\{1,\ldots,i\}} B_h$ is *connected*.

As noticed in [16], for every connected graph G, $\mathsf{mcns}(G) = \mathsf{cpw}(G) + 1$ (see also [1]). Notice that the above demand results to a break of symmetry: the fact that $\langle B_1, \ldots, B_q\rangle$ is a connected path-decomposition does not imply that the same holds for $\langle B_q, \ldots, B_1\rangle$ (while this is always the case for conventional path-decompositions). This *sense of direction* seems to be the source of all combinatorial particularities (and challenges) of connected pathwidth.

Computing connected pathwidth. It is easy to see that checking whether $\mathsf{cpw}(G) \leq k$ is an NP-complete problem: if we define G^* as the graph obtained from G after adding a new vertex adjacent with all the vertices of G, then observe that $\mathsf{pw}(G) = \mathsf{cpw}(G^*) - 1$. This motivates the question on the parameterized complexity of the problem. The first progress in this direction was done recently in [17] by Dereniowski, Osula, and Rzążewski who gave an $f(k) \cdot n^{O(k^2)}$ time algorithm. In [17, Conjecture 1], they conjectured that there is a fixed parameter algorithm checking whether $\mathsf{cpw}(G) \leq k$. The general question on the parameterized complexity of the connected variants of graph search was raised as an open question by Fedor V. Fomin during the GRASTA 2017 workshop [18].

A somehow dissuasive fact towards a parameterized algorithm for connected pathwidth is that connected pathwdith is not closed under minors and therefore it does not fit in the powerful algorithmic framework of Graph Minors (which is the case with pathwidth). The removal of an edge may increase the parameter. For instance, the connected pathwidth of the graph in Figure 1 has connected pathwidth 2 while if we remove the edge $\{x, y\}$ its connected pathwidth increases to 3. On the positive side, connected pathwidth is closed under contractions (see e.g., [1]), i.e, its value does not increase when we contract edges and, moreover, the yes-instances of the problem have bounded pathwidth, therefore they also have bounded treewidth. Based on these observations, the existence of a parameterized algorithm would be implied if we can prove that, for any k, the set $\mathcal{Z}_k$ of contraction-minimal[3] graphs with connected pathwidth more than k is *finite*: as contraction containment can be expressed

[3] For instance, the graph $G \setminus \{x, y\}$ from Figure 1 belongs in $\mathcal{Z}_2$.

in MSO logic, one should just apply Courcelle's theorem [14] to check whether some graph of $\mathcal{Z}_k$ is a contraction of G. The hurdle in this direction is that we have no idea whether $\mathcal{Z}_k$ is finite or not. The alternative pathway is to try to devise a linear parameterized algorithm by applying the algorithmic techniques that are already known for pathwidth.

The typical sequence technique. The main result of [8] was an algorithm that, given a path-decomposition Q of G of width at most k and an integer w, outputs, if exists, a path-decomposition of G of width at most w, in $2^{O(k(w+\log k))} \cdot n$ time. In this algorithm Bodlaender and Kloks introduced the celebrated *typical sequence technique*, a refined dynamic programming technique that encodes partial path/tree decompositions as a system of suitably compressed sequences of integers, able to encode all possible path-decompositions of width at most w (see also [15, 30]). This technique was later extended/adapted for the design of parametrized algorithms for numerous graph parameters such as branchwidth [9], linear-width [10], cutwidth [39], carving-width [38], modified cutwidth, and others [6,7,40]. In [6] this technique was viewed as a result of *un-nondeterminization*: a stepwise evolution of a trivial hypothetical non-deterministic algorithm towards a deterministic parameterized algorithm. A considerable generalization of the characteristic sequence technique was proposed in the PhD thesis of Soares [32] where this technique was implemented under the powerful meta-algorithmic framework of *q-branched Φ-width*. Non-trivial extensions of the typical sequence technique where proposed for devising parameterized algorithms for parameters on matroids such as matroid pathwidth [23], matroid branchwidth [25], as well as all the parameters on graphs or hypergraphs that can be expressed by them. Very recently Bodlaender, Jaffke, and Telle in [7] suggested refinements of the typical sequence technique that enabled the polynomial time computation of several width parameters on directed graphs. Finally, Bojańczyk and Pilipczuk suggested an alternative approach to the typical sequence technique, based on MSO transductions between decompositions [11].

Unfortunately, the above mentioned state of the art on the typical sequence technique is unable to encompass connected pathwidth. A reason for this is that the connectivity demand is a "global property" applying to *every* prefix of the path-decomposition which correspond to an unbounded number of subgraphs of arbitrary size.

Our result. In this paper we resolve *affirmatively* the conjecture that checking whether $\mathsf{cpw}(G) \leq k$ is fixed parameter tractable. Our main result is the following.

▶ **Theorem 1.** *One may construct an algorithm that given an n-vertex connected graph G, a path-decomposition $\mathsf{Q} = \langle B_1, \ldots, B_q \rangle$ of G of width at most k and an integer w, outputs a connected path-decomposition of G of width at most w or reports correctly that such an algorithm does not exist in $2^{O(k(w+\log k))} \cdot n$ time.*

To design an algorithm checking whether $\mathsf{cpw}(G) \leq k$ we first use the algorithms of [8] and [20], to build, if exists, a path decomposition of G of width at most k, in $2^{O(k^2)} \cdot n$ time. In case of a negative answer we know than $\mathsf{cpw}(G) > k$, otherwise we apply the algorithm of Theorem 1. The overall running time is dominated by the algorithm of Fürer in [20] which is $2^{O(k^2)} \cdot n$.

Our techniques. We now give a brief description of our techniques by focusing on the novel issues that we introduce. This description demands some familiarity with the typical sequence technique. Otherwise, the reader can go directly to the next section.

Let $\mathsf{Q} = \langle B_1, \ldots, B_q \rangle$ be a (nice) path-decomposition of G of width at most k. For every $i \in [q]$, we let $\mathbf{G}_i = (G_i, B_i)$ be the boundaried graph where $G_i = G[\bigcup_{h \in \{1,\ldots,i\}} B_h]$. We follow standard dynamic programming over a path-decomposition that consists in computing a representation of the set of partial solutions associated to $\mathbf{G}_i$, which in our case are *connected* path-decompositions of $\mathbf{G}_i$ of width at most w. The challenge is how to handle in a compact way the connectivity requirement of a path-decomposition of a graph that can be of arbitrarily large size.

A connected path-decomposition $\mathsf{P} = \langle A_1, \ldots, A_\ell \rangle$ of $\mathbf{G}_i$ is represented by means of a $(\mathbf{G}_i, \mathsf{P})$-*encoding sequence* $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$. For every $j \in [\ell]$, the element s_j of the sequence S is a triple $(\mathbf{bd}(\mathsf{s}_j), \mathbf{cc}(\mathsf{s}_j), \mathbf{val}(\mathsf{s}_j))$ where: $\mathbf{bd}(\mathsf{s}_i) = A_j \cap B_i$; $\mathbf{val}(\mathsf{s}_j) = |A_j \setminus B_i|$; and $\mathbf{cc}(\mathsf{s}_j)$ is the projection of the connected components of $G_i^j = G_i[\bigcup_{h \in \{1,\ldots,j\}} A_h]$ onto the subset of boundary vertices $B_i \cap V(G_i^j)$. To compress a $(\mathbf{G}_i, \mathsf{P})$-*encoding sequence* S, we identify a subset $\mathbf{bp}(\mathsf{S})$ of indexes, called *breakpoints*, such that $j \in \mathbf{bp}(\mathsf{S})$ if $\mathbf{bd}(\mathsf{s}_{j-1}) \neq \mathbf{bd}(\mathsf{s}_j)$ (type-1) or $\mathbf{cc}(\mathsf{s}_{j-1}) \neq \mathbf{cc}(\mathsf{s}_j)$ (type-2) or j is an index belonging to a typical sequence of the integer sequence $\langle \mathbf{val}(\mathsf{s}_b), \ldots, \mathbf{val}(\mathsf{s}_{c-1}) \rangle$ where $b, c \in [\ell]$ are consecutive type-1 or 2- breakpoints. We define $\mathsf{rep}(\mathsf{S})$ as the induced subsequence $\mathsf{S}[\mathbf{bp}(\mathsf{S})]$.

The novelty in this representation is the $\mathbf{cc}(\cdot)$ component which is a near-partition of the subset $B_i \cap V(G_i^j)$ of boundary vertices. The critical observation is that for every $j \in [\ell - 1]$, $\mathbf{cc}(\mathsf{s}_{j+1})$ is coarser than $\mathbf{cc}(\mathsf{s}_j)$. This, together with the known results on typical sequences, allows us to prove that the size of $\mathsf{rep}(\mathsf{S})$ is $O(kw)$ and that the number of representative sequences is $2^{O(k(w+\log k))}$. Finally, as in the typical sequence technique, we define a domination relation over the set of representative sequences. The DP algorithm over the path-decomposition Q consists then in computing a domination set $\mathbf{D}_w(G_{i+1})$ of the representative sequences of $\mathbf{G}_{i+1}$ from a domination set $\mathbf{D}_w(G_i)$ of the representative sequences of $\mathbf{G}_i$.

The above scheme extends the current state of the art on typical sequences as it further incorporates the encoding of the connectivity property. While this is indeed a "global property", it appears that its evolution with respect to the bags of the decomposition can be controlled by the second component of our encoding and this is done in terms of a sequence of a gradually coarsening partitions. This establishes a dynamic programming framework that can potentially be applied on the connected versions of most of the parameters where the typical sequence technique was used so far. Moreover, it may be the starting point of the algorithmic study of parameters where other, alternative to connectivity, global properties are imposed to the corresponding decompositions.

Consequences in connected graph searching. The original version of graph searching was the *edge searching* variant, defined [4] by Parsons [33,34], where the only differences with node searching is that a searcher can additionally slide along an edge and sliding is the only way to clean an edge. The corresponding search number is called *edge search number* and is denoted by $\mathsf{es}(G)$. If we additionally demand that the searching strategy is connected and monotone, then we define the *monotone connected edge search number* denoted by $\mathsf{mces}(G)$. As proved

[4] An equivalent model was proposed independently by Petrov [35]. The models of Parsons and Petrov where different but also equivalent, as proved by Golovach in [21,22]. The model of Parsons was inspired by an earlier paper by Breisch [12], titled *"An intuitive approach to speleotopology"*, where the aim was to rescue an (unlucky) speleologist lost in a system of caves. Notice that "unluckiness" cancels the speleologist's will of being rescued as, from the searchers' point of view, it imposes on him/her the status of an "evading entity". As a matter of fact, the connectivity issue appears even in the first inspiring model of the search game. In a more realistic scenario, the searchers cannot "teleport" themselves to non-adjacent territories of the caves while this was indeed permitted in the original setting of Parsons.

in [29], $\mathsf{es}(G) = \mathsf{pw}(G_{\mathsf{v}})$, where G_{v} is the graph obtained if we subdivide twice each edge of G. Applying the same reduction as in [29] for the monotone and connected setting, one can prove that $\mathsf{mces}(G) = \mathsf{cpw}(G_{\mathsf{v}})$. As we already mentioned, $\mathsf{mcns}(G) = \mathsf{cpw}(G_{\mathsf{v}}) + 1$. These two reductions imply that the result of Theorem 1 holds also for mcns and mces, i.e., the search numbers for the monotone and connected versions of both node and edge searching.

2 Preliminaries and definitions

2.1 Basic concepts

Sets and near-partitions. For an integer ℓ, we denote by $[\ell]$ the set $\{1, \ldots, \ell\}$. Let S be a finite set. A *near-partition* Q of S is a family of subsets $\{X_1, \ldots, X_k\}$ (with $k \leq |S| + 1$) of subsets of S, called *blocks*, such that $\bigcup_{i \in [k]} X_i = S$ and for every $1 \leq i < j \leq k$, $X_i \cap X_j = \emptyset$. Observe that a near-partition may contain several copies of the empty set. A *partition* of S is a near-partition with the additional constraint that if it contains the empty set, then this is the unique block. Let Q be a near-partition of a set S and Q' be a near-partition of a set S' such that $S \subseteq S'$. We say that Q is *thinner* than Q', or that Q' is *coarser* than Q, which we denote $\mathsf{Q} \sqsubseteq \mathsf{Q}'$, if for every block X of Q, there exists a block X' of Q' such that $X \subseteq X'$. For a near-partition $\mathsf{Q} = \{X_1, \ldots, X_\ell\}$ of S and a subset $S' \subseteq S$, we define the *projection of* Q *onto* S' as the near-partition $\mathsf{Q}_{|S'} = \{X_1 \cap S', \ldots, X_\ell \cap S'\}$. Observe that if Q is a partition, then $\mathsf{Q}_{|S'}$ may not be a partition: if several blocks of Q are subsets of $S \setminus S'$, then $\mathsf{Q}_{|S'}$ contains several copies of the emptyset.

Sequences. Let S be a set. A *sequence* of elements of S, denoted by $\alpha = \langle a_1, \ldots, a_\ell \rangle$, is a subset of S equipped with a total ordering: for $1 \leqslant i < j \leqslant \ell$, a_i occurs before a_j in the sequence α. The *length* of a sequence is the number of elements that it contains. Let $X \subseteq [\ell]$ be a subset of indexes of α. We define the *subsequence* of α *induced* by X as the sequence $\alpha[X]$ on the subset $\{a_i \mid i \in X\}$ such that, for $i, j \in X$, a_i occurs before a_j in $\alpha[X]$ if and only if $i < j$.

The *duplication* of the element a_j, with $j \in [\ell]$, in the sequence $\alpha = \langle a, \ldots, a_\ell \rangle$ yields the sequence $\alpha' = \langle a_1, \ldots, a_j, a_j, \ldots, a_\ell \rangle$ of length $\ell + 1$. A sequence β is an *extension* of the sequence α if it is either α or it results from a series of duplications on α. We define the set of extensions of α as: $\mathsf{Ext}(\alpha) = \{\alpha^* \mid \alpha^* \text{ is an extension of } \alpha\}$.

Let $\alpha = \langle a_1, \ldots, a_\ell \rangle$ be a sequence and $\alpha^* = \langle a_1, \ldots, a_p \rangle$ be an extension of α. If $p \leq \ell + k$, then α^* results from a series of at most k duplications and we say that α^* is a $(\leq k)$-*extension* of α. With the definition of an extension, every element of α^* is a copy of some element of α. We define the *extension surjection* as a surjective function $\delta_{\alpha^* \to \alpha} : [p] \to [\ell]$ such that if $\delta_{\alpha^* \to \alpha}(j) = i$ then $a^*_j = a_i$. An extension surjection $\delta_{\alpha^* \to \alpha}$ is a certificate that $\alpha^* \in \mathsf{Ext}(\alpha)$. Finally, we observe that if $\alpha^* \in \mathsf{Ext}(\alpha)$, then α is an induced subsequence of α^*. Moreover, if $\alpha^* \in \mathsf{Ext}(\alpha)$ and $\beta \in \mathsf{Ext}(\alpha^*)$, then β is an extension of α.

Graphs and boundaried graphs. Given a graph $G = (V, E)$ and a vertex set $S \subseteq V(G)$, we denote by $G[S]$ the *subgraph* of G that is *induced by the vertices of* S, i.e., the graph $(S, \{e \in E \mid e \subseteq S\})$. Also, if $x \in V$, we define $G \setminus x = G[V \setminus \{x\}]$. The *neighborhood* of a vertex v in G is the set of vertices that are adjacent to v in G and is denoted by $N_G(v)$.

A *boundaried graph* is a pair $\mathbf{G} = (G, B)$ such that G is a graph over a vertex set V and $B \subseteq V$ is a subset of distinguished vertices, called *boundary vertices*. We say that a boundaried graph $\mathbf{G} = (G, B)$ is *connected* if either G is connected and $B = \emptyset$ or, in case $B \neq \emptyset$, every connected component C of G contains some boundary vertex, that is $C \cap B \neq \emptyset$.

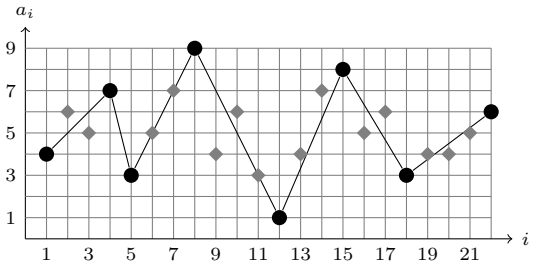

Figure 2 The black bullets forms the typical sequence $\mathsf{Tseq}(\alpha) = \langle 4,7,3,9,1,8,3,6\rangle$ of the sequence $\alpha = \langle 4,6,5,7,3,5,7,9,4,6,3,1,4,7,8,5,6,3,4,4,5,6\rangle$ of black and gray diamonds.

The definition of a connected path-decomposition also naturally extends to boundaried graphs as follows.

▶ **Definition 2.** *Let* $\mathsf{P} = \langle A_1, \ldots, A_\ell\rangle$ *be a path-decomposition of the boundaried graph* $\mathbf{G} = (G, B)$*. Then* P *is* connected *if, for every* $i \in [\ell]$*, the boundaried graph* $(G_i, V_i \cap B)$ *is connected, where* $V_i = \bigcup_{h\in[i]} A_h$ *and* $G_i = G[V_i]$*.*

2.2 Integer sequences

Let us recall the notion of *typical sequences* introduced by Bodlaender and Kloks [8] (see also [15, 30]).

▶ **Definition 3.** *Let* $\alpha = \langle a_1, \ldots, a_\ell\rangle$ *be an integer sequence. The* typical sequence $\mathsf{Tseq}(\alpha)$ *is obtained after iterating the following operations, until none is possible anymore:*

- *if for some* $i \in [\ell - 1]$*,* $a_i = a_{i+1}$*, then remove* a_{i+1} *from* α*;*
- *if there exists* $i, j \in [\ell]$ *such that* $i \leqslant j - 2$ *and* $\forall h$*,* $i < h < j$*,* $a_i \leq a_h \leq a_j$ *or* $\forall h$*,* $i < h < j$*,* $a_i \geq a_h \geq a_j$*, then remove the subsequence* $\langle a_{i+1}, \ldots, a_{j-1}\rangle$ *from* α*.*

As a typical sequence $\mathsf{Tseq}(\alpha) = \langle b_1, \ldots, b_i, \ldots, b_r\rangle$ is a subsequence of α, it follows that, for every $i \in [r]$, there exists $j_i \in [\ell]$ such that $b_i = a_{j_i}$. Herefater every such index j_i is called a *tip* of the sequence α.

If α and β are two integer sequences of same length ℓ, we say that $\alpha \leq \beta$ if for every $j \in [\ell]$, $a_j \leq b_j$.

▶ **Definition 4.** *Let* α *and* β *be two integer sequences. Then* $\alpha \preceq \beta$ *if there are* $\alpha^* \in \mathsf{Ext}(\alpha)$ *and* $\beta^* \in \mathsf{Ext}(\beta)$ *such that* $\alpha^* \leq \beta^*$*. Whenever* $\alpha \preceq \beta$ *and* $\beta \preceq \alpha$*, we say that* α *and* β *are* equivalent *which is denoted by* $\alpha \equiv \beta$*.*

We extend the definition of the $\leq$-relation and $\preceq$-relation on integer sequences to sequences of integer sequences. Let $\mathsf{P} = \langle \mathsf{L}_1, \ldots, , L_r\rangle$ and $\mathsf{Q} = \langle \mathsf{K}_1, \ldots, \mathsf{K}_r\rangle$ be two sequences of integer sequences such that for every $i \in [r]$, L_i and K_i have the same length. We say that $\mathsf{P} \leq \mathsf{Q}$ if for every $i \in [r]$, $\mathsf{L}_i \leq \mathsf{K}_i$. The set of *extensions* of P is $\mathsf{Ext}(\mathsf{P}) = \{\langle \mathsf{L}'_1, \ldots, \mathsf{L}'_r\rangle \mid i \in [r], \mathsf{L}'_i \in \mathsf{Ext}(\mathsf{L}_i)\}$. Finally we say that $\mathsf{P} \preceq \mathsf{Q}$ if there exist $\mathsf{P}' \in \mathsf{Ext}(\mathsf{P})$ and $\mathsf{Q}' \in \mathsf{Ext}(\mathsf{Q})$ such that $\mathsf{P}' \leq \mathsf{Q}'$. If $\mathsf{P} \preceq \mathsf{Q}$ and $\mathsf{Q} \preceq \mathsf{P}$, then we say that $\mathsf{P} \equiv \mathsf{Q}$. The relation $\equiv$ is an equivalence relation.

2.3 Boundaried sequences

We now define the main notion that will allow us to represent and manipulate (connected) path-decompositions of a boundaried graph $\mathbf{G} = (G, B)$ (see Subsection 3.1).

▶ **Definition 5** (B-boundaried sequence). *Let B be a finite set. A* B-boundaried sequence *is a sequence $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ such that for every $j \in [\ell]$, $\mathsf{s}_j = (\mathbf{bd}(\mathsf{s}_j), \mathbf{cc}(\mathsf{s}_j), \mathbf{val}(\mathsf{s}_j))$ is defined as follows:*

- *$\mathbf{bd}(\mathsf{s}_j) \subseteq B$ with the property that for every $x \in B$, the indices $j \in [l]$ such that $x \in \mathbf{bd}(\mathsf{s}_j)$ are consecutive;*
- *$\mathbf{cc}(\mathsf{s}_j)$ is a near-partition of $\bigcup_{i \leq j} \mathbf{bd}(\mathsf{s}_i) \subseteq B$ with the property that for every $j < \ell$, $\mathbf{cc}(\mathsf{s}_j) \sqsubseteq \mathbf{cc}(\mathsf{s}_{j+1})$;*
- *$\mathbf{val}(\mathsf{s}_j)$ is a positive integer.*

The width *of S is defined as $\mathsf{width}(\mathsf{S}) = \max_{j \in \ell}(|\mathbf{bd}(\mathsf{s}_j)| + \mathbf{val}(\mathsf{s}_j))$.*

▶ **Definition 6** (Connected B-boundaried sequence). *Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence for some finite set B. We say that S is* connected *if for every $i \in [\ell]$, $\mathbf{cc}(\mathsf{s}_i)$ is a partition of $\bigcup_{i \leq j} \mathbf{bd}(\mathsf{s}_i) \subseteq B$.*

Observe that if $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ is a connected B-boundaried sequence and if there exists some $i \in [\ell]$ such that $\mathbf{cc}(\mathsf{s}_i) = \{\emptyset\}$, then, for every $j \leq i$, $\mathbf{bd}(\mathsf{s}_j) = \emptyset$ and $\mathbf{cc}(\mathsf{s}_j) = \{\emptyset\}$.

▶ **Definition 7** (Breakpoints). *Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_j, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence for some finite set B. Then the index j, with $1 \leq j \leq \ell$, is a* breakpoint *of:*

- type-1 *if $j = 1$ or $\mathbf{bd}(\mathsf{s}_j) \neq \mathbf{bd}(\mathsf{s}_{j-1})$ or $j = \ell$;*
- type-2 *if it is not a type-1 breakpoint and $\mathbf{cc}(\mathsf{s}_j) \neq \mathbf{cc}(\mathsf{s}_{j-1})$;*
- type-3 *if it is not a type-1 nor a type-2 and j is a tip of the integer sequence $\langle \mathbf{val}(\mathsf{s}_{l_j}), \ldots, \mathbf{val}(\mathsf{s}_{r_j - 1}) \rangle$ where l_j and r_j are respectively the largest and smallest type-1 or type-2 breakpoints such that $l_j < j < r_j$.*

Let $\mathbf{bp}(\mathsf{S})$ be the set of breakpoints of S and $\mathbf{bp}_t(\mathsf{S})$ be the set of type-t breakpoints of S, for $t \in \{1, 2, 3\}$. The representative sequence $\mathsf{rep}(\mathsf{S})$ *of S is defined as $\mathsf{S}[\mathbf{bp}(\mathsf{S})]$.*

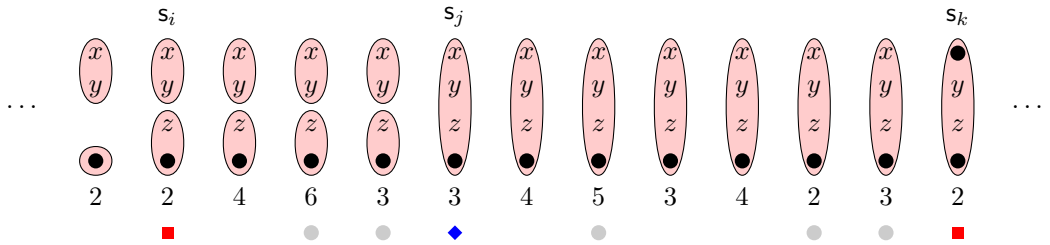

Figure 3 The part $\langle \mathsf{s}_{i-1}, \ldots \mathsf{s}_k \rangle$ of a B-boundaried sequence S for some set $B \supseteq \{x, y, z\}$. A bullet $\bullet$ at some index j represents an element of $\bigcup_{h<j} \mathbf{bd}(\mathsf{s}_h)$. Observe that at index k, x is indeed represented by a black bullet. For the index i, we have $\mathbf{bd}(\mathsf{s}_i) = \{x, y, z\}$, $\mathbf{cc}(\mathsf{s}_i) = \{\{x, y\}, \{z, \bullet\}\}$ and $\mathbf{val}(\mathsf{s}_i) = 2$. At every position j, only named elements belong to $\mathbf{bd}(\mathsf{s}_j)$. The red squares mark the type-1 breakpoints: at position i, element z is new, while at position k, element x is forgotten. The blue diamond at index j marks a type-2 breakpoint which corresponds to the merge of two parts of $\mathbf{cc}(\mathsf{s}_{i+4})$ into a single part. Finally, the grey bullets type-3 breakpoints corresponding to tips of the integer sequences $\langle \mathbf{val}(\mathsf{s}_i), \ldots \mathbf{val}(\mathsf{s}_{j-1}) \rangle$ and $\langle \mathbf{val}(\mathsf{s}_j), \ldots \mathbf{val}(\mathsf{s}_{k-1}) \rangle$.

Figure 3 illustrates the notions of B-boundaried sequence and breakpoints. Observe that $\mathsf{rep}(\mathsf{S})$ can be computed from the B-boundaried sequence S as in Definition 3 and is uniquely defined. The set of representative B-boundaried sequences of width at most w is defined as

$$\mathbf{Rep}_w(B) = \{\mathsf{rep}(\mathsf{S}) \mid \mathsf{S} \text{ is a } B\text{-boundaried sequence of width} \leq w\}.$$

▶ **Definition 8** (B-boundary model). *Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_j, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence. For every $j \in [\ell]$, we set $\dot{\mathsf{s}}_j = (\mathbf{bd}(\mathsf{s}_j), \mathbf{cc}(\mathsf{s}_j), \mathbf{t}(\mathsf{s}_j))$ with $\mathbf{t}(\mathsf{s}_j) = 1$ if $j \in \mathbf{bp}_1(\mathsf{S})$, $\mathbf{t}(\mathsf{s}_j) = 2$ if $j \in \mathbf{bp}_2(\mathsf{S})$ and $\mathbf{t}(\mathsf{s}_j) = 0$ otherwise. The B-*boundary model *of S, denoted by $\mathsf{model}(\mathsf{S})$, is the subsequence of $\dot{\mathsf{S}} = \langle \dot{\mathsf{s}}_1, \ldots, \dot{\mathsf{s}}_j, \ldots, \dot{\mathsf{s}}_\ell \rangle$ induced by $\mathbf{bp}_1(\mathsf{S}) \cup \mathbf{bp}_2(\mathsf{S})$.*

▶ **Lemma 9.** *Let* S *be a* B*-boundaried sequence. If* $\mathsf{S}^* \in \mathsf{Ext}(\mathsf{S})$*, then* $\mathsf{model}(S^*) = \mathsf{model}(S)$.

As in [8, 24], we will bound the number of representatives of B-boundaried sequences. For doing so, we bound the number of B-boundaried models and then use [8, Lemma 3.5] which gives an upper bound on the number of typical sequences. Taking into account the fact that there are $O(|B|)$ breakpoints and $|B|^{O(k)}$ different coarsening scenarios for the encoded near-partitions, we prove the following bound.

▶ **Lemma 10.** *Let* B *be a set of size* k*. Then,* $|\mathbf{Rep}_w(B)| = 2^{O(k(w+\log k)}$.

Notice that the notion of a B-boundary model corresponds to the one of *interval model* in [8]. Besides the B-boundary model of a sequence S, we introduce the *profile* of S, which corresponds to the concept of *list representation* in [8].

▶ **Definition 11** (Profile)**.** *Let* S *be a* B*-boundaried sequence of length* ℓ *and let* $1 = j_1 < \cdots < j_i < \cdots < j_r = \ell$ *be the subset of indices of* $[\ell]$ *that belong to* $\mathbf{bp}_1(\mathsf{S}) \cup \mathbf{bp}_2(\mathsf{S})$*. Then we set* $\mathsf{profile}(\mathsf{S}) = \langle \mathsf{L}_1, \ldots, \mathsf{L}_r \rangle$ *with, for* $i \in [r]$*,* $\mathsf{L}_j = \langle \mathbf{val}(\mathsf{s}_{j_i}), \ldots, \mathbf{val}(\mathsf{s}_{j_{i+1}-1}) \rangle$.

We introduce the domination relation over B-boundaried sequences. This allows us to compare B-boundaried sequences having the same model by means of their B-profiles.

▶ **Definition 12** (Domination relation)**.** *Let* $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_j, \ldots, \mathsf{s}_\ell \rangle$ *and* $\mathsf{T} = \langle \mathsf{t}_1, \ldots, \mathsf{t}_j, \ldots, \mathsf{t}_\ell \rangle$ *be two* B*-boundaried sequences such that* $\mathsf{model}(\mathsf{S}) = \mathsf{model}(\mathsf{T})$*. If* $\mathsf{profile}(\mathsf{S}) \leq \mathsf{profile}(\mathsf{T})$*, then we write* $\mathsf{S} \leq \mathsf{T}$*. And, we say that* S dominates T*, denoted by* $\mathsf{S} \preceq \mathsf{T}$*, if* $\mathsf{profile}(\mathsf{S}) \preceq \mathsf{profile}(\mathsf{T})$*. If we have* $\mathsf{profile}(\mathsf{S}) \preceq \mathsf{profile}(\mathsf{T})$ *and* $\mathsf{profile}(\mathsf{T}) \preceq \mathsf{profile}(\mathsf{S})$*, then we say that* S *and* T *are equivalent, which is denoted by* $\mathsf{S} \equiv \mathsf{T}$.

▶ **Lemma 13.** *Let* S *be a* B*-boundaried sequence. Then,*

1. $\mathsf{rep}(\mathsf{S}) \equiv \mathsf{S}$*,*
2. *if* $\mathsf{S}^* \in \mathit{Ext}(\mathsf{S})$*, then* $\mathsf{S}^* \equiv \mathsf{S}$*,*
3. $\mathsf{S} \preceq \mathsf{T}$ *if and only if* $\mathsf{rep}(\mathsf{S}) \preceq \mathsf{rep}(\mathsf{T})$*.*
4. *If* T *is a* B*-boundaried sequence such that* $\mathsf{S} \preceq \mathsf{T}$*, then there exist an extension* S^* *of* S *and an extension* T^* *of* T *such that* $\mathsf{S}^* \leq \mathsf{T}^*$*.*
5. *The relation* $\preceq$ *is transitive, and* $\equiv$ *is an equivalence relation (refering to boundary sequences).*

2.4 Operations on B-boundaried sequences

Given a finite set B, we define two operations on B-boundaried sequences that will be later used in the DP algorithm. The *projection* of a B-boundaried sequence S onto $B' \subsetneq B$ aims at changing the status of a boundary element from $B \setminus B'$ to the status of a non-boundary element. The second operation deals with the insertion in a B-boundaried sequence of a new boundary element x with respect to a subset $X \subseteq B$.

2.4.1 Projection of B-boundaried sequences

▶ **Definition 14** (Projection)**.** *Let* $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_i, \ldots, \mathsf{s}_\ell \rangle$ *be a* B*-boundaried sequence. For a subset* $B' \subseteq B$*, the* projection *of* S *onto* B' *is the* B'*-boundaried sequence* $\mathsf{S}_{|B'} = \langle \mathsf{s}_{1|B'}, \ldots, \mathsf{s}_{i|B'}, \ldots, \mathsf{s}_{\ell|B'} \rangle$ *such that for every* $i \in [\ell]$*:*

- $\mathbf{bd}(\mathsf{s}_{i|B'}) = \mathbf{bd}(\mathsf{s}_i) \cap B'$*;*
- $\mathbf{cc}(\mathsf{s}_{i|B'}) = \mathbf{cc}(\mathsf{s}_i)_{|B'}$*;*
- $\mathbf{val}(\mathsf{s}_{i|B'}) = \mathbf{val}(\mathsf{s}_i) + |\mathbf{bd}(\mathsf{s}_i) \setminus B'|$.

We observe that though the B-boundaried sequence S is connected, its projection $\mathsf{S}_{|B'}$ onto $B' \subseteq B$ may not be connected. This is the case if for some $j \in [\ell]$, the partition $\mathbf{cc}(\mathsf{s}_j)$ contains several blocks and at least one of them is a subset of $B \setminus B'$. Notice that the projection operation does not change the width of a sequence.

▶ **Lemma 15.** *Let B be a finite set and $B' \subsetneq B$. If S^* is an extension of a B-boundaried sequence S, then $\mathsf{S}^*_{|B'}$ is an extension of $\mathsf{S}_{|B'}$.*

▶ **Lemma 16.** *Let B be a finite set and $B' \subseteq B$. If S and T are B-boundaried sequences such that $\mathsf{S} \leq \mathsf{T}$, then $\mathsf{S}_{|B'} \leq \mathsf{T}_{|B'}$.*

Using Lemma 15 and Lemma 16, we prove the following:

▶ **Lemma 17.** *Let B be a finite set and $B' \subseteq B$. If S and T are B-boundaried sequences such that and $\mathsf{S} \preceq \mathsf{T}$, then $\mathsf{S}_{|B'} \preceq \mathsf{T}_{|B'}$.*

2.4.2 Insertion into a B-boundaried sequence

Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence and let X be a subset of B. An *insertion position* is a pair of indices (f_x, l_x) such that $1 \leq f_x \leq l_x \leq \ell$. An insertion position is *valid with respect to X in S* if $X \subseteq \bigcup_{f_x \leq j \leq l_x} \mathbf{bd}(\mathsf{s}_j)$.

▶ **Definition 18.** *Let $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle$ be a B-boundaried sequence and (f_x, l_x) be a valid insertion position with respect to $X \subseteq B$. Then $\mathsf{S}^x = \mathsf{Ins}(\mathsf{S}, x, X, f_x, l_x) = \langle \mathsf{s}^x_1, \ldots, \mathsf{s}^x_\ell \rangle$ is the $(B \cup \{x\})$-boundaried sequence such that for every $j \in [\ell]$:*

- *if $j < f_x$, then $\mathbf{bd}(\mathsf{s}^x_j) = \mathbf{bd}(\mathsf{s}_j)$; $\mathbf{cc}(\mathsf{s}^x_j) = \mathbf{cc}(\mathsf{s}_j)$ and $\mathbf{val}(\mathsf{s}^x_j) = \mathbf{val}(\mathsf{s}_j)$.*
- *if $f_x \leq j \leq l_x$, then $\mathbf{bd}(\mathsf{s}^x_j) = \mathbf{bd}(\mathsf{s}_j) \cup \{x\}$; $\mathbf{cc}(\mathsf{s}^x_j)$ is obtained by adding a new block $\{x\}$ to $\mathbf{cc}(\mathsf{s}_j)$ and then merging that new block with all the blocks of $\mathbf{cc}(\mathsf{s}_j)$ that contain an element of X (if any); $\mathbf{val}(\mathsf{s}^x_j) = \mathbf{val}(\mathsf{s}_j)$.*
- *and otherwise, $\mathbf{bd}(\mathsf{s}^x_j) = \mathbf{bd}(\mathsf{s}_j)$; $\mathbf{cc}(\mathsf{s}^x_j)$ is obtained by adding a new block $\{x\}$ to $\mathbf{cc}(\mathsf{s}_j)$ and then merging that new block with all the blocks of $\mathbf{cc}(\mathsf{s}_j)$ that contain an element of X (if any); $\mathbf{val}(\mathsf{s}^x_j) = \mathbf{val}(\mathsf{s}_j)$.*

We can show that if T is an extension of S, then, to every valid insertion position (f_x, l_x) with respect to some subset $X \subseteq B$ in S, one can associate a valid insertion position (f^*_x, l^*_x) with respect to X in T. The reverse is not true as illustrated by Figure 4.

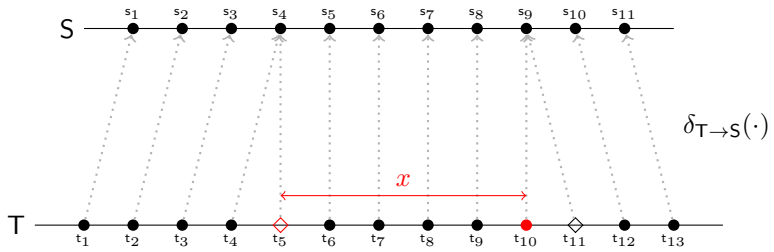

Figure 4 Let T be a 2-extension of the B-boundaried sequence S. Suppose that $(5, 10)$ is a valid insertion position with respect to some set $X \subseteq B$ in T. Observe that as $4 = \delta_{\mathsf{T}\to\mathsf{S}}(5)$ and $9 = \delta_{\mathsf{T}\to\mathsf{S}}(10)$, $(4, 9)$ is also a valid insertion position with respect to some for $X \subseteq B$ in S. However, $\mathsf{Ins}(\mathsf{T}, x, X, 5, 10)$ is not a 2-extension of $\mathsf{Ins}(\mathsf{S}, x, X, 4, 9)$.

However the next two lemmas provides an alternative by means of 2-extensions of S. The idea of the proof of Lemma 20 is illustrated by Figure 5.

▶ **Lemma 19.** *Let B and B' be finite sets with $B = B' \setminus \{x\}$ for some $x \in B'$. Let S and T be B-boundaried sequences such that $\mathsf{S} \leq \mathsf{T}$. If (f_x, l_x) is a valid insertion position with respect to a subset $X \subseteq B$ in T, then (f_x, l_x) is a valid insertion position with respect to X in S and $\mathsf{Ins}(\mathsf{S}, x, X, f_x, l_x) \leq \mathsf{Ins}(\mathsf{T}, x, X, f_x, l_x)$.*

▶ **Lemma 20.** *Let B and B' be finite sets with $B = B' \setminus \{x\}$ for some $x \in B'$. Let S and T be B-boundaried sequences such that $\mathsf{S} \preceq \mathsf{T}$. If (f_x^*, l_x^*) is a valid insertion position with respect to a subset $X \subseteq B$ in T, then there is a valid insertion position (f_x', l_x') in a (≤ 2)-extension R of S such that $\mathsf{Ins}(\mathsf{R}, x, X, f_x', l_x') \preceq \mathsf{Ins}(\mathsf{T}, x, X, f_x^*, l_x^*)$.*

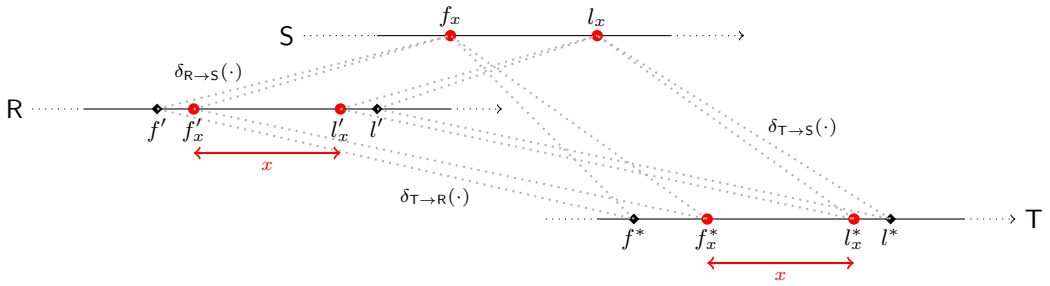

Figure 5 Let the B-boundaried sequence T be an extension of S that is certified by the surjective function $\delta_{\mathsf{T}\to\mathsf{S}}(\cdot)$ such that $f_x^* \neq f^*$, $l_x^* \neq l^*$ and $f^* = \min\{h \in [r] \mid \delta_{\mathsf{T}\to\mathsf{S}}(h) = f_x\}$, $l^* = \max\{h \in [r] \mid \delta_{\mathsf{T}\to\mathsf{S}}(h) = l_x\}$, $\delta_{\mathsf{T}\to\mathsf{S}}(f_x^*) = f_x$ and $\delta_{\mathsf{T}\to\mathsf{S}}(l_x^*) = l_x$. The B-boundaried sequence R is a 2-extension of S certified by the surjective function $\delta_{\mathsf{R}\to\mathsf{S}}(\cdot)$ such that $\delta_{\mathsf{R}\to\mathsf{S}}(f') = f_x$, $\delta_{\mathsf{R}\to\mathsf{S}}(f_x') = f_x$, $\delta_{\mathsf{R}\to\mathsf{S}}(l_x^*) = l_x$ and $\delta_{\mathsf{R}\to\mathsf{S}}(l') = l_x$. The fact that T is an extension of R can be certified by the surjective function $\delta_{\mathsf{T}\to\mathsf{R}}(\cdot)$ such that $\delta_{\mathsf{T}\to\mathsf{R}}(f^*) = f'$, $\delta_{\mathsf{T}\to\mathsf{R}}(f_x^*) = f_x'$, $\delta_{\mathsf{T}\to\mathsf{T}}(l_x^*) = l_x'$ and $\delta_{\mathsf{T}\to\mathsf{R}}(l^*) = l'$.

3 Computing the connected pathwidth

A (connected) path-decomposition $\mathsf{P} = \langle A_1, \ldots, A_\ell \rangle$ of a graph G is *nice* if $|A_1| = 1$ and $\forall i \in [p]$, $|A_{i-1} \vartriangle A_i| = 1$. A bag A_i, for $1 < 1 \leq \ell$, is called an *introduce* bag if $A_i \subsetneq A_{i-1}$ and a *forget* bag otherwise. As we will show, connected B-boundaried sequences are combinatorial objects designed in order to encode connected path-decompositions. Our algorithm is based on two routines. Forget Routine processes the forget bags by performing a projection operation on the B-boundaried sequences associated to those bags, while Insertion Routine handles the insertion bags by performing an insertion operation in the associated boundaried sequences.

3.1 Encoding a connected path-decomposition

▶ **Definition 21** ($(\mathbf{G}, \mathsf{P})$-encoding sequence). *Let $\mathsf{P} = \langle A_1, \ldots, A_\ell \rangle$ be a path-decomposition of the boundaried graph $\mathbf{G} = (G, B)$. A B-boundaried sequence $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_j, \ldots, \mathsf{s}_\ell \rangle$ is a $(\mathbf{G}, \mathsf{P})$-encoding sequence, if for every $j \in [\ell]$:*

- $\mathbf{bd}(\mathsf{s}_j) = A_j \cap B$*: the set of boundary vertices of (G, B) belonging to the bag A_j;*
- $\mathbf{cc}(\mathsf{s}_j) = \{V(C) \cap B \mid C$ *is a connected component of* $G_j\}$*;*
- $\mathbf{val}(\mathsf{s}_j) = |A_j \setminus B|$*: the number of non-boundary vertices in the bag A_j.*

It is worth to observe that $\mathbf{cc}(\mathsf{s}_j)$ is, in general, not a partition of A_j (see Figure 3). Also, notice that if G_j is connected and $B \cap V_j = \emptyset$, then $\mathbf{cc}(\mathsf{s}_j) = \{\emptyset\}$. Notice that if P is a connected path-decomposition, then S is a connected B-boundaried sequence.

▶ **Definition 22.** *Let* $\mathbf{G} = (G, B)$ *be a connected boundaried graph and* S *a* B*-boundaried sequence. We say that* S *is* realizable in $\mathbf{G}$ *if there is an extension* S^* *of* S *that is the* $(\mathbf{G}, \mathsf{P})$*-encoding sequence of some connected path-decomposition* P *of* $\mathbf{G}$.

Let us observe that if a B-boundaried sequence S is realizable, then S is connected. The set of *representative* B*-boundaried sequences of a connected boundaried graph* $\mathbf{G} = (G, B)$ *of width* $\leq w$ is defined as:

$$\mathbf{Rep}_w(\mathbf{G}) = \{\mathsf{rep}(\mathsf{S}) \mid \mathsf{S} \text{ of width} \leq w \text{ is realizable in } \mathbf{G} = (G, B)\}.$$

To compute the connected pathwidth of a graph, rather than computing $\mathbf{Rep}_w(\mathbf{G})$, we compute a subset $\mathbf{D}_w(\mathbf{G}) \subseteq \mathbf{Rep}_w(\mathbf{G})$, called *domination set*, such that for every representative B-boundaried sequence $\mathsf{S} \in \mathbf{Rep}_w(\mathbf{G})$, there exists a representative B-boundaried sequence $\mathsf{R} \in \mathbf{D}_w(\mathbf{G})$ where $\mathsf{R} \preceq \mathsf{S}$. Observe that a connected boundaried graph $\mathbf{G} = (G, B)$ has connected pathwidth at most w if and only if $\mathbf{D}_{w+1}(\mathbf{G}) \neq \emptyset$.

3.2 Forget Routine

Let $\mathbf{G} = (G, B)$ be a boundaried graph. If $x \in B$ is a boundary vertex, we denote by $B^{\overline{x}} = B \setminus \{x\}$. We define $\mathbf{G}^{\overline{x}} = (G, B^{\overline{x}})$, that is, while the graph G is left unchanged, we remove x from the set of boundary vertices. *Forget Routine* is described in Algorithm 1. Its correctness is proved in two steps. We first establish the *completeness* of the algorithm that is: for every connected path-decomposition P of $\mathbf{G}^{\overline{x}}$, there exists some B-boundaried sequence $\mathsf{S} \in \mathbf{D}_w(\mathbf{G})$ such that $\mathsf{rep}(\mathsf{S}_{|B\setminus\{x\}}) \preceq \mathsf{rep}(\mathsf{T})$ where T is the $(\mathbf{G}^{\overline{x}}, \mathsf{P})$-encoding sequence. For the soundness of the routine we prove that for every B-boundaried sequence $\mathsf{S} \in \mathbf{D}_w(\mathbf{G})$, $\mathsf{rep}(\mathsf{S}_{|B\setminus\{x\}}) \in \mathbf{D}_w(\mathbf{G}^{\overline{x}})$ if $\mathsf{S}_{|B\setminus\{x\}}$ is connected. The proofs of completeness and soundness rely both on Lemma 17. The time complexity is dominated by the bound on the number of representatives, given by Lemma 10.

Algorithm 1 Forget Routine.

Input: A boundaried graph $\mathbf{G} = (G, B)$, a vertex $x \in B$, and $\mathbf{D}_w(\mathbf{G})$.
Output: $\mathbf{D}_w(\mathbf{G}^{\overline{x}})$, a domination set of $\mathbf{Rep}_w(\mathbf{G}^{\overline{x}})$.

```
1 D_w(G^x̄) ← ∅;
2 foreach S ∈ D_w(G) do
3 |   if S_{|B\{x}} is connected, then add rep(S_{|B\{x}}) to D_w(G^x̄) ;
4 end
5 return D_w(G^x̄).
```

▶ **Theorem 23.** *Algorithm 1 computes* $\mathbf{D}_w(\mathbf{G}^{\overline{x}})$ *in* $2^{O(k(w+\log k))}$*-time, where* $k = |B|$.

3.3 Insertion Routine

Let $\mathbf{G} = (G, B)$ be a boundaried graph with $G = (V, E)$. For a subset $X \subseteq B$, we set $G^x = (V \cup \{x\}, E \cup \{xy \mid y \in X\})$ and $\mathbf{G}^x = (G^x, B^x)$ where $B^x = B \cup \{x\}$. Algorithm 2 is describing Insertion Routine (notations of Figure 5 are used in the pseudo-code). To prove its correctness, we proceed in two steps. We first establish the *completeness* of the algorithm: for every connected path-decomposition P^x of $\mathbf{G}^x$, the $(\mathbf{G}^x, \mathsf{P}^x)$-encoding sequence T^x is dominated by some B^x-boundaried sequence S^x that can be computed from a B-boundaried sequence S belonging to $\mathbf{D}_w(\mathbf{G})$. Then we argue about the *soundness* of Insertion Routine

Algorithm 2 Insertion Routine.

Input: A boundaried graph $\mathbf{G} = (G, B)$, a subset $X \subsetneq B$, and $\mathbf{D}_w(\mathbf{G})$.
Output: $\mathbf{D}_w(\mathbf{G}^x)$, a domination set of $\mathbf{Rep}_w(\mathbf{G}^x)$.

$\mathbf{D}_w(\mathbf{G}^x) \leftarrow \emptyset$;
foreach $\mathsf{S} = \langle \mathsf{s}_1, \ldots, \mathsf{s}_\ell \rangle \in \mathbf{D}_w(\mathbf{G})$ **do**
  foreach $f_x, l_x \in [\ell]$ *such that* $X \subseteq \bigcup_{f_x \leq j \leq l_x} \mathbf{bd}(\mathsf{s}_j)$ **do**
    foreach (≤ 2)-*extension* R *of* S *duplicating none, one or both of* s_{f_x} *and* s_{l_x} **do**
      let ℓ' be the length of R;
      set $f'_x = \max\{j \in [\ell'] \mid \delta_{\mathsf{R}\to\mathsf{S}}(j) = f_x\}$ and $l'_x = \min\{j \in [\ell'] \mid \delta_{\mathsf{R}\to\mathsf{S}}(j) = l_x\}$;
      set $\mathsf{S}^x = \mathsf{Ins}(\mathsf{R}, x, X, f'_x, l'_x)$;
      (observe that by construction (f'_x, l'_x) is valid with respect to X in R);
      if $\mathsf{width}(\mathsf{S}^x) \leq w$, **then** add $\mathsf{rep}(\mathsf{S}^x)$ to $\mathbf{D}_w(\mathbf{G}^x)$;
    end
  end
end
return $\mathbf{D}_w(\mathbf{G}^x)$.

that is: if S^x is generated from a B-boundaried $\mathsf{S} \in \mathbf{D}_w(\mathbf{G})$, then $\mathsf{rep}(\mathsf{S}^x)$ belongs to $\mathbf{D}_w(\mathbf{G}^x)$. The proofs of completeness and soundness rely both on Lemma 19, and Lemma 20. As for Forget Routine, the time complexity follows from Lemma 10.

▶ **Theorem 24.** *Algorithm 2 computes* $\mathbf{D}_w(\mathbf{G}^x)$ *in* $2^{O(k(w+\log k))}$*-time, where* $k = |B|$.

3.4 The dynamic programming algorithm

We are now in position to prove Theorem 1. We are given a nice path-decompositon $\mathsf{Q} = \langle B_1, \ldots, B_q \rangle$ of G of width at most k and for each $i \in [q]$, we consider the boundaried graphs $\mathbf{G}_i = (G[V_i], B_i)$, where $V_i = \bigcup_{1 \leq h \leq i} B_h$. We have a way to compute $\mathbf{D}_w(G_{i+1})$ from $\mathbf{D}_w(G_i)$, in $2^{O(k^2)} \cdot n$ time, using the algorithms of Theorem 23 or Theorem 24 depending on whether B_i is an insertion or a forget bag. We next describe the set $\mathbf{D}_{w+1}(\mathbf{G}_1)$. For this, we take the representative set $\mathbf{Rep}_{w+1}(\mathbf{G}_1)$ that consists for the following four possible connected B_1-boundaried sequences:

- $\mathsf{S}_1 = \langle (\{x\}, \{\{x\}\}, 0) \rangle$,
- $\mathsf{S}_2 = \langle (\emptyset, \{\emptyset\}, 0), (\{x\}, \{\{x\}\}, 0) \rangle$,
- $\mathsf{S}_3 = \langle (\emptyset, \{\emptyset\}, 0), (\{x\}, \{\{x\}\}, 0), (\emptyset, \{\{x\}\}, 0) \rangle$, and
- $\mathsf{S}_4 = \langle (\{x\}, \{\{x\}\}, 0), (\emptyset, \{\{x\}\}, 0) \rangle$.

As already noticed $\mathsf{cpw}(G) \leq k$ if and only if $\mathbf{D}_{w+1}(\mathbf{G}_q) \neq \emptyset$. This completes proof of the decision version of Theorem 1. In [8, Section 6] Bodlaender and Kloks explained how to turn their decision algorithm for pathwidth and treewidth to one that is able to construct, in case of a positive answer, the corresponding decomposition. It is straightforward to see that the modification of [8, Section 6] that transforms the decision algorithm for pathwidth to one that also constructs the corresponding path-decomposition also applies to our algorithm for connected pathwidth. This completes the proof of Theorem 1.

If we now use the result of Fürer [20] for constructing a path-decomposition of width at most k in $2^{O(k^2)} \cdot n$ time and taking into account that $\mathsf{pw}(G) \leq \mathsf{cpw}(G)$, we have the following.

▶ **Theorem 25.** *One may construct an algorithm that, given an* n*-connected graph* G *and a non-negative integer* k*, either outputs a connected path-decomposition of* G *of width at most* k *or correctly reports that such a decomposition does not exist in* $2^{O(k^2)} \cdot n$ *time.*

References

1 Isolde Adler, Christophe Paul, and Dimitrios M. Thilikos. Connected search for a lazy robber. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 150 of *Leibniz International Proceedings in Informatics*, pages 7:1–7:14, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.7.

2 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

3 Lali Barrière, Paola Flocchini, Fedor V. Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, and Dimitrios M. Thilikos. Connected graph searching. *Information and Computation*, 219:1–16, 2012. doi:10.1016/j.ic.2012.08.004.

4 Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In 14th Annual ACM Symposium on Parallel Algorithms and Architectures, *SPAA*, pages 200–209. ACM, 2002. doi:10.1145/564870.564906.

5 Lali Barrière, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M. Thilikos. Searching is not jumping. In 29th International Workshop on Graph-Theoretic Concepts in Computer Science, *WG*, volume 2880 of *Lecture Notes in Computer Science*, pages 34–45, 2003. doi:10.1007/978-3-540-39890-5_4.

6 Hans L. Bodlaender, Michael R. Fellows, and Dimitrios M. Thilikos. Derivation of algorithms for cutwidth and related graph layout parameters. *Journal of Computer and System Sciences*, 75(4):231–244, 2009. doi:10.1016/j.jcss.2008.10.003.

7 Hans L. Bodlaender, Lars Jaffke, and Jan Arne Telle. Typical sequences revisited - computing width parameters of graphs. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS*, volume 154 of *Leibniz International Proceedings in Informatics*, pages 57:1–57:16, 2020. doi:10.4230/LIPIcs.STACS.2020.57.

8 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.

9 Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In 24th International Colloquium on Automata, Languages and Programming, *ICALP*, volume 1256 of *Lecture Notes in Computer Science*, pages 627–637, 1997. doi:10.1007/3-540-63165-8_217.

10 Hans L. Bodlaender and Dimitrios M. Thilikos. Computing small search numbers in linear time. In First International Workshop on Parameterized and Exact Computation, *IWPEC*, volume 3162 of *Lecture Notes in Computer Science*, pages 37–48, 2004. doi:10.1007/978-3-540-28639-4_4.

11 Mikołaj Bojańczyk and Michal Pilipczuk. Optimizing tree decompositions in MSO. In 34th Symposium on Theoretical Aspects of Computer Science, *STACS*, volume 66 of *Leibniz International Proceedings in Informatics*, pages 15:1–15:13, 2017. doi:10.4230/LIPIcs.STACS.2017.15.

12 R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers (A publication of the Southwestern Region of the National Speleological Society)*, VI(5):72–78, 1967.

13 Gary Chartrand, Ping Zhang, Teresa W. Haynes, Michael A. Henning, Fred R. McMorris, and Robert C. Brigham. Graphical measurement. In *Handbook of Graph Theory*, pages 872–951. Chapman & Hall / Taylor & Francis, 2003. doi:10.1201/9780203490204.ch9.

14 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

15 Bruno Courcelle and Jens Lagergren. Equivalent definitions of recognizability for sets of graphs of bounded tree-width. *Mathematical Structure for Computer Sciecnce*, 6(2):141–165, 1996. doi:10.1017/S096012950000092X.

16 Dariusz Dereniowski. From pathwidth to connected pathwidth. *SIAM Journal on Discrete Mathematics*, 26(4):1709–1732, 2012. doi:10.1137/110826424.

17 Dariusz Dereniowski, Dorota Osula, and Paweł Rzążewski. Finding small-width connected path decompositions in polynomial time. *Theoretical Computer Science*, 794:85–100, 2019. doi:10.1016/j.tcs.2019.03.039.

18 Fedor V. Fomin. Complexity of connected search when the number of searchers is small. Open Problems of GRASTA 2017: The 6th Workshop on GRAph Searching, Theory and Applications, 2017.

19 Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008. doi:10.1016/j.tcs.2008.02.040.

20 Martin Fürer. Faster computation of path-width. In 27th International Workshop on Combinatorial Algorithms*, IWOCA*, Lecture Notes in Computer Science, pages 385–396, 2016. doi:10.1007/978-3-319-44543-4_30.

21 P. A. Golovach. Equivalence of two formalizations of a search problem on a graph (Russian). *Vestnik Leningrad. Univ. Mat. Mekh. Astronom.*, vyp. 1:10–14, 122, 1989. translation in Vestnik Leningrad Univ. Math. 22 (1989), no. 1, 13–19.

22 Petr A. Golovach (П. А. Головач). A topological invariant in pursuit problems, (Об одном топологическом инварианте в задачах преследования). *Differentsialânye Uravneniya (Differential Equations),* (Дифференц. уравнения), 25(6):923–929, 1989. URL: http://mi.mathnet.ru/de6861.

23 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Constructive algorithm for path-width of matroids. In 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA*, pages 1695–1704, 2016. doi:10.1137/1.9781611974331.ch116.

24 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. The "art of trellis decoding" is fixed-parameter tractable. *IEEE Transactions on Information Theory*, 63(11):7178–7205, 2017. doi:10.1109/TIT.2017.2740283.

25 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Finding branch-decompositions of matroids, hypergraphs, and more. In 45th International Colloquium on Automata, Languages, and Programming*, ICALP*, volume 107 of *Leibniz International Proceedings in Informatics*, pages 80:1–80:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.80.

26 Mamadou Moustapha Kanté, Christophe Paul, and Dimitrios M. Thilikos. A linear fixed parameter tractable algorithm for connected pathwidth. *CoRR*, abs/2004.11937, 2020. arXiv:2004.11937.

27 Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 1992. doi:10.1016/0020-0190(92)90234-M.

28 Lefteris M. Kirousis and Christos H. Papadimitriou. Interval graphs and seatching. *Discrete Mathematics*, 55(2):181–184, 1985. doi:10.1016/0012-365X(85)90046-9.

29 Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(3):205–218, 1986. doi:10.1016/0304-3975(86)90146-5.

30 Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In 18th International Colloquium on Automata, Languages and Programming*, ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 532–543, 1991. doi:10.1007/3-540-54233-7_161.

31 Rolf H. Möhring. Graph problems related to gate matrix layout and PLA folding. In *Computational graph theory*, volume 7 of *Comput. Suppl.*, pages 17–51. Springer, 1990.

32 Ronan Pardo Soares. *Pursuit-Evasion, Decompositions and Convexity on Graphs*. Theses, Université Nice Sophia Antipolis, November 2013. URL: https://tel.archives-ouvertes.fr/tel-00908227.

33 Torrence D. Parsons. Pursuit-evasion in a graph. In International Conference on Theory and applications of graphs, volume 642 of *Lecture Notes in Mathematics*, pages 426–441. Springer, 1978.

34 Torrence D. Parsons. The search number of a connected graph. In 9th Southeastern Conference on Combinatorics, Graph Theory and Computing, volume XXI of *Congress. Numer., XXI*, pages 549–554. Utilitas Math., 1978.

35 Nikolai N. Petrov (Н. Н. Петров). A problem of pursuit in the absence of information on the pursued, (Задачи преследования при отсутствии информации об убегающем). *Differentsial'nye Uravneniya (Differential Equations),* (Дифференц. уравнения), 18(8):1345–1352, 1468, 1982. URL: http://mi.mathnet.ru/de4628.

36 Neil Robertson and Paul D. Seymour. Graph minors. I. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

37 Neil Robertson and Paul D. Seymour. Graph minors. XX. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

38 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Constructive linear time algorithms for small cutwidth and carving-width. In 11th International Conference on Algorithms and Computation*, ISAAC*, volume 1969 of *Lecture Notes in Computer Science*, pages 192–203, 2000. doi:10.1007/3-540-40996-3_17.

39 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, 2005. doi:10.1016/j.jalgor.2004.12.001.

40 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth II: algorithms for partial w-trees of bounded degree. *Journal of Algorithms*, 56(1):25–49, 2005. doi:10.1016/j.jalgor.2004.12.003.