TU WIEN Informatics

Graphenparameter: Strukturelle Eigenschaften und Algorithmische Anwendungen

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

eingereicht von

Viktoriia Korchemna
Matrikelnummer 11931227

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Robert Ganian

Diese Dissertation haben begutachtet:

Christian Komusiewicz | Dušan Knop

Wien, 17. Oktober 2024

Viktoriia Korchemna

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Graph Parameters: Structural Properties and Algorithmic Applications

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Viktoriia Korchemna
Registration Number 11931227

to the Faculty of Informatics

at the TU Wien

Advisor: Robert Ganian

The dissertation has been reviewed by:

Christian Komusiewicz

Dušan Knop

Vienna, October 17, 2024

Viktoriia Korchemna

Erklärung zur Verfassung der Arbeit

Viktoriia Korchemna

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 17. Oktober 2024

Viktoriia Korchemna

Acknowledgements

First of all, I owe a deep sense of gratitude to my teacher and advisor Robert Ganian, for the exciting and challenging problems he suggested to me and for hours of brainstorming we spent together while working on them. In addition to scientific work itself, he always encouraged me to expand my horizons by visiting workshops, conferences, and research visits, and eventually such events became the most inspiring and fun part of my research.

I am also very grateful to Stefan Szeider, the head of Algorithms and Complexity Group, for creating a very friendly, open, and creative research atmosphere, which I could enjoy during my Ph.D. studies.

It is a pleasure for me to express the highest appreciation to all co-authors with whom I was fortunate to meet and work a lot: Cornelius Brand and Argirios Deligkas for their sense of humor and brave ideas that actually work, Eduard Eiben and Thekla Hamm for their kindness, patience, and deep structural insights, Karolina Okrasa for her optimism and pioneer spirit, Kirill Simonov, Simon Schierreich, and Michael Skotnica for their sincere interest and willingness to discuss new ideas.

I would like to say a special thank you to Anna Prianichnikova: not only for giving me a unique opportunity to receive Helmut Veith Stipend (which in fact enabled my education and future research at TU Wien) but also for her compassion and strong support she provided to my family and friends who moved to Austria because of the war in Ukraine.

Last but not least, I am grateful to all my teachers: my lyceum teacher Svitlana Kushnir, who had awakened a curiosity for mathematics and logic within me, my father Sergiy Ageyev and my lyceum teacher Ernest Volkov, who took me into first journeys to the world of computer science and algorithms, and of course my bachelor thesis advisor Andriy Oliynyk for providing a highly interactive introduction into computability and complexity theory and for guiding me through my first independent research.

Kurzfassung

Parametrisierte Komplexität [CFK+15] bietet eine reiche Werkzeugkiste, um mit unlösbaren Problemen umzugehen, die in verschiedenen Bereichen der Informatik auftreten. Eines der Hauptziele hierbei ist es, eine feinkörnigere Komplexitätsanalyse von rechnerisch schwierigen Problemen zu erhalten, die nicht nur auf der Eingabegröße, sondern auch auf einem bestimmten *Parameter* basiert ist. Die Analoga zu polynomial lösbareren Problemen hier sind die *parametrisierbare* (engl. *fixed-parameter tractable* oder *FPT*) Probleme, die einen Algorithmus mit einer Laufzeit von $f(k) \cdot n^{\mathcal{O}(1)}$ zulassen, wobei f eine berechenbare Funktion, n die Eingabegröße und k ein Parameter sind. In dieser Arbeit untersuchen wir die parametrisierte Komplexität von Problemen, deren Instanzen Graphendarstellungen zulassen, sodass die Parameter, die wir betrachten, mit den strukturellen Eigenschaften des Eingabegraphen verbunden sind. Wir konzentrieren uns auf dekompositionelle Graphenparameter, die es ermöglichen, Graphen entlang einer baumähnlichen Struktur in Komponenten zu zerlegen.

Unter den dekompositionellen Graphenparametern ist die *Baumweite* [RS86] am prominentesten, welche die Baumähnlichkeit eines Graphen anhand von Knoten-Trenner misst. Um Probleme anzugehen, die in Bezug auf sie keine FPT sind, werden häufig restriktivere auf Knoten-Schnitt (engl. vertex-cut) basierende Parameter wie Pfadweite [RS83], Baumtiefe [NdM12] oder die Knotenüberdeckungszahl [FPR+18] verwendet. Bei einigen Problemen ist es jedoch natürlich, stattdessen auf Kanten-Schnitt basierende Parameter zu betrachten, und hier ist die Auswahl viel begrenzter, was große Lücken im Verständnis der Komplexität dieser Probleme hinterlässt. Unser Ziel in dieser Arbeit ist es, diese Lücke in der Hierarchie der auf Kanten-Schnitt basierenden Graphenparameter zu schließen. Als unseren Hauptbeitrag führen wir einen neuen Parameter ein, die *slim Tree-Cut Width*, welche eine Einschränkung der *Tree-Cut Width* [Wol15] ist, und zeigen, dass sie perfekt für eine Reihe klassischer Graphenprobleme passt, bei denen die Parametrisierung durch letzteren nicht hilft.

Wir führen eine tiefe theoretische Untersuchung der strukturellen Eigenschaften der slim Tree-Cut Width durch. Insbesondere bieten wir drei äquivalente Charakterisierungen des Parameters: (i) tatsächliche *slim Tree-Cut Width*, definiert mit Hilfe von Baum-Schnitt-Zerlegungen (engl. tree-cut decompositions) und vielen Vorteilen der Tree-Cut Width ererbend, (ii) *super Edge-Cut Width*, die eine einfach zu verwendende Zerlegung in Form eines Spannbaums in Algorithmen bietet, und (iii) die Charakterisierung in Bezug auf

Blockzerlegungen, die es ermöglicht, Graphen in Komponenten mit kleiner Baumweite und maximalem Grad zu zerlegen. Wir bieten mehrere Beispiele, die bezeugen, dass slim Tree-Cut Width und andere auf Kanten-Schnitt basierende Parameter nicht nur in vielen klassischen Graphenproblemen, sondern auch in einigen grundlegenden Problemen, die aus der KI stammen, wie Datenvervollständigung und kausales Netzwerklernen, Tragfähigkeit erreichen helfen.

Abstract

Parameterized complexity [CFK+15] provides a rich toolbox for dealing with intractable problems that arise in different areas of computer science. One of the main aims here is to obtain a more fine-grained complexity analysis of computationally hard problems based not only on the input size but also on some *parameter*. The analogues of polynomially solvable problems here are *fixed-parameter tractable* problems, which admit an algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f, where n is an input size and k is a parameter. In this work, we investigate the parameterized complexity of problems whose instances admit graph representations, so the parameters we consider are tied to structural properties of the input graph. We focus on decompositional graph parameters, which allow to decompose graphs into components along some tree-like structure.

Among the decompositional graph parameters, the most prominent is *treewidth* [RS86], which measures the treelikeness of a graph in terms of vertex separators. To handle problems where it does not help to achieve fixed-parameter tractability, more restrictive vertex-cut based parameters such as pathwidth [RS83], treedepth [NdM12], or vertex cover number [FPR+18] are commonly used. However, for some problems, it is natural to consider edge-cut-based parameters instead, and here the choice is much more limited, leaving large gaps in the understanding of complexity of these problems. Our aim in this work is to fill this gap in the hierarchy of edge-cut based graph parameters. As our main contribution, we introduce a new parameter *slim tree-cut width* which is a restriction of *tree-cut width* [Wol15], and show that it is a perfect fit for a number of classical graph problems where the parameterization by the latter does not help.

We perform a deep theoretical study of the structural properties of slim tree-cut width. In particular, we provide three equivalent characterizations of the parameter: (i) actual *slim tree-cut width*, defined with the help of tree-cut decompositions and inheriting many advantages of the tree-cut width, (ii) *super edge-cut width*, providing an easy-to-use in algorithms decomposition in terms of a spanning tree, and (iii) the characterization in terms of *block decompositions*, allowing to decompose graph into components of small treewidth and maximum degree. We provide multiple examples witnessing that slim tree-cut width and other edge-cut based parameters help achieve tractability not only in many classical graph problems but also in some fundamental problems originating from AI, such as data completion and causal network learning.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Most fundamental problems that arise in various areas of computer science and beyond, such as constraint satisfaction, resource allocation, graph drawing, causal network learning, and many others, cannot be solved efficiently: based on common complexity-theoretical assumptions, these problems do not admit algorithms with polynomial running times. Despite their differences, all of these problems have a common characteristic: their complexity is derived from an exponential search space, even though the structure of a potential solution is often relatively simple. In particular, they are typically NP-hard [GJ79].

Parameterized complexity [DF13] provides a powerful set of tools to overcome the intractability of NP-hard problems by designing parameterized algorithms [CFK+15] which allow to efficiently solve some specific classes of instances. This can be achieved by using different approaches: one is to restrict the problem to instances of a specific input structure, while another is to require additional properties from solutions, narrowing the search space. This process often occurs gradually, when identification of some basic property (parameter) that enables a fast algorithm is followed by sequential relaxations, resulting in as general restrictions as possible.

Many problems admit a natural graph representation, which has led to the development of a rich hierarchy of structural graph parameters (see, e.g., Figure 1 in [BJK13]). Numerous works have been conducted in parameterized complexity, exploring the limits of tractability with respect to these parameters.

Treewidth [Bod05], introduced by Robertson and Seymour, is by far the most prominent parameter in this hierarchy, and it is known that many problems of interest are fixed-parameter tractable when parameterized by treewidth. However, in this work, we will primarily be interested in problems that lie on the other side of this spectrum: those which

remain intractable when parameterized by treewidth. One way to deal with such problems is to use a more restrictive decompositional parameter, for instance, treedepth [NdM12], and as we will see when dealing with synchronous dynamical systems, sometimes it is a perfect fit. However, for many problems (such as well-established Edge Disjoint Paths (EDP)) even vertex cover—the most restrictive parameter based on vertex separators—does not help achieve tractability. This motivates us to try an alternative approach—namely, to consider a decompositional parameter based on edge-cuts.

A first natural candidate for such a parameter would be tree-cut width, discovered by Wollan, who described it in terms of tree-cut decompositions—variations of tree decompositions based on edge cuts instead of vertex separators [Wol15]. Although it is true that "tree-cut decompositions share many of the natural properties of tree decompositions" [MW14], from the perspective of algorithmic design, tree-cut width seems to behave differently than an edge-cut based alternative to treewidth. To provide one example, consider the EDP problem mentioned above: Vertex Disjoint Paths is a classical example of a problem that is FPT parameterized by treewidth, and one should by all means expect a similar outcome for EDP parameterized by the analogue of treewidth based on edge cuts. However, EDP is W[1]-hard parameterized by tree-cut width. At the same time, it is fixed-parameter tractable w.r.t. treewidth plus maximum degree. In the latter parameterization, small degree and vertex separators allow to obtain small edge-cuts for free. However, restriction on the maximum degree is quite strong, as it might be violated even on trees and stars. And EDP is not an exception: out of twelve problems that are W[1]-hard parameterized by treewidth but fixed parameter-tractable w.r.t. treewidth plus maximum degree, where a tree-cut width parameterization has been pursued so far, only four are fixed-parameter tractable [GKS22, GKO21] while eight turn out to be W[1]-hard [BHKN19, GKS22, GK21, GO21, GÖP$^+$17] (see Table 3.1).

So, what is the algorithmic analogue of treewidth for edge cuts? In this work, we develop a candidate for such a measure, which we call *slim tree-cut width*. The name here is not random: our parameter is defined similarly to the tree-cut width but imposes stronger restrictions on the tree-cut decompositions, making them sparser or thinner. Crucially, the slim tree-cut width allows one to handle graphs with high degrees. As we show, it makes all eight problems mentioned above fixed-parameter tractable. Moreover, it turns out that the slim tree-cut width helps not only in classical graph-theoretical problems but also in fundamental problems arising in AI, such as causal network learning and data completion. Last but not least, it inherits all desirable structural properties of tree-cut width and yet admits an easy-to-use decomposition which is simply a spanning tree.

1.2 Organisation of Thesis

In Chapter 2 we introduce basic notions and techniques related to parameterized complexity and graph theory. In particular, we define such fundamental graph parameters as treewidth, treedepth, and tree-cut width, which will be used throughout our work. Most of the definitions provided in this chapter are rather classical, and for this reason, we

omit technical details, but refer an interested reader to a well-established literature.

Our contribution begins with Chapter 3, where we make an attempt to design an algorithmically-driven edge-cut based analogue of treewidth. We start by introducing a new decompositional parameter called *edge-cut width*, which, intuitively, measures the number of cycles intersecting in one vertex. Crucially, this parameter allows us to achieve tractability for all eight problems mentioned above. Moreover, we ensure that it can be computed in FPT time. However, despite its practical advantages and easy-to-use spanning tree decomposition, the parameter has some drawbacks. In particular, it is not closed under vertex and edge deletions. We provide these results—both positive and negative—in Section 3.1, which is based on our joint work with C. Brand, E. Ceylan, R. Ganian and C. Hatschka [BCG+22].

The next Section 3.2 emerged from our follow-up work with R. Ganian [GK22a, GK24] that targeted development of a more structurally complete edge-cut measure. *Slim tree-cut width*, introduced and studied here, fills the gaps left by its predecessor edge-cut width: it is closed under taking subgraphs and generalizes both edge-cut width and treewidth plus maximum degree, but is more restrictive than tree-cut width. Still, we show that all the tractability results achieved for the edge-cut width can be easily transferred to the slim tree-cut width. In this context, it can be considered as a unifying reason why the problems are FPT by both edge-cut width and degree treewidth but not tree-cut width.

We provide three equivalent characterizations of the slim tree-cut width. The first is based on tree-cut decomposition, similarly to the definition of tree-cut width. It allows us to derive a number of nice structural properties: the slim tree-cut width is closed under taking weak immersions, $k-$edge sums, admits a simple characterization in terms of excluded weak immersions and can be approximated in FPT time. The second characterization is in terms of *super edge-cut width*, obtained by a slight modification of the definition of edge-cut width. This definition is more convenient in the context of dynamic programming, and we use it to prove fixed-parameter tractability for a variant of maximum stable matching in Section 3.3.

In principle, fixed-parameter tractability of the rest of the problems studied in [BCG+22] w.r.t. super edge-cut width (and hence slim tree-cut width) could be established in a similar fashion, by generalizing FPT algorithms for the edge-cut width. However, as we have discovered later, slim tree-cut width admits one more equivalent characterization–namely, in terms of block decompositions–that allows us to derive tractability directly from known results for the degree treewidth. This set of results is based on our recent journal publication [GK24] and is provided in Section 3.4. We conclude the chapter with Section 3.5 comparing edge-cut width and slim tree-cut width and their relation to other recent edge-cut measures.

While most of the problems studied in Chapter 3 are classical graph-theoretical problems, in the next chapters we demonstrate that slim tree-cut width and other edge-cut based parameters are also extremely helpful for a variety of other problems, in particular those arising in artificial intelligence research.

In particular, Chapter 4, based on our recent joint work with R. Ganian and S. Szeider [GKS24], deals with the classical task of learning causal relationships (represented as a so-called causal graph) from observational data. Here, we investigate the complex relationship between the graph structure and the efficiency of constraint-based causal discovery algorithms. First, in Section 4.4 we provide a near-tight characterization of which causal graphs can potentially be efficiently recovered by a constraint-based causal discovery algorithm. Then, in Section 4.5 we explicitly construct a sequence of causal graphs on which the influential PC algorithm works extremely inefficiently. Finally, in Section 4.6 we formulate a new causal discovery algorithm which achieves fixed-parameter running time by considering the maximum number of edge-disjoint paths between variables in the (undirected) super-structure as the parameter.

In Chapter 5, we study a fundamental problem called k-means clustering, also related to data completion. In particular, we focus on the settings where most of the data is not known or simply irrelevant. Our approach is based on exploiting the structural properties of a graphical encoding of the missing entries. We consider two natural encodings—incidence and primal graphs—and provide three novel fixed-parameter algorithms: one for the clustering of bounded-domain data and two incomparable algorithms that target real-valued data. The first two algorithms exploit treewidth, while the latter uses slim tree-cut width and can be considered as a first step towards resolving a long-standing open problem for the real-valued data. This chapter is based on our joint work with R. Ganian, T. Hamm, K. Okrasa and K. Simonov [GHK$^+$22].

The last set of our results, provided in Chapter 6, is related to synchronous dynamic systems, well-established models that have been used to capture a range of phenomena in networks, including opinion diffusion, spread of disease, and product adoption. We study the three most notable problems in synchronous dynamic systems: whether the system will transition to a target configuration from a starting configuration, whether the system will reach convergence from a starting configuration, and whether the system is guaranteed to converge from every possible starting configuration. Although all three problems were known to be intractable in the classical sense, we initiate the study of their exact boundaries of tractability from the perspective of structural parameters of the network. In particular, we show that all three problems remain intractable even on instances of small constant treewidth. We complement this negative finding with fixed-parameter algorithms for the former two problems parameterized by treedepth. For the convergence guarantee, treedepth alone is known to be insufficient to obtain a similar algorithm. However, we design a fixed-parameter algorithm when additionaly parameterized by maximum in-degree. The results provided in this chapter are based on our joint work with E. Eiben, R. Ganian and T. Hamm [EGHK23].

For completeness, we declare that not all of the research work carried out during the Ph.D. studies is included in this thesis. In particular, for the sake of conciseness and coherence, scientific contributions to the areas of fine-grained complexity [GHK$^+$24], resource allocation [DEKS24], algebraic algorithms [BKS23, BKSS24], and approximation algorithms [KLS$^+$24] were not included in this manuscript.

CHAPTER 2

Preliminaries

We denote the set of natural numbers by $\mathbf{N}$. For $i \in \mathbf{N}$, let $[i]$ denote the set $\{1, \ldots, i\}$ of the first i natural numbers, and let $[i]_0 = [i] \cup \{0\}$.

2.1 Parameterized Complexity Theory

In parameterized algorithmics [CFK+15, DF13, Nie06] the running time of an algorithm is studied with respect to a parameter $k \in \mathbf{N}$ and input size n. The basic idea is to find a parameter that describes the structure of the instance such that the combinatorial explosion in the running time can be confined to this parameter. In this respect, the most favorable complexity class is FPT (*fixed-parameter tractable*) which contains all problems that can be decided by an algorithm running in time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a computable function. Algorithms with this running time are called *fixed-parameter* algorithms or just FPT algorithms.

A less favorable outcome is an XP algorithm, which is an algorithm running in time $\mathcal{O}(n^{f(k)})$. Problems admitting such algorithms form the class XP. Another important complexity class is W[1]. A proper definition of the class is relatively involved and is not needed to establish our results. For a general impression, let us mention that W[1] is contained in XP and showing W[1]-hardness of a problem rules out the existence of a fixed-parameter algorithm under the well-established assumption that W[1] $\neq$ FPT. This is usually done via a parameterized reduction from some known W[1]-hard problem. A *parameterized reduction* from a parameterized problem P to a parameterized problem Q is a function:

- which maps Yes-instances to Yes-instances and No-instances to No-instances,
- which can be computed in time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a computable function, and
- where the parameter of the output instance can be upper-bounded by some function of the parameter of the input instance.

Moreover, there also exists an even less favorable outcome than W[1]-hardness for a parameterized problem: it may turn out to be NP-hard even for a fixed value of the parameter. For a more comprehensive introduction to parameterized complexity, we refer the interested reader to the monograph by Cygan et al. [CFK+15].

2.2 Graphs and Directed Graphs

We use the standard terminology of Diestel [Die17]. Let G be a graph with the vertex (node) set V and the edge set E, we denote this by $G = (V, E)$. For a subset $V' \subseteq V$ of vertices of G, we denote by $G[V']$ the graph with the vertex set V' and the edge set E'' consisting of the edges of G with both endpoints in V'. That is, $G[V'] = (V', E'')$, where $E'' = \{e \in E : e \subseteq V'\}$. Moreover, for a subset E' of E, we denote by $G[E']$ a graph G with the set of vertices $\bigcup_{e' \in E'} e'$ and the set of edges E'. We say that $G[V']$ ($G[E']$) is a vertex-induced (edge-induced) subgraph of G induced by a set V' (E' respectively). A vertex-induced subgraph of G that is a tree is called an *induced subtree* of G.

In contrast, we use $G \setminus W$ ($G \setminus E'$) to denote the graph obtained from G by removing all the vertices in W (edges in E'). Here, when removing a vertex, we also remove all its incident edges, that is, the graph $G \setminus W$ has the vertex set $V \setminus W$ and the edge set $E \cap ((V \setminus W) \times (V \setminus W))$. In case W consists of a single vertex w, we may also write $G \setminus w$ to denote $G \setminus \{w\}$, similarly for edges. Moreover, if H is a subgraph of G, we denote by $G \setminus H$ the graph obtained from G by removing all vertices and edges of H. Moreover, we denote the graph obtained by adding a set V' of vertices (set E' of edges) to the set of vertices (edges) of G by $G \cup V'$ ($G \cup E'$ respectively). Furthermore, if G and H are two (not necessarily vertex and edge disjoint) graphs, we denote their union by $G \cup H$, that is, $G \cup H$ is a graph with the vertex set $V(G) \cup V(H)$ and the edge set $E(G) \cup E(H)$. For simplicity, we use the short notation xy for an edge $e = \{x, y\}$.

Given a graph G, we denote its vertex set by $V(G)$ and its edge set by $E(G)$. The *(open) neighborhood* of a vertex $x \in V(G)$ is the set $\{y \in V(G) \mid xy \in E(G)\}$ and is denoted by $N_G(x)$. For a vertex subset X, the neighborhood of X is defined as $\bigcup_{x \in X} N_G(x) \setminus X$ and is denoted by $N_G(X)$; we drop the subscript if the graph is clear from the context. If H is a subgraph of G, we denote it by $H \subseteq G$. *Contracting* an edge $\{x, y\}$ is the operation of replacing vertices x, y by a new vertex whose neighborhood is $(N(x) \cup N(y)) \setminus \{x, y\}$.

A *path* π between a vertex x_1 and a vertex x_n in G is a non-repeating sequence of vertices $x_1, \ldots, x_n$ such that for each pair of vertices x_i and x_{i+1}, there is an edge $x_i x_{i+1}$ in $E(G)$. If $x_1, \ldots, x_n$ is a path in G and G contains an edge between x_1 and x_n, we say that $x_1, \ldots, x_n, x_1$ is a *cycle* in G. Note that any path π in G defines both the set of vertices $V_\pi = \{x_1, \ldots, x_n\}$ and the set of edges $E_\pi = \{x_1x_2, \ldots, x_{n-1}x_n\}$, which form a subgraph (V_π, E_π) of G; sometimes we will identify π with this subgraph.

A graph is called a *forest* if it has no cycles. A graph is called *connected* if there is a path between any pair of its distinct vertices. A connected forest is called a *tree*. We refer to a tree with some fixed vertex (called *root*) as *rooted tree*. A *rooted forest* is a

disjoint union of rooted trees. Note that every graph G is a disjoint union of connected graphs, we call these graphs *connected components* of G. In particular, the connected components of a forest are trees.

Let x, y and z be some vertices of G. An *edge-cut* in G which separates x from y is a set E' of edges such that x and y lie in different connected components of $G \setminus E'$; we say that E' *touches* a vertex z if it contains at least one edge incident to y.

Let x, y and z be three distinct vertices of G such that $xy, yz \in E(G)$. To *lift* the pair of edges xy, yz means to delete the edges xy and yz from G and add (if it doesn't exist yet) a new edge xz. We say that G contains H as *weak immersion* (denoted $H \leq_I G$) if and only if H can be obtained from G by a sequence of edge deletion, vertex deletion, and lifting operations.

For a natural number k, we say that a graph G is a *k-edge sum* of vertex-disjoint graphs G_1 and G_2 if there exist vertices $x_i \in V(G_i)$ of degree k for $i = 1, 2$ and a bijection $\pi : N_{G_1}(x_1) \to N_{G_2}(x_2)$ such that G is obtained from $(G_1 \setminus x_1) \cup (G_2 \setminus x_2)$ by adding an edge $(x, \pi(x))$ for every $x \in N_{G_1}(x_1)$. In this case, we write $G = G_1 \oplus_k G_2$. Observe that the same pair of graphs may produce different k-edge sums.

We say that a tree T is a *spanning tree* of a connected graph G if $V(T) = V(G)$ and $E(T) \subseteq E(G)$. In case G is not connected and consists of connected components $G_1, \ldots, G_l$, we say that a graph F is a *maximal spanning forest* of G if F is a disjoint union of trees $T_1, \ldots, T_l$ such that T_i is a spanning tree of G_i, $i \in [l]$.

To obtain some of our results, we will also need to consider directed graphs, i.e., digraphs. For a digraph D, we denote by $V(D)$ and $A(D)$ its sets of vertices and arcs respectively. If D is a digraph with vertex set V and arc set A, we denote this by $D = (V, A)$. For an arc $(x, y) \in A$ we may use a short notation xy. The *in-neighborhood* of a vertex $x \in V(D)$ is the set $\{y \in V(D) \mid yx \in A(D)\}$, and the *out-neighborhood* of x is the set $\{y \in V(D) \mid xy \in A(D)\}$. The elements of these sets are called the *in-* and *out-neighbors* of x, and the sizes of these sets are called the *in-degree* and *out-degree* of x, respectively. The same as for graphs, if the digraph H is a subgraph of a digraph D, we denote this as $H \subseteq D$. The notions of deleting a vertex set, an arc set, or a subgraph, as well as taking a union of two digraphs are defined analogously to the case of graphs.

A *skeleton* (sometimes called the *underlying undirected graph*) $\underline{D}$ of a directed graph D is the simple graph obtained by replacing each arc in D with an undirected edge. An *(undirected) path* between x_1 and x_n in digraph D is a path between x_1 and x_n in the skeleton of D. The path is *directed* if for each pair of vertices x_i and x_{i+1}, there is an arc $x_i x_{i+1}$ in $A(D)$. If such a path exists, we say that x_n is *reachable* from x_1 or, equivalently, x_n is a *descendant* of x_1 in D (x_1 is an *ancestor* of x_n in D). As a special case for $n = 1$, every vertex is reachable from (is a descendant and an ancestor of) itself. Similarly to the case of graphs, each directed path defines a subgraph of D, and we will often identify them for simplicity.

2.3 Structural Graph Parameters

A natural approach to parameterizing problems dealing with graphs is to consider fundamental structural graph parameters, such as the well-established notion of treewidth [RS86]. In the context of digraphs, it is common to consider the structural parameters of the underlying undirected graph.

Among basic graph parameters, one usually distinguishes the feedback vetex and edge numbers, the vertex cover number. An edge set $E' \subseteq E(G)$ is a *feedback edge set* of G if $G \setminus E'$ is a forest. The *feedback edge number* of a graph G, denoted by $\mathrm{fen}(G)$, is the smallest integer k such that G has a feedback edge set of size k. The notions of *feedback vertex set* and *feedback vertex number* of G, are defined analogously. A vertex set $V' \subseteq V(G)$ is a *vertex cover* of G if for each $e \in E(G)$ we have $e \cap V' \neq \emptyset$. The *vertex cover number* of a graph G, denoted by $\mathrm{vcn}(G)$, is the smallest integer k such that G admits a vertex cover of size k.

Let $\mathcal{G}$ denote the class of all finite graphs. Given two graph parameters $\alpha, \beta : \mathcal{G} \to \mathbf{N}$, we say that α *dominates* β if there exists a function p such that for each graph $G \in \mathcal{G}$, it holds that $\alpha(G) \leq p(\beta(G))$. If α dominates β but β does not dominate α, we often say that β is *more restrictive* than α (or, equivalently, α is *more general* than β. As an example, treewidth is more general than vertex cover number. Two parameters that dominate each other are called *asymptotically equivalent*.

2.3.1 Treewidth

Treewidth is a well-established fundamental graph parameter which intuitively captures the "tree-likeness" of a graph [RS86]. One way to define it is in terms of tree decompositions. A *tree decomposition* $\mathcal{T}$ of a graph $G = (V, E)$ is a pair (T, χ), where T is a tree (whose vertices we call *nodes*) rooted at a node r and χ is a function that assigns each node t a set $\chi(t) \subseteq V$ such that:

- For every $uv \in E$ there is a node t where $u, v \in \chi(t)$.
- For every vertex $v \in V$, the set of nodes t satisfying $v \in \chi(t)$ forms a subtree of T.
- $|\chi(\ell)| = 1$ for every leaf ℓ of T and $|\chi(r)| = 0$.

The tree decomposition $\mathcal{T}$ is called *nice* if T has only three types of non-leaf nodes:

- **Introduce node:** a node t with exactly one child t' such that $\chi(t) = \chi(t') \cup \{v\}$ for some vertex $v \notin \chi(t')$.
- **Forget node:** a node t with exactly one child t' such that $\chi(t) = \chi(t') \setminus \{v\}$ for some vertex $v \in \chi(t')$.
- **Join node:** a node t with two children t_1, t_2 such that $\chi(t) = \chi(t_1) = \chi(t_2)$.

Definition 2.1 (Treewidth)**.** *The* width *of a tree decomposition* (T, χ) *is defined as* $\mathrm{tw}(T, \chi) = \max_{t \in V(T)}(|\chi(t)| - 1)$*, and the* treewidth *of the graph* G*, denoted* $\mathrm{tw}(G)$*, is the minimum width of a tree decomposition of* G*.*

We let T_t denote the subtree of T rooted at a node t, and we use $\chi_t^{\downarrow}$ to denote the set $\bigcup_{t' \in V(T_t)} \chi(t')$. The sets $\chi(t)$ are commonly called *bags*. Any tree decomposition can be transformed into a nice tree decomposition of the same width.

Fixed-parameter algorithms are known for computing (nice) tree decompositions of optimal width and linearly many nodes [KL23, Bod96, Klo94], albeit more efficient fixed-parameter approximation algorithms are often used to achieve better running times [Kor22, BDD+16]. The notion of *pathwidth* is defined similarly with the only difference that in a tree decomposition (T, χ), T must be a path. In particular, pathwidth is more restrictive than treewidth. To see that vertex cover number is more restrictive than pathwidth (and hence treewidth), assume that a graph G admits vertex cover V' of size k. Consider a tree decomposition (T, χ) such that T is a path of length $|V(G)| - k$ and each bag contains V' plus a unique vertex from $V(G) \setminus V'$. Then the size of each bag is equal to $k + 1$, so $\mathrm{tw}(T, \chi) = k + 1$. Conversely, trees have treewidth one but arbitrary large vertex cover numbers.

2.3.2 Treedepth

Treedepth is a parameter closely related to treewidth—in particular, the treedepth of a graph is lower-bounded by its treewidth. A useful way of thinking about graphs of bounded treedepth is that they are (sparse) graphs with no long paths. We formalize the parameter below.

Let F be rooted forest. For a node x in a tree T of F, the *height* (or *depth*) of x in F is the number of nodes in the path from the root of T to x. The *height of a rooted forest* is the maximum height of a node of the forest.

Definition 2.2 (Treedepth)**.** *Let the* closure *of a rooted forest* F *be the graph* $\lambda(F) = (V_c, E_c)$*, where* $V_c = V(F)$ *and* $E_c = \{xy \mid x \text{ is an ancestor of } y \text{ in some } T \subseteq F\}$*. A* treedepth decomposition *of a graph* G *is a rooted forest* F *such that* $G \subseteq \lambda(F)$*. The* treedepth $\mathrm{td}(G)$ *of a graph* G *is the minimum height of a treedepth decomposition of* G*.*

It is known that an optimal-width treedepth decomposition can be computed by a fixed-parameter algorithm [NdM12, RRVS14, NPS22] and also, e.g., via a SAT encoding [GLOS19]; hence, in our algorithms we assume that such a decomposition is provided on the input.

2.3.3 Tree-Cut Width

The notion of tree-cut decompositions was introduced by Wollan [Wol15], see also subsequent work by Marx and Wollan [MW14]. A family of subsets $X_1, \ldots, X_k$ of X is a *near-partition* of X if they are pairwise disjoint and $\bigcup_{i=1}^{k} X_i = X$, allowing the possibility of $X_i = \emptyset$.

Definition 2.3. *A* tree-cut decomposition *of G is a pair $(T, \mathcal{X})$ which consists of a rooted tree T and a near-partition $\mathcal{X} = \{X_t \subseteq V(G) \mid t \in V(T)\}$ of $V(G)$. A set in the family $\mathcal{X}$ is called a* bag *of the tree-cut decomposition.*

For any node t of T other than the root r, let $e(t) = ut$ be the unique edge incident to t on the path to r. Let T_u and T_t be the two connected components in $T \setminus e(t)$ which contain u and t, respectively. Note that $(\bigcup_{q \in T_u} X_q, \bigcup_{q \in T_t} X_q)$ is a near-partition of $V(G)$, and we use E_t to denote the set of edges with one endpoint in each part. We define the *adhesion* of t ($\mathrm{adh}(t)$) as $|E_t|$; we explicitly set $\mathrm{adh}(r) = 0$ and $E_r = \emptyset$. The adhesion of $(T, \mathcal{X})$ is then $\mathrm{adh}(T, \mathcal{X}) = \max_{t \in V(T)} \mathrm{adh}(t)$.

The *torso* of a tree-cut decomposition $(T, \mathcal{X})$ at a node t, written as H_t, is the graph obtained from G as follows. If T consists of a single node t, then the torso of $(T, \mathcal{X})$ at t is G. Otherwise, let $T_1, \ldots, T_\ell$ be the connected components of $T - t$. For each $i = 1, \ldots, \ell$, the vertex set $Z_i \subseteq V(G)$ is defined as the set $\bigcup_{b \in V(T_i)} X_b$. The torso H_t at t is obtained from G by *consolidating* each vertex set Z_i into a single vertex z_i (this is also called *shrinking* in the literature). Here, the operation of consolidating a vertex set Z into z is to substitute Z by z in G, and for each edge e between Z and $v \in V(G) \setminus Z$, adding an edge zv in the new graph. We note that this may create parallel edges.

The operation of *suppressing* (also called *dissolving* in the literature) a vertex v of degree at most 2 consists of deleting v, and when the degree is two, adding an edge between the neighbors of v. Given a connected graph G and $X \subseteq V(G)$, let the *3-center* of (G, X) be the unique graph obtained from G by exhaustively suppressing vertices in $V(G) \setminus X$ of degree at most two. Finally, for a node t of T, we denote by $\tilde{H}_t$ the 3-center of (H_t, X_t), where H_t is the torso of $(T, \mathcal{X})$ at t. Let the *torso-size* $\mathrm{tor}(t)$ denote $|\tilde{H}_t|$.

Definition 2.4. *The width of a tree-cut decomposition $(T, \mathcal{X})$ of G is defined as $\mathrm{tcw}(T, \mathcal{X}) = \max_{t \in V(T)} \{\mathrm{adh}(t), \mathrm{tor}(t)\}$. The tree-cut width of G, or $\mathrm{tcw}(G)$ in short, is the minimum $\mathrm{tcw}(T, \mathcal{X})$ over all tree-cut decompositions $(T, \mathcal{X})$ of G.*

Without loss of generality, we shall assume that $X_r = \emptyset$. We conclude this section with some notation related to tree-cut decompositions. Given a tree node t, let T_t be the subtree of T rooted at t. Let $Y_t = \bigcup_{b \in V(T_t)} X_b$, and let G_t denote the vertex-induced subgraph $G[Y_t]$. A node $t \neq r$ in a rooted tree-cut decomposition is *thin* if $\mathrm{adh}(t) \leq 2$ and *bold* otherwise.

A tree-cut decomposition $(T, \mathcal{X})$ is *nice* if it satisfies the following condition for every thin node $t \in V(T)$: $N(Y_t) \cap (\bigcup_{b \text{ is a sibling of } t} Y_b) = \emptyset$. The intuition behind nice tree-cut decompositions is that we restrict the neighborhood of thin nodes in a way which

facilitates dynamic programming. Every tree-cut decomposition of width k can be transformed into a nice tree-cut decomposition of the same width in cubic time [GKS22]. Moreover, the resulting nice decomposition has the following property. For a node t, let $B_t = \{b \text{ is a child of } t \mid |N(Y_b)| \leq 2 \wedge N(Y_b) \subseteq X_t\}$ denote the set of thin children of t whose neighborhood is a subset of X_t, and let $A_t = \{a \text{ is a child of } t \mid a \notin B_t\}$ be the set of all other children of t. Then $|A_t| \leq 2k + 1$ for every node t [GKS22].

We refer to previous work [MW14, Wol15, KOP+18, GKS22] for a detailed comparison of tree-cut width to other parameters. Here, we mention only that tree-cut width is dominated by treewidth and dominates treewidth plus maximum degree, which we denote $\operatorname{degtw}(G)$ and may also refer to as *degree treewidth*. Tree-cut width also dominates the feedback edge number.

Lemma 2.1 ([GKS22, MW14, Wol15])**.** *For every graph G, it holds that*

$$\operatorname{tw}(G) \leq 2\operatorname{tcw}(G)^2 + 3\operatorname{tcw}(G),\ \operatorname{tcw}(G) \leq \operatorname{fen}(G) + 1 \text{ and } \operatorname{tcw}(G) \leq 4\operatorname{degtw}(G)^2.$$

CHAPTER 3

Refined Edge-Cut Measures

Tree-cut width parameterizations were typically considered for problems which are not fixed-parameter tractable (FPT) w.r.t. treewidth, but are FPT w.r.t. feedback edge number and also FPT w.r.t. treewidth plus maximum degree. The twelve candidate problems where tree-cut width has been considered are shown in Table 3.1. However, out of these twelve problems, only four are fixed-parameter tractable with respect to the tree-cut width, while eight turn out to be W[1]-hard.

Our main goal in this chapter is to develop a new edge-cut based measure which yields fixed-parameter tractability for all of these problems.

3.1 Edge-Cut Width

Our first candidate for such a new edge-cut based parameter is edge-cut width. On the algorithmic side, edge-cut width has precisely the properties one could hope to see in an edge-based analogue to treewidth: not only does it yield fixed-parameter algorithms for all twelve "candidate" problems, but it is also based on a very simple type of decomposition that is much easier to use than tree-cut decompositions.

3.1.1 Algorithmically-Driven Definition

The definition of edge-cut width is based on the notion of *local feedback edge set*, introduced in a joint work with R. Ganian [GK21]. For a graph G and a maximal spanning forest T of G, let the *local feedback edge set* at $v \in V$ be

$$E^{G,T}_{\mathtt{loc}}(v) = \{uw \in E(G) \setminus E(T) \mid \text{the path between } u \text{ and } w \text{ in } T \text{ contains } v\}.$$

Definition 3.1. *The edge-cut width of the pair* (G,T) *is* $\mathrm{ecw}(G,T) = 1+\max_{v\in V} |E^{G,T}_{\mathtt{loc}}(v)|$, *and the edge-cut width of* G (*denoted* $\mathrm{ecw}(G)$) *is the smallest edge-cut width among all possible maximal spanning forests* T *of* G.

Problem	tcw	ecw	degtw
Capacitated Vertex Cover	FPT [GKS22]	FPT	FPT
Capacitated Dominating Set	FPT [GKS22]	FPT	FPT
Imbalance	FPT [GKS22]	FPT	FPT
Bounded Degree Deletion	FPT [GKO21]	FPT	FPT
Edge Disjoint Paths	W[1]-hard	FPT	FPT
List Coloring	W[1]-hard	FPT	FPT
Precoloring Extension	W[1]-hard	FPT	FPT
Boolean Constraint Satisfaction	W[1]-hard	FPT	FPT
Bayesian Network Structure Learning	W[1]-hard	FPT	FPT
Polytree Learning	W[1]-hard	FPT	FPT
Min. Changeover Cost Arborescence	W[1]-hard	FPT	FPT
Max. Stable Roommates (MSRTIL)	W[1]-hard	FPT	FPT

Table 3.1: The twelve candidate problems and their complexity w.r.t. edge-cut based parameters tree-cut width, edge-cut width and degree treewidth. As we will see in next sections, slim tree-cut width provides a unified explanation for why these problems are FPT w.r.t. both ecw and degtw, and lifts these results to more general inputs. The complexity of Bayesian Network Structure Learning and Polytree Learning was studied in [GK21]. In particular, the results obtained there immediately imply fixed-parameter tractability with respect to edge-cut width, see [BCG+22] for more details and for the rest of FPT results for the edge-cut width.

Notice that the definition increments the edge-cut width of T by 1. This "cosmetic" change may seem arbitrary, but it matches the situation for treewidth (where the width is the bag size minus one) and allows trees to have a width of 1. Moreover, defining edge-cut width in this way provides a more concise description of the running times for our algorithms, where the records will usually depend on a set that is one larger than $|E^{G,T}_{\mathtt{loc}}(v)|$. We note that the predecessor to edge-cut width, called the *local feedback edge number* (lfen(G)) [GK21], was defined without this cosmetic change and hence is equal to edge-cut width minus one.

Proposition 3.1 ([BCG+22])**.** *For every graph G,* $\mathrm{tcw}(G) \leq \mathrm{ecw}(G) \leq \mathrm{fen}(G) + 1$.

In fact, it was shown in [BCG+22] that the gaps in both inequalities can be arbitrary large, see Figure 3.1 for a simple example of the second one.

While Lemma 2.1 and Proposition 3.1 together imply that $\mathrm{tw}(G) \leq 2\,\mathrm{ecw}(G)^2 + 3\,\mathrm{ecw}(G)$, one can in fact show that the gap is linear. This will also allow us to provide a better running time bound in Section 3.1.2.

Lemma 3.2 ([BCG+22])**.** *For every graph G,* $\mathrm{tw}(G) \leq \mathrm{ecw}(G)$.

Proof. Let T be a spanning tree of G such that $\mathrm{ecw}(G) = \mathrm{ecw}(G, T)$. We arbitrarily pick a root r in T and construct the tree decomposition (T, χ) of G as follows. At first,

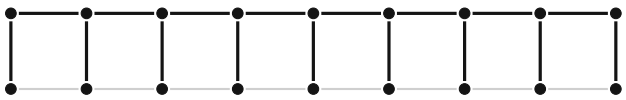

Figure 3.1: Example of a graph G with a spanning tree T (thick black) such that $\mathrm{ecw}(G) = \mathrm{ecw}(G, T) = 3$. The feedback edge number of G can be made arbitrarily large in this fashion.

for every $v \in V(G)$, we add to $\chi(v)$ the vertex v and the parent of v in T (if it exists). Obviously, after this step each vertex v of G appears in some bag and every edge of T is contained as a subset in some bag. Moreover, v appears only in $\chi(v)$ and in the bags of children of v in T, which results in a connected subtree of T.

To complete the construction, we process feedback edges one by one. For every $e \in E(G) \setminus E(T)$, we arbitrarily choose an endpoint u of $e = uw$ and add u to each bag $\chi(v)$ such that $u \in E^{G,T}_{\mathtt{loc}}(v)$. Note that any such step does not violate the connectivity condition. Indeed, we add u to the bags of all vertices which lie on the path between the endpoints of e in T. In particular, the path hits u whose bag $\chi(u)$ initially contained u. Finally, both endpoints of e appear in $\chi(w)$. In the resulting decomposition, for each $v \in V(G)$ it holds that $|\chi(v)| \leq 2 + E^{G,T}_{\mathtt{loc}}(v) \leq 1 + \mathrm{ecw}(G)$. Hence the width of (T, χ) is at most $\mathrm{ecw}(G)$. □

Crucially, as we show in the next section, a spanning tree resulting in a decomposition of optimal width can be efficiently computed. This makes all our algorithms obtained in [BCG+22] directly implementable in practice. We will not describe the algorithms here, but note that they all share the same framework, which is a dynamic programming along the spanning tree. The key idea here is to traverse nodes of the spanning tree in a leaf-to-root fashion, at each node storing the information about partial solutions of the instance restricted to corresponding subtree. The main observation is that small edge-cut width guarantees that the number of edges "leaving" every such subtree (which we call *boundary*) is bounded. Moreover, for every node, most of its children have only one such edge. We will formally define the notion of boundary graph in Section 3.3 in context of the slim tree-cut width, where we also apply this framework to design FPT algorithm for Maximum Stable Matching.

3.1.2 FPT Algorithm for Computing Edge-Cut Width

In this section we consider the question of computing edge-cut width along with an optimal "decomposition" (i.e., spanning tree). We provide an explicit fixed-parameter algorithm for this task.

By Lemma 3.2, the treewidth of G can be linearly bounded by $\mathrm{ecw}(G)$. The algorithm uses this to perform dynamic programming on a tree decomposition (T, χ) of G. For a

node $t \in V(T)$, we denote by G_t the vertex-induced subgraph of G induced by $\chi_t^{\downarrow}$, that is, $G_t = G[\chi_t^{\downarrow}]$.

Lemma 3.3. *Given an n-vertex graph G of treewidth k and a bound w, it is possible to decide whether G has edge-cut width at most w in time $k^{\mathcal{O}(wk^2)} \cdot n$. If the answer is positive, we can also output a spanning tree of G of edge-cut width at most w.*

Using the relation between treewidth and edge-cut width above, we immediately obtain:

Theorem 3.4. *Given a graph G, the edge-cut width* $\mathrm{ecw}(G)$ *can be computed in time* $\mathrm{ecw}(G)^{\mathrm{ecw}(G)^3} \cdot n$.

Proof of Lemma 3.3. Without loss of generality, we assume that G is connected. Using one of the approximation algorithms [BDD+16, Kor21], we first compute a nice tree decomposition (T, χ) with root $r \in V(T)$ of width $k = \mathcal{O}(\mathrm{tw}(G))$ in time $2^{\mathcal{O}(k)} \cdot n$.

On a high level, the algorithm relies on the fact that if G has edge-cut width at most w, then at each bag $\chi(t)$ the number of unique paths contributing to the edge-cut width of vertices in $\chi(t)$ is upper-bounded by $|\chi(t)|w \leq kw$. Otherwise, at least one of the vertices in $\chi(t)$ would lie on more than w cycles. We can use this to branch on how these at most kw edges are routed through the bag.

At each vertex $t \in T$ of the tree decomposition, we store *records* that consist of:

- an acyclic subset F of edges of $G[\chi(t)]$,
- a partition $\mathcal{C}$ of $\chi(t)$, and
- two multisets `future`, `past` of sequences of vertex-pairs (u, v) from $\chi(t)$, with the following property:
  - Every vertex of $\chi(t)$ appears on at most w distinct u-v paths, where (u, v) is a pair of vertices in a sequence in `future` or `past`.
  - v_i and u_{i+1} are not connected by an edge in $\chi(t)$.

The semantics of these records are as follows: For every spanning tree of width at most w, the record describes the intersection of the solution with $G[\chi(t)]$, and the intersection of every fundamental cycle of this solution with $G[\chi(t)]$. We encode the path that a cycle takes through $G[\chi(t)]$ via a sequence of vertex pairs that indicate where the path leaves and enters $G[\chi(t)]$ from the outside (it may be that these are the same vertex). More precisely, `past` contains those cycles that correspond to an edge that has already appeared in G_t, whereas `future` corresponds to those cycles that correspond to an edge not in G_t. In particular, this allows to reconstruct on how many cycles a vertex of $\chi(t)$ lies. The partition $\mathcal{C}$ says which vertices of $\chi(t)$ are connected via the solution in G_t.

To be more precise, let $t \in T$ and let S be an acyclic subset of edges of G that has width at most w on G_t (that is, each vertex of S lies on at most w fundamental cycles of S in G_t). We call such S *partial solutions* at t. Then, we let the *t-projection* of S be defined as $(F, \mathcal{C}, \texttt{future}, \texttt{past})$, where

- $F = S \cap G[\chi(t)]$.
- $\mathcal{C}$ is a partition of F according to the connected components of S in G_t.
- Let C_e be a fundamental cycle of S in G corresponding to the edge $e \in G - S$. Then, there is a sequence $P_e = ((u_1, v_1), \ldots, (u_t, v_t))$ in either $\texttt{future}$ or $\texttt{past}$ of vertex pairs such that the intersection of C_e with S traverses F along the unique u_i-v_i paths in the order they appear in P_e (note that $u_i = v_i$ is possible, in which case the path contains just the vertex u_i).
- For each fundamental cycle C_e of S in G, if $e \in G_t$, then $P_e \in \texttt{past}$ holds, otherwise, $P_e \in \texttt{future}$ holds.

Note that P_e can (and often will) be the empty sequence $P_e = \emptyset$. Moreover, we assume that the correspondence between $\texttt{future} \cup \texttt{past}$ and the edges in $G - S$ is bijective, in the sense that if two edges e, e' produce the same sequence $P_e = P_{e'}$, then P_e and $P_{e'}$ occur as two separate copies in $\texttt{future} \cup \texttt{past}$.

The encoding length of a single record is $\mathcal{O}(wk^2 \log k)$, dominated by the at most kw sequences P_e of k pairs of vertices each, with indices having $\mathcal{O}(\log k)$ bits. Overall, the number of records is hence bounded by $2^{\mathcal{O}(wk^2 \log k)}$.

For each $t \in T$, we store a set of records $\mathcal{R}(t)$ that has the property that $\mathcal{R}(t)$ contains the set of all t-projections of spanning trees of width at most w (that is, projections of solutions of the original instance). In addition, we require for every record in $\mathcal{R}(t)$ that there is a partial solution S of G_t of width at most w that agrees with $F, \mathcal{C}$ and $\texttt{past}$ of the record. In this case, we call $\mathcal{R}(t)$ *valid*. Supposing correctness of this procedure, G is a YES-instance if and only if $(F_r, \mathcal{C}_r, \texttt{past}_r, \texttt{future}_r) \in \mathcal{R}(r)$, with $F_r = \mathcal{C}_r = \texttt{future}_r = \emptyset$, $\texttt{past}_r = \{\emptyset^{m-n}\}$, and a NO-instance otherwise.

We compute $\mathcal{R}(t)$ bottom-up along the nice tree decomposition depending on the type of the node t as follows:

At a leaf-node, per convention, $\chi(t) = \emptyset$, and since G_t is the empty graph, any spanning tree S has width at most w on G_t. This implies that any t-projection $(F, \mathcal{C}, \texttt{past}, \texttt{future})$ of such S satisfies $F = \mathcal{C} = \texttt{past} = \emptyset, \texttt{future} = \{\emptyset^{n-m}\}$. It therefore suffices to set $\mathcal{R}(t) = \{(\emptyset, \emptyset, \emptyset, \{\emptyset^{n-m}\})\}$, and this is valid.

At an introduce-node, let the vertex introduced at t be $v \in G$, and let s be the unique child of t in T. By definition, $\chi(t) = \chi(s) \cup \{v\}$. We assume by inductive hypothesis that $\mathcal{R}(s)$ is valid. Consider now any solution S of width at most w on G_t. This solution will

be of width at most w also on G_s. Hence, since $\mathcal{R}(s)$ is assumed valid, there is a record $(F_s, \mathcal{C}_s, \texttt{past}_s, \texttt{future}_s)$ corresponding to the s-projection of S.

We first branch over the way that the edges incident with v in $G[\chi(t)]$ extend F_s. Call this new set of edges E_v. During this process, we discard any choice of E_v that connects vertices within the same connected component as indicated by $\mathcal{C}_s$.

Furthermore, we discard any choice that implies cycles in the solution via `future`: If there is an entry in $\texttt{future}_s$ that contains two consecutive pairs $(u, u'), (w, w')$ such that u' and w are now in the same component of $\mathcal{C}$ (that is, were connected by adding v to G_s), and one of u' or w is not a neighbor of v, then this would imply two u'-w paths: u' and w, but not any of the vertices on the paths u'-v and v-w lie on the fundamental cycle corresponding to the entry in $\texttt{future}_s$ containing $(u, u'), (w, w')$, yielding two paths: One through the cycle, the other through v via E_v. Therefore, this choice of E_v can be discarded.

Then, for every edge (v, u) incident with v that was not chosen into E_v, there must be a sequence of pairs P in $\texttt{future}_s$ such that the last vertex in the last pair of the sequence P is u, otherwise we may discard E_v (since the corresponding fundamental cycle wasn't reflected in $\texttt{future}_s$.) We branch over all ways of choosing $P_1, \ldots, P_d$ for each edge $e_1, \ldots, e_d$ incident to v that is not in E_v. For each $i = 1, \ldots, d$, if $P = P_i$ just consists of the single pair (u, u), we add the single pair (v, v) to P, and move P to `past` (since the feedback edge (v, u) is now part of G_t). Otherwise, if the first pair (w, w') in P is distinct from (u, u), we add the pair (v, w') to P, remove (w, w') from P, and add P to `past`.

We now update `past` and `future` as follows: If there is a consecutive pair $(u, u'), (w, w')$ in an element of $\texttt{past}_s$ or $\texttt{future}_s$ such that u' and w are neighbors of v, replace the subsequence $(u, u'), (w, w')$ by (u, w'): any other choice of connecting u' and w through a path than directly via v would imply a cycle. In any case, add the resulting sequence to `past` or `future`, respectively.

We then branch over the choices of extending fundamental cycles along v: For each pair in a sequence in `past` or `future` that contains a neighbor u of v connected via E_v, branch over whether or not to route this fundamental cycle via v by replacing (u, w) by (v, w) or (w, u) by (w, v), respectively.

If during any of the choices for $E_v, P_1, \ldots, P_d$ and the extensions of the fundamental cycles via v, the solution would have to route more than w cycles over any vertex of $\chi(t)$ (as can be checked by tracing out all the pairs in the sequences now contained in `future` and `past`), discard the choice. If there is no way to choose the above without exceeding the width bound, discard the entire choice of record and consider the next record in $\mathcal{R}(s)$.

If this is not the case, then, for a choice of E_v (i.e., how to extend F_s), $P_1, \ldots, P_d$ (i.e., how to route the new edges in G_t in `past`) and a choice of extending the existing cycles in $\texttt{past}_s$ and $\texttt{future}_s$ to in—or exclude v, we branch over how many additional fundamental cycles v outside of G_t will be part of, and add as many copies of the sequence

consisting just of (v, v) to $\texttt{future}$, simultaneously decreasing the multiplicity of $\emptyset$ in $\texttt{future}_s$ by as many, and adding the result to $\texttt{future}$.

Finally, add $(F_s \cup E_v, \mathcal{C}, \texttt{past}, \texttt{future})$ to $\mathcal{R}(t)$, and consider the next entry of $\mathcal{R}(s)$. Since any partial solution of width at most w on G_t will have to extend its s-projection in one of the above ways, this generates all possible t-projections (and possibly some additional records with the same $F, \mathcal{C}, \texttt{past}$). In particular, the generated set $\mathcal{R}(t)$ is valid. This completes the description of the introduce step.

The running time of this step is dominated by branching over the sequences $P_1, \ldots, P_d$. Since $d \leq k$ and there are at most kw sequences in total, we have $(kw)^k = 2^{\mathcal{O}(wk \log k)}$ choices at most, for each of the $2^{\mathcal{O}(wk^2 \log k)}$ records in $\mathcal{R}(s)$, and processing each choice only adds a lower-order term in the running time. Therefore, this step takes time $2^{\mathcal{O}(wk^2 \log k)}$.

At a forget-node, let the vertex forgotten at t be $v \in G$, and let s be the unique child of t in T. By definition, $\chi(t) = \chi(s) \setminus \{v\}$. We assume by inductive hypothesis that $\mathcal{R}(s)$ is valid, and let $(F_s, \mathcal{C}_s, \texttt{past}_s, \texttt{future}_s) \in \mathcal{R}(s)$.

If $\{v\} \in \mathcal{C}$ (that is, v is a single component in the intersection of any solution that projects to the current record with $\chi(t)$), then discard the choice for the record and consider the next element of $\mathcal{R}(s)$. In this case, the component that contains v in any partial solution conforming with the record could never be completed to form a connected subgraph.

If (v, v) appears as part of a sequence in $\texttt{future}_s$ or $\texttt{past}_s$, remove (v, v) from the sequence. Furthermore, if (v, u) is part of any sequence in $\texttt{past}_s$ or $\texttt{future}_s$ for some $u \neq v$, replace (v, u) by (v', u), where v' is the next vertex on the unique v-u path in F_s (and $u = v'$ is possible). In both cases, add the resulting sequence (which is possibly equal to the empty sequence) to $\texttt{future}$ or $\texttt{past}$, respectively. If the empty sequence would be added to $\texttt{future}$, discard the current record (since there is no way of closing this fundamental cycle in the future that can involve v).

We remove all edges involving v from F_s to obtain F and update $\mathcal{C}_s$ by removing v from all sets it appears in, thereby obtaining $\mathcal{C}$. We add $(F, \mathcal{C}, \texttt{past}, \texttt{future})$ to $\mathcal{R}(t)$. Since $G_t = G_s$, the set of solutions that contribute to the set of t-projections and s-projections doesn't change; we hence only have to update the s-projections to become t-projections, as we did, in order to obtain a valid set $\mathcal{R}(t)$.

The running time of this step is dominated by the running time at the introduce-nodes.

At a join-node, let s and s' be the two children of t in T. We consider all pairs of records in $\mathcal{R}(s)$ and $\mathcal{R}(s')$. If $F_s \neq F_{s'}$ or $\texttt{future}_s \neq \texttt{future}_{s'}$, we discard the current choice. Consider the transitive closures of the reachability relations on $\chi(t)$ as induced by $\mathcal{C}_s$ and $\mathcal{C}_{s'}$, respectively. If their union (as multigraphs) produces a cycle (which could be two parallel edges (u, v) and (u, v) for some $u, v \in \chi(t) = \chi(s) = \chi(s')$), any solution that s-projected and s'-projected to $\mathcal{C}_s$ and $\mathcal{C}_{s'}$, respectively, would be cyclic on G_t. Hence, we may discard this choice of records.

If none of the above happens, we set $\texttt{past} = \texttt{past}_s \cup \texttt{past}_{s'}$ as multisets, and check if this results in any of the vertices of $\chi(t)$ coming to lie on more than w fundamental cycles.

If this is the case, we discard the current choice of records. If not, let $\mathcal{C}$ be finest common coarsening of the partitions $\mathcal{C}_s$ and $\mathcal{C}_{s'}$ (that is, the result of merging any two components that share a vertex, and exhausting this process). We let $F = F_s(= F_{s'})$, $\texttt{future} = \texttt{future}_s(= \texttt{future}_{S'})$, and set $\mathcal{R}(t) = (F, \mathcal{C}, \texttt{past}, \texttt{future})$.

By a similar token as in the previous cases, this produces all possible t-projections of solutions of G_s and $G_{s'}$ that are also solutions for G_t of width at most w, and hence a valid set $\mathcal{R}(t)$.

Since we have to consider pairs of records that differ in `past`, and `past` dominates the size of the records, the running time at the join-nodes dominates the running time at the introduce-nodes, and is bounded by $2^{\mathcal{O}(wk^2 \log k)}$.

Overall, the running time of the algorithm is bounded by $2^{\mathcal{O}(wk^2 \log k)} \cdot n$. By keeping one representative of a partial solution of G_t per record at each node t that t-projects to the current record, we can successively build a solution of width at most w. □

3.1.3 Structural Properties

Decomposition for the edge-cut width is simply a spanning tree. This makes it easier to work with in dynamic programming applications than, e.g., tree-cut decompositions. However, as we will see in this section, edge-cut width has some structural drawbacks. In particular, it is not closed under vertex or edge deletions and is incomparable to degtw.

Lemma 3.5. *For each $m \in \mathbf{N}$, there exists a graph G_m of degree at most 3, tree-cut width at most 2, and edge-cut width at least $m + 1$.*

Proof. We start from two regular binary trees Y and Y' of depth m, i.e., rooted binary trees where every node except leaves has precisely two children and the path from any leaf to the root contains m edges. We glue Y and Y' together by identifying each leaf of Y with a unique leaf of Y', preserving distances between any pair of leaves (see the left part of Figure 3.2 for an illustration). It remains to show that the resulting graph, which we denote G_m, has the desired properties.

Consider an arbitrary spanning tree T of G_m. There exists a unique path $\pi \subseteq T$ between the roots r and r' of Y and Y'. Observe that $G_m \setminus \pi$ is a disjoint union of m graphs G_l, $l \in [m-1]$. We add to every such G_l two edges which connect it with π and denote the resulting graph by B_l. Then every B_l contains at least one edge that contributes to the local feedback edge set of $q \in V(\pi)$, where q is a leaf in Y and Y'. Indeed, fix any $l \in [m-1]$ and denote by a_l and a'_l the vertices of B_l intersecting π in Y and Y' respectively. As T is a tree, $T \setminus q$ is a union of two trees: one containing a_l and another containing a'_l. Hence every vertex ob B_l is connected to precisely one of a_l and a'_l in $T - q$. In particular, there exists an edge e_l of B_l such that one endpoint of e_l is connected to a_l and another is connected to a'_l in $T \setminus q$. Then e_l belongs to the local feedback edge set of every vertex of π that lies between a_l and a'_l, in particular, to the local feedback edge

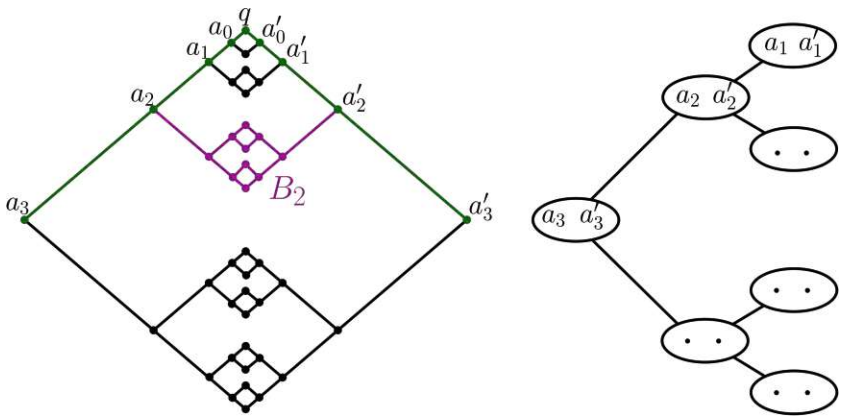

Figure 3.2: **Left**: Graph G_4, where the roots of Y and Y' are a_3 and a'_3, the path π is green and B_2 is violet. **Right**: Fragment of the tree-cut decomposition (Y^*, χ) of G_4.

set of q. As B_l and $B_{l'}$ don't share edges for any $l \neq l'$, this results in $|E^{G_m,T}_{\texttt{loc}}(q)| \geq m$. Since the inequality holds for any choice of T, we may conclude that $\mathrm{ecw}(G_m) \geq m + 1$.

To compute the tree-cut width of G_m, consider its tree-cut decomposition (Y^*, χ) where Y^* is a regular binary tree of depth m and χ is defined as follows. Let $h : V(Y^*) \to V(Y)$ and $h' : V(Y^*) \to V(Y')$ be bijections such that (1) if y is a leaf of Y^*, then $h(y)$ and $h'(y)$ are identified leaves of Y and Y', and (2) if y_1 is a parent of y_2 in Y^*, then $h(y_1)$ is a parent of $h(y_2)$ in Y and $h'(y_1)$ is a parent of $h'(y_2)$ in Y'. Further, for every node y of Y^* we define its bag to be $X_y = \{h(y), h'(y)\}$ (see the right part of Figure 3.2 for the illustration). Observe that the adhesion of every node as well as size of each bag is at most 2, and all the children are thin, therefore, $\mathrm{tcw}(G_m) = 2$. □

Since it is known that treewidth dominates tree-cut width (see Lemma 2.1), Lemma 3.5 implies that edge-cut width does not dominate degtw. Conversely, it is easy to build graphs with unbounded degtw and bounded edge-cut width (e.g., consider the class of stars). Hence, we obtain that edge-cut width is incomparable to degtw. An illustration of the parameter hierarchy including edge-cut width is provided in Figure 3.3.

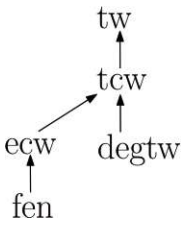

Figure 3.3: Position of edge-cut width in the hierarchy of graph parameters. Here an arrow from parameter β to parameter α represents the fact that α dominates β, i.e., there exists a function p such that for each graph G, $\alpha(G) \leq p(\beta(G))$. We use fen to denote the feedback edge number.

Last but not least, we show that—also somewhat surprisingly—edge-cut width is not closed under edge or vertex deletion.

For the edge-deletion case, we refer readers to Figure 3.4 which illustrates a graph G along with a spanning tree witnessing that $\mathrm{ecw}(G) \leq 4$. Conversely, any spanning tree T of $G \setminus ac$ must contain both edges ab_i and b_ic for some $i \in \{1, 2, 3\}$. We will assume that those edges are ab_1 and b_1c, since the other cases are symmetrical. Then T contains precisely one edge of each pair (ab_2, b_2c) and (ab_3, b_3c). The other, "missing" edge from each pair contributes to the local feedback edge set of b_1. Together with two missing edges of 3-cycles that intersect b_1, this results in $|E_{\mathtt{loc}}^{G\setminus ac,T}(b_1)| \geq 4$ and, since a similar situation happens for any choice of a spanning tree, we conclude that $\mathrm{ecw}(G \setminus ac) \geq 5$. The vertex deletion case can be argued analogously using the graph obtained from G by subdividing the edge ac.

Corollary 3.6. *There exist graphs G and H such that* $\mathrm{ecw}(G \setminus e) > \mathrm{ecw}(G)$ *and* $\mathrm{ecw}(H \setminus v) > \mathrm{ecw}(H)$ *for some $e \in E(G)$ and $v \in V(H)$.*

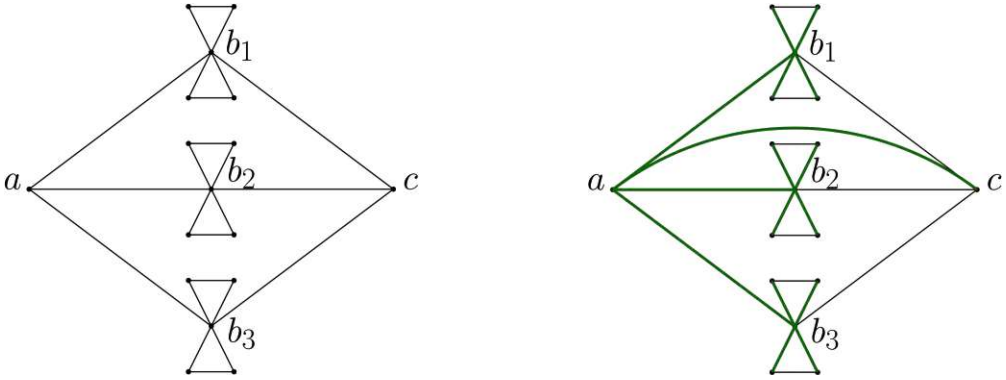

Figure 3.4: **Left**: Graph $G \setminus ac$ of $\mathrm{ecw}(G \setminus ac) \geq 5$. **Right**: Green tree witnessing that $\mathrm{ecw}(G) \leq 4$.

3.2 Slim Tree-Cut Width

While the structural drawbacks we have seen in the last section seem highly unfavorable, here we will show that they can be easily overcome by a slight modification of the definition of the edge-cut width. Basically, it suffices to consider the spanning tree of any supergraph of a given graph, resulting in a strictly more general parameter *super edge-cut width*. This parameter combines the advantages of tree-cut width and edge-cut width while avoiding all the shortcomings listed above. It admits at least three equivalent characterizations.

Slim tree-cut width, which we distinguish as a primal characterization of super edge-cut width, is defined similarly to tree-cut width, by altering the threshold for when a torso vertex is suppressed (dissolved) in the tree-cut decomposition from 3 to 2. If we go further and make this threshold equal to 1, this leads to an even more restrictive parameter 0-*tree-cut width*. However, as we will see, it is asymptotically equivalent to the degree treewidth, serving as a strong evidence that the slim tree-cut width is a perfect choice of an intermediate edge-cut based measure between degree treewidth and tree-cut width.

3.2.1 Definitions and Comparison

Let $(T, \mathcal{X})$ be some tree-cut decomposition of G. Given a connected graph Q and $X \subseteq V(Q)$, let the *2-center* of (Q, X) be the unique graph obtained from Q by exhaustively deleting vertices in $V(Q) \setminus X$ of degree at most one. For a node t of T, we denote by $\bar{H}_t^2$ the 2-center of (H_t, X_t), where H_t is the torso of $(T, \mathcal{X})$ at t. We denote $|\bar{H}_t^2|$ by $\mathrm{tor}_2(t)$.

Definition 3.2. *The slim width of a tree-cut decomposition $(T, \mathcal{X})$ of a graph G is* $\mathrm{stcw}(T, \mathcal{X}) = \max_{t \in V(T)}\{\mathrm{adh}(t), \mathrm{tor}_2(t)\}$. *The slim tree-cut width of G, or* $\mathrm{stcw}(G)$ *in short, is the minimum slim width of $(T, \mathcal{X})$ over all tree-cut decompositions $(T, \mathcal{X})$ of G.*

Naturally extending the notions of 2- and 3-center for a connected graph Q and $X \subseteq V(Q)$, we define the *1-center* of (Q, X) as the graph obtained from Q by deleting isolated vertices in $V(Q) \setminus X$. For a node t of T, we denote by $\bar{H}_t^1$ the 1-center of (H_t, X_t), where H_t is the torso of $(T, \mathcal{X})$ at t. Let us denote $|\bar{H}_t^1|$ by $\mathrm{tor}_1(t)$.

Definition 3.3. *The* 0*-width of a tree-cut decomposition $(T, \mathcal{X})$ of G is defined as* $\mathrm{tcw}_0(T, \mathcal{X}) = \max_{t \in V(T)}\{\mathrm{adh}(t), \mathrm{tor}_1(t)\}$. *The* 0*-tree-cut width of G, or* $\mathrm{tcw}_0(G)$ *in short, is the minimum* $\mathrm{tcw}_0(T, \mathcal{X})$ *over all tree-cut decompositions $(T, \mathcal{X})$ of G.*

It follows from the definitions that for any tree-cut decomposition $(T, \mathcal{X})$ of G, for each node t of T, $\mathrm{tor}(t) \leq \mathrm{tor}_2(t) \leq \mathrm{tor}_1(t)$. In particular, the width of $(T, \mathcal{X})$ is upper-bounded by its slim width, while the latter does not exceed the 0-width of $(T, \mathcal{X})$.

Corollary 3.7. *For any graph G,* $\mathrm{tcw}(G) \leq \mathrm{stcw}(G) \leq \mathrm{tcw}_0(G)$.

The gaps in these inequalites can be arbitrarily large—and, more strongly, tcw_0 is a more restrictive parameter than stcw, which is in turn more restrictive than tcw. Indeed, for the comparison of tcw_0 and stcw consider the class of stars which have slim tree-cut width 1. Let S_r denote the star with r leaves (i.e., the complete bipartite graph $K_{1,r}$).

Lemma 3.8. *For every positive integer $r \geq 1$,* $\mathrm{tcw}_0(S_{r^2}) \geq r$.

Proof. Let $(T, \mathcal{X})$ be a tree-cut decomposition of S_{r^2} of 0-width k where the bags of leaves are non-empty. Let t be the node of T such that X_t contains the vertex of degree r^2. Observe that t has at most $\mathrm{tor}_1(t) - |X_t| \leq k - |X_t|$ children. For every child t' of t, $Y_{t'}$ contains at most $\mathrm{adh}(t') \leq k$ vertices of S_{r^2}. In total, Y_t contains at most $|X_t| + k \cdot (k - |X_t|) \leq k^2$ vertices of S_{r^2}. Together with at most $\mathrm{adh}(t) \leq k$ vertices outside of Y_t, S_{r^2} has at most $k \cdot (k+1)$ vertices and hence $k \geq r$. □

To show the gap between stcw and tcw, let us denote by W_r the graph on $2r+1$ vertices consisting of r triangles sharing one vertex; here we call such graphs windmills, and refer to Figure 3.5 later for an illustration. The class of windmills has tree-cut width 2 but, as the following lemma shows, unbounded slim tree-cut width.

Lemma 3.9. *For every positive integer $r \geq 1$, $\mathrm{stcw}(W_{r^2}) \geq r$.*

Proof. The case $r = 1$ is straightforward. For $r \geq 2$, assume, to the contrary, that there exists a tree-cut decomposition $(T, \mathcal{X})$ of W_{r^2} of slim width at most $r-1$. Let t be the node of T such that X_t contains the vertex of degree $2r^2$. Without loss of generality, we assume that all the leaves of T have non-empty bags. Then the adhesion of any child t' of t is at least two, as $Y_{t'}$ contains some vertex v of W_{r^2} and the two edge-disjoint paths from v to the high-degree vertex in t each contribute to $\mathrm{adh}(t')$. Hence, t has at most $\mathrm{tor}_2(t) \leq r-1$ children. Moreover, for every child t' of t, $Y_{t'}$ intersects at most $\frac{r-1}{2}$ distinct triangles of W_{r^2}, since each such triangle contributes 2 to $\mathrm{adh}(t')$. Hence, for every child t' of t, $Y_{t'}$ contains at most $r-1$ vertices of W_{r^2}. In total, $Y_t \setminus X_t$ contains at most $(r-1)^2$ vertices of W_{r^2}. Since both $\mathrm{adh}(t)$ and $|X_t|$ are upper-bounded by $r-1$ and the former bounds the number of vertices outside of Y_t by $r-1$, this would mean that W_{r^2} has at most $(r-1)^2 + 2r - 2$ vertices, a contradiction with the definition of W_{r^2}. □

Given a graph G and its nice tree-cut decomposition $(T, \mathcal{X})$ of width at most k, let us denote by $B_t^{(2)}$ the set of children of t from B_t with adhesion precisely two; notice that $B_t^{(2)}$ does not necessarily contain all children of t with adhesion precisely two, since some may lie in A_t. Observe that for every fixed vertex t of T, if x is an element of 2-center of the torso at t and $x \notin X_t$, then x corresponds either to the parent of t in T or to some child of t from $A_t \cup B_t^{(2)}$. Hence $\mathrm{tor}_2(t) \leq 1 + |X_t| + |A_t| + |B_t^{(2)}| \leq 3k + 2 + |B_t^{(2)}|$.

Corollary 3.10. *Let G be a graph with tree-cut decomposition $(T, \mathcal{X})$ of width at most k. Then for each node t of T it holds that $|B_t^{(2)}| \geq \mathrm{tor}_2(t) - 3k - 2$.*

3.2.2 Characterization in Terms of Forbidden Weak Immersions

Naturally extending the result of Wollan for tree-cut width [Wol15], we show that both slim and 0-tree-cut width are closed under weak immersions.

Theorem 3.11. *If G and H are graphs such that $H \leq_I G$, then $\mathrm{stcw}(H) \leq \mathrm{stcw}(G)$ and $\mathrm{tcw}_0(H) \leq \mathrm{tcw}_0(G)$.*

Proof. It is sufficient to proof the statement when H is obtained from G by precisely one edge deletion, isolated vertex deletion or lifting a pair of edges. Let $(T, \mathcal{X})$ be a tree-cut decomposition of G of minimum slim (or 0-) width. Then $(T, \mathcal{X})$ is also a tree-cut decomposition of $G \setminus e$ for any edge e of G with the same or smaller slim (0-) width. Similarly for the isolated vertex deletion: we just need to delete the vertex from the corresponding bag. It remains to consider the case $H = G \setminus \{xy, yz\} \cup \{xz\}$ for some edges $xy, yz \in E(G)$.

Notice that the lifting operation doesn't increase adhesion of any node t of T: if the edge xz has endpoints in different connected components of $T \setminus e(t)$, then so does at least one of the edges xy or yz. To see that $\mathrm{tor}_2(t)$ and $\mathrm{tor}_1(t)$ do not increase either, denote

by Q_G and Q_H the torsos at t in $(T, \mathcal{X})$ for graphs G and H respectively. Every vertex of Q_G corresponds to a non-empty subset of the vertices of G. Depending on how the vertices x, y and z are split among these subsets, it holds that either $E(Q_H) \subseteq E(Q_G)$ (which yields the same or smaller 1-center and 2-center) or Q_H is obtained from Q_G by splitting a pair of edges. For the latter, observe that $v \in V(Q_G) \setminus X_t$ is not in the 2-center of (Q_G, X_t) if and only if v belongs to some induced subtree of Q_G connected to the rest of Q_G by at most one edge. It is not hard to see that lifting the pair of edges preserves the property. For the 1-center the situation is even simplier: isolated vertices of Q_G remain isolated. □

Recall that the weak immersion relation $\leq_I$ is a transitive, reflexive and antisymmetric relation on the set of finite graphs, i.e., a partial order. The previous theorem showed that stcw is monotone with respect to $\leq_I$. Our next goal is to find graphs of simple structure but large slim (or 0-) tree-cut width, such that forbidding them as weak immersions bounds the corresponding width of a graph. Wollan in [Wol15] characterized such graphs for tree-cut width. Namely, he established the following dichotomy:

Theorem 3.12. *(a) If G is a graph such that $H_{2r^2} \leq_I G$ for some $r \geq 3$, then $\mathrm{tcw}(G) \geq r$. (b) There exists a function $f : \mathbf{N} \to \mathbf{N}$ such that if $\mathrm{tcw}(G) \geq f(r)$, then $H_r \leq_I G$, $r \in \mathbf{N}$.*

Here H_r denotes the r-wall, the graph which can be obtained from the $r \times r$ grid by deleting every second vertical edge in each row, see [Wol15] for the definition and Figure 3.5 for an illustration. We are going to complete the family of excluded immersions to provide similar characterizations for 0-tree-cut width and slim tree-cut width. Recall that the families of stars S_r and windmils W_r have unbounded 0- and slim tree-cut width, respectively (Lemmas 3.8 and 3.9). Combining this with Theorem 3.11, we immediatedly obtain:

Lemma 3.13. *For every positive integer r, if $\mathrm{stcw}(G) < r$ ($\mathrm{tcw}_0(G) < r$), then G does not admit W_{r^2} (S_{r^2}, respectively) as a weak immersion.*

As we will show in the remainder of this section, excluding W_r (S_r) as a weak immersion along with H_r is actually sufficient to bound slim tree-cut width (0-tree-cut width).

Theorem 3.14. *If G is a graph such that $H_{2r^2} \leq_I G$ for some $r \geq 3$ or $S_{r^2} \leq_I G$ for some $r \geq 1$, then $\mathrm{tcw}_0(G) \geq r$. Moreover, there exists a function $h : \mathbf{N} \to \mathbf{N}$ such that if $\mathrm{tcw}_0(G) \geq h(r)$, then $H_r \leq_I G$ or $S_r \leq_I G$.*

Proof. If $H_{2r^2} \leq_I G$ for some $r \geq 3$, we have that $\mathrm{tcw}(G) \geq r$ by Theorem 3.12 and hence $\mathrm{tcw}_0(G) \geq r$. In case $S_{r^2} \leq_I G$, the lower bound follows from Lemma 3.13.

Let f be the function given by Theorem 3.12. We define h by $h(r) = r \cdot f(r) + 3 \cdot f(r) + 2$. Assume that G is a graph such that $\mathrm{tcw}_0(G) \geq h(r)$. If $\mathrm{tcw}(G) \geq f(r)$, we immediatedly conclude that $H_r \leq_I G$ by Theorem 3.12. Otherwise, let $(T, \mathcal{X})$ be a nice tree-cut

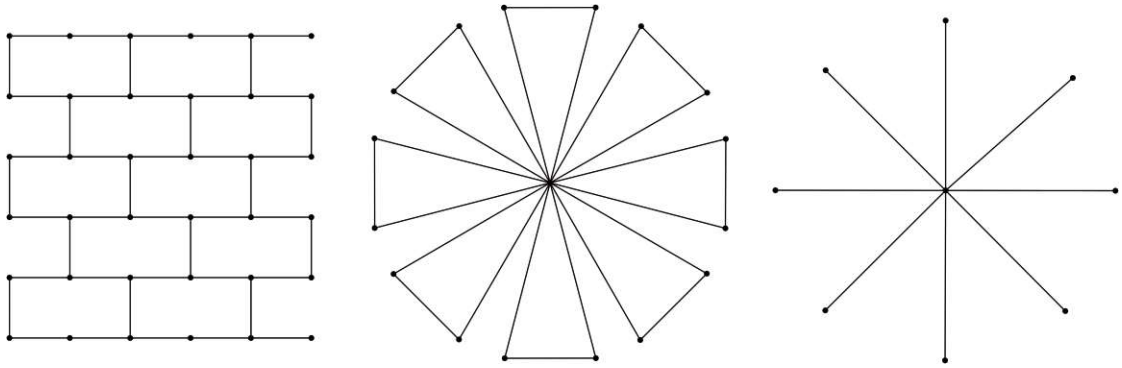

Figure 3.5: Illustrations of forbidden weak immersions for the graphs with bounded standard, slim or 0-tree-cut width. Left: 6-wall H_6, Middle: windmill W_8, Right: star S_8.

decomposition of G of width at most $f(r)$ with leaves having non-empty bags. There exists a node t of T such that $\mathrm{tor}_1(t) \geq h(r)$, in particular, $B_t \geq r \cdot f(r)$. As the size of X_t is at most $f(r)$, some vertex of X_t has degree of at least r and hence $S_r \leq_I G$. □

Before providing similar characterization for slim tree-cut width, we introduce a simple technical modification of tree-cut decompositions, which will also be used later for establishing the connection between slim tree-cut width and edge-cut width. The aim is, roughly speaking, to avoid the situation where a thin child has adhesion 2, even though it consists of two completely independent components each of which could be a thin child of adhesion 1. Formally, let $(T, \mathcal{X})$ be a nice tree-cut decomposition of G. We say that a node t with parent t' in T is *decomposable* if the following conditions hold:

- $t \in B_{t'}$ and there exist two edges e_1 and e_2 between G_t and $G \setminus G_t$ in G;
- the endpoints of e_1 and e_2 in G_t belong to different connected components of G_t.

Lemma 3.15. *Any nice tree-cut decomposition of G can be transformed into a nice tree-cut decomposition of the same tree-cut width with no decomposable nodes.*

Proof. Let $(T', \mathcal{X}')$ be a nice tree-cut decomposition of G with at least one decomposable node. Let t be a decomposable node of T' with minimum distance to the root, and let e_1 and e_2 be the edges between G_t and $G \setminus G_t$ in G. We create a copy T'_{t^*} of the rooted subtree T'_t where the copy of $s \in T'_t$ is $s^* \in T'_{t^*}$. We then connect t^* to the parent of t. Let G_1 be the connected component of G_t containing an endpoint of e_1. For every $s \in V(T'_t)$ we set $X_s = X'_s \cap V(G_1)$ and $X_{s^*} = X'_s \setminus X_s$. For the rest of nodes s of T' we set $X_s = X'_s$. Finally, we exhaustively remove empty bags which are leaves and denote the obtained tree by T. Observe that the resulting decomposition $(T, \mathcal{X})$ is nice and its width is not greater than the width of $(T', \mathcal{X}')$. Moreover, our transformation doesn't create any decomposable nodes outside of subtrees rooted in t and t^*; both t and t^* have an adhesion of one and hence are not decomposable. Therefore, after a finite number of

such steps we obtain some nice tree-cut decomposition of G of the same width but with no decomposable nodes. □

Further, as a technical term, we will refer to nice decompositions with no decomposable nodes as *very nice* decompositions.

Corollary 3.16. *Every tree-cut decomposition can be transformed into a very nice tree-cut decomposition in quartic time, without increasing the width.*

Proof. Let $(T'', \mathcal{X}'')$ be a tree-cut decomposition of G of width k. We transform $(T'', \mathcal{X}'')$ into a nice tree-cut decomposition $(T', \mathcal{X}')$ of width at most k (this can be done in cubic time, see [GKS22] for details). Further, we apply Lemma 3.15 on $(T', \mathcal{X}')$. This requires at most quartic time, since every node of T' is decomposed at most once and every such decomposition can be performed in cubic time. Then the resulting decomposition $(T, \mathcal{X})$ is very nice and has width of at most k. □

With this transformation in hand, we are now ready to fully characterize forbidden weak immersions for graphs of bounded slim tree-cut width.

Theorem 3.17. *If G is a graph such that $H_{2r^2} \leq_I G$ for some $r \geq 3$ or $W_{r^2} \leq_I G$ for some $r \geq 1$, then $\mathrm{stcw}(G) \geq r$. Moreover, there exists a function $g : \mathbf{N} \to \mathbf{N}$ such that if $\mathrm{stcw}(G) \geq g(r)$, then $H_r \leq_I G$ or $W_r \leq_I G$.*

Proof. If $H_{2r^2} \leq_I G$ for some $r \geq 3$, we have that $\mathrm{tcw}_2(G) \geq r$ by Theorem 3.12 and hence $\mathrm{stcw}(G) \geq r$. In case $W_{r^2} \leq_I G$, the lower bound follows from Lemma 3.13.

Let f be the function given by Theorem 3.12. We define g by $g(r) = 2r \cdot f^2(r) + 3 \cdot f(r) + 2$. Assume that G is a graph such that $\mathrm{stcw}(G) \geq g(r)$. If $\mathrm{tcw}(G) \geq f(r)$, we immediatedly conclude that $H_r \leq_I G$ by Theorem 3.12. Otherwise, by Corollary 3.16 there exists a very nice tree-cut decomposition $(T, \mathcal{X})$ of G of width at most $f(r)$. Let us pick a node t of T such that $\mathrm{tor}_2(t) \geq g(r)$. By Corollary 3.10 we have that the size of $B_t^{(2)}$ is at least $g(r) - 3 \cdot f(r) - 2 = 2r \cdot f^2(r)$. Since $(T, \mathcal{X})$ is very nice, all the children of t in $B_t^{(2)}$ are non-decomposable. Recall that for every $t' \in B_t^{(2)}$, the neighbourhood of $Y_{t'}$ in G is a one- or two-element subset of X_t, and hence $Y_{t'}$ provides a path between some (possibly equal) vertices of X_t. As the size of X_t is at most $f(r)$, G contains either $2r$ cycles intersecting in one vertex of X_t or $2r$ paths between two vertices of X_t. Since every such pair of paths can be transformed into a cycle by lifting the pair of their first edges, in both cases we have $W_r \leq_I G$. □

3.2.3 k-Edge Sums

Another natural property Wollan [Wol15] established for tree-cut width is that the parameter is closed under the operation of taking k-edge sum for small k:

Lemma 3.18 ([Wol15])**.** *Let G, G_1, and G_2 be graphs such that $G = G_1 \oplus_k G_2$. If G_j has a tree-cut decomposition $(T_j, \mathcal{X}_j)$ for $j = 1, 2$, then G has a tree-cut decomposition $(T, \mathcal{X})$ such that $\operatorname{adh}(T, \mathcal{X}) = \max\{k, \operatorname{adh}(T_1, \mathcal{X}_1), \operatorname{adh}(T_2, \mathcal{X}_2)\}$. Moreover, for every $t \in V(T)$, the torso H_t of t in $(T, \mathcal{X})$ is isomorphic to the torso of some vertex of $(T_1, \mathcal{X}_1)$ or $(T_2, \mathcal{X}_2)$.*

Based on this result for optimal decompositions $(T_1, \mathcal{X}_1)$ and $(T_2, \mathcal{X}_2)$, we immediatedly obtain the upper bound on 0- and slim tree-cut width for k-edge sums:

Corollary 3.19. *Let G, G_1 and G_2 be graphs such that $G = G_1 \oplus_k G_2$. Then it holds that $\operatorname{stcw}(G) \leq \max\{k, \operatorname{stcw}(G_1), \operatorname{stcw}(G_2)\}$ and $\operatorname{tcw}_0(G) \leq \max\{k, \operatorname{tcw}_0(G_1), \operatorname{tcw}_0(G_2)\}$.*

In particular, if both G_1 and G_2 have 0-, slim or standard width of at most ω and $k \leq \omega$, we may conclude that the corresponding width of G is at most ω.

3.2.4 Super Edge-Cut Width

In this section, we study one more alternative characterization of slim tree-cut width and 0-tree-cut width. In particular, we observe that the latter is asymptotically equivalent to maximum degree plus treewidth. This provides an interesting connection between tree decompositions and tree-cut decompositions, but essentially rules out its study as a means of establishing novel tractability results. For slim tree-cut width, however, we obtain a characterization in terms of super edge-cut width that ties it to the previously studied edge-cut width.

Characterization of 0-Tree-Cut Width. Wollan [Wol15] showed that a bound on the treewidth and maximum degree implies a bound on the tree-cut width of a graph:

Proposition 3.20. *Let G be a graph with maximal degree d and treewidth w. Then there exists a tree-cut decomposition of adhesion at most $(2w + 2)d$ such that every torso has at most $(d + 1)(w + 1)$ vertices.*

In particular, as $\operatorname{tor}_1(t) \leq |H_t| \leq (d + 1)(w + 1) \leq (2w + 2)d$ for every node t of T, we have $\operatorname{tcw}_0(G) \leq (2w + 2)d$. In the following proposition, we show that the converse is true as well: bounded tcw_0 implies bounded treewidth and maximum degree of a graph.

Proposition 3.21. *Let G be a graph with $\operatorname{tcw}_0(G) = k$. Then every vertex of G has degree of at most $k^2 + 2k$ and $\operatorname{tw}(G) \leq 2k^2 + 3k$.*

Proof. By Lemma 2.1 and Corollary 3.7 we have $\mathrm{tw}(G) \leq 2\,\mathrm{tcw}(G)^2+3\,\mathrm{tcw}(G) \leq 2k^2+3k$. Since $\mathrm{tcw}_0(G) = k$, Lemma 3.8 implies that G does not contain $S_{(k+1)^2}$ as a weak immersion, in particular, degree of any vertex of G is at most $k^2 + 2k$. □

Corollary 3.22. 0*-tree-cut width is asymptotically equivalent to maximum degree plus treewidth.*

This also implies an immediate connection to cutwidth—another edge-cut based parameter, defined in terms of linear orderings. A *linear ordering* of G is a bijective mapping $f : V \to \{1, \ldots, n\}$. The *cut* at vertex v with respect to f (denoted $\mathrm{cut}(v)$) is $|\{(u, w) \in E \mid f(u) \leq f(v) < f(w)\}|$. The cutwidth of a linear ordering f is defined as $\max_{v \in V(G)} \mathrm{cut}(v)$. The *cutwidth* of G is the minimum cutwidth of f over all linear orderings f of G.

Observe that the degree of every vertex v of G is at most two times the cutwidth of G: neighbours of v that go after v in the ordering contribute to $\mathrm{cut}(v)$, while the rest of the neighbours contribute to $\mathrm{cut}(w)$ where $f(w) = f(v) - 1$. It is known that both the pathwidth and the treewidth of a graph is upper-bounded by its cutwidth [KS93], and in particular this implies that the class of binary trees has unbounded cutwidth. Hence, cutwidth is a strictly more restrictive parameter than degtw (and hence tcw_0).

Characterization of Slim Tree-Cut Width. Recall that edge-cut width is a parameter that is defined over spanning trees in the input graph G, which serve as the corresponding decompositions. Let us now consider a slight generalization of this where we consider not only spanning trees over G, but of any supergraph of G. Such a generalization would—unlike edge-cut width itself—trivially be closed under both vertex and edge deletion. For our considerations, let us denote this parameter *super edge-cut width* ($\mathrm{sec}(G)$):

$$\mathrm{sec}(G) = \min\{\mathrm{ecw}(H, T) \mid H \supseteq G \text{ and } T \text{ is a maximal spanning forest of } H\}.$$

If $H \supseteq G$ is a supergraph of G and T is a maximal spanning forest of H such that $\mathrm{ecw}(H, T) \leq k$, we say that T *witnesses* $\mathrm{sec}(G) \leq k$. Observe that there always exists a connected witness, i.e., a tree. Indeed, if H consists of $m > 1$ connected components, we can arbitrarily extend it to a connected graph H^* by adding $m - 1$ edges. The addition of these edges to T then results in the tree T^* witnessing $\mathrm{sec}(G) \leq k$. Moreover, notice that any witness of $\mathrm{ecw}(G) \leq k$ is also a witness of $\mathrm{sec}(G) \leq k$.

Corollary 3.23. *For every graph* G, $\mathrm{sec}(G) \leq \mathrm{ecw}(G)$.

However, graphs of constant super edge-cut width can have arbitrarily large edge-cut width, as will become clear at the end of the section. A slight modification of the proof of Proposition 3.1 yields:

Proposition 3.24. *For every graph* G, $\mathrm{tcw}(G) \leq \mathrm{sec}(G)$.

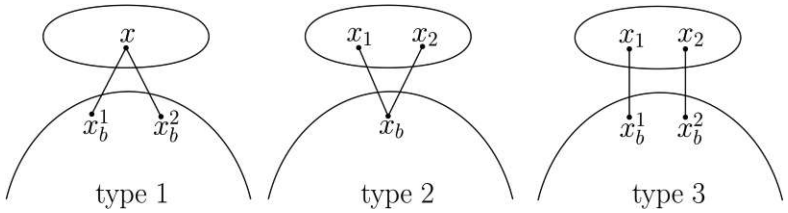

Figure 3.6: Possible configurations of edges between thin child $b \in B_t^{(2)}$ and its parent t.

Proof. Let Q be the supergraph of G and let T be a spanning tree of Q such that $\mathrm{ecw}(Q,T) = \mathrm{sec}(G)$. We construct a tree-cut decomposition $(T,\mathcal{X})$ of G where each bag contains at most one vertex, notably by setting $X_t = \{t\}$ for each $t \in V(G)$ and $X_t = \emptyset$ for each $t \in V(Q) \setminus V(G)$. Fix any node t in T other than the root, let u be the parent of t in T. All the edges of $G \setminus ut$ with one endpoint in the rooted subtree T_t and another outside of T_t belong to $E_{loc}^{Q,T}(t)$, so $\mathrm{adh}_T(t) \leq |E_{loc}^{Q,T}(t)| + 1 \leq \mathrm{sec}(G)$.

Let H_t be the torso of $(T,\mathcal{X})$ in t, then $V(H_t) = X_t \cup \{z_1 ... z_l\}$ where z_i correspond to connected components of $T \setminus t$, $i \in [l]$. In $\tilde{H}_t$, only z_i with degree at least 3 are preserved. But all such z_i are the endpoints of at least two edges in $|E_{loc}^{Q,T}(t)|$, so $\mathrm{tor}(t) = |V(\tilde{H}_t)| \leq 1 + |E_{loc}^{Q,T}(t)| \leq \mathrm{sec}(G)$. Thus $\mathrm{tcw}(G) \leq \mathrm{sec}(G)$. □

To represent a deeper connection between tree-cut decompositions and super edge-cut width, it will be convenient to work with very nice decompositions introduced in Section 3.2.2.

Proposition 3.25. *Let $(T,\mathcal{X})$ be a very nice tree-cut decomposition of G of width at most k. Then for each node t of T, $|B_t^{(2)}| \leq k \cdot \mathrm{sec}(G)$. In particular, it holds that* $\mathrm{stcw}(G) \leq \mathrm{sec}(G)^2 + 4 \cdot \mathrm{sec}(G)$.

Proof. Assume that T^* is a spanning tree of $H \supseteq G$ such that $\mathrm{sec}(G) = \mathrm{ecw}(H,T^*)$. For any node t of T and $b \in B_t^{(2)}$, b has one of three types (see Figure 3.6):

1. $N(Y_b) = \{x\}$ for some $x \in X_t$, x is connected to distinct x_b^1 and x_b^2 from Y_b;
2. $N(Y_b) = \{x_1, x_2\}$ for $x_1 \neq x_2$, x_1 and x_2 are connected to the same $x_b \in Y_b$;
3. $N(Y_b) = \{x_1, x_2\}$ for $x_1 \neq x_2$, x_1 and x_2 are connected to distinct x_b^1 and x_b^2 from Y_b respectively;

Let us start with the first type. If $x_b^i x$ doesn't belong to T^* for $i = 1$ or $i = 2$, then $x_b^i x \in E_{loc}^{H,T^*}(x)$. Otherwise, x_b^1 and x_b^2 are connected via x in T^*. Then $T^*[Y_b]$ has precisely two connected components. As b is not decomposable, there exists a path p between x_b^1 and x_b^2 in G_b containing precisely one edge outside of T^*. This edge contributes to $E_{loc}^{H,T^*}(x)$.

As T^* is a tree, there can be at most $|X_t| - 1 \leq k - 1$ thin children b of the second type such that x_b is adjacent to two elements of X_t in T^*. For the rest of b of the second type, there exists $x \in X_t$ such that $xx_b \in G \setminus T^* \subseteq H \setminus T^*$ and therefore $xx_b \in E_{loc}^{H,T^*}(x)$.

Let b be a thin node of the third type. If x_1^b and x_2^b are connected via a path in $T^*[Y_b]$, we can apply the same argument as for the second type. Otherwise, $T^*[Y_b]$ has precisely two connected components and, analogously to the first type, there exists an edge in $G_b \cup \{x_1x_1^b, x_2x_2^b\}$ that belongs to $E_{loc}^{H,T^*}(x_1)$.

To conclude, any node of $B_t^{(2)}$ either increases $E_{loc}^{H,T^*}(x)$ for some $x \in X_t$ or creates a path in T^* between two vertices of X_t. Since T^* is a tree, $|X_t| \leq k$ and $|E_{loc}^{H,T^*}(x)| \leq \sec(G) - 1$ for every $x \in X_t$, the size of $B_t^{(2)}$ is at most $(k-1) + \sum_{x \in X_t} |E_{loc}^{H,T^*}(x)| \leq k \cdot \sec(G) - 1$. Then $\mathrm{tor}_2(t) \leq |A_t| + |X_t| + 1 + |B_t^{(2)}| \leq 3k + 1 + k \cdot \sec(G) \leq k \cdot (\sec(G) + 4)$. Since the bound holds for every node t of T, we may conclude that the slim width of $(T, \mathcal{X})$ is at most $k \cdot (\sec(G) + 4)$. By Proposition 3.24 and Corollary 3.16, there exists a very nice tree-cut decomposition of G of width $k \leq \sec(G)$, therefore $\mathrm{stcw}(G) \leq \sec(G)^2 + 4 \cdot \sec(G)$. □

Hence, slim tree-cut with of any graph is upper-bounded by a quadratic function of its super edge-cut width. Next, we show that the converse statement holds as well:

Proposition 3.26. *For every graph G, $\sec(G) \leq 3 \cdot (\mathrm{stcw}(G) + 1)^2$. Moreover, given a tree-cut decomposition of G of slim width k, it is possible to compute a supergraph $Q \supseteq G$ and its spanning tree T witnessing $\sec(G) \leq 3(k+1)^2$ in cubic time.*

Proof. Let $(T_0, \mathcal{X}_0)$ be a tree-cut decomposition of G of slim width k. We start by transforming it into a nice tree-cut decomposition $(T, \mathcal{X})$ in cubic time as in [GKS22]. The transformation procedure acts on the 2-centers of torsos only by contracting some edges. Recall that $v \in V(H_t) \setminus X_t$ is not in the 2-center of (H_t, X_t) if and only if v belongs to some induced subtree of H_t connected to the rest of H_t by at most one edge. Since contracting an edge either preserves the property or merges v with some other vertex, it doesn't increase $\mathrm{tor}_2(t)$ for any node t of T. In particular, the slim width of $(T, \mathcal{X})$ is at most k.

Let $\Omega \subseteq \mathcal{X}$ be the set of empty bags of $(T, \mathcal{X})$, we construct $Q \supseteq G$ along with its tree-cut decomposition $(T, \mathcal{X}')$ as follows. Firstly, we add to G vertices v_t for every $t \in \Omega$. We define $X'_t = \{v_t\}$ if $X_t = \emptyset$ and $X'_t = X_t$ otherwise. For every node $t \in T$, construct an arbitrary tree T_t^* over X'_t and add its edges to Q. Further, we process every edge $e = pt \in E(T)$ such that p is the parent of t in T and either $N(Y_t) \not\subseteq X_t$ or $\mathrm{adh}(t) > 1$ as follows. If G doesn't contain an edge between X'_t and X'_p, we add to $E(Q)$ arbitrary edge with endpoints in X'_t and X'_p. This increases the adhesion of e by at most one.

Now we proceed to the choice of the spanning tree T^* in Q. For every $t \in T$ other than the root, let p be the parent of t in T. If $\mathrm{adh}(t) = 1$ and $N(Y_t) \subseteq X_t$, we denote by e_t the unique edge between Y'_t and X'_p in Q. Otherwise, let e_t be arbitrary edge of Q with endpoints in X'_t and X'_p. We then construct T^* by gluing together all T_t^* via

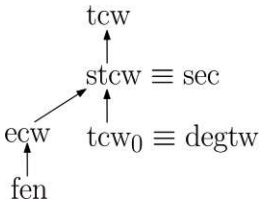

Figure 3.7: Position of slim and 0-tree-cut width in the hierarchy of edge-cut based parameters. An arrow from p to q represents the fact that p is more restrictive than q, while asymptotic equivalence is depicted by $\equiv$.

edges e_t, that is, $T^* = (\cup_{t\in V(T)} T_t^*) \bigcup (\cup_{t\in V(T)\setminus r}\{e_t\})$. Obviously the construction can be performed in cubic time; we will show that $\mathrm{sec}(Q, T^*) \leq 3(k+1)^2$.

To this end, fix any node t of T and $x \in X'_t$ and denote $E_{loc}(x) = E_{loc}^{Q,T^*}(x)$. If T^* contains more than one edge between Y'_t and rest of T^*, then all but one of them are the unique edges connecting Q'_q to the rest of Q for some descendants q of t in T. Hence, they don't belong to any path in T^* between the endpoints of some feedback edge $e \in E(Q) \setminus E(T^*)$. Therefore, every edge of $E_{loc}(x)$ has at least one endpoint in Y'_t. The number of edges in $E_{loc}(x)$ with both endpoints in X'_t is at most $|X'_t| \cdot (|X'_t| - 1) \leq k \cdot (k-1)$. Every edge with one endpoint in X'_t and another outside of Y'_t contributes to the adhesion of t in $(T, \mathcal{X}')$, so their number is bounded by $k+1$.

Finally, if $e = yz \in E_{loc}(x)$ contains an endpoint y in $Y'_t \setminus X'_t$, then $y \in Y'_q$ for some child q of t. Then Q contains a cycle intersecting Y'_q and $x \in X_t$. In particular, by construction of Q we may conclude $q \in A_t \cup B_t^{(2)}$ w.r.t. the decomposition $(T, \mathcal{X})$. By the same arguments as for the node t, we conclude that at most one edge between Y'_q and the rest of T^* belongs to any path in T^* between the endpoints of some feedback edge $e \in E(Q)\setminus E(T^*)$, so $z \notin Y'_q$ and e contributes to the adhesion of q in $(T, \mathcal{X}')$. In particular, $E_{loc}(x)$ contains at most $\mathrm{adh}(q) + 1$ edges with an endpoint in Y'_q. In total, at most $\max_{q\in A_t}(\mathrm{adh}(q)+1)\cdot|A_t| + \max_{q\in B_t^{(2)}}(\mathrm{adh}(q)+1)\cdot|B_t^{(2)}| \leq (k+1)(2k+1)+3k = 2k^2+6k+1$ edges in $E_{loc}(x)$ have an endpoint in $Y'_t\setminus X'_t$, so $|E_{loc}(x)| \leq k\cdot(k-1)+(k+1)+2k^2+6k+1 = 3k^2+6k+2$ and hence $\mathrm{sec}(Q, T^*) \leq 3k^2+6k+3 = 3(k+1)^2$. □

Corollary 3.27. sec *and* stcw *are asymptotically equivalent.*

The results of this section are summarized in Figure 3.7. In particular, the graph family provided in [BCG+22, Lemma 2] shows that graphs of constant super edge-cut width may have arbitrarily large edge-cut width.

3.2.5 Cubic Approximation

In this section we show how to efficiently construct a tree-cut decomposition of a graph G with slim width bounded by a cubic function of its optimal value $\mathrm{stcw}(G)$. As a starting point for our approximation, we use the following result of Kim, Oum, Paul, Sau and Thilikos:

Theorem 3.28 ([KOP+18])**.** *There exists an algorithm that, given a graph G and $\omega \in \mathbf{N}$, either outputs a tree-cut decomposition of G with width at most 2ω or correctly reports that no tree-cut decomposition of G with width at most ω exists in $2^{\mathcal{O}(\omega^2 \cdot log\omega)} \cdot n^2$ steps.*

As an observant reader might have already noticed, if G has bounded slim tree-cut width, it imposes some restrictions on the structure of possible decompositions of G of small (standard) tree-cut width. This fact enables us to construct an efficient approximation for $\mathrm{stcw}(G)$.

Theorem 3.29. *There exists an algorithm that, given a graph G and $\omega \in \mathbf{N}$, either outputs a tree-cut decomposition of G with slim width at most $6(\omega+1)^3$ or correctly reports that no tree-cut decomposition of G with slim width at most ω exists in $2^{\mathcal{O}(\omega^2 \cdot log\omega)} \cdot n^4$ steps.*

Proof. Given a graph G and $\omega \in \mathbf{N}$, let us run the algorithm from Theorem 3.28. If it reports that $\mathrm{tcw}(G) > \omega$, we may conclude that $\mathrm{stcw}(G) > \omega$ by Corollary 3.7. In case the algorithm returns a tree-cut decomposition $(T', \mathcal{X}')$ of width at most 2ω, we invoke Corollary 3.16 to transform this decomposition into a very nice decomposition $(T, \mathcal{X})$ of the same width in at most quartic time. By Proposition 3.25, we have that $|B_t^{(2)}| \leq 2\omega \cdot \mathrm{sec}(G)$ for each node t of T. If for some node t the size of $B_t^{(2)}$ exceeds $6\omega \cdot (\omega+1)^2$, then $\mathrm{sec}(G) > 3(\omega+1)^2$ and by Proposition 3.26 we may correctly report that $\mathrm{stcw}(G) > \omega$. Otherwise, $\mathrm{tor}_2(t) \leq 1 + |X_t| + |A_t| + |B_t^{(2)}| \leq 1 + 2\omega + (4\omega+1) + 6\omega \cdot (\omega+1)^2 \leq 6(\omega+1)^3$ for any node t of T. Hence, the slim width of $(T, \mathcal{X})$ is at most $6(\omega+1)^3$. □

Combining this result with Proposition 3.26 shows that instead of using a tree-cut decomposition of the input graph G to design fixed-parameter algorithms—as was done in past dynamic programming algorithms that utilized tree-cut width—we can perform dynamic programming along a spanning tree T of a supergraph H of G. Both H and T can be computed from G in a pre-processing stage by using Theorem 3.29 and then Proposition 3.26, and using a spanning tree instead of a tree-cut decomposition typically leads to significantly more concise (and conceptually cleaner) algorithms.

Corollary 3.30. *There exists an algorithm that, given a graph G and $\omega \in \mathbf{N}$, either outputs a supergraph H of G along with a spanning tree T such that $\mathrm{ecw}(H, T) = \mathcal{O}(\omega^6)$ or correctly reports that the slim tree-cut width of G is greater than ω in $2^{\mathcal{O}(\omega^2 \cdot log\omega)} \cdot n^4$ steps.*

The cost for this simplification is the quadratic gap between the widths of these decompositions. We note that this situation is somewhat analogous to how one still typically uses clique-width [CMR00] as a general and easy-to-use parameterization for various problems (especially when aiming for instances with higher edge-densities), even though rank-width [Oum05] and Boolean-width [BTV11] are asymptotically equivalent parameterizations which have been shown to yield more efficient algorithms [GH10]—there, the gap is even exponential.

3.3 Algorithmic Application: Maximum Stable Matching

Having established its structural properties, we now turn to the algorithmic aspects of the slim tree-cut width. We exploit slim tree-cut width to obtain FPT algorithms for problems which remain W[1]-hard w.r.t. tree-cut width (see Table 3.1). In a joint work with Brand et al. [BCG+22] we showed that these problems are fixed-parameter tractable w.r.t. edge-cut width. Although the functional gap between edge-cut width and slim tree-cut width may be arbitrarily large, it is not hard to lift each of these tractability results to the super edge-cut width, and, hence, slim tree-cut width (recall that stcw and sec are asymptotically equivalent).

To illustrate this, here we provide an FPT algorithm for one of the problems from Table 3.1, which is Maximum Stable Roommates with Ties and Incomplete Lists (MSRTIL). This problem originates from the area of computational social choice [BHKN19]. It is W[1]-hard when parameterized by tree-cut width [BHKN19] but FPT when parameterized by degree treewidth [BHKN19, AGR+18].

In MSRTIL, we are given a set of *agents* V, where each agent $v \in V$ has a *preference* $\mathcal{P}_v = (P_v, \preceq_v)$. The agents $P_v \subseteq V \setminus \{v\}$ are called *acceptable (for v)* and $\preceq_v$ is a linear order on P_v with ties. Let $u, w \in P_v$. If $u \prec_v w$, then we say that v *strongly* prefers u to w; otherwise we say that v *weakly prefers* w to u.

We represent this problem via the undirected *acceptability graph* G, which contains a vertex for each agent in V and an edge between two agents if and only if both appear in the preference lists of the other. A set $M \subseteq E(G)$ is called a *matching* if no two edges in M share an endpoint. If the edge $\{v, w\}$ is contained in M, then we say v is *matched* to w and denote this as $M(v) = w$ and vice versa. In case a vertex v is not incident to any edge in M, then v is *unmatched* resp. $M(v) = \perp$ (where we assume $\perp$ to be less preferable than all acceptable neighbors of v). An edge $\{v, w\} \in E(G) \setminus M$ is *blocking* for M (we also say v, w form a *blocking pair*) if $w \prec_v M(v)$ and $v \prec_w M(w)$. A matching is *stable* if it does not admit a blocking pair.

Maximum Stable Roommates with Ties and Incomplete Lists (MSRTIL)	
Input:	A set of agents V, a preference profile $\mathcal{P} = (\mathcal{P}_v)_{v \in V}$, and an integer π.
Question:	Is there a stable matching of (V, P) of cardinality at least π?

Theorem 3.31. *MSRTIL is fixed-parameter tractable when parameterized by the slim tree-cut width of the acceptability graph.*

Proof. As a starting point, we apply Corollary 3.30 to compute a supergraph H of G along with its spanning tree T such that $\mathrm{ecw}(H, T) = \mathcal{O}(\mathrm{stcw}(G)^6)$. Let r be an arbitrarily chosen root in T. For each node $v \in V(T)$, we will use T_v to denote the subtree of T rooted at v. Without loss of generality, we will assume that G and H are connected.

The central notion used in our algorithm is that of a *boundary graph*, which plays a similar role as the bags in tree decompositions. Intuitively, the boundary graph contains all the edges which leave T_v (including the vertices incident to these edges). For each $v \in V(T)$, the *boundary graph* $\partial(v)$ of T_v is the edge-induced subgraph of G induced by those edges which have precisely one endpoint in T_v.

Observe that for each $v \in V(T)$, $|E(\partial(v))| \leq \mathrm{ecw}(H,T)$ and $|V(\partial(v))| \leq 2\,\mathrm{ecw}(H,T)$. It will also sometimes be useful to speak of the graph induced by the vertices that are "below" v in T, so we set $\mathcal{Y}_v = V(T_v)$ and $G_v = G[\mathcal{Y}_v]$; we note that $v \in \mathcal{Y}_v$ for every vertex v of G. Observe that $\partial(v)$ acts as a separator between vertices outside of $\mathcal{Y}_v \cup V(\partial(v))$ and vertices in $\mathcal{Y}_v \setminus V(\partial(v))$

We are now ready to design a dynamic programming procedure that proceeds the spanning tree T of H in a leaf-to-root fashion, computing records at each node, so that the record for the root will contain a solution to our problem. We start by defining the syntax of the records. For $v \in V(T)$, let a *signature* at v be a mapping $E(\partial(v)) \to \{\texttt{matched}, \texttt{unsafe}, \texttt{safe}\}$. Clearly, the number of signatures at v is upper-bounded by 3^k, where $k = \mathrm{ecw}(H,T)$.

To make it easier to describe the semantics of the records, let us first define the graph $\mathcal{G}_v$ as the non-disjoint union of G_v and $\partial(v)$; we recall that $\partial(v)$ contains both vertices in G_v and vertices adjacent to these, and that $E(\partial(v))$ forms an edge-cut separating G_v from the rest of G.

We are now ready to define the semantics of the records. A matching M in $\mathcal{G}_v$ is called a *partial solution* if there is no blocking edge for M in $E(\mathcal{G}_v)$; in other words, we explicitly forbid the edges in the boundary graph from forming blocking pairs in partial solutions. Each partial solution M *corresponds* to a signature $\texttt{sig}$ at v defined as follows:

- for each $e \in M \cap E(\partial(v))$, $\texttt{sig}(e) = \texttt{matched}$,
- for each $e = ab \in E(\partial(v)) \setminus M$ such that $a \in \mathcal{Y}_v$ and there exists $ac \in M$ such that $c \prec_a b$, $\texttt{sig}(e) = \texttt{safe}$, and
- $\texttt{sig}(e) = \texttt{unsafe}$ otherwise.

Intuitively, the signature of M captures the following information about M: which edges of the boundary graph are matched, and for those which are not matched it stores whether they are "safe" (meaning that the endpoint in $\mathcal{Y}_v$ will never form a blocking pair with that edge), or "unsafe" (meaning that the endpoint in $\mathcal{Y}_v$ could later form a blocking pair with that edge, depending on the preferences and matching of the endpoint outside of $\mathcal{Y}_v$).

We define $\mathcal{R}(v)$ to be a mapping from the set of all signatures at v to $\mathbf{N} \cup \{-\infty\}$, where (1) if there is no partial solution corresponding to a signature τ, then $\mathcal{R}(v)(\tau) \mapsto -\infty$, and otherwise (2) $\mathcal{R}(v)$ maps τ to the size of the largest partial solution in $\mathcal{G}_v$ whose

signature is τ. To avoid any confusion, we remark that when applying addition to the images of $\mathcal{R}(v)$, we let $-\infty + x = -\infty$ for each $x \in \mathbf{N} \cup \{-\infty\}$.

If we can compute $\mathcal{R}(r)$ for the root r of a spanning tree T witnessing that $\mathrm{ecw}(G, T) \leq k$, then by definition each partial solution is also a stable matching in the instance. Hence, it suffices to check whether $\mathcal{R}(r)(\emptyset) \geq \pi$; if this is the case, then we output "Yes", and otherwise we can safely output "No". At this point, it suffices to compute $\mathcal{R}(v)$ for each $v \in V(G)$ in leaf-to-root fashion along T.

If v is a leaf, we may assume without loss of generality that v is a vertex of G. First, we add the mapping $(E(\partial(v)) \mapsto \texttt{unsafe}) \mapsto 0$ to $\mathcal{R}(v)$, which corresponds to the empty partial solution. Then, for each $vw \in E(\partial(v))$ we construct a signature τ_w which assigns vw to `matched`, and for each neighbor u of v in G other than w either assigns vu to `safe` (if v weakly prefers w to u) or assigns vu to `unsafe` (if v strongly prefers u to w). For each τ_w constructed in this way, we set $\mathcal{R}(v)(\tau_w) = 1$.

If v is an internal node, there are two possibilities: either v is a vertex of G or it is a vertex of $H \setminus G$. In first case, we begin by branching over all edges $vw \in E(G)$ incident to v, and for each such edge we proceed by restricting our attention to all partial solutions which contain vw. In both cases, we also have a separate branch to deal with all partial solutions where v remains unmatched; in case $v \in H \setminus G$ this will be the only branch. We will begin by dealing with this branch.

Branch 1: v remains unmatched. Assume that for some child u of v it holds that $E(\partial(u)) = \{uv\}$. Then, in particular, u and v are vertices of G and $uv \in E(G)$. For every such child u of v, we observe that only partial solutions at u with the signature $uv \mapsto \texttt{safe}$ can be extended to a partial solution at v; indeed, $uv \mapsto \texttt{matched}$ would violate our assumption that v remains unmatched, while $uv \mapsto \texttt{unsafe}$ would, by definition, lead to a blocking pair. For brevity, let us set `simple-size` to be the sum of all $\mathcal{R}(u)(uv \mapsto \texttt{unsafe})$ over all vertices u with a single-edge boundary graph. In case there are no such u (in particular, if $v \notin V(G)$) we set $\texttt{simple-size} = 0$.

We observe that at this point only at most $2k$ children of v remain to be processed, say $x_0, \ldots, x_\ell$. We proceed by simultaneously branching over all of the at most 3^k signatures for each of these children, resulting in a total branching factor of 3^{k^2}; each branch can be represented as a tuple $(\texttt{sig}_{x_0}, \ldots, \texttt{sig}_{x_\ell})$. We now discard all tuples that are not well-formed, where a tuple is well-formed if the following conditions hold:

- it contains no signature that maps an edge incident to v to either `unsafe` or `matched` (as before, these edges may only be mapped to `safe`);
- for each edge $ab \in E(G)$ such that $a \in \mathcal{Y}_{x_i}$ and $b \in \mathcal{Y}_{x_j}$, $i, j \in [\ell]$, the signatures of x_i and x_j must either (a) both map that edge to `matched`, or (b) both map that edge to `safe`, or (c) map that edge to `safe` once and `unsafe` once (signatures must be consistent).

For all remaining tuples, we set `branching-size` to $\sum_{i\in[\ell]} \mathcal{R}(x_i)(\texttt{sig}_{x_i})$. We also identify a unique signature $\texttt{sig}^*$ corresponding to the current branch as follows: each edge in $\partial(v)$ incident to v is mapped to `unsafe`, and each edge e in $\partial(v)$ not incident to v must have an endpoint in $\mathcal{Y}_{x_i}$ for some x_i and is mapped to $\texttt{sig}_{x_i}(e)$. At this point, we update $\mathcal{R}(v)(\texttt{sig}^*)$ as follows: if the value of $\mathcal{R}(v)(\texttt{sig}^*)$ computed so far is greater than `simple-size` + `branching-size`, then we do nothing, and otherwise we set that value to `simple-size` + `branching-size`. We now proceed to the next branch, i.e., choice of neighbor of v.

Branch 2: v is matched to w. Recall that this can only happen if $v, w \in V(G)$ and $vw \in E(G)$. We will in principle follow the same steps as in the previous branch, but with a few extra complications. Let us begin by distinguishing whether (1) w itself is a child of v such that $E(\partial(u)) = \{uv\}$, (2) w is in $\mathcal{Y}_{x_i}$ for some child x_i of v not satisfying this property (including the case where $w = x_i$), or (3) $w \notin \mathcal{Y}_v$. In the first case, we set the child w aside and initiate `simple-size` $= \mathcal{R}(w)(wv \mapsto \texttt{matched})$. In the second case, we will later (in the appropriate branching step) discard all signatures of x_i which do not map wv to `matched`. In the third case, we will take this into account when constructing $\texttt{sig}^*$.

Next, for each child u of v such that $E(\partial(u)) = \{uv\}$ (other than w, in case (1)), we distinguish whether v weakly prefers w to u, or not. For each u where this holds, we observe that any partial solution at u that does not use v can be safely extended to a partial solution at v—hence, we increase `simple-size` by $\max(\mathcal{R}(u)(uv \mapsto \texttt{unsafe}), \mathcal{R}(u)(uv \mapsto \texttt{safe}))$. Furthermore, for each u where v strongly prefers u to w we observe that a partial solution at u can only be extended to one at v if it matches u in a way which prevents the creation of a blocking pair with v. Hence, in this case, we increase `simple-size` by $\mathcal{R}(u)(uv \mapsto \texttt{safe})$.

In the second step, we once again proceed by simultaneously branching over all of the at most 3^k signatures for the remaining children $x_0, \ldots, x_\ell$ of v. As before, this results in a total branching factor of 3^{k^2}, and each branch can be represented as a tuple $(\texttt{sig}_{x_0}, \ldots, \texttt{sig}_{x_\ell})$. We now discard all tuples that aren't well-formed, where a tuple is well-formed if the following conditions hold:

- for each edge ab such that $a \in \mathcal{Y}_{x_i}$ and $b \in \mathcal{Y}_{x_j}$, $i, j \in [\ell]$, the signatures of x_i and x_j must either (a) both map that edge to `matched`, or (b) both map that edge to `safe`, or (c) map that edge to `safe` once and `unsafe` once (signatures must be consistent);
- in case (2), the edge vw is mapped to `matched` in the appropriate signature;
- the tuple contains no signature that maps any edge incident to v (other than vw) to `matched`;
- for no edge vz where $z \in \mathcal{Y}_{x_i}$ for some $i \in [\ell]$ such that v strongly prefers z to w, the signature of x_i maps vz to `unsafe` (as this would create a blocking pair).

For all remaining tuples, we set `branching-size` to $\sum_{i\in[\ell]} \mathcal{R}(x_i)(\mathtt{sig}_{x_i})$ in cases (1) and (2); in case (3), we set it to $\sum_{i\in[\ell]} \mathcal{R}(x_i)(\mathtt{sig}_{x_i}) + 1$. We also identify a unique signature $\mathtt{sig}^*$ corresponding to the current branch as follows: each edge $vc \in E(\partial(v))$ is mapped to `unsafe` if v strongly prefers c to w, and `safe` otherwise (with the exception of $c = w$ in case (3), where vw must be mapped to `matched`). Furthermore, each edge e in $\partial(v)$ not incident to v must have an endpoint in $\mathcal{Y}_{x_i}$ for some x_i and is mapped to $\mathtt{sig}_{x_i}(e)$. At this point, we update $\mathcal{R}(v)(\mathtt{sig}^*)$ as follows: if the value of $\mathcal{R}(v)(\mathtt{sig}^*)$ computed so far is greater than `simple-size+branching-size`, then we do nothing, and otherwise we set that value to `simple-size+branching-size`. We then proceed to the next branch, i.e., choice of neighbor of v.

The correctness of the algorithm can be shown by induction; it is not difficult to verify that the computation of the records is correct at the leaves, and for non-leaves one uses the assumption that the records of the children are correct. The crucial point is that every partial solution at a child that corresponds to a certain signature can be extended to a partial solution at the parent if the verified conditions hold, which justifies the correctness of adding up the appropriate values for the children. The running time is upper-bounded by $3^{k^2} \cdot n^2$. □

3.4 Block Decompositions and more Algorithmic Applications

One way to design FPT algorithms for the slim tree-cut width, used in Section 3.3, is to consider an asymptotically equivalent parameter super edge-cut width and proceed dynamic programming along the spanning tree decomposition. But sometimes we can obtain the algorithm almost immediately, if the problem is known to be FPT by the treewidth plus maximum degree. This is based on one crucial advantage that slim tree-cut width pocesses, comparing to its predecessor edge-cut width. This advantage is related to block decompositions, which we use in this section to provide one more characterization of the slim tree-cut width.

After establishing some structural properties of block decompositions of graphs with bounded slim tree-cut width, we will exploit them to obtain the FPT results for the rest of the problems for which we provided FPT algorithms w.r.t. the edge-cut width in our earlier work [BCG+22]. The approach used here allows us to lift these previous algorithmic results to the more general setting of slim tree-cut width without requiring the technical and lengthy description of entirely new dynamic programming procedures. Intuitively, we obtain these results by showing that every graph of bounded slim tree-cut width can be decomposed into a tree of 2-connected blocks of bounded treewidth and degree, and this structure can be processed in a leaf-to-root fashion. We provide formal definitions in the following subsection.

3.4.1 Block Decompositions and Slim Tree-Cut Width

For a connected graph $G = (V, E)$, we say that $e \in E$ is a *bridge* if $G \setminus e$ is disconnected. We introduce an equivalence relation $\equiv$ on V such that for any $v, w \in V$, $v \equiv w$ if and only if the shortest path between v and w in G doesn't contain bridges. For $v \in V(G)$, let us denote by $\bar{v}$ the equivalence class containing v. We construct the *block decomposition* Υ_G of G which is a graph with nodes and edges defined as follows:

- the set of nodes $V(\Upsilon_G)$ is the set of equivalence classes $\{\bar{v} \mid v \in V\}$,
- for every bridge $e = (v, w)$ of G, add an edge $\bar{e} = (\bar{v}, \bar{w})$ to $E(\Upsilon_G)$.

While a block decomposition is well-defined for any graph and always forms a tree, in general it does not provide any algorithmic insights. For instance, if there are no bridges in G, then Υ_G consists of a unique node containing all the vertices of G. However, the situation is completely different if we consider graphs of bounded slim tree-cut width. Crucially, we can establish the following property:

Lemma 3.32. *Let G be a graph such that* $\mathrm{stcw}(G) \leq k$. *Then, for every $\bar{v} \in V(\Upsilon_G)$,* $\mathrm{tcw}_0(G[\bar{v}]) \leq k$.

Proof. Consider a tree-cut decomposition (T, χ) of $G[\bar{v}]$ of slim width at most k. By definition, the adhesion in each node t of T is at most k. The graph $G[\bar{v}]$ is connected and does not contain bridges, so for every node t both $\mathrm{tor}_1(t)$ and $\mathrm{tor}_2(t)$ are equal to the size of the torso at t, in particular $\mathrm{tor}_1(t) = \mathrm{tor}_2(t) \leq \mathrm{stcw}(G[\bar{v}]) \leq k$ and hence $\mathrm{tcw}_0(G[\bar{v}]) \leq k$. □

Along with Proposition 3.21 this immediatedly provides upper bounds for the maximum degree and treewidth of any induced subgraph $G[\bar{v}]$:

Corollary 3.33. *For a graph G and every $\bar{v} \in V(\Upsilon_G)$, maximal degree of $G[\bar{v}]$ is at most* $\mathrm{stcw}(G)^2 + 2\,\mathrm{stcw}(G)$ *and* $\mathrm{tw}(G[\bar{v}]) \leq 2\,\mathrm{stcw}(G)^2 + 3\,\mathrm{stcw}(G)$.

Next, we describe how this result can be used to lift fixed-parameter tractability from degree treewidth to slim tree-cut width.

3.4.2 Edge Disjoint Paths

Let us start from the Edge Disjoint Paths problem, which was shown to be FPT w.r.t. degree treewidth [GOR21] but W[1]-hard with respect to tree-cut width [GO21].

Edge Disjoint Paths (EDP)	
Input:	A graph G and a set P of *terminal pairs*, i.e., a set of subsets of $V(G)$ of size two.
Question:	Is there a set of pairwise edge disjoint paths connecting every terminal pair in P?

Consider an arbitrary instance (G, P) of EDP. Assume that $e \in E(G)$ is a bridge, and G_1 and G_2 are connected components of $G \setminus e$. If P contains at least two terminal pairs with one vertex in G_1 and another in G_2, we have a NO-instance, since any path connecting such a pair would be routed through e. Otherwise, if both endpoints of any terminal pair belong to the same connected component, we may solve the problem independently on G_1 and G_2, restricting the set of terminal pairs to their vertex sets. Finally, if there exists precisely one terminal pair $(v_1, v_2) \in P$ such that $v_1 \in V(G_1)$ and $v_2 \in V(G_2)$, then a path between v_1 and v_2 in a potential solution must contain the edge e. Let x_i be the endpoint of e that belongs to G_i, $i = 1, 2$. To solve the original instance (G, P), it is sufficient to independently solve the instances (G_i, P_i), where P_i contains all the pairs from P with both endpoints in $V(G_i)$ plus the pair (v_i, x_i), $i = 1, 2$.

Iteratively applying this argument for every bridge of G, we obtain the set of instances $(G[\bar{v}], P_v)$, $\bar{v} \in V(\Upsilon_G)$, such that (G, P) is a YES-instance if and only if every $(G[\bar{v}], P_v)$ is a YES-instance. Since by Corollary 3.33 the degree treewidth of each $G[\bar{v}]$ is bounded by a quadratic function of $\mathrm{stcw}(G)$, we may now apply the fixed-parameter algorithm from [GOR21].

Corollary 3.34. *Edge Disjoint Paths is fixed-parameter tractable w.r.t. the slim tree-cut width.*

3.4.3 Minimum Changeover Cost Arborescence

Some problems cannot be decomposed along bridges in such a simple way. In this case, instead of solving independent instances for each $G[\bar{v}]$, we try to iteratively compress the original instance, every time decreasing the number of nodes in the block decomposition. For example, let us consider the problem Minimum Changeover Cost Arborescence, which is W[1]-hard when parameterized by the tree-cut width [GÖP+17] but FPT when parameterized by degree treewidth [GSSZ16].

An *arborescence* is a directed tree with a designated vertex r (often called the *root*), which contains a directed path from each vertex to r. Given an arborescence T with root r and an edge $e \in E(T)$ we denote with $\mathtt{succ}(e)$ the edge incident to e on the path from v to the root r. For an edge e incident to the root we define $\mathtt{succ}(e) = e$.

A function $\mathtt{cost} : X^2 \to \mathbf{N}$ is called a *changeover cost function* if it satisfies the following:

1. $\mathtt{cost}(x_1, x_2) = \mathtt{cost}(x_2, x_1)$ for each $x_1, x_2 \in X$, and
2. $\mathtt{cost}(x, x) = 0$ for each $x \in X$.

The *total changeover costs* of an arborescence T are now defined as

$$\sum_{e \in E(T)} \mathtt{cost}(e, \mathtt{succ}(e)).$$

Minimum Changeover Cost Arborescence (MinCCA)	
Input:	A directed graph $D = (V, A)$, a root $r \in V(D)$, an arc coloring $\texttt{col} : A(D) \to X$, and a changeover cost function $\texttt{cost} : X^2 \to \mathbf{N}$.
Question:	What is an arborescence of D minimizing the total changeover costs?

Let $\mathcal{I} = (D, r, \texttt{col}, \texttt{cost})$ be instance of MinCCA, and let G be the skeleton of D. Since the root r is fixed, we can determine for each bridge e of the block decomposition Υ_G which orientation e must have in the arborescence. In particular, for any leaf t of Υ_G we can restrict $\mathcal{I}$ to the vertices contained in t, plus the bridge e_t connecting t to its parent s in Υ_G. Let v_t and v_s be the endpoints of e_t that belong to t and s, respectively. The skeleton of the restricted instance has bounded degree treewidth, and hence an optimal arborescence $T[t]$ can be constructed in FPT time by applying the known algorithm from [GSSZ16]. After obtaining $T[t]$, we delete from G all vertices of t, and for each vertex v such that $v_s v \in A$, we update $v_s v$ by changing its color to the new color $c_{v_s v}$ such that for any other color c, $\texttt{cost}(c_{v_s v}, c) = \texttt{cost}(\texttt{col}(e_t), \texttt{col}(v_s v)) + \texttt{cost}(\texttt{col}(v_s v), c)$. Iteratively applying this rule, we obtain an optimal arborescence $T[t]$ for each node t of Υ_G that respects additional costs from incoming bridges. An optimal arborescence T can then be obtained by gluing all the individual $T[t]$'s together.

Corollary 3.35. *MinCCA is fixed-parameter tractable when parameterized by the slim tree-cut width of the skeleton of G.*

3.4.4 List Coloring and Precoloring Extension

Another problem where some kind of iterative compression along the block decomposition works is List Coloring. A *coloring* col is a mapping from the vertex set of a graph to a set of colors; a coloring is *proper* if for every pair of adjacent vertices a, b, it holds that $\texttt{col}(a) \neq \texttt{col}(b)$. The problem can then be formulated as follows:

List Coloring	
Input:	A graph $G = (V, E)$ and for each $v \in V$ a list $L(v)$ of permitted colors.
Question:	Does G admit a proper coloring col where for each vertex v it holds $\texttt{col}(v) \in L(v)$?

List Coloring is one more example of a problem that is W[1]-hard w.r.t. tree-cut width but FPT w.r.t. degree treewidth [GKS22].

Consider an instance $\mathcal{I} = (G, L)$ of List Coloring. If Υ_G consists of a unique node, we conclude that G has bounded degree treewidth and therefore $\mathcal{I}$ can be solved in FPT time [GKS22]. Otherwise, we show how to iteratively compress the leaves of Υ_G.

Let $\bar{v} \in V(\Upsilon_G)$ be a leaf adjacent to $\bar{w}$, where $vw \in E(G)$. First, for each color $c \in L(v)$, we create an instance $\mathcal{I}_c = (G[\bar{v}], L_c)$ where L_c coinsides with L on $\bar{v}$ except $L_c(v) = \{c\}$. Since $G[\bar{v}]$ has bounded degree treewidth, every such instance can be solved in FPT

time [GKS22]. Observe that $L'(v) = \{c \in L(v) \mid \mathcal{I}_c \text{ is a YES-instance}\}$ is the set of all colors that v receives in some proper coloring of $G[\bar{v}]$ that agrees with L. If $L'(v) = \emptyset$, $\mathcal{I}$ is obviously a NO-instance. If the cardinality of $L'(v)$ is at least 2, removing $\bar{v}$ from $V(G)$ results in an instance that is equivalent to $\mathcal{I}$: any coloring of $G \setminus \bar{v}$ can be extended to $\bar{v}$ so that v and w receive different colors. Finally, if $L'(v)$ contains a unique color c, we remove $\bar{v}$ from $V(G)$ and, in case $c \in L(w)$, remove c from $L(w)$. In each case, we reduce the number of nodes of the block decomposition by one. For the last node, we simply apply the FPT algorithm from [GKS22].

Precoloring Extension is another coloring problem from Table 3.1, where a graph is given with a partial coloring, and the task is to extend it to the proper coloring. As was mentioned in [BCG+22], this problem can be considered as a special case of List Coloring.

Corollary 3.36. *List Coloring and Precoloring Extension are fixed-parameter tractable w.r.t. slim tree-cut width.*

3.4.5 Boolean CSP

Finally, let us consider the classical constraint satisfaction problem over a Boolean domain [SS10], which is W[1]-hard w.r.t. tree-cut width [GKS22]. An instance $\mathcal{I}$ of Boolean CSP is a tuple (X, C), where X is a finite set of *variables* and $\mathcal{C}$ is a finite set of *constraints*. Each constraint in $\mathcal{C}$ is a pair (S, R), where the *constraint scope* S is a non-empty sequence of distinct variables of X, and the *constraint relation* R is a relation over $\{0, 1\}$ (given as a set of tuples) whose arity matches the length of S. An *assignment* is a mapping from the set X of variables to $\{0, 1\}$. An assignment σ satisfies a constraint $C = ((x_1, \ldots, x_n), R)$ if $(\sigma(x_1), \ldots, \sigma(x_n)) \in R$, and σ satisfies the Boolean CSP instance if it satisfies all its constraints. An instance $\mathcal{I}$ is satisfiable if it is satisfied by some assignment.

Boolean CSP	
Input:	A set of variables X and a set of constraints $\mathcal{C}$.
Question:	Is there an assignment $\sigma : X \to \{0, 1\}$ satisfying all constraints in $\mathcal{C}$?

We represent this problem via the *incidence graph* G, whose vertex set is $X \cup \mathcal{C}$ and which contains an edge between a variable and a constraint if and only if the variable appears in the scope of the constraint. Boolean CSP is known to be fixed-parameter tractable when parameterized by the degree treewidth of G [SS10]. Similarly as before, to lift the result to slim tree-cut width, it is sufficient to provide a reduction compressing a leaf of Υ_G in a given instance $\mathcal{I}$. We distinguish the following two cases.

Case 1: *$\bar{v} \in V(\Upsilon_G)$ is a leaf adjacent to $\bar{c}$, where v, c are the unique variable in $\bar{v}$ and constraint in $\bar{c}$, respectively, such that v appears in the scope of c.* We create two instances $\mathcal{I}_0$ and $\mathcal{I}_1$, where $\mathcal{I}_i$ is obtained from $\mathcal{I}$ by restricting G to $\bar{v}$ and adding a new constraint with the scope (v) which is satisfied if and only if $\sigma(v) = i$.

- if both $\mathcal{I}_0$ and $\mathcal{I}_1$ are NO-instances, clearly $\mathcal{I}$ is a NO-instance as well.
- if both $\mathcal{I}_0$ and $\mathcal{I}_1$ are YES-instances, we simply remove from $\mathcal{I}$ all the variables and constraints that belong to $\bar{v}$ and remove v from the scope of c (i.e., we forget corresponding coordinate in the constraint relation). For correctness, observe that any satisfying assignment σ of the modified instance can be extended to v to satisfy c. Then, if $\sigma(v) = i$, any solution to $\mathcal{I}_i$ provides an extension of σ to the variables in $\bar{v}$,
- if for some $i \in \{0, 1\}$, $\mathcal{I}_i$ is a YES-instance and $\mathcal{I}_{1-i}$ is a NO-instance, we remove from $\mathcal{I}$ all the variables and constraints that belong to $\bar{v}$, same as before. But now, before removing v from the scope of c, we delete from the constraint relation of c all the tuples which assign to v the value $1 - i$. This also results in an equivalent instance, since any sattisfying assignment for $\mathcal{I}$ must map v to i.

Case 2: *$\bar{c} \in V(\Upsilon_G)$ is a leaf adjacent to $\bar{v}$, and c, v are the unique constraint in $\bar{c}$ and variable in $\bar{v}$, respectively, such that v appears in the scope of c.* For each $i \in \{0, 1\}$, we try to satisfy all the constraints in $\bar{c}$ while assigning to v the value i. For this, we solve the instances $\mathcal{I}_0$ and $\mathcal{I}_1$ where $\mathcal{I}_i$ is obtained from $\mathcal{I}$ by deleting from c all the tuples that set v equal to $1 - i$, removing v from the scope of c and then removing all the variables and constraints outside of $\bar{c}$. There can be three possible outcomes:

- if both $\mathcal{I}_0$ and $\mathcal{I}_1$ are NO-instances, we conclude that $\mathcal{I}$ is a NO-instance.
- if both $\mathcal{I}_0$ and $\mathcal{I}_1$ are YES-instances, we simply remove from $\mathcal{I}$ all the variables and constraints that belong to $\bar{c}$.
- if for some $i \in \{0, 1\}$, $\mathcal{I}_i$ is a YES-instance and $\mathcal{I}_{1-i}$ is a NO-instance, any potential solution should assign to v the value i. Hence we can safely delete from the constraints of $\mathcal{I}$ all the tuples setting v equal to $1 - i$, and then remove variables and constraints that belong to $\bar{c} \cup \{v\}$.

Corollary 3.37. *Boolean CSP is fixed-parameter tractable w.r.t. the slim tree-cut width of the incidence graph.*

3.5 Discussion

The contributions of this chapter are mainly conceptual: they provide a possible resolution to the search for an alternative to treewidth for edge cuts which is both structurally sound and exhibits the expected (and desired) algorithmic properties. Slim tree-cut width can be viewed as the "missing link" which explains why the problems depicted in Table 3.1 admit fixed-parameter algorithms that exploit dynamic programming along small edge cuts w.r.t. both edge-cut width (as a generalization of the feedback edge number) and treewidth plus maximum degree. We firmly believe that there are many more problems of interest where edge-cut based parameters may help push the frontiers

of tractability. On this front, the alternative characterization via the edge-cut width of a supergraph provides decompositions which are better suited for dynamic programming than tree-cut decompositions. The problem of computing optimal decompositions for slim tree-cut width remains, as in the case of tree-cut width [KOP+18], a prominent open question. Moreover, we believe that the ideas used to obtain a 2-approximation algorithm for tree-cut width could also be used to obtain an improved constant-factor approximation for slim tree-cut width. In this context, edge-cut width, a predecessor of the slim tree-cut width, has a practical advantage—it can be computed in FPT time along with a witnessing spanning tree decomposition of minimal width.

3.5.1 Edge-cut Width or Slim Tree-cut Width?

Given the ease of transferring dynamic programming algorithms from edge-cut width or degree treewidth to slim tree-cut width, an inquisitive reader might be wondering whether it is not possible to formally prove that *every* problem which is FPT w.r.t. former is also FPT w.r.t. the latter. That is, however, not true in general: one can construct entirely artificial problems which do not behave in this way.

To illustrate this on a high level for edge-cut width, let us consider an arbitrary graph problem PP which remains NP-hard even on trees (as an example, the Firefighter problem [FKMR07]) and can be solved on general n-vertex graphs in time $\tau(n)$. Moreover, let $\gamma(n)$ denote the time required to compute the slim tree-cut width of a graph G via an exhaustive brute force search, and let ψ be a function which dominates both τ and γ. We now define an artificial new problem PP′ as follows:

- every n-vertex graph G such that $\psi(\mathrm{ecw}(G)) \leq n$ is a YES-instance, and otherwise
- G is a YES-instance if and only if G is a YES-instance of Firefighter.

Then PP′ is FPT parameterized by edge-cut width. Indeed, given an instance (G, k) of PP′, one can attempt to run a brute-force search to determine the edge-cut width (which is promised to be at most k) with a time-out of $\psi(\psi(k))$. If the algorithm times out, this implies that $\psi(\mathrm{ecw}(G)) \leq n$ and we correctly output "Yes". If not, we proceed by calling a brute-force algorithm to solve Firefighter on G, and this must once again complete in time at most $\psi(\psi(k))$. Conversely, PP′ remains NP-hard even on graph classes with constant $\mathrm{stcw}(G)$—consider, for instance, the class of all graphs with two connected components, one of which is a tree and the other a graph from the class with constant slim tree-cut width but unbounded edge-cut width (one such class is depicted in Figure 2 of [BCG+22]). On some inputs from this class, PP′ will ask for a solution to the Firefighter problem (which is NP-hard on trees) but the parameter $\mathrm{stcw}(G)$ will remain constant.

3.5.2 Other edge-cut based measures

Recently, Chang et al. [CKL23] introduced new edge-cut based measures, called *α-edge-crossing width* and *edge-crossing width*. The parameters are defined in terms of tree-cut decompositions similarly to tree-cut width, but use the notions of *thickness* and *crossing number* instead of torso size and adhesion respectively.

The *thickness* of a tree-cut decomposition (T, X) is simply the maximal cardinality of a bag, while the *crossing number* of a bag t measures the number of edges that have endpoints in different components of $T \setminus t$. For a positive integer α, the *α-edge-crossing width* of a graph G, denoted by $\mathrm{ecrw}_\alpha(G)$, is the minimum crossing number over all tree-cut decompositions of G whose thicknesses are at most α. The edge-crossing width of (T, X) is the maximum of the crossing number and the thickness of (T, X). The edge-crossing width of G, denoted by $\mathrm{ecrw}(G)$, is the minimum edge-crossing width over all tree-cut decompositions of G, and it turns out to be asymptotically equivalent to the *tree-partition width*.

Both α-edge-crossing width, for any $\alpha \in \mathbf{N}$, and edge-crossing width are more general than slim tree-cut width but more restrictive than treewidth. Somewhat surprisingly, α-edge-crossing width is incomparable to tree-cut width [CKL23, Lemmas 3.7 and 3.8]. Along with the fact that Precoloring Extension is FPT when parameterized by ecrw_α for any fixed α, it would be natural to ask whether ecrw_α allows to achieve tractability for any other problem from Table 3.1.

For completeness, we also note that an article exploring a different parameter that is aimed at providing an edge-based alternative to treewidth was recently authored by Magne, Paul, Sharma and Thilikos [MPST23]; the parameter is based on different ideas and is incomparable to both tree-cut width and slim tree-cut width.

CHAPTER 4

Edge-Cut Based Parameters in Causal Network Learning

Causality has become a crucial topic in AI research for overcoming the limitations of machine learning systems that are typically based on correlation. While *causal inference* focuses on estimating the effect of a known or hypothesized causal relationship, *causal discovery* (the subject of this chapter) aims to uncover new causal relationships directly from data, often without prior hypotheses.

The causal structure over a set of random variables can be explicitly expressed as a *causal graph*, which is a directed graph whose vertices are the variables, and where an arc $X \to Y$ indicates that X is a direct cause of Y. Causal discovery is the problem of identifying as much as possible about the causal graph given a dataset of measurements over the underlying variables; in some cases, part of the causal relationships among the variables is provided from expert knowledge [NZZZ21, CSCD16, TBA06] which can be presented in terms of a *super-structure* [PIM08, OS13, GK21, GK22b], a graph containing all edges which are known or allowed to be part of the causal graph in some orientation.

An important family of algorithms for causal discovery are *constraint-based algorithms* which work with a (usually very large) set of observational data and repeatedly perform independence checks between pairs of variables, i.e., test $X \perp Y \mid Z$. Here, it is crucial to distinguish how one models these tests: one commonly assumes these are performed by an oracle which can either (1) determine whether X and Y are conditionally independent via a single query, or (2) test independence under a specific assignment of the conditioning set Z. Here, we take the latter approach—which we from now on refer to as the *fine-grained oracle model*—as it allows us to more faithfully model settings where, e.g., the data needs to be obtained via on-the-fly experiments or when the amount of data is excessively large. We note that under the fine-grained oracle model, performing independence tests

for very large choices of Z becomes infeasible—a fact which also corresponds well to the known unreliability of such independence tests [WL21, TP20].

Our main goal in this chapter is to draw a connection between the graph structure and the efficiency of constraint-based causal discovery algorithms. Our main contributions include (i) a near-tight characterization of which causal graphs can potentially be efficiently recovered by a constraint-based causal discovery algorithm, (ii) the explicit construction of a sequence of causal graphs on which the influential PC algorithm works extremly inefficiently, and (iii) the formulation of a new causal discovery algorithm which achieves fixed-parameter running time by considering the maximum number of edge-disjoint paths between variables in the (undirected) super-structure as the parameter.

4.1 Related Work

The *PC algorithm*, proposed by Peter and Clarke [SGS00], is a constraint-based algorithm that had a monumental impact on causal discovery and served as the basis for several other constraint-based algorithms; in fact, it is considered the "default algorithm for attempting causal discovery" [Sha23]. The statistical independence check used by the PC algorithm (and many other constraint-based causal discovery algorithms) is based on so-called d-separating sets (note that "d" here stands for "dependence" and is not a number). Since the running time of the test in the fine-grained oracle model is exponential in the size of the d-separating set, the PC algorithm performs the tests with gradually increasing d-separating sets, starting from checking unconditional independence (which corresponds to checking for the existence of an empty d-separating set).

There are two main approaches to causal discovery: constraint-based and score-based. Score-based causal discovery was extensively studied in the literature, both in terms of standard and parameterized complexity [OS13, KP15, GK21, GKM21, GK22b]. In this chapter we focus on constraint-based causal discovery. Its complexity is well-studied in terms of the number of conditional independence tests. As Claassen, Mooij and Heskes ([CMH13]) noted, the PC algorithm allows learning causal networks whose node degree is upper-bounded by k via at most $n^{2(k+2)}$ independence tests. Recent works have also explored other constraint-based causal discovery algorithms, including the refinement ED-PC of the PC algorithm by Wienöbst and Liskiewicz ([WL21]) and MARVEL [MAGK21]; the latter achieves nearly-tight bounds in the number of independence tests performed.

However, these results do not carry over to the fine-grained oracle model since the algorithms assume the possibility of performing efficient independence tests with large conditioning sets; the sizes of these sets are only upper-bounded by the maximal in-degree. The importance of reducing the sizes of conditioning sets was also mentioned in many other articles (see, e.g., the discussion in the work of [MAJ+22]). For instance, [TP20] aimed to bound the number of oracle calls and the size of d-separating sets. They did so by considering the so-called *moral graph* of the hidden causal graph and showed that whenever the moral graph has treewidth at most k, each pair of variables admits a

d-separating set of size at most $k + 1$. This is incomparable to our results: we show that the size of a d-separating set is upper-bounded by a function of the maximum edge-cut of a skeleton of the hidden causal graph (which allows us to obtain strong bounds even for skeletons with high-degree nodes), but it is easy to observe that a bound on the treewidth of the moral graph implies a bound on the degree of the skeleton. An example of an instance where our bounds supersede previous results can be seen, e.g., in Figure 4.5. Moreover, it is unclear whether one can efficiently compute moral graphs in the fine-grained oracle model.

It is known that the smallest d-separating set between X and Y corresponds to the smallest vertex separator in a moral graph of D restricted to ancestors of X and Y [TP98]. We show that this characterization cannot be extended to skeletons (and, therefore, neither to the super-structure setting) by constructing a family of networks with skeletons containing a vertex separator of size 3 between every pair of variables but arbitrarily large smallest d-separating sets between some pair of variables (Theorem 4.5). We summarize the discussed related work and some of our results, providing upper-bounds on the size of the smallest d-separating set, in Table 4.1.

4.2 Our Contribution

The PC and other constraint-based algorithms proposed for causal discovery over the last two decades have been refined and analyzed from statistical and empirical perspectives (see the discussion of related work in Section 4.1). However, relatively little is known about the complexity-theoretic foundations of constraint-based causal discovery algorithms in the fine-grained oracle model. We perform the first complexity-theoretic analysis in this more refined setting by attacking the following three fundamental research questions.

Q1 *Which structural properties of the causal graph guarantee small d-separating sets between any pair of variables?* A small d-separating set is a prerequisite for applying constraint-based causal discovery algorithms efficiently and provides guarantees for their running times.

Q2 *How large are the d-separating sets discovered by the PC algorithm compared to smallest d-separating sets?* Let us refer to the difference between the size of a d-separating set identified by the PC algorithm and the size of a smallest such set as the "approximation error". The smaller the approximation error, the closer the PC algorithm's time complexity is to the time complexity of an idealistic constraint-based algorithm.

Q3 *Which properties of the provided super-structure guarantee efficient causal discovery?* We consider this question with respect to the PC algorithm, but also for new constraint-based algorithms that we design to exploit the properties of the super-structure better.

separation property	skeleton	moral graph
vertex sep. of size $\leq k$	**unrestricted**	(restricted to ancestors of X and Y): $\leq k$
treewidth $\leq k$	**unrestricted**	$\leq k+1$
edge cut of size $\leq k$	$\mathbf{\leq 2^{k^2}}$	$\leq 2k$ (see first row)

Table 4.1: Upper bounds for the size of the smallest d-separating set between a pair X, Y of variables, provided for different separation properties of a skeleton or moral graph. Our results are marked in bold. Here **unrestricted** means that the minimal size of a d-separating set can be arbitrary large, even if the parameter k is a small constant.

Addressing Q1: We obtain a near-tight characterization of which causal graphs admit a small d-separating set for each pair of vertices. In particular, if the number of edge-disjoint paths between each pair of vertices in the skeleton of the causal graph (i.e., the undirected graph underlying the causal graph) is bounded, then there is a d-separating set between each pair of vertices of bounded size (Theorem 4.3). At the same time, the statement cannot be strengthened to simply speak of edge-disjoint directed paths (Observation 4.4) or to the number of vertex-disjoint paths in the skeleton (Theorem 4.5). Conversely, if a causal graph admits a d-separating set of size at most ℓ for each pair of non-adjacent vertices, then for each pair of non-adjacent vertices, the number of directed vertex-disjoint paths is at most ℓ (Observation 4.8, Corollary 4.9). We show that this result is tight in the sense that it cannot be improved to a bound on the number of vertex-disjoint or edge-disjoint paths in the skeleton (Lemma 4.10) or to a bound on the number of directed edge-disjoint paths in the causal graph (Lemma 4.11). These results are presented in Section 4.4, see Table 4.1 for a short summary.

Addressing Q2: We first observe that no constraint-based causal discovery algorithm can run faster than in time n^k, where n is the number of variables and k is the size of a smallest d-separating set between any pair of variables (Lemma 4.13). In other words, the size of a smallest d-separating set is a hard complexity bound for all constraint-based algorithms.

In the literature, it has been claimed that the PC algorithm will find and use a smallest d-separating set between every pair of variables [CMH13, Subsection 3.1]. We, in fact, show that this is not the case: the PC algorithm might fail to run efficiently due to failing to find a small d-separating set even when such a set exists. This would be relatively easy to show when expert knowledge is provided, but our construction works even in the base case with no expert knowledge (i.e., when the super-structure is a complete graph). In particular, we construct a non-trivial class of instances where the PC algorithm only discovers a d-separating set of size linear in the total number of variables, even though there exists a d-separating set of size 3 (Theorem 4.14). This direction is investigated within our Section 4.5.

Addressing Q3: We have now seen two obstacles to efficient causal discovery: if the d-separating set is large, then no constraint-based algorithm will run efficiently, but even if it is small, the PC algorithm might not run efficiently. Here, we consider whether natural graph-theoretic structural restrictions on the super-structure can help us address both of these obstacles to guarantee efficient causal discovery for cases where some (suitable) expert knowledge is available. In particular, we aim to avoid the exponential blowup of n^k by designing FPT algorithms for the problem.

A typical restriction one would consider in the setting of fixed-parameter tractability would be to consider super-structures of bounded treewidth [RS86]; however, as our first result in this direction we prove that using treewidth (as well as many other more restrictive variants of it that have been used in this context) as a parameter cannot yield fixed-parameter tractability for causal discovery, for any constraint-based algorithm (Corollary 4.16). In contrast, we show that the maximum degree in the super-structure is a parameter which guarantees that the PC algorithm will terminate in fixed-parameter time (Theorem 4.17). Unfortunately, the restriction to super-structures of bounded degree only is very strong, and so this result begs the question of whether tractability can be extended to super-structures of unbounded degree.

We show that by a slight modification of the PC algorithm, we can lift the tractability for the bounded-degree case to parameterizing by the maximum degree within each 2-connected component (Theorem 4.18); it is worth noting that fixed-parameter tractability for the PC algorithm without our modification cannot be guaranteed in this case (Corollary 4.19). We conclude by developing a new causal discovery algorithm that guarantees a fixed-parameter running time in a much more general setting—in particular, there it suffices to parameterize by the maximum number of edge-disjoint paths between any pair of variables (Theorem 4.20)—and again match this with a lower bound showing that the PC algorithm (even the modified one) cannot solve this case efficiently (Corollary 4.21). This third set of results is presented in Section 4.6.

4.3 Basic Notions, d-Separation and the PC algorithm

A *causal graph* is a directed acyclic graph $\mathcal{D}$ over a set $V(\mathcal{D})$ of variables (represented as vertices) whose arcs represent causal relationships between these variables. Here we consider variables to range over a finite and fixed domain of size B. A vertex (or variable) X_i is a *collider* on a path $\mathcal{P}$ if the path contains $X_{i-1} \to X_i \leftarrow X_{i+1}$; otherwise it is a *non-collider* [GZS19].

A *super-structure* $\mathcal{G}$ is a graph on the same vertex set as the causal graph $\mathcal{D}$ which forms a supergraph of the skeleton $\underline{\mathcal{D}}$ of $\mathcal{D}$, i.e., $\underline{\mathcal{D}}$ can be obtained from $\mathcal{G}$ by removing some set of edges. In the context of causal discovery, it is used to represent all potential dependencies between variables (known, e.g., from prior or expert knowledge).

Conditional Independence and d-Separation. Let $\mathcal{D}$ be a causal graph and P be a probability distribution over its vertices. $\mathcal{D}$ and P satisfy the *Causal Markov Condition* if and only if every vertex X of $\mathcal{D}$ is conditionally independent of its non-descendants given its parents. $\mathcal{D}$ and P satisfy the *Faithfulness Condition* if and only if every conditional independence relation true in P is entailed by the Causal Markov Condition applied to $\mathcal{D}$ [SGS00]. The two conditions—Faithfulness and Causal Markov—are common assumptions in causal discovery, and we also assume these to hold in the remainder of Chapter 4.

Let $\mathcal{S}$ be a subset of variables in $\mathcal{D}$, and let X and Y be two different variables outside of $\mathcal{S}$. We say that X is *d-separated* from Y conditional on $\mathcal{S}$ if and only if all paths between X and Y in $\underline{D}$ are *blocked* by $\mathcal{S}$. A path $\mathcal{P}$ is *blocked* by $\mathcal{S}$ if at least one of the following two conditions holds:

- $\mathcal{P}$ contains a non-collider which is in $\mathcal{S}$, or
- $\mathcal{P}$ contains a collider which has no descendants in $\mathcal{S}$.

Assuming the Causal Markov Condition and the Faithfulness Condition, X is d-separated from Y by $\mathcal{S}$ if and only if X and Y are conditionally independent given $\mathcal{S}$ [GZS19]. Further, we will denote both of these facts (namely, conditional independence and d-separation of X from Y given $\mathcal{S}$) by $X \perp Y \mid \mathcal{S}$.

The PC algorithm. The PC algorithm is one of the basic algorithms used to determine the Markov Equivalence Class (i.e., a partially oriented skeleton) of the causal graph. It receives as input a set $V(\mathcal{D})$ of variables, possibly a super-structure of the (hidden) causal graph over $V(\mathcal{D})$, and an oracle that can check conditional (in)dependencies between variables X, Y w.r.t. a subset of $V(\mathcal{D}) \setminus \{X, Y\}$.

In its entirety, the PC algorithm consists of two phases: the first phase (called the *learning phase*) computes a skeleton, while the second obtains a partial orientation of the skeleton obtained in the first phase. The key distinction is that while the second phase can be implemented as a polynomial-time postproceessing routine, the learning phase can in general take exponential time. Hence, in this complexity-theoretic study we focus our attention solely to the learning phase, which forms the bottleneck in the worst-case running time of the PC algorithm. This phase consists of the following sequence of procedures (algorithm from [GZS19], modified for the case of expert knowledge):

1. Construct a complete undirected graph over the provided set of variables.
2. Eliminate edges between variables that are either unconditionally independent or are known to be independent due to expert knowledge (non-adjacent in the super-structure).
3. Set $i = 1$ and repeat the following:

- for each ordered pair of variables (X, Y) having an edge between them, and for each subset $\mathcal{S} \subseteq V(\mathcal{D}) \setminus \{X, Y\}$ of size i of variables adjacent to Y, eliminate the edge between X and Y if $X \perp Y \mid \mathcal{S}$.
- increment i by 1.
- if the obtained skeleton has maximum degree at most i, return it as the answer.

After the skeleton was returned, let $k \leq i$ be the maximum size of sets that need to be considered in Step 3 in order for the PC algorithm to discover the whole skeleton of the causal graph. We remark that if k is known in advance, it can be used to provide a basic upper bound on the running time of the PC algorithm, since we may safely stop the process once all sets of size at most k have been checked.

Indeed, observe that there are only $\mathcal{O}(n^k)$ subsets of size k, and that checking the independence of X and Y conditional on every such set $\mathcal{S}$ requires the performance of an independence test for X and Y for every assignment of variables from $\mathcal{S}$. Since the number of such assignments is upper-bounded by $B^{|S|}$, the time required to process one pair of variables for an independence test conditioned on a set of size k is upper-bounded by $B^k \cdot n^{\mathcal{O}(k)}$.

Various variants of the PC algorithm have been proposed over the last two decades [GZS19]. We will refer to all such variants—and in particular, all complete algorithms that are based on checking the conditional independence of pairs of variables in terms of d-separation in a learning phase, followed by an orientation phase (such as the FCI algorithm [SGS00])—as *constraint-based* causal discovery algorithms.

4.4 Characterizing Small d-Separating Sets

The aim of this section is to characterize necessary and sufficient conditions for the existence of bounded-size d-separating sets in a hidden causal graph $\mathcal{D}$. While one would at first glance expect that such conditions would be closely tied to structural properties that depend on the orientation of the arcs in $\mathcal{D}$ (and, in fact, some of our results provide precisely such a relationship), here we provide positive results for conditions that are oblivious to the orientation of the arcs, i.e., conditions that solely depend on the undirected skeleton $\underline{\mathcal{D}}$. There are two reasons for this:

1. It is known to be impossible to identify the exact hidden causal graph $\mathcal{D}$ using constraint-based algorithms, but such algorithms can reliably compute $\underline{\mathcal{D}}$ [Ebe17, Section 2]. This makes the undirected skeleton a tangible and guaranteed output of such algorithms.
2. We provide concrete examples and constructions which show that the results linking the existence of bounded-size d-separating sets to undirected structures in $\underline{\mathcal{D}}$ cannot be lifted to analogous directed structures in $\mathcal{D}$.

As a first basic step, it will be useful to provide a lemma that can be seen as an independent short proof of the correctness of the PC algorithm. Let X and Y be two distinct variables in $\mathcal{D}$, and let V_{XY} be the union of vertices of all undirected paths between X and Y in $\underline{\mathcal{D}}$. We denote by $\mathcal{D}_{XY}$ and $\underline{\mathcal{D}}_{XY}$ the vertex-induced subgraphs of $\mathcal{D}$ and $\underline{\mathcal{D}}$ respectively induced by V_{XY} in $\underline{\mathcal{D}}$. For the following, it will be useful to recall that the causal graphs considered here are acyclic.

Note that the only difference between $\mathcal{D}_{XY}$ and $\underline{\mathcal{D}}_{XY}$ is that the former preserves edge orientations from $\mathcal{D}$.

Lemma 4.1. *Let X and Y be distinct and non-adjacent variables such that X is not reachable from Y in $\mathcal{D}$. Let $\mathcal{S}$ be the set of all in-neighbors of Y in $\mathcal{D}_{XY}$. Then $X \perp Y \,|\, \mathcal{S}$.*

Proof. Let $\mathcal{Q}$ be the set of all vertices of $\mathcal{D}$ reachable from Y via directed paths in $\mathcal{D}$, and observe that $\mathcal{S} \cap \mathcal{Q} = \emptyset$. To see that $\mathcal{S}$ is a d-separating set for X and Y, consider an arbitrary undirected path $\mathcal{P}$ between X and Y in $\underline{\mathcal{D}}_{XY}$. If $\mathcal{P}$ contains an arc ZY for some in-neighbor Z of Y, then Z belongs to $\mathcal{S}$ and occurs as a non-collider in $\mathcal{P}$, so it blocks $\mathcal{P}$. Otherwise, $\mathcal{P}$ contains an arc YZ for some out-neighbor Z of Y, in which case $Z \in \mathcal{Q}$. Since X is not reachable from Y, there is some collider W on $\mathcal{P}$ that is reachable from Z. In particular, all descendants of W belong to $\mathcal{Q}$, so they are not contained in $\mathcal{S}$. Hence, W blocks $\mathcal{P}$. □

Lemma 4.1 immediately implies that if the causal graph has in-degree ℓ, then every pair of variables admits a d-separating set of size at most ℓ. In particular, bounded maximum in-degree in $\mathcal{D}$—and hence also bounded maximum degree in $\underline{\mathcal{D}}$—is a sufficient condition for the existence of small d-separating sets. Naturally, such a condition is far from necessary: for instance, the class of simple stars with all edges oriented towards the center has arbitrarily large maximum in-degree even though each pair of non-adjacent variables admits a d-separating set of size 0.

As our next task, we will provide a strictly more general, comparing to the maximum in-degree, sufficient condition on $\underline{\mathcal{D}}$ for the existence of small d-separating sets—specifically, the existence of a bounded number of edge-disjoint paths between each pair of variables. The next lemma provides the first step towards this goal:

Lemma 4.2. *Let X and Y be non-adjacent vertices in $\underline{\mathcal{D}}$ such that X is not reachable from Y in $\mathcal{D}$. Assume that $\underline{\mathcal{D}}$ admits an edge-cut of size k which separates X and Y and does not touch X. Then X and Y can be d-separated by a set of size at most $2k$ in $\mathcal{D}$.*

Proof. Let $\mathcal{C}$ be the set of endpoints of the edge-cut of size k separating X from Y, and observe that $|\mathcal{C}| \leq 2k$. As before, we define $\mathcal{Q}$ to be the set of all vertices of $\mathcal{D}$ reachable from Y via directed paths in $\mathcal{D}$.

We show that $\mathcal{S} = \mathcal{C} \setminus \mathcal{Q}$ is a d-separating set for X and Y. Indeed, let $\mathcal{P}$ be an arbitrary undirected path between X and Y. If $\mathcal{P}$ contains vertices from $\mathcal{Q} \setminus \{Y\}$, then at least

one such vertex—say Z—must be a collider since X is not reachable from Y in $\mathcal{D}$. In this case, also all descendants of Z in $\mathcal{D}$ belong to $\mathcal{Q}$. By the construction of $\mathcal{S}$ we have $\mathcal{Q} \cap S = \emptyset$, and hence Z blocks $\mathcal{P}$.

The remaining case is that $\mathcal{P} \cap \mathcal{Q} = \{Y\}$; here, let e be an arc of $\mathcal{P}$ that belongs to the selected edge-cut of size k. In the subcase where Y is an endpoint of e, another endpoint Z of e must be an in-neighbor of Y since $Z \notin \mathcal{Q}$. In particular, Z is a non-collider that belongs to $\mathcal{S}$, and hence Z blocks $\mathcal{P}$. Otherwise both endpoints of e are in $\mathcal{C} \setminus \mathcal{Q} = \mathcal{S}$. At least one of these is a non-collider and hence this element of $\mathcal{S}$ once again blocks $\mathcal{P}$, as desired. □

We note that it is crucial that the edge-cut considered in Lemma 4.2 does not touch X: Figure 4.1 depicts a causal graph admitting an edge-cut $\{XZ, XW\}$ of size two, but requires large d-separating set for X and Y. Indeed, to block the path $X \to Z \to Y$, we need to include Z into a d-separating set. Because of this, we cannot block any of the paths $X \to W \leftarrow V_i \to Y$ by excluding the collider W and its descendants from the d-separating set. Hence, all of the variables V_i, $i \in [n]$, must be part of the d-separating set.

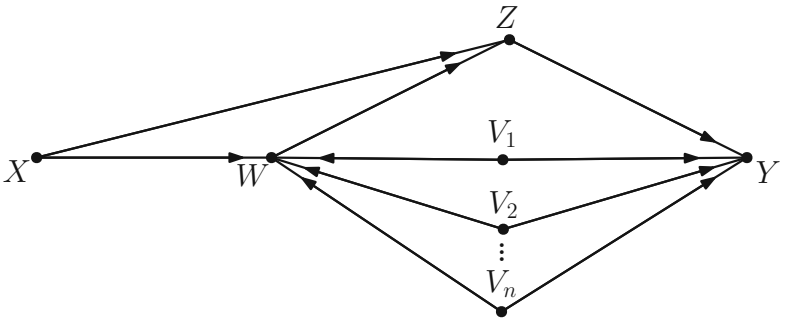

Figure 4.1: An example of a causal graph where X and Y are separated by two edges XZ and XW, but any d-separating set for X and Y has size at least $n + 1$.

Interestingly, when aiming for a sufficient condition that would guarantee the existence of small d-separating sets between *all* pairs of variables, we show that the existence of arbitrary edge-cuts of bounded size is sufficient (regardless of which variables they touch).

Theorem 4.3. *If every pair of vertices X and Y can be separated in $\underline{\mathcal{D}}$ by an edge-cut of size at most k, then every pair of non-adjacent variables can be d-separated in $\mathcal{D}$ by a set of size at most $2k^2$.*

Proof. Let X and Y be an arbitrary pair of non-adjacent (i.e., conditionally independent) variables. Assume w.l.o.g. that X is not reachable from Y in $\mathcal{D}$. Let $\mathcal{E}'$ be the edge-cut of size at most k that separates X from Y in $\underline{\mathcal{D}}$. We modify $\mathcal{E}'$ to exclude X from the set of endpoints as follows. For every edge $e = XZ$ of $\mathcal{E}'$, we replace e by an edge-cut of size at most k that separates Z from Y in $\underline{\mathcal{D}} \setminus \{X\}$. The size of the resulting edge-cut $\mathcal{E}$

can be upper-bounded by k^2, as in the worst case it could happen that every edge $e \in \mathcal{E}'$ is replaced by k new edges not incident to X. Crucially, $\mathcal{E}$ does not touch X. At this point, we apply Lemma 4.2 to X, Y and $\mathcal{E}$ to obtain the claimed d-separating set of size at most $2k^2$. □

Notice that the bound above is imposed on the number of undirected edge-disjoint paths. By contrast, restricting the number of directed edge-disjoint paths does not suffice: as a simple example, consider a causal graph $\mathcal{L}_n$ that consists of two variables X and Y plus n vertex-disjoint paths of length 2 between them, where every vertex other than X and Y is a source. In particular, X and Y are sinks.

Observation 4.4. *$\mathcal{L}_n$ does not contain a directed path between any pair of non-adjacent variables, but the only d-separating set for X and Y has size n.*

However, this still raises the question of whether Theorem 4.3 could be strengthened by simply requiring the existence of a vertex-separator of size at most k (instead of having an edge-cut of size at most k) between each pair of variables. Below, we answer this in the negative:

Theorem 4.5. *There exists a family $\{\mathcal{D}_i : i \in \mathbb{N}\}$ of causal graphs with $|V(\mathcal{D}_i)| = \mathcal{O}(4^i)$, $i \in \mathbf{N}$, such that each $\mathcal{D}_i$ contains a pair of non-adjacent variables with no d-separating set of size smaller than $2^{i-1} + 1$, and yet each pair of variables in $\mathcal{D}_i$ admits a vertex separator of size 3.*

Proof. For convenience, let us define $\mathcal{D}_0$ to simply be a causal graph consisting of a single arc. If the edge begins in X and ends in Y, we say that $\mathcal{D}_0$ has *orientation* XY. We construct the family by induction; to this end, let us assume that the causal graph $\mathcal{D}_i$, $i \in \mathbb{N} \cup \{0\}$, has already been defined and has a special pair of vertices called its orientation.

For the construction itself, take two vertex-disjoint copies of $\mathcal{D}_i$ with orientations XR and $R'Y$ and glue them together by identifying R and R', and denote the obtained graph as $\mathcal{D}_i^{\text{repeat}}$. Take another two vertex-disjoint copies of $\mathcal{D}_i$ with orientations $M'X'$ and MY' and glue them together by identifying M and M', and denote the graph as $\mathcal{D}_i^{\text{mirror}}$. Finally, glue together $\mathcal{D}_i^{\text{repeat}}$ and $\mathcal{D}_i^{\text{mirror}}$ by identifying X' with X and Y' with Y. The resulting graph is $\mathcal{D}_{i+1}$ with orientation XY. An illustration of the construction—which may be viewed as "fractal" in nature—is provided in Figure 4.2.

By construction, for every $i \in \mathbb{N}$, the graph $\mathcal{D}_{i+1}$ contains 4 copies of $\mathcal{D}_i$ as subgraphs. If M and R are the gluing points for the copies of $\mathcal{D}_i$ in $\mathcal{D}_i^{\text{mirror}} \subseteq \mathcal{D}_{i+1}$ and $\mathcal{D}_i^{\text{repeat}} \subseteq \mathcal{D}_{i+1}$ respectively, then the orientations of these four copies are XR, RY, MX and MY. It also follows from the construction that Y is a sink, X and R have precisely one outgoing edge, and M has precisely two outgoing edges in $\mathcal{D}_{i+1}$. Moreover, Y is reachable from X, and both X and Y are reachable from M.

Claim 4.6. *For every $i \in \mathbb{N}$, $\mathcal{D}_i$ admits a vertex separator of size at most 3 between any pair of vertices.*

Proof of the Claim: For $\mathcal{D}_1$ the claim is trivial. Let A and B be two vertices of $\mathcal{D}_i$, $i \geq 2$. Let $j \in [i] \cup \{0\}$ be minimal such that A and B belong to the same copy of $\mathcal{D}_j$ in $\mathcal{D}_i$, further in this proof we will refer to it simply as $\mathcal{D}_j$. Let QW be the orientation of $\mathcal{D}_j$, and let M and R be the gluing vertices of its subgraphs $\mathcal{D}_{j-1}$ in $\mathcal{D}_{j-1}^{\text{mirror}}$ and $\mathcal{D}_{j-1}^{\text{repeat}}$ respectively.

If $\{A, B\} \cap \{Q, W\} = \emptyset$, we add Q and W to the vertex separator to hit every path between A and B that contains vertices outside of $\mathcal{D}_j$. In case A and B are both in $\mathcal{D}_{j-1}^{\text{repeat}}$ (both in $\mathcal{D}_{j-1}^{\text{mirror}}$), they cannot coincide with R or M due to the minimality of j, and so we may add R (or M, respectively) to the vertex separator in order to intersect the remaining paths inside $\mathcal{D}_j$.

Otherwise, without loss of generality we may assume that $A = Q$. Then we form the vertex separator by taking R and M to intersect the paths inside $\mathcal{D}_j$, plus (in case $j < i$) one of the two vertices determining the orientation of $\mathcal{D}_{j+1} \supseteq D_j$ other than Q and W; see Figure 4.3 for an illustration. ■

Claim 4.7. *For every $i \in \mathbb{N}$, the causal graph $\mathcal{D}_i$ with orientation XY contains no d-separating set between X and Y of size smaller than $2^{i-1} + 1$.*

Proof of the Claim: For $i = 1$ the claim obviously holds, since in $\mathcal{D}_1$ the only d-separating set between X and Y has size $2 = 2^0 + 1$. For the inductive step, assume that the statement holds for $\mathcal{D}_i$ and consider the graph $\mathcal{D}_{i+1}$ with orientation XY and gluing points M and R for the copies of $\mathcal{D}_i$ in $\mathcal{D}_i^{\text{mirror}} \subseteq \mathcal{D}_{i+1}$ and $\mathcal{D}_i^{\text{repeat}} \subseteq \mathcal{D}_{i+1}$, as defined in the construction. Let $\mathcal{S}$ be some minimal d-separating set for X and Y in $\mathcal{D}_{i+1}$. Observe that if $R \in \mathcal{S}$, then R only blocks the paths where it occurs as a non-collider. But all such paths contain the unique out-neighbor of R (which by construction has in-degree of one), so we can replace R in $\mathcal{S}$ by its out-neighbor. Similarly, if M belongs to S, we replace it by its only two out-neighbors. This results in a d-separating set $\mathcal{S}'$ for X and Y of size at most $|\mathcal{S}| + 1$ that does not contain M and R.

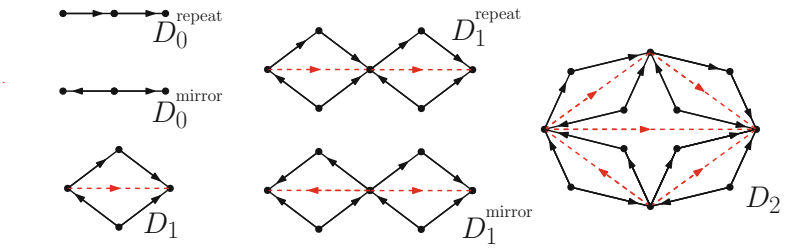

Figure 4.2: Construction of $\mathcal{D}_{i+1}$ from 4 copies of $\mathcal{D}_i$ for $i = 0, 1$. The red dashed arcs depict the orientations.

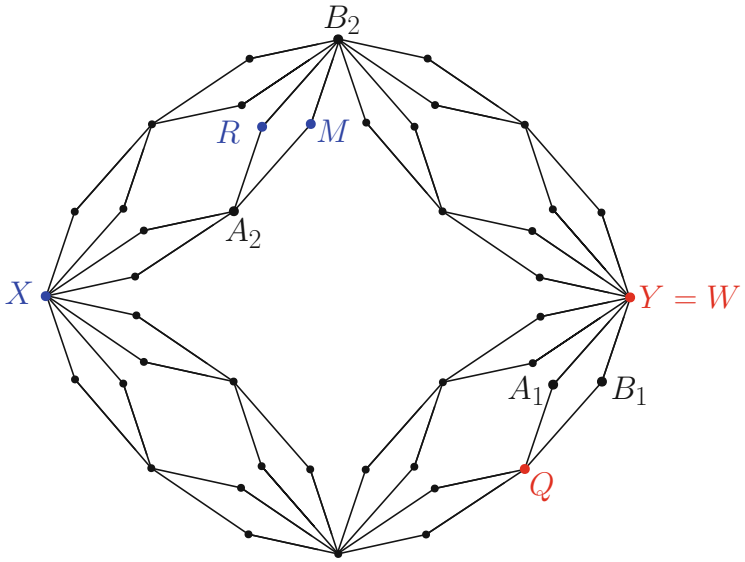

Figure 4.3: Vertex separators in the skeleton of $\mathcal{D}_3$ from the proof of Claim 4.6. Red vertices form the vertex separator between A_1 and B_1, while blue vertices form the vertex separator between A_2 and B_2. In both cases, $j = 1$.

Since $\mathcal{S}'$ blocks every path in $\mathcal{D}_i^{\text{mirror}} \subseteq \mathcal{D}_{i+1}$, it should block every path in at least one of the copies of $\mathcal{D}_i$ in $\mathcal{D}_i^{\text{mirror}}$. Indeed, assume that there are paths $\mathcal{P}_1$ in one copy and $\mathcal{P}_2$ in another copy which are not blocked by $\mathcal{S}'$. Since their concatenation $\mathcal{P}_1\mathcal{P}_2$ is blocked by $\mathcal{S}'$ and $M \notin \mathcal{S}'$, M must be a collider on $\mathcal{P}_1\mathcal{P}_2$ such that none of its descendants belong to $\mathcal{S}'$. Recall that X and Y are reachable from M via paths in the corresponding copies of $\mathcal{D}_i$. The concatenation of these paths forms the path between X and Y that consists only of descendants of M and does not contain colliders, so it is not blocked by $\mathcal{S}'$, a contradiction. This implies that the restriction $\mathcal{S}'_M$ of $\mathcal{S}'$ to $\mathcal{D}_i^{\text{mirror}} \subseteq \mathcal{D}_{i+1}$ forms a d-separating set for at least one of the copies of $\mathcal{D}_i$ in $\mathcal{D}_i^{\text{mirror}}$. In particular, S'_M has size of at least $|V(\mathcal{D}_i)|$ by the inductive assumption.

To show that $\mathcal{S}'$ also blocks every path in at least one of the copies of $\mathcal{D}_i$ in $\mathcal{D}_i^{\text{repeat}}$, we distinguish two cases. Firstly, assume that none of the descendants of R in $\mathcal{D}_i^{\text{repeat}}$ belongs to $\mathcal{S}'$. Let $\mathcal{P}_0$ be the directed path from R to Y in the copy of $\mathcal{D}_i \subseteq \mathcal{D}_i^{\text{repeat}}$ with orientation RY. Consider an arbitrary path $\mathcal{P}$ from X to R in the copy of $\mathcal{D}_i \subseteq \mathcal{D}_i^{\text{repeat}}$ with orientation XR. Their concatenation $\mathcal{P}\mathcal{P}_0$ is blocked by $\mathcal{S}'$, and by our assumptions $\mathcal{P}_0 \cap \mathcal{S}' = \emptyset$. Therefore, $\mathcal{P}$ is blocked by $\mathcal{S}'$. Since $\mathcal{P}$ was chosen arbitrarily, we conclude that the restriction $\mathcal{S}'_R$ of $\mathcal{S}'$ to $\mathcal{D}_i^{\text{repeat}} \subseteq \mathcal{D}_{i+1}$ forms a d-separating set for the copy of $\mathcal{D}_i$ with orientation XR. It remains to consider the case when $\mathcal{S}'$ contains descendants of R. Assume that there are paths $\mathcal{P}_1$ in one copy of $\mathcal{D}_i \subseteq \mathcal{D}_i^{\text{repeat}}$ and $\mathcal{P}_2$ in another copy which are not blocked by $\mathcal{S}'$. Since R does not belong to $\mathcal{S}'$ but has descendants in $\mathcal{S}'$, their concatenation $\mathcal{P}_1\mathcal{P}_2$ is not blocked by $\mathcal{S}'$, a contradiction. Hence, $\mathcal{S}'_R$ forms the d-separating set for at least one of the copies of $\mathcal{D}_i$ in $\mathcal{D}_i^{\text{repeat}}$. In particular, $\mathcal{S}'_R$ has size of at least $2^{i-1} + 1$ by the inductive assumption.

Therefore, $|\mathcal{S}| \geq |\mathcal{S}'| - 1 = |\mathcal{S}'_R| + |\mathcal{S}'_M| - 1 \geq 2(2^{i-1} + 1) - 1 = 2^i + 1$, which by the minimality of $\mathcal{S}$ concludes the proof. ■

The theorem follows directly from the above two claims. □

Theorems 4.3 and 4.5 imply that while having a bounded number of undirected edge-disjoint paths between variables guarantees the presence of a bounded-size d-separating set, this is not the case for undirected vertex-disjoint paths. Hence, having a bounded number of undirected edge-disjoint paths can be seen as a *sufficient* condition for the existence of a bounded-size d-separating set. Our aim in the rest of this section is to identify a condition which is *necessary*.

We begin by making a simple observation concerning a necessary condition tied to the hidden causal graph (as opposed to the skeleton). In particular, since a d-separating set must intersect each directed path between a pair of variables in the causal graph, we obtain:

Observation 4.8. *If a causal graph $\mathcal{D}$ admits a d-separating set of size k for each pair of non-adjacent variables, then for each pair of non-adjacent variables the number of directed vertex-disjoint paths is bounded by the same bound k.*

In terms of conditions on the actual skeleton $\underline{\mathcal{D}}$, we note the following. If the number of vertex-disjoint paths between two non-adjacent variables X and Y in $\underline{\mathcal{D}}$ is large, then there exists an acyclic orientation of $\underline{\mathcal{D}}$ with no small d-separating set between X and Y (this occurs, e.g., when all of the X-Y paths are oriented from X to Y). Hence, having a small number of vertex-disjoint paths between each pair of non-adjacent variables is necessary to guarantee that every orientation of $\underline{\mathcal{D}}$ admits a small d-separating set. In other words:

Corollary 4.9. *Let $\underline{\mathcal{D}}$ be an undirected graph. Assume that, in every acyclic orientation $\mathcal{D}$ of $\underline{\mathcal{D}}$, every pair of non-adjacent variables can be d-separated in $\mathcal{D}$ by a set of size k. Then every pair of non-adjacent variables can be separated in $\underline{\mathcal{D}}$ by a vertex separator of size at most k.*

Finally, the fact that Corollary 4.9 references *every* orientation $\mathcal{D}$ of $\underline{\mathcal{D}}$ is crucial. Indeed:

Lemma 4.10. *There exists a family of undirected graphs $\mathcal{H} = \{\underline{\mathcal{X}}_i \mid i \in \mathbf{N}\}$ with the following properties:*

- *Each $\underline{\mathcal{X}}_i$ admits an acyclic orientation $\mathcal{X}_i$ where every pair of non-adjacent variables can be d-separated by a set of size 0, and*
- *$\underline{\mathcal{X}}_i$ contains a pair of non-adjacent variables A, B whose minimum vertex separator has size i.*

Proof. Let $\underline{\mathcal{X}_i}$ be the graph consisting of two variables A, B and i vertex-disjoint A-B paths, each containing a single internal vertex V_j, $j \in [i]$. Clearly, this family satisfies the latter condition of the lemma. For the former condition, it suffices to consider the orientation $\mathcal{X}_i$ which makes each variable V_j other than A and B a sink. □

As the final result in this section, we also exclude the possibility of generalizing Observation 4.8 to a bound on the number of directed edge-disjoint paths (as opposed to a bound on the number of directed vertex-disjoint paths).

Lemma 4.11. *There exists a family $\{\mathcal{D}'_i : i \in \mathbb{N}\}$ of causal graphs with $|V(\mathcal{D}'_i)| = \mathcal{O}(4^i)$, $i \in \mathbf{N}$, such that each $\mathcal{D}_i$ contains 2^i directed edge-disjoint paths between some pair of non-adjacent variables, and yet each pair of non-adjacent variables can be d-separated by a set of size at most 3.*

Proof. For every $i \in \mathbf{N}_0$, we define the causal graph $\mathcal{D}'_i$ as follows. The skeleton of $\mathcal{D}'_i$ coincides with the skeleton of $\mathcal{D}_i$ from Theorem 4.5, and if such $\mathcal{D}_i$ has orientation XY, in $\mathcal{D}'_i$ we orient all the arcs on every path from X and Y towards Y. In particular, $\mathcal{D}'_{i+1}$ can be obtained by gluing together 4 copies of $\mathcal{D}'_i$, $i \in \mathbf{N}_0$.

Unlike in $\mathcal{D}_i$, we can find constant-sized d-separating sets between each pair of variables in $\mathcal{D}'_i$:

Claim 4.12. *For every $i \in \mathbf{N}_0$, $\mathcal{D}'_i$ admits a d-separating set of size at most 3 for any pair of non-adjacent variables.*

Proof of the Claim: For $\mathcal{D}'_0$ the claim is trivial. Let A and B be two non-adjacent vertices of $\mathcal{D}'_i$, $i \in \mathbb{N}$. Let $j \in [i]$ be minimal such that A and B belong to the same copy of $\mathcal{D}'_j$ in $\mathcal{D}'_i$; for brevity, we will refer to this particular copy of $\mathcal{D}'_j$ simply as $\mathcal{D}'_j$. Let QW be the orientation of $\mathcal{D}'_j$, and let M and R be the gluing vertices of its subgraphs $\mathcal{D}'_{j-1}$ other than Q and W.

By minimality of j, X and B belong to different copies of $\mathcal{D}'_{j-1}$. We can always block the paths between A and B that lie inside D'_j by a set $\mathcal{S}_1$ of size at most 2. Namely, if $A = Q$ or $B = W$, we choose $\mathcal{S}_1 = \{R, M\}$. If $A \neq Q$, $B \neq W$ and the copies of $\mathcal{D}'_j$ containing A and B are glued in one of the vertices R or M, then any path between A and B inside $\mathcal{D}'_j$ will intersect either this vertex as a non-collider or W as a collider. Otherwise any path between A and B inside $\mathcal{D}'_j$ will intersect either Q as a non-collider or W as a collider. In either of these cases, this allows us to choose $\mathcal{S}_1$ of size 1.

Let us define the d-separating set to be $\mathcal{S} = \mathcal{S}_1 \cup \{T\} \setminus \{A\}$, where T is present if and only if $i > j$ and denotes the source of $\mathcal{D}'_{j+1} \supseteq \mathcal{D}'_j$. In this case we denote the sink of this $\mathcal{D}'_{j+1}$ by U. see Figure 4.4 for the illustration. Since $\mathcal{S}$ does not contain any descendant of W (including W itself), it also blocks all the paths between A and B that lie inside $\mathcal{D}'_j$.

Let $\mathcal{P}$ be a path between A and B that is not contained in $\mathcal{D}'_j$. Then $\mathcal{P}$ must cross T and U. If $A \neq T$, then $\mathcal{P}$ contains $T \in \mathcal{S}$ as a non-collider. Otherwise $W \neq U$ and the

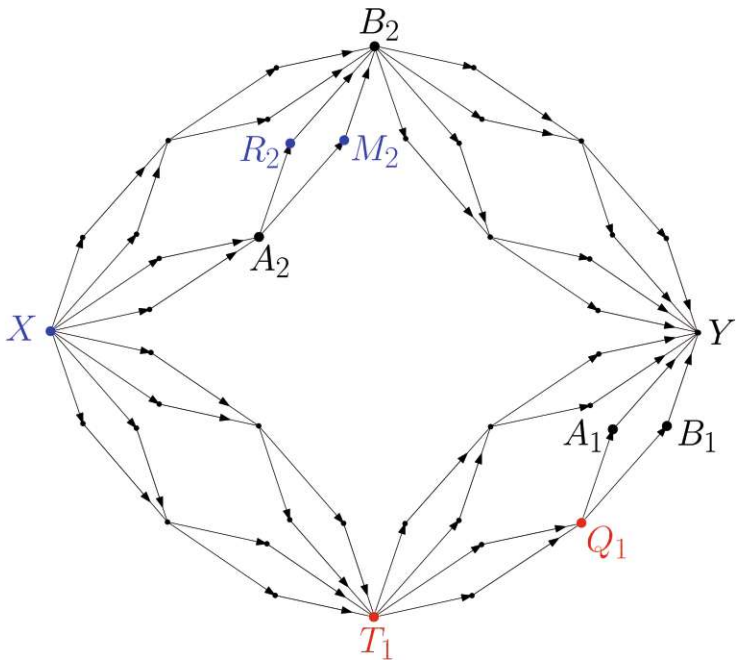

Figure 4.4: Two examples of d-separating sets in $\mathcal{D}'_3$ from the proof of Claim 4.12. Red vertices d-separate A_1 and B_1, blue vertices d-separate A_2 and B_2. For each pair A_i and B_i, the notations are same as in the proof, but with the lower index i.

subpath of $\mathcal{P}$ between W and U is oriented towards Y. Since A is not reachable from Y, the subpath of $\mathcal{P}$ between U and A contains a collider of $\mathcal{P}$ reachable from W. In particular, the collider has no descendants in $\mathcal{S}$ and therefore blocks $\mathcal{P}$. ■

The lemma now follows from the fact that each causal graph $\mathcal{D}_i$ consists of 2^i edge-disjoint paths from X to Y. □

4.5 Limits of Tractability for the PC Algorithm

While the previous section was aimed at identifying necessary and sufficient conditions for the existence of a small d-separating set, having a small d-separating set itself is merely a *necessary* condition for efficient causal discovery. Indeed, in this section we show that the existence of a small d-separating set cannot on its own guarantee efficient running times for causal discovery.

We begin with a lower bound ruling out fixed-parameter causal discovery by any constraint-based algorithm.

Lemma 4.13. *There exists a class of causal graphs* $\mathcal{H} = \{\mathcal{D}_{i,j} \mid i, j \in \mathbf{N}; j \leq i\}$ *such that:*

1. *each* $\mathcal{D}_{i,k}$ *contains* $n = i + 3$ *vertices,*
2. *each* $\mathcal{D}_{i,k}$ *has a single pair* X, Y *of non-adjacent vertices with a d-separating set of size* k, *and*
3. *every constraint-based algorithm needs to test* $\Omega(n^{k-1})$ *subsets to determine whether* X *and* Y *are conditionally independent in* $\mathcal{D}_{i,k}$.

Proof. Let each $\mathcal{D}_{i,k}$ be a complete digraph minus the arc XY, where X is a source and Y has k incoming and $n - k - 1$ outgoing arcs. Furthermore, there is an arc from every in-neighbor of Y to every out-neighbor (i.e., none of the in-neighbors of Y is a descendant of any of Y's out-neighbors). The remaining arcs are oriented arbitrarily in a way which preserves acyclicity.

Observe that $\mathcal{D}_{i,k}$ contains precisely one d-separating set between X and Y: it contains every in-neighbor Z of Y (as these are the only non-colliders on the path XZY), and contains no out-neighbor W of Y (as these are the only colliders on the path XWY). Therefore, the set consists of precisely k in-neighbors of Y. Before discovering this set, any constraint-based algorithm satisfying the properties specified in Section 4.3 would in the worst case need to test all of the $\Omega(n^{k-1})$ subsets of size at most $k - 1$. □

While Lemma 4.13 may already seem disheartening, the lower bound it provides only becomes truly problematic as k grows above a small constant. In other words, at this point it would still seem plausible that the PC algorithm achieves a running time of, e.g., $n^{\mathcal{O}(k)}$, where k is the size of the largest d-separating set in the causal graph. As our second result in this section, we show that the situation is in fact much worse—there is a class of causal graphs for which the PC algorithm will necessarily run in exponential time, even though all pairs of variables admit very small d-separating sets between them.

Theorem 4.14. *For each* $n \geq 2$, *there exists a causal graph* $\mathcal{T}_n$ *on* $\mathcal{O}(2^n)$ *vertices such that* $\mathcal{T}_n$ *admits d-separating sets of size at most 3 between every pair of non-adjacent variables, but the PC algorithm only discovers a d-separating set of size* $2^n + 2$ *for at least one pair of variables.*

Proof. To obtain $\mathcal{T}_n$, we start from two full binary trees $\mathcal{T}_n^1$ and $\mathcal{T}_n^2$ with 2^n leaves each, where all the arcs are oriented from the roots to the leaves. We glue $\mathcal{T}_n^1$ and $\mathcal{T}_n^2$ together at their roots and denote the resulting common root by O_n. Moreover, we introduce a variable W_n and add an incoming arc to it from every leaf of $\mathcal{T}_n^1$. Symetrically, we introduce a variable E_n and add an incoming arc to it from every leaf of $\mathcal{T}_n^2$. Finally, we add two variables N_n and S_n along with out-going arcs to X and Y from both of them, see Figure 4.5 for an illustration.

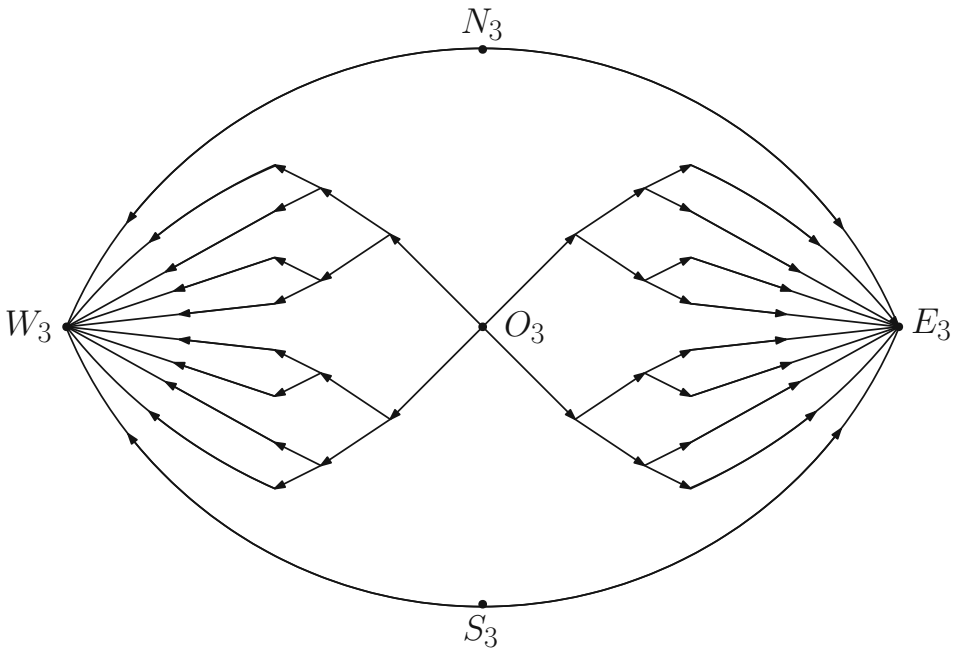

Figure 4.5: The causal graph $\mathcal{T}_3$, where E_3 and W_3 admit a d-separating set of size 3 but the PC algorithm only discovers a d-separating set of size 10.

Claim 4.15. *For every $n \geq 2$, the sizes of minimal d-separating sets between non-adjacent variables are as follows:*

- *0 between S_n or N_n and any vertex other that E_n and W_n;*
- *1 between any pair of vertices from $V(\mathcal{T}_n^1) \cup V(\mathcal{T}_n^2)$, between W_n (E_n) and any non-root vertex of $\mathcal{T}_n^2$ ($\mathcal{T}_n^1$);*
- *2 between O_n and E_n or W_n;*
- *3 between W_n and E_n, between W_n and any non-root vertex of $\mathcal{T}_n^1$, between E_n and any non-root vertex of $\mathcal{T}_n^2$.*

For every such pair except for W_n and E_n, the minimal d-separating set can be chosen from the neighborhood.

Proof of the Claim: First, notice that E_n and W_n are sinks. In particular, S_n and N_n are d-separated by an empty set from every vertex other than E_n and W_n.

Furthermore, any two non-adjacent vertices X and Y from $V(\mathcal{T}_n^1) \cup V(\mathcal{T}_n^2)$ are d-separated by the parent of one of them in the corresponding tree: any path between X and Y passes either through the parent or through the sink E_n or W_n. The same holds for W_n and any non-root vertex of $\mathcal{T}_n^2$, as well as E_n and any non-root vertex of $\mathcal{T}_n^1$.

In order to d-separate O_n from W_n (or from E_n), it suffies to take the two children of O_n in $\mathcal{T}_n^1$ (in $\mathcal{T}_n^2$ respectively), which matches the lower bound given by the number of vertex-disjoint paths not containing a collider.

Between W_n and any non-root vertex X of $\mathcal{T}_n^1$, there are 3 such paths: one of them passes through the parent of X and then through the root O_n, while two others follow the directions of arcs. In particular, the three neighbours of X form the d-separating set of the minimal size. Same holds for E_n and any non-root vertex of $\mathcal{T}_n^2$.

Finally, the only d-separating set for W_n and E_n of size 3 consists of S_n, N_n and O_n. Here, S_n and N_n should be necessarily added to any d-separating set as the only internal vertices on the paths of length 2. If we look for a d-separating set between W_n and E_n that contains only their neighbors, we have to also select all the leaves of one of the trees $\mathcal{T}_n^1$ or $\mathcal{T}_n^2$, resulting in a set of size at least $2^n + 2$. ■

Now, let us analyze how the PC algorithm will discover the d-separating sets in $\mathcal{T}_n$. In the first stage, it searches for d-separating sets of size 0, which will disconnect S_n and N_n from all the other vertices except E_n and W_n.

Next, it finds all the d-separating sets of size 1 and 2 (these can be chosen from the respective neighborhoods by Claim 4.15) and removes the corresponding edges.

Further, PC proceeds to d-separating sets of size 3. The only such set for E_n and W_n cannot be discovered since O_n is not their neighbor anymore. However, any other pair of non-adjacent vertices of $\mathcal{T}_n$ that remains connected until this stage, becomes disconnected since it admits a d-separating set of size 3 chosen from the neighborhood by Claim 4.15. Therefore, after this stage we obtain an undirected graph consisting of the skeleton of $\mathcal{T}_n$ plus one additional edge W_nE_n. To eliminate this final edge, the PC algorithm will need to add N_n and S_n along with all the leaves of $\mathcal{T}_n^1$ or $\mathcal{T}_n^2$ to the d-separating set, resulting in a set of size $2^n + 2$. □

4.6 Utilizing the Super-Structure for Fixed-Parameter Algorithms

In our final section, we show that the lower bounds arising from Lemma 4.13 and Theorem 4.14 can both be circumvented if we are provided with a super-structure possessing suitable structural properties. In particular, our aim is to show that causal discovery can be carried out in fixed-parameter time not with respect to the size of a maximum d-separating set (as this was ruled out in Lemma 4.13), but instead with respect to a structural measure of a provided super-structure.

A first natural graph-theoretic measure of the super-structure would be the well-established treewidth. However, we can immediately rule out the use of treewidth (and also many other related graph parameters) by recalling the graph class $\mathcal{L} = \{\mathcal{L}_i \mid i \in \mathbf{N}\}$ used in Observation 4.4. Indeed, X and Y form a vertex cover of each graph $\mathcal{L}_i \in \mathcal{L}$, and hence all graphs in that class have treewidth at most 2. However, the only d-separating set

between X and Y has size i. Thus, by setting $\underline{\mathcal{L}}'_i$ to be a super-structure for $\mathcal{L}_i$ that contains each edge in $\underline{\mathcal{L}}_i$ plus the edge XY, we obtain the corollary below. In particular, each $\mathcal{L}_i$ consists of X, Y, and a set of i sources adjacent to both X and Y, and $\underline{\mathcal{L}}'_i$ adds a "superfluous" edge XY to the skeleton of $\mathcal{L}_i$.

Corollary 4.16. *For every $i \in \mathbf{N}$, there exists a super-structure $\underline{\mathcal{L}}'_i$ of $\mathcal{L}_i$ with a vertex cover of size 2 such that no constraint-based causal discovery algorithm can compute the skeleton of the hidden causal graph $\mathcal{L}_i$ in time less than B^i.*

We remark that the same construction rules out efficient algorithms based on not only the treewidth and the vertex cover number, but also treedepth, tree-cut width and a range of other graph parameters typically used to achieve fixed-parameter tractability. However, as we will see in the rest of the section, there are natural structural properties of the super-structure which guarantee efficient causal discovery. The first such property is the maximum degree in the super-structure. It may be worth noting that it is exceedingly rare for a computational problem to be fixed-parameter tractable w.r.t. the maximum degree and yet intractable w.r.t. parameters such as the vertex cover number.

Theorem 4.17. *If the PC algorithm is provided an n-variable super-structure $\mathcal{G}$ of maximum degree k, then it will discover the skeleton $\underline{\mathcal{D}}$ of the hidden causal graph in time $2^k B^k \mathcal{O}(nk)$, where B is the size of domain.*

Proof. By Lemma 4.1, for any two independent variables X and Y such that X is not reachable from Y, the set $\mathcal{S}$ of all in-neighbors of Y in $\mathcal{D}_{XY}$ forms a d-separating set for X and Y. Hence for every pair of variables X and Y there are at most 2^k subsets of neighbors of Y in the super-structure, and testing whether a fixed subset of size $i \leq k$ forms a d-separating set takes time $\mathcal{O}(B^i)$.

Note that we only need to perform this procedure for those X and Y that are adjacent in $\mathcal{G}$. Since $\mathcal{G}$ has maximum degree k, there are at most nk such pairs. □

Next, we show that the tractability result of Theorem 4.17 can be extended to the setting where only the maximum degree of each 2-connected component is bounded. To this end, we define a new constraint-based algorithm PC*. For a super-structure $\mathcal{G}$ and two variables $X, Y \in V(\mathcal{G})$, let $\mathcal{G}_{XY}$ be the subgraph of $\mathcal{G}$ induced by all paths between X and Y in $\mathcal{G}$. PC* takes $\mathcal{G}$ (which may, in the worst case, be the complete graph) as input and proceeds similarly to the PC algorithm, with only a single modification: while looking for potential d-separating sets for X and Y, it only checks those subsets of neighbors of Y that belong to $\mathcal{G}_{XY}$.

Theorem 4.18. *Given a super-structure $\mathcal{G}$ of $\mathcal{D}$ over n variables, PC* computes the skeleton of $\mathcal{D}$ in time $2^k B^k \mathcal{O}(nk)$, where k is the maximum degree over all 2-connected components of $\mathcal{G}$.*

Proof. Let X and Y be two non-adjacent variables of $\mathcal{D}$ such that X is not reachable from Y in $\mathcal{D}$. By Lemma 4.1, X and Y admit the d-separating set $\mathcal{S}$ consisting of in-neighbors of Y in $\mathcal{D}_{XY}$. Since $\underline{\mathcal{D}}_{XY} \subseteq \mathcal{G}_{XY}$, $\mathcal{S}$ has size of at most k and it will be necessarily discovered by PC*.

Again, we only test conditional independence for those X and Y that are adjacent in $\mathcal{G}$. Since the maximum degree of any 2-connected component of $\mathcal{G}$ is k, there are at most nk such pairs contained in the same connected component, plus at most n pairs with endpoints in different 2-connected components. □

Even though the difference between the PC and PC* algorithms might seem rather subtle, the latter immediately extends the class of instances for which the skeleton can be computed efficiently.

To illustrate this, consider for instance the class $\{\mathcal{L}_k^n \mid n, k \in \mathbf{N}\}$ of causal graphs obtained from the graphs $\mathcal{L}_k$ (cf. Observation 4.4) by introducing $n-k$ new in-neighbors of X and $n-k$ new in-neighbors of Y. Assume that the super-structure $\mathcal{G}$ of the hidden causal graph $\mathcal{L}_k^n$, provided in the input, consists of the skeleton of $\mathcal{L}_k^n$ plus one extra edge XY, i.e., XY is the only edge to be eliminated. Then X and Y can be d-separated by a set $\mathcal{S}$ of size k consisting of all their neighbors inside of $\mathcal{L}_k \subseteq \mathcal{L}_k^n$. However, the PC algorithm will check all subsets of neighbors of size at most $k-1$ before it discovers $\mathcal{S}$, and there are $\Omega(n^{k-1})$ of these. By contrast, the PC* algorithm will only check subsets contained in the subgraph of $\mathcal{G}$ induced by all paths between X and Y in $\mathcal{G}$, which is precisely the skeleton of $\mathcal{L}_k$. Therefore, the PC* algorithm will not test more than 2^k subsets before discovering $\mathcal{S}$.

Corollary 4.19. *If $\mathcal{L}_k^n$ is a hidden causal graph and $\mathcal{G}$ contains the only extra edge between its pair of high-degree vertices, then the* PC *algorithm requires $\Omega(n^{k-1})$ iterations to eliminate the single superfluous edge in $\mathcal{G}$, while the* PC* *algorithm computes the skeleton in time $2^k B^k \mathcal{O}(nk)$.*

As our final contribution, we provide a new causal discovery algorithm called PC_{cut} that features fixed-parameter runtime guarantees not only on all instances where the PC and PC* algorithms do, but also on more general classes of inputs. To this end, it will be useful to recall that a bound on the size of smallest edge-cuts between each pair of variables is a sufficient condition for the existence of small d-separating sets (see Theorem 4.3). While causal graphs with this property need not be efficiently discoverable by constraint-based algorithms, this changes when dealing with a provided super-structure.

The PC_{cut} algorithm alters the original PC algorithm by testing conditional independence via subsets selected from a smallest edge-cut between the variables X, Y, instead of simply checking the neighborhood of these two variables. More precisely, we define the PC_{cut} algorithm as follows. It begins by performing the same initial two steps as the PC algorithm (see Section 4.3). Afterwards, it performs a subroutine that starts with $i := 1$ and increments this up to a value of $2k^2$, where k upper-bounds the size of a smallest

edge-cut. For each choice of i, it runs over all ordered pairs of adjacent variables, and for each such pair (X, Y) it computes an edge-cut $\mathcal{E}$ of minimum size between X and Y in the graph computed so far (but without the edge XY) that does not touch X (this can be done, e.g., via the Ford-Fulkerson algorithm where we make the edges incident to X undeletable). Let $\mathcal{S}^{\mathcal{E}}$ be the set of variables incident to the edges in $\mathcal{E}$. We now perform essentially the same procedure the PC algorithm originally performs over the neighborhood of X and Y, but on the set $\mathcal{S}^{\mathcal{E}}$; in particular, for all subsets $\mathcal{S}$ of $\mathcal{S}^{\mathcal{E}}$ of size i, we test whether $X \perp Y \mid \mathcal{S}$ and if this test succeeds we eliminate the edge XY.

Theorem 4.20. *When given a super-structure $\mathcal{G}$ of an n-variable causal graph $\mathcal{D}$ such that each pair of variables can be separated by an edge-cut of size at most k in $\mathcal{G}$, the* $\mathrm{PC}_{\mathrm{cut}}$ *algorithm will compute the skeleton $\underline{\mathcal{D}}$ of $\mathcal{D}$ in time at most $B^{\mathcal{O}(k^2)} \cdot n$.*

Proof. Assume w.l.o.g. that X is not reachable from Y in $\mathcal{D}$. It follows from the proof of Theorem 4.3 that whenever X and Y are conditionally independent in $\mathcal{D}$, some subset of $\mathcal{S}^{\mathcal{E}}$ is a d-separating set for them. Indeed, the subset can be formed by restricting $\mathcal{E}$ to $\underline{\mathcal{D}}$ and removing vertices reachable from Y in $\mathcal{D}$. While we are not aware of the edge orientations when working with $\underline{\mathcal{D}}$, the $\mathrm{PC}_{\mathrm{cut}}$ algorithm tests all subsets of $\mathcal{S}^{\mathcal{E}}$.

Crucially, $|\mathcal{S}^{\mathcal{E}}|$ is upper-bounded by $2k^2$, since the size of an edge-cut that does not touch X is upper-bounded by k^2 due to the existence of an edge-cut of size at most k between each pair of variables (see also the proof of Theorem 4.3). Therefore, at most $2^{2k^2} \leq B^{2k^2}$ sets will be tested. Testing independence with respect to each choice of $\mathcal{S}$ takes time $\mathcal{O}(B^{2k^2})$.

Let us estimate the number of edges in $\mathcal{G}$. Pick any pair X, Y of varibles that belong to the same connected component of $\mathcal{G}$. By the theorem assumption, we can choose at most k edges to disconnect X from Y, and this will increase the number of connected components of $\mathcal{G}$ by at least one. Hence, after disconnecting at most n pairs of varibles we will obtain edgeless graph. Therefore, $\mathcal{G}$ has at most nk edges, which upper-bounds the number of pairs of variables for which the conditional independence should be tested. □

We conclude this section by noting that there exist classes of inputs where $\mathrm{PC}_{\mathrm{cut}}$ will outperform both the PC and PC* algorithms by an arbitrarily large factor.

Indeed, consider the class of instances where the hidden causal graphs are the graphs $\mathcal{T}_n$ defined in the proof of Theorem 4.14. Let the super-structure $\mathcal{G}$ be the skeleton of $\mathcal{T}_n$ plus one extra edge W_nE_n. Observe that $\mathcal{G}$ admits an edge-cut of size at most 5 between each pair of variables. Indeed, in every pair other than E_n and W_n this claim holds since one vertex of the pair has degree at most 4. For E_n and W_n, the claim also holds due to the existence of the edge-cut W_nE_n, W_nN_n, W_nS_n plus the two edges of $\mathcal{T}_n^1$ adjacent to O_n. By Theorem 4.20, $\mathrm{PC}_{\mathrm{cut}}$ will compute the skeleton in time at most $B^{\mathcal{O}(1)} \cdot 2^n$ for every n. In contrast, since every d-separating set between W_3 and E_3 that lies in the neighborhoods of these two variables has size $\Omega(2^n)$, both the PC and PC* algorithms will necessarily need to test $2^{\Omega(2^n)}$ subsets before they discover such a set.

Corollary 4.21. *If $\mathcal{T}_n$ is a hidden causal graph and $\mathcal{G}$ contains the only extra edge between its pair of high-degree vertices,* PC *and* PC* *need $2^{\Omega(2^n)}$ iterations to eliminate the single extra edge, while* PC_{cut} *computes the skeleton in time $B^{\mathcal{O}(1)} \cdot 2^n$.*

4.7 Discussion

In this chapter we have studied the relationship between the graph structure of causal graphs and the efficiency of constraint-based causal discovery algorithms from various angles. We have provided a near-tight characterization of the causal graphs that admit small d-separating sets, pointed out obstacles for the PC algorithm to run efficiently, and established the fixed-parameter tractability of causal discovery exploiting properties of the super-structure.

The crucial parameter we used here is the maximum number of edge-disjoint paths between each pair of variables. Given as a promise on the hidden causal graph structure, it allows to bound the sizes of d-separating sets. Moreover, in case the super-structure of a hidden network with bounded value of this parameter is provided, we can learn hidden causal relationships efficiently. We leave as an intriguing open question for future research whether a skeleton having a large number of edge-disjoint paths between some pair of variables implies that there is an orientation with a large d-separating set between some pair of variables.

CHAPTER 5

Structural Parameters that make Clustering FPT

The problem of k-means clustering, also closely related to data completion, is another fundamental task in machine learning and data analysis, one that has been the focus of extensive empirical as well as theoretical research. In general, the aim in k-means clustering is to partition the rows in an input matrix A into k clusters and compute one center per cluster so as to minimize the within-cluster sum of squares—or, when viewed as a decision problem, achieve a within-cluster sum of squares of at most a specified target value ℓ.

In this chapter, we focus on the settings where most of the rows of A are not known or simply irrelevant, and exploit the structural properties of a graphical encoding of the missing entries. Namely, we consider two natural encodings—incidence and primal graphs—and provide three novel fixed-parameter algorithms: (i) parameterized by treewidth of the incidence graph for bounded-domain data, (ii) parameterized by treewidth of the primal graph for real-valued data and (iii) parameterized by the slim tree-cut width of the incidence graph for real-valued data.

5.1 Related Work

Depending on the setting, the matrix A could be real-valued [Llo82, MDRF20, FGS21] or contain integers from a bounded domain; for instance, Fomin et al. [FGP18] studied the case where A was binary, and Ganian et al. [GKOS18] the bounded-domain case where $\ell = 0$.

It is well known that k-means clustering is NP-complete, even when restricted to $k = 2$ clusters [DFK+04, ADHP09]. Parameterized complexity paradigm provides a more fine-grained look into the complexity of the problem [CAdMRR18, FGP18].

Apart from classical clustering problems, one direction that has been gaining traction in recent years is the study of clustering in settings where some of the data is not known or simply irrelevant—a task that essentially combines data completion with clustering [WLH+19]. Indeed, several authors have studied the parameterized complexity of various clustering problems for data with missing entries [GKOS18, KFN20, KFN21, EGK+21], and Eiben et al. [EFG+21] obtained a fixed-parameter approximation algorithm specifically for (k-)Means Clustering with Missing Entries (MCME).

While their approaches and techniques differ, all of these works target the case where nearly all entries are known, i.e., where the unknown entries only occur in a precisely defined "sparse" way. At the same time, very little is known about the complexity of MCME when the number of unknown entries is large; such situations are of practical relevance in, e.g., recommender systems and predictive analytics. For example, in the classical Netflix Prize challenge where the task was to predict user ratings for movies based on previous ratings, only about 1% of the user-movie pairs were originally supplied with a rating.[3] The central mission of this section is to push the boundaries of tractability for MCME to instances where *known* entries are sparse.

5.2 Our Contribution

A natural approach for handling instances of MCME with many missing entries would be to invert the parameterizations that have been developed for tackling instances where almost all entries are known. For instance, Eiben et al. [EGK+21] and Ganian et al. [GKOS18] obtained several fixed-parameter clustering algorithms by using a parameter called the *covering number*, which is the minimum number of rows and columns needed to cover all the unknown entries. Equivalently, if we use an auxiliary binary matrix W (the *mask*) to specify which entries of A are relevant/known[4], the covering number is simply the minimum number of rows and columns needed to cover all the 0's in W. It would be tempting to instead parameterize by the number of rows and columns needed to cover all the 1's in W; such an "inverse covering number" would lead to fixed-parameter algorithms targeting instances where all known entries occur in a few columns and rows.

However, we show that in our setting it is in fact possible to do much better than that. Our approach is based on using the *incidence graph* representation of W, which is the graph containing one vertex for each column and for each row of W and where an edge connects row a with column b if and only if $W[a, b] = 1$. Observe that the inverse covering number considered in the previous paragraph would simply be the size of a minimum vertex cover in W, while the covering number is the size of a minimum vertex cover in the complement of W. It is worth noting that in graph-theoretic settings, the size of a minimum vertex cover is considered a highly restrictive parameterization, one that is used primarily when more desirable parameterizations fail. In contrast, the by far

[3] The dataset is available at `https://www.kaggle.com/netflix-inc/netflix-prize-data`.

[4] The mask W has the same size as the matrix A. For each relevant/known entry of A, the corresponding entry of W is 1, and rest of the entries of W are 0's.

most prominent parameter used for graphs is treewidth, which has found ubiquitous applications throughout computer science and which is less restrictive than the size of a minimum vertex cover. As our first result, we use the treewidth of the incidence graph of the mask W (the *incidence treewidth*) to prove:

Result 1: Bounded-domain MCME is fixed-parameter tractable when parameterized by the incidence treewidth of the mask.

Result 1 is noteworthy not only due to the use of treewidth instead of the inverse covering number, but also because incidence treewidth is the *only* parameter required to achieve tractability. In particular, unlike previous algorithms for clustering incomplete (bounded-domain or real-valued) data by Eiben et al. [EFG+21, EGK+21], Ganian et al. [GKOS18], and Koana et al. [KFN20, KFN21], Result 1 can also be applied to instances where the number k of clusters is large. As we will see later, we will be able to retain this benefit in all our algorithms.

Unfortunately, an extension of Result 1 towards real-valued instances seems difficult at this point. Indeed, a fixed-parameter algorithm for real-valued MCME parameterized by the incidence treewidth of the mask would imply, as a special case, fixed-parameter tractability parameterized by the number d of columns. However, the existence of a fixed-parameter algorithm even for k-Means (corresponding to the restriction of MCME where all entries are known) when parameterized by d is a long-standing open problem dating back to Inaba et al.'s celebrated XP algorithm parameterized by $k + d$ [IKI94].

Instead, we show that one can obtain fixed-parameter algorithms for real-valued MCME by using the treewidth of a different graph representation of W. In particular, we use the *primal graph*—a well-established counterpart to the incidence graph representation in settings such as Boolean satisfiability [SS09], constraint satisfaction [SS10] or integer programming [GO19]. In our context, the primal graph of W contains a vertex for each row of W, with edges connecting pairs of rows that share at least one coordinate which is known/relevant according to the mask, and the primal treewidth is simply the treewidth of this graph.

Result 2: Real-valued MCME is fixed-parameter tractable when parameterized by the primal treewidth of the mask.

Note that, due to the nature of the primal graph representation, the primal treewidth is a more restrictive parameter than the incidence treewidth (in the sense of being bounded on fewer classes of instances). In fact, the primal treewidth of a mask may in certain cases be unboundedly large even when its incidence graph is just a tree. It would hence be useful to also have an algorithm for real-valued MCME that can exploit properties of the incidence graph, even for properties that are more restrictive than incidence treewidth.

However, the same obstacle towards extending Result 1 to the real-valued setting also applies to all traditional structural parameters that are, intuitively, based on small vertex separators in the graph—including pathwidth [RS83], treedepth [NdM12], the feedback vertex number (e.g., [MNN20]) and even the vertex cover number (e.g., [GN19, FPR+18]),

which is the most restrictive vertex separator based parameter. In spite of this, as our third and final result we identify a parameter of the incidence graph that yields fixed-parameter tractability of real-valued MCME: the slim tree-cut width.

Result 3: Real-valued MCME is fixed-parameter tractable when parameterized by the slim tree-cut width of the incidence graph of the mask.

We note that the parameters used in Results 2 and 3 are orthogonal and the gap between them may be arbitrarily large. Because of this, both results are incomparable and give rise to different tractable classes for the problem.

5.3 Notation and Problem Definition

Recall that for an integer i, we denote by $[i]$ the set $\{1, 2, \ldots, i\}$ and $[i]_0 = [i] \cup \{0\}$. In contrast, if p is an equivalence relation on domain X, we denote by $[p]$ the set of equivalence classes of p and by $|[p]|$ the cardinality of $[p]$, i.e., the number of equivalence classes of p. For every $x \in X$, $[x]_p \in [p]$ is the equivalence class containing x.

Let A be a matrix with n rows and d columns. We use V_A and C_A to denote the set of row and column indices of A, i.e., $V_A = [n]$ and $C_A = [d]$. Let $M_A = \max_{v \in V_A} \max_{c \in C_A} |A[v, c]|$. For two matrices A, B, we denote by $A - B$ the entry-wise subtraction of the two matrices, i.e., $(A - B)[i, j] = A[i, j] - B[i, j]$, and similarly by $A \circ B$ the entry-wise product of the two matrices, i.e., $(A \circ B)[i, j] = A[i, j] \cdot B[i, j]$. For a matrix $A \in \mathbb{R}^{n \times d}$ we denote its squared Frobenius norm by $||A||_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{d} A[i, j]^2$. We can now formalize our problems of interest (abbreviated as MCME):

Means Clustering with Missing Entries	
Input:	Matrix $A \in D^{n \times d}$ over a domain $D \subseteq \mathbb{R}$, binary matrix W (the *mask*), integers k and ℓ.
Task:	Determine if there exists a matrix B over D containing at most k distinct rows such that $\|W \circ (A - B)\|_F^2 \leq \ell$.

We distinguish between Bounded-domain MCME and Real-valued MCME depending on whether $D = [z]_0$ for some fixed integer z, or D is the set of reals. In particular, for $z = 1$ this coincides with Binary k-Means Clustering [FGP18]. In this case, minimizing the Frobenius norm also minimizes the sum of Hamming distances for each row to its cluster center. Using the Frobenius norm also for $z > 1$ is not only consistent with the continuous case, but also allows us to capture further applications, e.g., when the data matrix to cluster is composed of user ratings each taking a small numeric value.

While we state MCME as a decision problem for complexity-theoretic reasons, every algorithm presented here is constructive and can output a solution matrix B if one exists.

In this formulation, the distinct rows of B are the centers of the sought-after clusters while ℓ is the target upper-bound on the sum of squares. For $V' \subseteq V_A$, we call a mapping

$\varphi : V' \to [k]$ a *cluster-assignment* w.r.t. B if for every $v_1, v_2 \in V'$ such that $\varphi(v_1) = \varphi(v_2)$, it holds that $B[v_1] = B[v_2]$. Notice that the existence of a cluster assignment w.r.t. a matrix B' implies that B' has at most k distinct rows. The *clusters* associated with φ are then sets of the form $\{v \in V \mid \varphi(v) = i\}$ for some $i \in [k]$.

In our algorithms, we often construct the solution matrix dynamically by "merging" smaller matrices; we formalize this procedure below. Let $\{B_i \mid i \in [m]\}$ be a set of matrices, where each B_i has row labels V_i, column labels C_i and is associated with a cluster-assignment $\varphi_i : V_i \to [k]$ and the following two *consistency conditions* hold:

- $\varphi_i(v) = \varphi_j(v)$ for every $v \in V_i \cap V_j$, $i, j \in [m]$,
- if $\varphi_i(v_i) = \varphi_j(v_j)$ for some $v_i \in V_i$ and $v_j \in V_j$, then $B_i[v_i, c] = B_j[v_j, c]$ for every $c \in C_i \cap C_j$.

Let $\varphi = \bigcup_{i \in [m]} \varphi_i$. We define the *composition* of (B_i, φ_i), $i \in [m]$, to be the matrix B^* with row labels $V = \bigcup_{i \in [m]} V_i$, column labels $C = \bigcup_{i \in [m]} C_i$ and entries as follows. For every $v \in V$ and $c \in C$, pick any $i \in [m]$ such that $c \in C_i$ and there exists $v_i \in V_i$ with $\varphi(v_i) = \varphi(v)$. We set $B^*[v, c_i] = B_i[v_i, c_i]$. If there is no such $i \in [m]$, we simply set $B^*[v, c]$ to some arbitrary but uniform default value—in our case, we will always use 0. Observe that if both consistency conditions hold, then both B^* and φ are well-defined and that φ is a cluster-assignment w.r.t. B^*.

Claim 5.1. *B^* is well-defined and admits the cluster-assignment φ.*

Proof. To see that B^* is well-defined, pick $v \in V$ and $c \in C_i \cap C_j$. If there exist $v_i \in V_i$ and $v_j \in V_j$ such that $\varphi(v_i) = \varphi(v)$ and $\varphi(v_j) = \varphi(v)$, then $\varphi(v_i) = \varphi(v_j)$, so $B_i[v_i, c] = B_j[v_j, c]$ by the consistency conditions. Moreover, φ is the cluster-assignment w.r.t. B^*. Indeed, consider $u, v \in V$ with $\varphi(u) = \varphi(v)$ and $c \in C$. If $c \in C_i$ for some $i \in [m]$ such that there exists $v_i \in V_i$ with $\varphi(v_i) = \varphi(v) = \varphi(u)$, then $B^*[v, c] = B_i[v_i, c] = B^*[u, c]$. If there is no such $i \in [m]$, we have $B^*[v, c] = 0 = B^*[u, c]$. □

Observe that if $C_i \cap C_j = \emptyset$ for every $i \neq j$, we can skip the second consistency condition: in this case it just requires $B_i[v] = B_i[u]$ whenever $\varphi_i(v) = \varphi_i(u)$, which holds as φ_i is the cluster-assignment w.r.t. B_i.

Sparse Matrices. It follows from definition of MCME that entries of the input matrix A and the target matrix B are only relevant when the respective entry of the mask W is set to "1". We thus assume that only those entries are stored explicitly; in particular, the size of the input is therefore linear in the number of non-zero entries of W. All the structural parameters in our considerations automatically imply that the mask W is sparse, in the sense that it has only $\mathcal{O}_k(n + d)$ non-zero entries, where the constant under $\mathcal{O}_k(\cdot)$ depends on the respective parameter k. All algorithms we provide except one are linear-time in the number of non-zero entries of the mask, and the above allows us to

state their running times in the form $(n+d)\cdot f(k)$, where f is a certain function of the respective parameter k.

5.4 Solving Bounded-Domain MCME by Exploiting the Mask

Since Bounded-domain MCME is NP-complete [DFK+04, ADHP09], it is natural to attempt and circumvent this lower bound by exploiting structural measures of the inputs to obtain fixed-parameter tractability. Eiben et al. [EFG+21] recently obtained a fixed-parameter approximation algorithm for Bounded-domain requiring the simultaneous parameterization by three measures: the number of clusters, the desired approximation ratio, and a technical measure that captures the sparsity of the missing data (i.e., the "0" entries in W).

In this section, we will present a fixed-parameter algorithm for Bounded-domain MCME which is aimed at the complementary case where the relevant data is sparse. To formalize this, we consider the *incidence graph* representation G_I of W which is defined as follows: $V(G_I) = V_W \cup C_W$, and $E(G_I) = \{ab \mid a \in V_W, b \in C_W, W[a,b] = 1\}$. Figure 5.1 later on can also be used as an example of the representation.

Intuitively, each "1" entry in W will correspond to an edge in G_I, and hence "structurally sparse" incidence graphs correspond to settings where most data is unknown. There is a well-studied hierarchy of structural graph parameters which measure, in a certain sense, the sparsity of graphs (see, e.g., Figure 1 in [BJK13]). Treewidth will be our parameter of choice here, as the best known and also most general parameter in this hierarchy. However, it is worth noting that the use of the incidence graph may provide a new perspective on previous algorithms for clustering problems—for instance, the so-called *covering number* parameter [GKOS18, EGK+21] is simply the vertex cover number of the complement of G_I.

The main goal of this section is to establish the fixed-parameter tractability of MCME parameterized by $\mathrm{tw}(G_I)$. As a first step towards this goal, we obtain a dynamic programming algorithm that handles the simpler case where k is also part of the parameterization.

Theorem 5.2. *Bounded-Domain MCME is fixed-parameter tractable when parameterized by $k + \mathrm{tw}(G_I)$.*

Proof. We begin by applying the recent 2-approximation algorithm for treewidth designed by Korhonen [Kor22] to compute a nice tree decomposition (T, χ) of G_I of width $q \leq 2\,\mathrm{tw}(G_I) + 1$ in time $2^{\mathcal{O}(\mathrm{tw}(G_I))}(n+d)$. Let r be the root of T. Given a node t of T, let V_t and C_t be the sets of vectors and coordinates in $\chi(t)$, respectively. Moreover, let $C_t^{\downarrow}$ and $V_t^{\downarrow}$ be the restrictions of $\chi_t^{\downarrow}$ to the sets of coordinates and vectors.

To prove the theorem, we will design a leaf-to-root dynamic programming algorithm which will compute and store a set of records at each node t of T, whereas once we

ascertain the records for r we will have the information required to output a correct answer. Intuitively, the records will store the cluster centers restricted to the coordinates that appear in the bag, the partition of the vectors in the bag into clusters and the sum of minimum distances from vectors in $V_t^{\downarrow}$ to the cluster centers along the coordinates in $C_t^{\downarrow}$.

Formally, the records will have the following structure. We call a pair $(\texttt{cent}, \texttt{part})$ a *snapshot* in t if the following holds:

- $\texttt{cent} : [k] \times C_t \to D$,
- $\texttt{part} : V_t \to [k]$.

Let $S(t)$ be the set of all snapshots of t. The record $\mathcal{R}_t$ of t is then a mapping from $S(t)$ to the set $\mathbb{R}^+$ of non-negative reals. Observe that $|S(t)| \leq |D|^{k(q+1)}k^{q+1}$.

To introduce the semantics of our records, let $\mathcal{B}_t$ be the set of all matrices with row labels $V_t^{\downarrow}$, column labels $C_t^{\downarrow}$ and entries of domain D. Let B_t be a matrix in $\mathcal{B}_t$. We define the *partial weighted distance* from B_t to A in t as follows:

$$\texttt{pwd}(B_t, t) = \sum_{v \in V_t^{\downarrow}} \sum_{c \in C_t^{\downarrow}} W[v, c] \cdot (A[v, c] - B_t[v, c])^2.$$

For $B_t \in \mathcal{B}_t$, we say that $(\texttt{cent}, \texttt{part}) \in S(t)$ is the *snapshot of* B_t in t if there is a cluster-assignment φ w.r.t. B_t such that the following conditions hold:

- $\texttt{part} = \varphi|_{V_t}$,
- for every $c \in C_t$ and $v \in V_t^{\downarrow}$, $B_t[v, c] = \texttt{cent}[\varphi(v), c]$.

Recall that the existence of a cluster-assignment implies, in particular, that B_t has at most k distinct rows. We are now ready to define the record $\mathcal{R}_t$. For each snapshot $(\texttt{cent}, \texttt{part}) \in S(t)$, we set $\mathcal{R}_t(\texttt{cent}, \texttt{part}) = \tau$ if there exists $B_t \in \mathcal{B}_t$ such that:

- $(\texttt{cent}, \texttt{part})$ is the snapshot of B_t in t,
- $\texttt{pwd}(B_t, t) = \tau$, and
- $\forall B'_t \in \mathcal{B}_t$ such that $(\texttt{cent}, \texttt{part})$ is the snapshot of B'_t, $\texttt{pwd}(B'_t, t) \geq \tau$.

In this case we say that B_t *witnesses* $\mathcal{R}_t(\texttt{cent}, \texttt{part}) = \tau$.

Recall that for the root r, we assume $\chi(r) = \emptyset$. Hence $S(r)$ contains only one element $(\emptyset, \emptyset)$, and $\mathcal{R}_r(\emptyset, \emptyset)$ is equal to the minimum value of $\|W \circ (A - B)\|_F^2$ that can be achieved by any matrix B with entries of domain D containing at most k distinct rows. In other words, the instance is a YES-instance if and only if $\mathcal{R}_r(\emptyset, \emptyset) \leq \ell$. To prove the theorem,

it now suffices to show that the records can be computed in a leaf-to-root fashion by proceeding along the nodes of T. We distinguish the following cases:

t is a leaf node. Let $\chi(t) = \{v\}$ where v is a vector. By definition, $S(t) = \{(\emptyset, \texttt{part}) | \texttt{part} : [1] \to [k]\}$ and $\mathcal{R}_t(\emptyset, \texttt{part}) = 0$ for every $(\emptyset, \texttt{part}) \in S(t)$ as there are no coordinates to sum over. Let $\chi(t) = \{c\}$ for some coordinate c. Then $S(t) = \{(\texttt{cent}, \emptyset) | \texttt{cent} : [k] \times [1] \to D\}$ and $\mathcal{R}_t(\texttt{cent}, \emptyset) = 0$ for each $(\texttt{cent}, \emptyset) \in S(t)$.

t is a forget node. Let t' be the child of t in T and $\chi(t) = \chi(t') \setminus \{v\}$ for some vector v. We set $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) := \min_{i \in [k]} \mathcal{R}_{t'}(\texttt{cent}, \texttt{part} \cup (v, i))$ for each $(\texttt{cent}, \texttt{part}) \in S(t)$.

For correctness, it will be useful to observe that $\mathcal{B}_t = \mathcal{B}_{t'}$. If $\mathcal{R}_t(\texttt{cent}, \texttt{part}) = \tau$, then there exists a matrix B_t which witnesses this. But then B_t also admits a snapshot $(\texttt{cent}, \texttt{part} \cup (v, i))$ at t' for some $i \in [k]$ and witnesses $\mathcal{R}_{t'}(\texttt{cent}, \texttt{part} \cup (v, i)) \leq \tau$. So in our algorithm $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) \leq \mathcal{R}_{t'}(\texttt{cent}, \texttt{part} \cup (v, i)) \leq \tau$. Conversely, if $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) = \tau$, then there exists a snapshot $(\texttt{cent}, \texttt{part} \cup (v, i))$ at t' for some $i \in [k]$ such that $\mathcal{R}_{t'}(\texttt{cent}, \texttt{part} \cup (v, i)) = \tau$. Existence of a matrix witnessing the value of $\mathcal{R}_{t'}(\texttt{cent}, \texttt{part} \cup (v, i))$ implies that $\mathcal{R}_t(\texttt{cent}, \texttt{part}) \leq \tau$. Hence, we can correctly set $\mathcal{R}_t = \mathcal{R}^0_t$.

If $\chi(t) = \chi(t') \setminus \{c\}$ for some coordinate c, we set:

$$\mathcal{R}_t(\texttt{cent}, \texttt{part}) = \min\{\mathcal{R}_{t'}(\texttt{cent}', \texttt{part}) | \texttt{cent} \text{ is obtained from } \texttt{cent}' \text{ by deleting } c\}.$$

Correctness can be argued similarly to the case of a forgotten vector.

t is an introduce node (introducing a vector). Let t' be the child of t in T and let $\chi(t) = \chi(t') \cup \{v_0\}$ for some vector v_0. Fix a snapshot $(\texttt{cent}, \texttt{part})$ in $S(t)$, let $i = \texttt{part}(v_0)$ and $\texttt{part}' = \texttt{part} \setminus (v_0, i)$. We will denote by Δ_0 the sum of distances from v_0 to the i-th cluster center along the coordinates in C_t, i.e., $\Delta_0 = \sum_{c \in C_t} W[v_0, c] \cdot (A[v_0, c] - \texttt{cent}[i, c])^2$. We set $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) = \mathcal{R}_{t'}(\texttt{cent}, \texttt{part}') + \Delta_0$.

For correctness, assume that $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) = \tau = \mathcal{R}_{t'}(\texttt{cent}, \texttt{part}') + \Delta_0$. Construct a matrix B_t from the witness $B_{t'}$ of $\mathcal{R}_{t'}(\texttt{cent}, \texttt{part}')$ with a cluster-assignment φ' by adding a new row with a label v_0 as follows. If $i = \varphi'(v)$ for some $v \in V^{\downarrow}_{t'}$, we set $B_t[v_0] := B_t[v]$. Otherwise we define $B_t[v_0, c] := \texttt{cent}[i, c]$ for $c \in C_t$ and $B_t[v_0, c] := 0$ in the rest of coordinates c. Note that in both cases $B_t[v_0, c] = \texttt{cent}[i, c]$ for every $c \in C_t$: in the second case it follows from the definition, while in the first one we have $B_t[v_0, c] := B_t[v, c] = B_{t'}[v, c] = \texttt{cent}[\varphi'(v), c] = \texttt{cent}[i, c]$. The matrix B_t with a cluster-assignment $\varphi = \varphi' \cup (v_0, i)$ has a snapshot $(\texttt{cent}, \texttt{part})$ in t and $\texttt{pwd}(B_t, t) = \sum_{v \in V^{\downarrow}_t} \sum_{c \in C^{\downarrow}_t} W[v, c] \cdot (A[v, c] - B_t[v, c])^2 = \texttt{pwd}(B_{t'}, t') + \sum_{c \in C^{\downarrow}_t} W[v_0, c] \cdot (A[v_0, c] - B_t[v_0, c])^2$. As $\texttt{pwd}(B_{t'}, t') = \mathcal{R}_{t'}(\texttt{cent}, \texttt{part}')$, $W[v_0, c] = 0$ for every forgotten coordinate c and $B_t[v_0, c] := \texttt{cent}[i, c]$ for every $c \in C_t$, we have $\texttt{pwd}(B_t, t) = \mathcal{R}_{t'}(\texttt{cent}, \texttt{part}') + \Delta_0 = \tau$. So B_t witnesses that $\mathcal{R}_t(\texttt{cent}, \texttt{part}) \leq \tau$.

For the converse, assume that $\mathcal{R}_t(\texttt{cent}, \texttt{part}) = \tau$. Then there exists a matrix B_t in $\mathcal{B}_t$ admitting the snapshot $(\texttt{cent}, \texttt{part})$ in t such that $\texttt{pwd}(B_t, t) = \tau$. Let φ be the

corresponding cluster-assignment w.r.t. B_t. We construct $B_{t'}$ from B_t by deletion of the row v_0. Then $B_{t'}$ with cluster-assignment $\varphi' = \varphi \setminus \{(v_0, i)\}$ has a snapshot $(\texttt{cent}, \texttt{part}')$ in t and witnesses that $\mathcal{R}_{t'}(\texttt{cent}, \texttt{part}') \leq \texttt{pwd}(B_{t'}, t') = \tau - \Delta_0$. Therefore in our algorithm $\mathcal{R}^0_t \leq \mathcal{R}_{t'}(\texttt{cent}, \texttt{part}') + \Delta_0 \leq \tau$. Hence, we can correctly set $\mathcal{R}_t = \mathcal{R}^0_t$.

t is an introduce node (introducing a coordinate). Let t' be the child of t in T and let $\chi(t) = \chi(t') \cup \{c_0\}$ for some coordinate c_0. Fix a snapshot $(\texttt{cent}, \texttt{part})$ in $S(t)$, we wil denote by Δ_0 the sum of distances from $v \in V_t^{\downarrow}$ to the corresponding cluster center along the coordinate c_0, i.e., $\Delta_0 = \sum_{v \in V_t} W[v, c_0] \cdot (A[v, c_0] - \texttt{cent}[\texttt{part}(v), c_0])^2$. Let $\texttt{cent}'$ be the restriction of $\texttt{cent}$ to all the coordinates except c_0. We set $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) = \mathcal{R}_{t'}(\texttt{cent}', \texttt{part}) + \Delta_0$.

For correctness, assume that $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) = \tau = \mathcal{R}_{t'}(\texttt{cent}', \texttt{part}) + \Delta_0$. We construct a matrix B_t from a witness $B_{t'}$ of $\mathcal{R}_{t'}(\texttt{cent}', \texttt{part})$ with cluster-assignment φ' by adding a column c_0. For every $v \in V_t^{\downarrow}$, we set $B_t[v, c] = B_{t'}[v, c]$ for all $c \in C_{t'}^{\downarrow}$ and $B_t[v, c_0] = \texttt{cent}[\varphi'(v), c_0]$. Then B_t with the same cluster-assignment φ' has a snapshot $(\texttt{cent}, \texttt{part})$ in t and $\texttt{pwd}(B_t, t) = \texttt{pwd}(B_{t'}, t') + \Delta_0 = \mathcal{R}_{t'}(\texttt{cent}', \texttt{part}) + \Delta_0 = \tau$. Therefore B_t witnesses that $\mathcal{R}_t(\texttt{cent}, \texttt{part}) \leq \tau$. Conversely, assume that $\mathcal{R}_t(\texttt{cent}, \texttt{part}) = \tau$. Then there exists a matrix B_t in $\mathcal{B}_t$ with the snapshot $(\texttt{cent}, \texttt{part})$ in t such that $\texttt{pwd}(B_t, t) = \tau$. Let $B_{t'}$ be obtained from B_t by deletion of the column with the label c_0. Then $B_{t'}$ witnesses $\mathcal{R}_{t'}(\texttt{cent}', \texttt{part}) \leq \tau - \Delta_0$, so in our algorithm $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) = \mathcal{R}_{t'}(\texttt{cent}', \texttt{part}) + \Delta_0 \leq \tau$. Hence, we can correctly set $\mathcal{R}_t = \mathcal{R}^0_t$.

t is a join node. Let t_1, t_2 be the two children of t in T, recall that $\chi(t_1) = \chi(t_2) = \chi(t)$ and $\chi(t_1)^{\downarrow} \cap \chi(t_2)^{\downarrow} = \chi(t)$. For every $(\texttt{cent}, \texttt{part})$ in $S(t)$ we set $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) = \mathcal{R}_{t_1}(\texttt{cent}, \texttt{part}) + \mathcal{R}_{t_2}(\texttt{cent}, \texttt{part}) - \texttt{doublecount}$, where $\texttt{doublecount}$ is equal to $\sum_{v \in V_t} \sum_{c \in C_t} W[v, c] \cdot (A[v, c] - \texttt{cent}[\texttt{part}(v), c])^2$.

For correctness, assume that $\mathcal{R}^0_t(\texttt{cent}, \texttt{part}) = \tau = \tau_1 + \tau_2 - \texttt{doublecount}$, where $\tau_1 = \mathcal{R}_{t_1}(\texttt{cent}, \texttt{part})$, $\tau_2 = \mathcal{R}_{t_2}(\texttt{cent}, \texttt{part})$. Let B_i with the cluster-assignment φ_i be the witness of $\mathcal{R}_{t_i}(\texttt{cent}, \texttt{part}) = \tau_i$, $i = 1, 2$. We obtain a matrix B_t with cluster-assignment $\varphi = \varphi_1 \cup \varphi_2$ as a composition of (B_1, φ_1) and (B_2, φ_2). To check the consistency conditions, observe that the sets of common row and column labels of B_1 and B_2 are V_t and C_t respectively. Recall that $\varphi_1|_{V_t} = \texttt{part} = \varphi_2|_{V_t}$. Moreover, if $v_1 \in V_1$ and $v_2 \in V_2$ are such that $\varphi_1(v_1) = \varphi_2(v_2)$, then we have $B_1[v_1, c] = \texttt{cent}[\varphi_1(v_1), c] = \texttt{cent}[\varphi_2(v_2), c] = B_2[v_2, c]$ for every $c \in C_t$.

Note that for every $v \in V_t$, $\varphi(v) = \varphi_1(v) = \texttt{part}$. Pick $c \in C_t$ and $v \in V_t^{\downarrow}$, then $v \in V_{t_i}^{\downarrow}$ for some $i \in \{1, 2\}$ and so $B_t[v, c] = B_i[v, c] = \texttt{cent}[\varphi_i(v), c] = \texttt{cent}[\varphi(v), c]$. Therefore $(\texttt{cent}, \texttt{part})$ is a snapshot of B_t in t. Recall that $\texttt{pwd}(B_t, t) = \sum_{v \in V_t^{\downarrow}} \sum_{c \in C_t^{\downarrow}} W[v, c] \cdot (A[v, c] - B_t[v, c])^2$. Here $W[v, c] = 0$ for every $v \in V_{t_i}^{\downarrow}$ and $c \notin C_{t_i}^{\downarrow}$, $i = 1, 2$. So $\texttt{pwd}(B_t, t) = \sum_{v \in V_{t_1}^{\downarrow}} \sum_{c \in C_{t_1}^{\downarrow}} W[v, c] \cdot (A[v, c] - B_t[v, c])^2 + \sum_{v \in V_{t_2}^{\downarrow}} \sum_{c \in C_{t_2}^{\downarrow}} W[v, c] \cdot (A[v, c] - B_t[v, c])^2 - \sum_{v \in V_t} \sum_{c \in C_t} W[v, c] \cdot (A[v, c] - B_t[v, c])^2 = \texttt{pwd}(B_1, t_1) + \texttt{pwd}(B_2, t_2) - \texttt{doublecount} = \tau_1 + \tau_2 - \texttt{doublecount} = \tau$. Hence B_t witnesses $\mathcal{R}_t(\texttt{cent}, \texttt{part}) \leq \tau$.

For the converse, assume that $\mathcal{R}_t(\texttt{cent},\texttt{part}) = \tau$ and B_t is a matrix witnessing this. Let B_i be the restriction of B_t to rows $V_{t_i}^{\downarrow}$ and columns $C_{t_i}^{\downarrow}$, $i = 1, 2$. Then B_1 and B_2 have a snapshot $(\texttt{cent},\texttt{part})$, so $\mathcal{R}_t^0(\texttt{cent},\texttt{part}) \leq \mathcal{R}_{t_1}(\texttt{cent},\texttt{part}) + \mathcal{R}_{t_2}(\texttt{cent},\texttt{part}) - \texttt{doublecount} \leq \texttt{pwd}(B_1) + \texttt{pwd}(B_2) - \texttt{doublecount} = \texttt{pwd}(B) = \tau$. Hence the resulting record $\mathcal{R}_t = \mathcal{R}_t^0$ is correct, which concludes the correctness proof of the algorithm.

To bound the runtime of the algorithm, observe that at each node t we compute the record $\mathcal{R}_t$ for $|S(t)| \leq |D|^{kq}k^q = |D|^{\mathcal{O}(k\cdot\mathrm{tw}(G_I))}$ entries, where each entry is computed in time at most quadratic in $\mathrm{tw}(G_I)$. Since the tree decomposition is nice and has only $\mathcal{O}(n+d)$ nodes, the runtime of the algorithm is upper-bounded by $(n+d)\cdot|D|^{\mathcal{O}(k\cdot\mathrm{tw}(G_I))}$. □

Towards proving fixed-parameter tractability without involving k, we consider the case where k is significantly larger than $\mathrm{tw}(G_I)$. We prove that this case always admits a "perfect" solution, which can also be computed efficiently.

Lemma 5.3. *If $k \geq (\mathrm{tw}(G_I)+1)\cdot|D|^{\mathrm{tw}(G_I)+1}$, then there exists a matrix B over domain D containing at most k distinct rows such that $\|W \circ (A-B)\|_F^2 = 0$. Moreover, given a nice tree decomposition of width $q-1$ such that $k \geq q\cdot|D|^q$, such a matrix B can be computed in time $(n+d)\cdot|D|^{\mathcal{O}(q)}$.*

Proof. We will prove the second claim of the lemma, from which the first one follows. To this end, let us assume that we are given a nice tree decomposition $\mathcal{T} = (T,\chi)$ of G_I of width $q-1$ and $k \geq q\cdot|D|^q$. We denote the root of T by r and define C_t, V_t, $C_t^{\downarrow}$ and $V_t^{\downarrow}$ analogously to the proof of Theorem 5.2. We will explicitly construct $q\cdot|D|^q$ cluster centers yielding a zero sum of squares, so that after processing the node t, each of the centers is defined on coordinates $c \in C_t^{\downarrow}$. The remaining clusters will be empty, thus for the rest of the proof we assume $k = q\cdot|D|^q$.

Formally, we will attach to every node t the pair (R_t,α_t) where R_t is the matrix with the row labels $[k]$ and the column labels $C_t^{\downarrow}$, and α_t is a mapping from $V_t^{\downarrow}$ to $[k]$. Intuitively, R_t stores the restrictions of cluster centers to processed coordinates, while α_t assigns a cluster index to every row label $v \in V_t^{\downarrow}$. Throughout the dynamic programming procedure, we will maintain the following invariants:

1. $\alpha_t(u) \neq \alpha_t(v)$ for any two distinct u and v from V_t,
2. for every $\omega \in D^{C_t}$, there are precisely $\frac{k}{|D|^{|C_t|}}$ indices $i \in [k]$ such that $R_t[i][C_t] = \omega$, where $R_t[i][C_t]$ is the restriction of $R_t[i]$ to the coordinate set C_t,
3. $\sum_{v\in V_t^{\downarrow}}\sum_{c\in C_t^{\downarrow}} W[v,c]\cdot(A[v,c] - R_t[\alpha_t(v),c])^2 = 0$.

Assume that the invariants hold in the root r, let us define the matrix B by setting $B[v] = R_r[\alpha_r(v)]$. Then B has at most k distinct rows and the third invariant yields that

$\sum_{v\in V_A}\sum_{c\in C_A} W[v,c]\cdot(A[v,c]-B[v,c])^2 = 0$. In particular, for $q = \mathrm{tw}(G_I)+1$ we get the statement of the lemma. Hence it is sufficient to construct (R_t, α_t) in a leaf-to-root fashion, for each of the following cases.

t is a leaf node. If $\chi(t) = \{v_0\}$ for some row label v_0, we set $\alpha_t(v_0) = 1$ and $R_t[i] = \emptyset$ for every $i \in [k]$. Otherwise, $\chi(t) = \{c\}$ for a coordinate c. Then α_t should be the empty mapping, for every $i \in [k]$ we set $R_t[i,c]$ equal to some $\gamma \in D$ so that every $\gamma \in D$ appears in R_t precisely $\frac{k}{|D|}$ times. It is easy to see that all the invariants are satisfied.

t is a forget node. Let t' be the child of t in T, we keep $R_t = R_{t'}$ and $\alpha_t = \alpha_{t'}$. The first and third invariants are obviously preserved. In case $\chi(t) = \chi(t') \setminus \{c_0\}$ for a coordinate c_0, we have $C_{t'} = C_t \cup \{c_0\}$. By the second invariant for t', for every $\gamma \in D$ and $\omega \in D^{C_t}$ the number of $i \in [k]$ with $R_{t'}[i][C_t] = \omega$ and $R_{t'}[i][c_0] = \gamma$ is $\frac{k}{|D|^{|C_{t'}|}}$. Therefore the number of $i \in [k]$ with $R_t[i][C_t] = \omega$ is precisely $|D|\cdot\frac{k}{|D|^{|C_{t'}|}} = \frac{k}{|D|^{|C_t|}}$.

t is an introduce node (introducing a vector). Let t' be the child of t in T and $\chi(t) = \chi(t') \cup \{v_0\}$ for some row label v_0. We then set $R_t = R_{t'}$ and $\alpha_t = \alpha_{t'} \cup (v_0, i_0)$ where $i_0 \in [k]$ is such that $R_{t'}[i_0][C_t] = A[v_0][C_t]$ and $\alpha_{t'}(v) \neq i_0$ for any $v \in V_{t'}$. Such an index always exists: by the second invariant for t' there are $\frac{k}{|D|^{|C_t|}} \geq \frac{k}{|D|^q} = q > |V_{t'}|$ indices i such that $R_{t'}[i][C_t] = A[v_0][C_t]$, so at least one of them is not equal to any $\alpha_{t'}(v)$, $v \in V_{t'}$. As the third invariant holds in t', we have $\sum_{v\in V_t^\downarrow}\sum_{c\in C_t^\downarrow} W[v,c]\cdot(A[v,c]-R_t[\alpha_t(v),c])^2 = \sum_{c\in C_t^\downarrow} W[v_0,c]\cdot(A[v_0,c]-R_t[\alpha_t(v_0),c])^2$. Here $W[v_0,c] = 0$ for every $c \in C_t^\downarrow \setminus C_t$ because such c never appears in the same bag with v_0. Moreover, for $c \in C_t$, $R_t[\alpha_t(v_0),c] = R_t[i_0,c] = A[v_0,c]$, so every summand is equal to zero.

t is an introduce node (introducing a coordinate). In case if t' is the child of t in T and $\chi(t) = \chi(t') \cup \{c_0\}$ for some coordinate c_0, we set $\alpha_t = \alpha_{t'}$, $R_t[i][C_{t'}] = R_{t'}[i][C_{t'}]$ for every $i \in [k]$ and define R_t in the column c_0 as follows. For every vector $\omega \in D^{C_{t'}}$, let g^ω be the set of indices $i \in [k]$ such that $R_{t'}[i][C_{t'}] = \omega$. Recall that by the second invariant for t', $|g^\omega| = \frac{k}{|D|^{|C_t-1|}}$. We subdivide g^ω into $|D|$ groups $\{g^\omega_\gamma \mid \gamma \in D\}$ of the same size $\frac{k}{|D|^{|C_t|}}$ so that if $\alpha_t(v) \in g^\omega$ for some $v \in V_t$, then $\alpha_t(v) \in g^\omega_\gamma$ with $\gamma = A[v,c_0]$. It is always possible as α_t is injective on V_t (so the index γ is determined uniquely for $\alpha_t(v) \in g^\omega$) and $\frac{k}{|D|^{|C_t|}} \geq \frac{k}{|D|^q} = q \geq |V_t|$ (so potentially one of the groups can contain $\alpha_t(v)$ for all $v \in V_t$). For every $\gamma \in D$ and $i \in g^\omega_\gamma$ we then set $R_t[i,c_0] = \gamma$ and by this satisfy the second invariant in t. The following validates the third one: $\sum_{v\in V_t^\downarrow}\sum_{c\in C_t^\downarrow} W[v,c]\cdot(A[v,c]-R_t[\alpha_t(v),c])^2 = \sum_{v\in V_t^\downarrow} W[v,c_0]\cdot(A[v,c_0]-R_t[\alpha_t(v),c_0])^2 = \sum_{v\in V_t} W[v,c_0]\cdot(A[v,c_0]-R_t[\alpha_t(v),c_0])^2$. Fix $v \in V_t$, let $\gamma = A[v][C_{t'}]$. Then $\alpha_t(v) \in g^\omega$, so $\alpha_t(v) \in g^\omega_\gamma$ with $\gamma = A[v,c_0]$. Therefore $R_t[\alpha_t(v),c_0] = \gamma = A[v,c_0]$, i.e., every summand is equal to zero.

t is a join node. Let t_1, t_2 be the two children of t in T, recall that in this case $\chi(t_1) = \chi(t_2) = \chi(t)$. By the well-known separation property of tree decompositions, $\chi(t_1)^\downarrow \cap \chi(t_2)^\downarrow = \chi(t)$ [DF13, CFK+15]. Without loss of generality we may assume

that α_{t_1} and α_{t_2} coincide on V_t and $R_{t_1}[i,c] = R_{t_2}[i,c]$ for every $i \in [k]$ and $c \in C_t$. In the case it does not hold, we achieve it by permuting the rows of R_{t_2}: by the second invariant, the sets of rows R_{t_1} and R_{t_2} are the same when restricted to C_t, thus there is a permutation that achieves $R_{t_1}[i,c] = R_{t_2}[i,c]$ for every $i \in [k]$ and $c \in C_t$. Furthermore, by the third invariant, for each $v \in V_t$, α_{t_2} maps v to a row that coincides with v on C_t. Together with the first invariant it means that for each $\omega \in D^{C_t}$, α_{t_2} acts injectively from $V_t^\omega = \{v \in V_t \ : \ A[v][C_t] = \omega\}$ to the set of rows of R_{t_2} that coincide with ω on C_t. Since the same holds for α_{t_1} and R_{t_1}, for each $\omega \in D^{C_t}$ there is a permutation on the rows of R_{t_2} equal to ω on C_t that makes α_{t_1} and α_{t_2} agree on V_t^ω. Combinining the above permutations, applying the resulting permutation π_t of $[k]$ on the rows of R_{t_2}, and replacing α_{t_2} with $\pi_t \circ \alpha_{t_2}$ leads to the desired property.

Then, we can correctly define $\alpha_t = \alpha_{t_1} \cup \alpha_{t_2}$ and construct R_t as follows:

- for every $i \in [k]$ and $c \in C_{t_1}^\downarrow$, $R_t[i,c] = R_{t_1}[i,c]$,
- for every $i \in [k]$ and $c \in C_{t_2}^\downarrow$, $R_t[i,c] = R_{t_2}[i,c]$.

(R_t, α_t) obviously satisfies the first and second invariants, let us verify the third one. Note that $W[v,c] = 0$ for every $v \in V_{t_j}^\downarrow$ and $c \notin C_{t_j}^\downarrow$, $j = 1, 2$. Therefore:

$\sum_{v \in V_t^\downarrow,\, c \in C_t^\downarrow} W[v,c] \cdot (A[v,c] - R_t[\alpha_t(v),c])^2 \leq \sum_{v \in V_{t_1}^\downarrow,\, c \in C_{t_1}^\downarrow} W[v,c] \cdot (A[v,c] - R_{t_1}[\alpha_{t_1}(v),c])^2$
$+ \sum_{v \in V_{t_2}^\downarrow,\, c \in C_{t_2}^\downarrow} W[v,c] \cdot (A[v,c] - R_{t_2}[\alpha_{t_2}(v),c])^2 = 0 + 0 = 0$ as desired.

We now estimate the running time of our procedure. To show the desired time bound that is linear in terms of $n + d$, we further specify how the entries (R_t, α_t) are stored. For a node t, we store separately the mapping α_t on the forgotten row indices $V_t^\downarrow \setminus V_t$ as a linked list of pairs $(v, \alpha_t(v))$ where $v \in V_t^\downarrow \setminus V_t$, and separately the values of α on the row indices of V_t; for the V_t part, the precise method of storage is not relevant for our desired running time bound as there are at most q such entries in each node. Analogously, for each $i \in [k]$, the row $R_t[i]$ is stored separately on the columns of C_t and $C_t^\downarrow \setminus C_t$, where the latter is stored as a linked list of pairs $(c, R_t[i,c])$, $c \in C_t^\downarrow \setminus C_t$.

With the storage method above in mind, observe first that for all nodes except join nodes the manipulations on the entries in the bag take time $\mathcal{O}(kq)$, while the entries for the forgotten row and column indices are simply copied from the child node. Since the entry at a child node is never reused further in the algorithm, the corresponding linked lists can be copied by reference in constant time each.

For a join node, finding a suitable permutation of R_{t_2} that matches with R_{t_1} on C_t takes time $\mathcal{O}(k^2 q)$ by comparing the values in the columns of C_t for each pair of rows from R_{t_1} and R_{t_2}, and the same time bound covers tweaking this permutation to obtain π_t that also makes α_{t_1} and α_{t_2} agree on V_t. Applying π_t to rearrange the rows of R_{t_2} takes time $\mathcal{O}(kq)$ since we pass the representations of $R_{t_2}[i][C_t^\downarrow \setminus C_t]$ by reference. Now, to compose π_t with α_{t_2}, instead of computing the image value-by-value which would take linear time,

we do the following constant-time deferred application. For the linked list that stores the values of α_{t_2} on $V_t^{\downarrow} \setminus V_t$, we consider another type of node that contains a single permutation π_t of $[k]$. This node in interpreted as follows: any pair (v, i) that comes after this node is deemed to represent the pair $(v, \pi_t(i))$ instead. Using permutation nodes, π_t can by applied to α_{t_2} in $\mathcal{O}(k+q)$ by computing the values on V_t explicitly and putting a permutation node π_t at the beginning of the linked list for $V_t^{\downarrow} \setminus V_t$, and another permutation node π_t^{-1} at the end of this list. Finally, the union of $\alpha_{t_1}$and α_{t_2} is computed in time $\mathcal{O}(q)$ by appending the linked list that stores the images of α_{t_2} on $V_{t_2}^{\downarrow} \setminus V_{t_2}$ to the linked list of α_{t_1} on $V_{t_1}^{\downarrow} \setminus V_{t_1}$, and storing the values on V_t separately. In the same fashion, for each $i \in [k]$ we construct the row $R_t[i]$ by concatenating the respective linked lists of $R_{t_1}[i]$ and $R_{t_2}[i]$.

After processing the root node r, the entries of B can be restored efficiently in the following way. Recall that only $\mathcal{O}((n+d) \cdot q)$ entries of W are "1", as the graph G_I has a tree decomposition of width $q-1$; only the corresponding $\mathcal{O}((n+d) \cdot q)$ entries of B are relevant, others need not to be restored. First, in time $\mathcal{O}(kd)$ we retrieve the rows of R_r from the computed representation and store each row as an array indexed by the column indices. The cluster assignment can be retrieved from the representation of α_r in time $\mathcal{O}((n+d) \cdot k)$ by traversing through the corresponding linked list while maintaining the composition of all occured permutation nodes (observe that at most $2(n+d)$ permutation nodes were created throughout the execution). Afterwards each relevant entry $B[v, c]$ can be computed in constant time, by retrieving the cluster assignment $\alpha_r(v)$ of v and the value $R_r[\alpha_r(v), c]$ from the respective row array. Since the number of nodes in the tree decomposition is bounded by $\mathcal{O}(n+d)$ and processing of each node takes time $\mathcal{O}(k^2 q)$, the total running time is thus bounded by

$$\mathcal{O}((n+d) \cdot k^2 q) = (n+d) \cdot |D|^{\mathcal{O}(q)},$$

as desired. □

At this point, we can prove the main result of this section.

Theorem 5.4. *Bounded-domain MCME is fixed-parameter tractable when parameterized by* $\mathrm{tw}(G_I)$.

Proof. Given an instance of Bounded-Domain MCME, we construct the incidence graph G_I and then apply the 2-approximation algorithm for treewidth [Kor22] to compute a tree decomposition of G_I of width $q - 1 \leq 2\,\mathrm{tw}(G_I) + 1$ in time $2^{\mathcal{O}(\mathrm{tw}(G_I))} \cdot (n+d)$. At this point we proceed by comparing q and k, as follows:

- if $k \geq q \cdot |D|^q$, then we can correctly output "Yes" (along with a suitable witness) by using Lemma 5.3;
- if $k \leq q \cdot |D|^q$, then we instead invoke Theorem 5.2 to solve the instance.

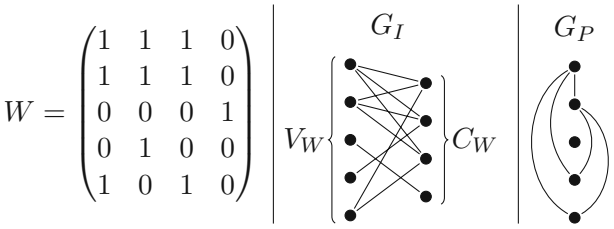

Figure 5.1: Example of a mask W (left) together with its incidence graph (middle) and its primal graph (right). Vertices from top to bottom correspond to rows from top to bottom and in the case of C_W columns from left to right. In this example $\mathrm{tw}(G_I) = 3$ and $\mathrm{tw}(G_P) = 2$.

The running time in the former case is upper-bounded by $(n+d) \cdot |D|^{\mathcal{O}(\mathrm{tw}(G_I))}$, while in the latter case the bound is $(n+d) \cdot |D|^{\mathcal{O}(k \cdot \mathrm{tw}(G_I))} \leq (n+d) \cdot |D|^{|D|^{\mathcal{O}(\mathrm{tw}(G_I))}}$. □

5.5 Handling Real-Valued Matrices with Treewidth

We now turn our attention to Real-valued MCME. As we mentioned before, a fixed-parameter algorithm for the problem parameterized by the incidence treewidth of W would resolve a long-standing open problem as a special case, and so far remains beyond reach. However, we can still make tangible progress on the problem by considering the treewidth of a different representation of W. In particular, the *primal graph* G_P of W is a graph containing one vertex for each row of W, where two rows a, b are adjacent if and only if there is a column e such that $W[a,e] = W[b,e] = 1$. An example comparing primal and incidence graphs is provided in Figure 5.1.

Our aim in this section is to prove the next theorem.

Theorem 5.5. *Real-valued MCME is fixed-parameter tractable when parameterized by* $\mathrm{tw}(G_P)$.

Before we proceed to the proof, we will first introduce some additional notation that will be useful in the context of real-valued matrices. For every $V' \subseteq V_A$, let $\mu(V', c) = \mathrm{argmin}_x f(x)$ for $f(x) = \sum_{v \in V'} W[v,c] \cdot (A[v,c] - x)^2$ and let $\Delta_c(V') = f(\mu(V', c))$. Intuitively, if vectors from V' form a cluster, then $\Delta_c(V')$ is a minimal sum of distances to the cluster center along the coordinate c.

Observation 5.6. *For each coordinate* c, *if* $V'' = \{z \in V' \mid W[z,c] = 1\} \neq \emptyset$, *then*

$$\mu(V', c) = \frac{\sum_{v \in V''} A[v,c]}{|V''|}, \text{ and } \Delta_c(V') = \sum_{v \in V''} (A[v,c])^2 - \frac{(\sum_{v \in V''} A[v,c])^2}{|V''|}.$$

Proof. As W is a binary matrix, we can rewrite $f(x)$ as follows:

$$f(x) = \sum_{v \in V''} (A[v,c] - x)^2 = |V''|x^2 - 2 \sum_{v \in V''} A[v,c] \cdot x + \sum_{v \in V''} (A[v,c])^2.$$

By assumption, $|V''| \neq 0$, so $f(x)$ is a quadratic function and attains a minimum at an extreme point, i.e., at $x = \frac{\sum_{v \in V''} A[v,c]}{|V''|}$. The minimal value of f can be computed as $\sum_{v \in V''} (A[v,c])^2 - \frac{(\sum_{v \in V''} A[v,c])^2}{|V''|}$. □

Now we are ready to present the proof of Theorem 5.5.

Proof of Theorem 5.5. As our initial step, we once again apply the 2-approximation algorithm [Kor22] to compute a nice tree decomposition (T, χ) of G_P of width $q \leq 2\,\mathrm{tw}(G_P) + 1$. We keep the notations r, V_t, and $V_t^{\downarrow}$ from the previous section. Note that now the bags in the tree decomposition contain only vectors, so $V_t = \chi(t)$. In contrast to the previous section, for every node t we denote by $C_t^{\downarrow}$ the set of all *processed* in t coordinates, i.e., coordinates c such that $W[v_c, c] \neq 0$ for some $v_c \in V_t^{\downarrow} \setminus V_t$. A crucial observation is the following: if $c \in C_t^{\downarrow}$, then for all vectors $v \notin V_t^{\downarrow}$ it holds that $W[v,c] = 0$, otherwise v and v_c would appear in the same bag. In other words, values of the cluster centers on $c \in C_t^{\downarrow}$ are not meaningful for vectors introduced outside of T_t.

We will design a leaf-to-root dynamic programming algorithm which computes a set of records $\mathcal{R}_t$ at each node t of T. For each way p of partioning the vectors in the bag V_t into at most k clusters, the record $\mathcal{R}_t$ stores the minimum cost of clustering the vectors of $V_t^{\downarrow}$ in the coordinates of $C_t^{\downarrow}$, considering only the partitions of $V_t^{\downarrow}$ that extend p.

Formally, for a node t of T let $\mathcal{P}_t$ be the set of all equivalence relations $p \subseteq V_t \times V_t$ with at most k equivalence classes. The record $\mathcal{R}_t$ of t is a mapping from $\mathcal{P}_t$ to $\mathbb{R}^+$. Observe that $|\mathcal{P}_t| \leq (q+1)^{q+1}$.

To introduce the semantics of our records, let $\mathcal{B}_t$ be the set of all real matrices with the row labels $V_t^{\downarrow}$ and the column labels $C_t^{\downarrow}$. Let B_t be a matrix in $\mathcal{B}_t$. We define the *partial weighted distance* from B_t to A in t as follows:

$$\mathtt{pwd}(B_t, t) = \sum_{v \in V_t^{\downarrow}} \sum_{c \in C_t^{\downarrow}} W[v,c] \cdot (A[v,c] - B_t[v,c])^2.$$

We also denote the number of *forgotten clusters* by $\mathtt{fcl}(B_t, t)$, that is, the number of distinct rows in B_t not equal to any $B_t[v]$, $v \in V_t$. We say that $p \in \mathcal{P}_t$ with $|[p]| \leq k$ equivalence classes is the *t-partition* of B_t if the following conditions hold:

- for every $v, w \in V_t$ such that $(v,w) \in p$, $B_t[v] = B_t[w]$,
- $\mathtt{fcl}(B_t, t)$ is at most $k - |[p]|$.

The last two conditions are in fact equivalent to existence of a cluster-assignment φ w.r.t. B_t such that for every $u, v \in V_t$, $\varphi(u) = \varphi(v)$ if and only if $(u,v) \in p$.

We are now ready to define the record $\mathcal{R}_t$. For each $p \in \mathcal{P}_t$, we set $\mathcal{R}_t(p) = \tau$ if there exists $B_t \in \mathcal{B}_t$ such that:

- p is the t-partition of B_t,
- $\mathtt{pwd}(B_t, t) = \tau$, and
- $\forall B'_t \in \mathcal{B}_t$ such that p is the t-partition of B'_t, $\mathtt{pwd}(B'_t, t) \geq \tau$.

In this case we say that B_t *witnesses* that $\mathcal{R}_t(p) = \tau$.
Observe that $\mathcal{R}_t(p) < \infty$ since any $p \in \mathcal{P}_t$ is a t-partition of the zero matrix, and that $\mathcal{R}_t(p)$ is well-defined (in the sense that $\inf_{B_t \in \mathcal{B}_t} \mathtt{pwd}(B_t, t)$ is achieved on $\mathcal{B}_t$), since $\mathtt{pwd}(B_t, t)$ is minimized on a matrix constructed by Observation 5.6 from an optimal cluster-assignment, and there is only a finite number of those.

Recall that for the root r of T, we assume $V_r = \emptyset$. Hence $\mathcal{P}_r$ contains only one element $\emptyset$, and $\mathcal{R}_r(\emptyset)$ is equal to the minimum value of $\|W \circ (A - B)\|_F^2$ that can be achieved by any real matrix B containing at most k distinct rows. Indeed, if some coordinate c is not processed in r, then $W[v, c] = 0$ for every row label v. Let us extend a witness B_r of $\mathcal{R}_r$ to a matrix B with column set C_A by setting $B[v, c] = 0$ for every row label v and every $c \notin C_r^{\downarrow}$. Then $\mathcal{R}_r(\emptyset) = \sum_{v \in V_r^{\downarrow}} \sum_{c \in C_r^{\downarrow}} W[v, c] \cdot (A[v, c] - B_r[v, c])^2 = \sum_{v \in V_A} \sum_{c \in C_A} W[v, c] \cdot (A[v, c] - B[v, c])^2$. Hence the instance is a YES-instance if and only if $\mathcal{R}_r(\emptyset) \leq \ell$.

We will show how to compute the records in a leaf-to-root fashion by proceeding along the nodes of T, in each of the following cases:

t is a leaf node. Let $V_t = \{v\}$, then $\mathcal{P}_t$ contains only one equivalence relation $p = \{(v, v)\}$ and $\mathcal{R}_t(p) = 0$ as there are no processed coordinates.

t is a forget node. Let t' be the child of t in T and $V_t = V_{t'} \setminus \{v_0\}$ for some vector v_0, we denote $C_{\mathtt{new}} = C_t^{\downarrow} \setminus C_{t'}^{\downarrow}$. By branching over all $p' \in \mathcal{P}(t')$ such that $p'|_{V_t \times V_t} = p$, we compute

$$\mathcal{R}_t^0(p) = \min_{p'} \left(\mathcal{R}_{t'}(p') + \Delta(p') \right), \text{ where } \Delta(p') = \sum_{[v] \in [p']} \sum_{c \in C_{\mathtt{new}}} \Delta_c([v]).$$

For correctness, assume that $\mathcal{R}_t^0(p) = \tau = \mathcal{R}_{t'}(p') + \Delta(p')$ for some $p' \in \mathcal{P}(t')$ such that $p'|_{V_t \times V_t} = p$. Construct a matrix B_t from the witness $B_{t'}$ of $\mathcal{R}_{t'}(p')$ by adding a new column for every label $c \in C_{\mathtt{new}}$ as follows. Fot every $v \in V_{t'}$, we set $B_t[v, c] = \mu([v]_{p'}, c)$. If $u \in V_{t'}^{\downarrow} \setminus V_{t'}$ is such that $B_{t'}[u] = B_{t'}[v]$ for some $v \in V_{t'}$, we set $B_t[u, c] = \mu([v]_{p'}, c)$. Otherwise we set $B_{t'}[u, c] = 0$. To see that p is a t-partition of B_t, assume firstly that there is $v \neq v_0$ such that $(v_0, v) \in p'$, then $\mathtt{fcl}(B_t, t) = \mathtt{fcl}(B_{t'}, t')$ and $||[p]|| = ||[p']||$. Otherwise $\mathtt{fcl}(B_t, t) \leq \mathtt{fcl}(B_{t'}, t') + 1$ and $||[p]|| = ||[p']|| - 1$; in any case p is a t-partition of B_t. Recall that

$$\begin{aligned} \mathtt{pwd}(B_t, t) &= \sum_{v \in V_t^{\downarrow}} \sum_{c \in C_t^{\downarrow}} W[v, c] \cdot (A[v, c] - B_t[v, c])^2 \\ &= \mathtt{pwd}(B_{t'}, t') + \sum_{v \in V_t^{\downarrow}} \sum_{c \in C_{\mathtt{new}}} W[v, c] \cdot (A[v, c] - B_t[v, c])^2. \end{aligned}$$

As $C_{\mathtt{new}} \cap C_{t'}^{\downarrow} = \emptyset$, we have $W[v,c] = 0$ for all $c \in C_{\mathtt{new}}$ and $v \in V_{t'}^{\downarrow} \setminus V_{t'}$. Therefore,

$$\mathtt{pwd}(B_t, t) = \mathtt{pwd}(B_{t'}, t') + \sum_{v \in V_{t'}} \sum_{c \in C_{\mathtt{new}}} W[v,c] \cdot (A[v,c] - \mu([v]_{p'}, c))^2$$
$$= \mathcal{R}_{t'}(p') + \Delta(p') = \tau,$$

and thus B_t witnesses that $\mathcal{R}_t(p) \leq \tau$.

For the converse, assume that $\mathcal{R}_t(p) = \tau$ and B_t is a matrix in $\mathcal{B}_t$ witnessing this. Let $B_{t'}$ be obtained from B_t by deleting the columns with labels in $C_{\mathtt{new}}$. We construct $p' \in \mathcal{P}(t')$ from p by adding v_0 to an equivalence class of arbitrary $v \in V_{t'}$ such that $B_t[v_0] = B_t[v]$. If there is no such v, we create a new equivalence class $\{v_0\}$. In any case p' is a t'-partition of $B_{t'}$: if $||[p']|| = ||[p]||$, then $\mathtt{fcl}(B_{t'}, t') = \mathtt{fcl}(B_t, t)$; in case $||[p']|| = ||[p]|| + 1$, we have $\mathtt{fcl}(B_{t'}, t') = \mathtt{fcl}(B_t, t) - 1$. Therefore $B_{t'}$ witnesses that

$$\mathcal{R}_{t'}(p') \leq \mathtt{pwd}(B_{t'}, t') = \tau - \sum_{v \in V_{t'}} \sum_{c \in C_{\mathtt{new}}} W[v,c] \cdot (A[v,c] - B_t[v,c])^2$$
$$\leq \tau - \sum_{[v] \in [p']} \sum_{c \in C_{\mathtt{new}}} \Delta_c([v]) = \tau - \Delta(p').$$

So in our algorithm $\mathcal{R}_t^0(p) \leq \mathcal{R}_{t'}(p') + \Delta(p') \leq \tau$. Hence, we can correctly set $\mathcal{R}_t = \mathcal{R}_t^0$.

t is an introduce node. Let t' be the child of t in T and let $V_t = V_{t'} \cup \{v_0\}$ for some vector v_0. For every p in $\mathcal{P}(t)$, we set $\mathcal{R}_t^0(p) = \mathcal{R}_{t'}(p')$ where $p' = p|_{V_{t'} \times V_{t'}}$.

For correctness, it will be useful to observe that $C_t^{\downarrow} = C_{t'}^{\downarrow}$. Recall from the first paragraph of the proof that $W[v_0, c] = 0$ for every $c \in C_{t'}^{\downarrow}$. Assume $\mathcal{R}_t(p) = \tau$ with a witness B_t, let us obtain $B_{t'}$ from B_t by deleting the row with label v_0. Then $B_{t'}$ admits the t'-partition p' and $\mathtt{pwd}(B_{t'}, t') = \mathtt{pwd}(B_t, t)$, so $B_{t'}$ witnesses that $\mathcal{R}_{t'}(p') \leq \tau$.

For the converse, if $\mathcal{R}_{t'}(p') = \tau$ and $B_{t'}$ is a matrix witnessing this, let us add to $B_{t'}$ a row with the label v_0 so that p is a t-partition of the resulting matrix B_t. For this, if $(v_0, v) \in p$ for some $v \in V_{t'}$, we set $B_t[v_0] := B_{t'}[v]$, which results in $||[p]|| = ||[p']||$ and $\mathtt{fcl}(B_t, t) = \mathtt{fcl}(B_{t'}, t')$. Otherwise $||[p]|| = ||[p']|| + 1$; we make $B_t[v_0]$ equal to any row of $B_{t'}$ that is not among $\{B_{t'}[v] \mid v \in V_{t'}\}$ (if such a row exists) and by this achieve $\mathtt{fcl}(B_t, t) = \mathtt{fcl}(B_{t'}, t') - 1$. If all the rows of $B_{t'}$ are among $\{B_{t'}[v] \mid v \in V_{t'}\}$, we set $B_t[v_0]$ equal to any of them. Then p is a t-partition of B_t and B_t witnesses $\mathcal{R}_t(p) \leq \tau$, so we can correctly set $\mathcal{R}_t = \mathcal{R}_t^0$.

t is a join node. Let t_1, t_2 be the children of t in T. For every p in $\mathcal{P}_t$ we set $\mathcal{R}_t^0(p) = \mathcal{R}_{t_1}(p) + \mathcal{R}_{t_2}(p)$.

For correctness, assume that $\mathcal{R}_t^0(p) = \tau = \tau_1 + \tau_2$ where $\tau_1 = \mathcal{R}_{t_1}(p)$, $\tau_2 = \mathcal{R}_{t_2}(p)$. Let B_i be a witness of $\mathcal{R}_{t_i}(p) = \tau_i$ and let φ_i be the cluster-assignment w.r.t. B_i such that for every $u, v \in V_t$, $\varphi_i(u) = \varphi_i(v)$ if and only if $(u, v) \in p$, $i = 1, 2$. Without loss of generality

we may assume that $\varphi_1|_{V_t} = \varphi_2|_{V_t}$. As $V_{t_1}^{\downarrow} \cap V_{t_2}^{\downarrow} = V_t$ and $C_{t_1}^{\downarrow} \cap C_{t_2}^{\downarrow} = \emptyset$, there exists a composition of (B_1, φ_1) and (B_2, φ_2), we denote it by B_t. Recall that B_t admits the cluster-assignment $\varphi = \varphi_1 \cup \varphi_2$, in particular, p is a t-partition of B_t. As $W[v, c] = 0$ for every $c \in C_{t_i}^{\downarrow}$ and $v \notin V_{t_i}^{\downarrow}$, $i = 1, 2$, we have

$$\begin{aligned}\mathtt{pwd}(B_t, t) = \sum_{v \in V_{t_1}^{\downarrow}} \sum_{c \in C_{t_1}^{\downarrow}} W[v,c] \cdot (A[v,c] - B_t[v,c])^2 + \sum_{v \in V_{t_2}^{\downarrow}} \sum_{c \in C_{t_2}^{\downarrow}} W[v,c] \cdot (A[v,c] - B_t[v,c])^2 \\ = \mathtt{pwd}(B_1, t_1) + \mathtt{pwd}(B_2, t_2) = \tau_1 + \tau_2 = \tau.\end{aligned}$$

Hence B_t witnesses that $\mathcal{R}_t(p) \leq \tau$.

For the converse, assume that $\mathcal{R}_t(p) = \tau$ and B_t is a matrix witnessing this. Let B_i be the restriction of B to rows $V_{t_i}^{\downarrow}$ and columns $C_{t_i}^{\downarrow}$, $i = 1, 2$. Then p is a t-partition of B_1 and B_2 and in our algorithm $\mathcal{R}_t^0(p) = \mathcal{R}_{t_1}(p) + \mathcal{R}_{t_2}(p) \leq \mathtt{pwd}(B_1) + \mathtt{pwd}(B_2) = \mathtt{pwd}(B_t) = \tau$. So the resulting record $\mathcal{R}_t = \mathcal{R}_t^0$ is correct, which concludes the correctness proof of the algorithm.

To estimate the time complexity, recall that in a forget node we branch over at most $(q+1)^{q+1} = \mathrm{tw}(G_P)^{\mathcal{O}(\mathrm{tw}(G_P))}$ partitions p'. Computing $\Delta(p')$ requires time at most $|C_{\mathtt{new}}| \cdot (q+1)^2 \leq |C_{\mathtt{new}}| \cdot \mathcal{O}(\mathrm{tw}(G_P)^2)$. Observe that each column appears in $C_{\mathtt{new}}$ of at most one forget node in T, as all rows that have a value of "1" in the same column form a clique. Thus, the processing time of all forget nodes is upper-bounded by $d \cdot \mathrm{tw}(G_P)^{\mathcal{O}(\mathrm{tw}(G_P))}$. For the rest of the nodes, it is easy to see that the processing time of each node is upper-bounded by $\mathrm{tw}(G_P)^{\mathcal{O}(\mathrm{tw}(G_P))}$. Since the nice tree decomposition $\mathcal{T}$ has $\mathcal{O}(n)$ nodes, the total running time is at most $(n+d) \cdot \mathrm{tw}(G_P)^{\mathcal{O}(\mathrm{tw}(G_P))}$. □

5.6 An Incidence-Graph Based Algorithm for Real-Valued MCME

While Theorem 5.5 significantly expands the previously known boundaries of tractability for MCME, the algorithm's performance strongly depends on the structural properties of the primal graph. In general, primal graph representations are known to be denser than incidence graph representations, and this may make them unsuitable for the application of structure-based algorithms on certain instances (see, e.g., the example below Theorem 5.7).

As our final result, we show that although a fixed-parameter algorithm for Real-Valued MCME parameterized by $\mathrm{tw}(G_I)$ remains beyond our reach, we can exploit a different parameter of the incidence graph to achieve fixed-parameter tractability—namely, the slim tree-cut width.

Theorem 5.7. *Real-Valued MCME is fixed-parameter tractable when parameterized by* $\mathrm{stcw}(G_I)$.

We note that it is not difficult to show that $\mathrm{stcw}(G_I)$ and $\mathrm{tw}(G_P)$ are pairwise incomparable parameterizations. Indeed, an $n \times 1$ mask with first entry "0"and rest of entries

"1" has $\mathrm{stcw}(G_I) = 1$ but $\mathrm{tw}(G_P) = n - 1$. Conversely, consider, for some integer m, an $(m^2 + 1) \times 2m^2$ mask W such that the first row v_0 of W consists only of "1" entries, while for each $i \in [m^2]$ the $(i + 1)$-th row has "1" entries on precisely two positions: i and $i + m^2$. Then G_P is a star with center in v_0, so $\mathrm{tw}(G_P) = 1$. However, G_I consists of m^2 edge-disjoint cycles intersecting in v_0, so by Lemma 3.13 its slim tree-cut width is at least m.

We proceed by introducing some additional terminology that will be useful in the coming arguments. To simplify our algorithm, we will work with super edge-cut width, a parameter that is asymptotically equivalent to the slim tree-cut width (see Corollary 3.27). Let H be a supergraph of G and let T be a fixed rooted spanning tree of H such that $\mathrm{ecw}(H) = \mathrm{ecw}(H, T) = \mathrm{sec}(G)$, denote the root by r. For $t \in V(T)$, let T_t be the subtree of T rooted at t. We define the *boundary* $\delta(t)$ of t to be the set of endpoints of all edges in H with precisely one endpoint in $V(T_t)$ (observe that the boundary can never have a size of 1). t is called *closed* if $|\delta(t)| \leq 2$ and *open* otherwise. The notion of boundary was introduced in a joint work with R. Ganian [GK21], where we also establish its basic properties. Combining them with the fact that $\mathrm{sec}(G) = \mathrm{ecw}(H) = \mathrm{lfen}(H) + 1$ (see Section 3.1.1), we immediately obtain:

Proposition 5.8. *Let H be a supergraph of G and let T be a spanning tree of H such that $\mathrm{sec}(G) = \mathrm{ecw}(H) = \mathrm{ecw}(H, T)$. For any node t of T the following holds:*

1. *For every closed child t' of t in T, it holds that $\delta(t') = \{t, t'\}$ and tt' is the only edge between $V(T_{t'})$ and $V(H) \setminus V(T_{t'})$ in H.*
2. *$|\delta(t)| \leq 2\,\mathrm{sec}(G)$.*
3. *Let $\{t_i | i \in [j]\}$ be the set of all open children of t in T. Then $j \leq 2\,\mathrm{sec}(G)$ and $\delta(t) \subseteq \bigcup_{i=1}^{j} \delta(t_i) \cup \{t\} \cup N_H(t)$.*

To prove Theorem 5.7, we will provide a leaf-to-root dynamic programming algorithm which stores information about optimal partitions of $\delta(t)$ into clusters. On a very intuitive level, Point 1. of Proposition 5.8 allows the algorithm to handle the closed children in a greedy manner, Point 2. ensures that the size of the records is bounded, and Point 3. is used in the dynamic programming step to compute records for a parent based on the records of its children. Furthermore, one can observe that $|\bigcup_{i=1}^{j} \delta(t_i)|$ is upper-bounded by a linear function of $\mathrm{sec}(G)$, which will be useful to ascertain the runtime bound for the algorithm.

Observation 5.9. *For each node t in T, $|\bigcup_{i=1}^{j} \delta(t_i)| \leq 4\,\mathrm{sec}(G)$.*

Proof. By Point 2. of Proposition 5.8, the number of vertices in $\bigcup_{i=1}^{j} \delta(t_i)$ that belong to $\delta(t)$ is at most $|\delta(t)| \leq 2\,\mathrm{sec}(G)$. If $v \in \delta(t_i) \setminus \delta(t)$ for some $i \in [j]$, then $v = t_i$. So the number of such v is at most $j \leq 2\,\mathrm{sec}(G)$ by Point 3. of Proposition 5.8. In total, we have $|\bigcup_{i=1}^{j} \delta(t_i)| \leq 4\,\mathrm{sec}(G)$. □

We are now ready to establish the claimed result by providing an algorithm for Real-Valued MCME with running time $(kn^2 + d) \cdot \sec(G_I)^{\mathcal{O}(\sec(G_I))}$, assuming that the supergraph H and its spanning tree T such that $\sec(G) = \mathrm{ecw}(H, T)$ are provided. If this is not the case, we recall that Corollary 3.30 allows to compute supergraph H of G along with its spanning tree T such that $\mathrm{ecw}(H, T) = \mathcal{O}(\mathrm{stcw}(G)^6)$. Let r be an arbitrarily chosen root in T. For each node $v \in V(T)$, we will use T_v to denote the subtree of T rooted at v.

Proof of Theorem 5.7. We will design a leaf-to-root dynamic procedure computing the set of records for every node of T. Intuitively, the record in t will store the minimum sum of distances of processed vectors, computed along the processed coordinates, to their cluster centers for every possible partition of vectors in $\delta(t)$ into clusters.

Formally, let $V_t^{\downarrow}$ and $C_t^{\downarrow}$ be the sets of vectors and coordinates of T_t respectively, we denote by V_t and C_t the restrictions of $\delta(t)$ to vectors and coordinates respectively, then $\delta(t) = V_t \sqcup C_t$. Let $\mathcal{P}_t$ be the set of all equivalence relations $p \subseteq V_t \times V_t$ with at most k equivalence classes. The record $\mathcal{R}_t$ of t is a mapping from $\mathcal{P}_t$ to $\mathbb{R}^+$. Observe that $|\mathcal{P}_t| \leq \sec(G_I)^{\mathcal{O}(\sec(G_I))}$.

To introduce the semantics of our records, let $\mathcal{B}_t$ be the set of all real matrices with the row labels $V_t^{\downarrow} \cup V_t$ and the column labels $C_t^{\downarrow}$. Let B_t be a matrix in $\mathcal{B}_t$. We define the *partial weighted distance* from B_t to A in t as follows:

$$\mathtt{pwd}(B_t, t) = \sum_{v \in V_t \cup V_t^{\downarrow}} \sum_{c \in C_t^{\downarrow}} W[v, c] \cdot (A[v, c] - B_t[v, c])^2.$$

For a matrix $B_t \in \mathcal{B}_t$, we say that $p \in \mathcal{P}_t$ is the *t-partition* of B_t if the following two conditions hold:

- for every $v, w \in V_t$ such that $(v, w) \in p$, $B_t[v] = B_t[w]$,
- the number $\mathtt{fcl}(B_t, t)$ of pairwise different rows in B_t not equal to any $B[v]$, $v \in V_t$, is at most $k - |[p]|$.

Note that the existence of a t-partition automatically implies that B_t has at most k distinct rows. We are now ready to define the record $\mathcal{R}_t$. For each $p \in \mathcal{P}_t$, we set $\mathcal{R}_t(p) = \tau$ if there exists $B_t \in \mathcal{B}_t$ such that:

- p is the t-partition of B_t,
- $\mathtt{pwd}(B_t, t) = \tau$, and
- $\forall B'_t \in \mathcal{B}_t$ such that p is the t-partition of B'_t, $\mathtt{pwd}(B'_t, t) \geq \tau$.

In this case we say that B_t *witnesses* that $\mathcal{R}_t(p) = \tau$.

Recall that for the root $r \in T$, the boundary is empty, in particular $V_r = \emptyset$, so $\mathcal{P}_r$ contains only one element $\emptyset$, and $\mathcal{R}_r(\emptyset)$ is equal to the minimum value of $\|W \circ (A - B)\|_F^2$ that can be achieved by any real matrix B containing at most k distinct rows. Hence the instance is a YES-instance if and only if $\mathcal{R}_r(\emptyset) \leq \ell$. We will compute the records in a leaf-to-root fashion by proceeding along the nodes of T. Without loss of generality we may assume that all of them are vertices of G_I, that is, $V(H) = V(G_I)$. Indeed, if t is a leaf of T and $t \notin V(G_I)$, we can delete t from T and H (note that this does not change $\mathrm{ecw}(H, T)$). Next, we can remove from H all the edges that do not belong to $G_I \cup T$. We say that a vector v is a *fake vector* if $A[v, c] = W[v, c] = 0$ for each $c \in C_A$. Analogously, we say that a coordinate c is a *fake coordinate* of A if $A[v, c] = W[v, c] = 0$ for each $v \in V_A$. Note that removing fake vectors and coordinates results in an equivalent instance. Hence, we can make the following modifications, processing T from leaves to the root:

- If $t \in V(T)$ does not belong to G_I and has coordinate as a child, we modify our instance by adding to V_A fake vector v_t. Moreover, for each child v of v_t which is a vector, we subdivide an edge vv_t in T and H by adding a fake coordinate $c_{t,v}$.
- If $t \in V(T)$ does not belong to G_I and has vector as a child, we modify our instance by adding to C_A fake coordinate c_t. Moreover, for each child c of c_t which is a coordinate, we subdivide an edge cc_t in T and H by adding a fake vector $v_{t,c}$.

Note that each edge of the modified graph H connects a vector and a coordinate, since before we ensured that $H = G_I \cup T$. From now on, we will denote by G_I the incidence graph of the modified instance. Therefore, we may assume that T and G_I have the same sets of vertices, in particular, each node of T is either vector or coordinate.

Now we are ready to compute our records. Since for any leaf vector v, there are no coordinates in T_v, we can set $\mathcal{R}_v(p) = 0$ for every $p \in \mathcal{P}_v$.

Let $t = c$ be a (maybe leaf) coordinate and let $v_1, \ldots, v_m$ be all its children in T. We assume that there is $j \in [m]$ such that $v_1, \ldots, v_j$ are open and the rest are closed. Furthermore, we asssume that the closed children are sorted by the coordinate c, i.e., $A[v_i, c] \leq A[v_{i+1}, c]$, $i \in [m-1] \setminus [j]$, except for the closed children v with $W[v, c] = 0$, which appear at the end. Recall from Proposition 5.8 that $j \leq 2\,\mathrm{sec}(G_I)$. Let us denote $V_0 = N_H(c) \setminus \{v_1, \ldots, v_m\}$, and for each $i \in [m]$, $V_i = V_{v_i}$, $C_i = C_{v_i}$, $V_i^{\downarrow} = V_{v_i}^{\downarrow}$, and $C_i^{\downarrow} = C_{v_i}^{\downarrow}$. Observe that $V_t \subseteq \bigcup_{i \in [j]_0} V_i$.

For every $p_0 \in \mathcal{P}_t$, we initiate by setting $\mathcal{R}_t(p_0) = dnM_A + 1$ and branch over all equivalence relations p on $\bigcup_{i \in [j]_0} V_i$ such that $|[p]| \leq k$ and $p|_{V_t} = p_0$. In every branch we construct a partition p^* on the domain $D^* = \bigcup_{i \in [m]_0} V_i$ extending p to the set of closed children. Intuitively, p^* will provide a subdivision of the interval $[j+1, m]$ into at most k subintervals such that closed children v_{i_1} and v_{i_2} belong to the same cluster if i_1 and i_2 belong to the same subinterval.

To find the optimal (i.e., minimizing the sum of distances in the coordinate c) subdivision, we will compute the dynamic programming table $\mathcal{M}(i, S, k_{\mathtt{cl}})$ for every subset $S \subseteq [p]$ and non-negative integers $i \in [j, m]$ and $k_{cl} \leq k - |[p]|$, that stores the minimum sum of distances along the coordinate c for the closed children $v_{j+1} \dots v_i$ if they are partitioned into $|S| + k_{\mathtt{cl}}$ clusters with the following property: $k_{\mathtt{cl}}$ clusters contain only closed children, and for each equivalence class $[v] \in S$, there is a cluster intersecting the set $N_H(c) \setminus \{v_{j+1}, \dots, v_m\}$ by exactly $[v]$. We initiate by setting $\mathcal{M}(j, \emptyset, 0) = 0$ and $\mathcal{M}(j, S, k_{\mathtt{cl}}) = dnM_A + 1$ whenever $S \neq \emptyset$ or $k_{\mathtt{cl}} \neq 0$. Further, we set $\mathcal{M}(i, \emptyset, 0) = dnM_A + 1$ for $i > j$ and then for remaining triples $S \subseteq [p]$, $i \in [j+1, m]$ and $k_{cl} \leq k - |[p]|$ compute:

$$\mathcal{M}_1(i, S, k_{\mathtt{cl}}) = \min_{j \leq i' < i} \Big(\mathcal{M}(i', S, k_{\mathtt{cl}} - 1) + \Delta_c(v_{(i',i]}) \Big),$$

$$\mathcal{M}_2(i, S, k_{\mathtt{cl}}) = \min_{j \leq i' \leq i} \min_{[v] \in S} \Big(\mathcal{M}(i', S \setminus \{[v]\}, k_{\mathtt{cl}}) + \Delta_c([v] \cup v_{(i',i]}) \Big),$$

where $v_{(i',i]} = \{v_y : y \in [i] \setminus [i']\}$. Recall that for a subset $V' \subseteq V_A$, $\Delta_c(V')$ is a minimal sum of distances to the cluster center along the coordinate c assuming that vectors from V' form a cluster, that is, $\Delta_c(V') = \min_x \sum_{v \in V'} W[v, c] \cdot (A[v, c] - x)^2$. As special cases, we set $\mathcal{M}_1(i, S, k_{\mathtt{cl}}) = dnM_A + 1$ if $k_{\mathtt{cl}} = 0$, and $\mathcal{M}_2(i, S, k_{\mathtt{cl}}) = dnM_A + 1$ if $S = \emptyset$. Intuitively, $\mathcal{M}_1$ corresponds to the case where the children in $v_{(i',i]}$ form a separate cluster, while $\mathcal{M}_2$ represents the addition of $v_{(i',i]}$ to the cluster of open children from $[v]$. In particular, $\mathcal{M}_2$ captures an option when no closed children are added to the cluster of $[v]$, namely, when $i = i'$. We set $\mathcal{M}(i, S, k_{\mathtt{cl}}) = \min(\mathcal{M}_1(i, S, k_{\mathtt{cl}}), \mathcal{M}_2(i, S, k_{\mathtt{cl}}))$.

When all the entries of $\mathcal{M}$ are computed, let $\Delta_c(p) = \min_{0 \leq k_{\mathtt{cl}} \leq k - |[p]|} \mathcal{M}(m, [p], k_{\mathtt{cl}})$, which is equal to some $\mathcal{M}(m, [p], k^0_{\mathtt{cl}})$. A simple back-tracking allows to construct the partition p^* of $D^* = \bigcup_{i \in [m]_0} V_i$ extending p such that $|[p^*]| = |[p]| + k^0_{\mathtt{cl}}$ and $\sum_{[v] \in [p^*]} \Delta_c([v]) = \Delta_c(p)$. We try to improve $\mathcal{R}^0_t(p_0)$ by $\sum_{i \in [m]} \mathcal{R}_{v_i}(p^*|_{V_i}) + \Delta_c(p)$.

For correctness, assume that in our algorithm $\mathcal{R}^0_v(p) = \tau = \sum_{i \in [m]} \mathcal{R}_{v_i}(p^*|_{V_i}) + \Delta_c(p)$. For every $i \in [m]$, let $p_i = p^*|_{V_i}$ and let B_i be the witness of $\mathcal{R}_{v_i}(p_i)$. We define the matrix B_0 with the only column label c and row labels V_0 by setting $B_0[v, c] = \mu([v]_{p^*}, c)$. Then, in particular, $\sum_{u \in [v] \cap V_0} W[u, c] \cdot (A(u, c) - B_0(u, c))^2 = \Delta_c([v])$ for every $[v] \in [p^*]$. For every $i \in [m]_0$, $p_i = p^*|_{V_i}$ and therefore we can construct cluster-assignments φ_i w.r.t. B_i such that

- for every $u, v \in V_i$, $\varphi_i(u) = \varphi_i(v)$ if and only if $(u, v) \in p^*$, $i \in [m]_0$;
- for every $v \in V_{i_1} \cap V_{i_2}$, $\varphi_{i_1}(v) = \varphi_{i_2}(v)$, $i_1, i_2 \in [m]_0$.

Note that the sets of column labels of B_i are pairwise disjoint, so there exists a composition B_t of (B_i, φ_i), $i \in [m]_0$. Recall that B_t admits the cluster-assignment $\varphi = \bigcup_{i \in [m]_0} \varphi_i$, has

row labels $\bigcup_{i\in[m]}(V_i \cup V_i^{\downarrow}) \cup V_0 = V_t \cup V_t^{\downarrow}$ and column labels $\bigcup_{i\in[m]} C_i^{\downarrow} \cup \{c\} = C_t^{\downarrow}$. Let us compute $\mathtt{pwd}(B_t, t) = \sum_{v\in V_t\cup V_t^{\downarrow}} \sum_{c'\in C_t^{\downarrow}} W[v,c'] \cdot (A[v,c'] - B_t[v,c'])^2$.

Observe that if $c' \in C_i^{\downarrow}$ and $v \notin V_i \cup V_i^{\downarrow}$, then $W[v,c'] = 0$, $i \in [m]$. The set of v such that $W[v,c] = 1$ is subset of V_0. Therefore $\mathtt{pwd}(B_t, t)$ can be computed as $\sum_{i\in[m]} \sum_{v\in V_i\cup V_i^{\downarrow}} \sum_{c'\in C_i^{\downarrow}} W[v,c'] \cdot (A[v,c'] - B_i[v,c'])^2 + \sum_{v\in V_0} W[v,c] \cdot (A[v,c] - B_0[v,c])^2 = \sum_{i\in[m]} \mathtt{pwd}(B_i, v_i) + \sum_{[v]\in[p^*]} \Delta_c([v]) = \sum_{i\in[m]} \mathcal{R}_{v_i}(p_i) + \Delta_c(p) = \tau$. Hence B_t witnesses that $\mathcal{R}_v^0(p) \leq \tau$.

For the converse, assume that $\mathcal{R}_t(p_0) = \tau$ and B_t is a matrix witnessing this. There exists a cluster-assignment φ w.r.t. B_t such that for every $u, v \in V_t$, $\varphi(u) = \varphi(v)$ if and only if $(u,v) \in p_0$. We define the equivalence relation p^* on domain D^* as follows: $(u,v) \in p^*$ if and only if $\varphi(u) = \varphi(v)$. In particular, p^* extends p_0 and $|[p]| \leq k$.

Claim 5.10. *If u and v are closed children of c, $W[v,c] = W[u,c] = 1$ and $A[u,c] \leq A[v,c]$ but $B_t[u,c] \geq B_t[v,c]$, then there exists a witness $B'_t \in \mathcal{B}_t$ of $\mathcal{R}_t(p_0) \leq \tau$ with cluster-assignment φ' such that:*

- *$\varphi'(u) = \varphi(v)$, $\varphi'(v) = \varphi(u)$;*
- *φ' coincides with φ outside of $V_u^{\downarrow} \cup V_v^{\downarrow}$;*
- *$B'_t[u,c] = B_t[v,c]$, $B'_t[v,c] = B_t[u,c]$;*
- *$B'_t[v_i,c] = B_t[v_i,c]$ for every $v_i \notin \{u,v\}$, $i \in [m]$;*

Proof. Let us define φ' as follows:

- if $w \in V_u^{\downarrow} \cup V_v^{\downarrow}$ and $\varphi(w) = \varphi(u)$, then $\varphi'(w) = \varphi(v)$;
- if $w \in V_u^{\downarrow} \cup V_v^{\downarrow}$ and $\varphi(w) = \varphi(v)$, then $\varphi'(w) = \varphi(u)$;
- for the rest of $w \in V_t \cup V_t^{\downarrow}$, $\varphi'(w) = \varphi(w)$;

We define the matrix B'_t with row and column labels $V_t \cup V_t^{\downarrow}$ and $C_t^{\downarrow}$:

- if $\varphi'(w) \neq \varphi'(v)$ and $\varphi'(w) \neq \varphi'(u)$, we set $B'_t[w] = B_t[w]$, $w \in V_t \cup V_t^{\downarrow}$;
- $B'_t[v,c'] = B_t[v,c']$ for $c' \in C_v^{\downarrow} \cup C_u^{\downarrow}$; $B'_t[v,c'] = B_t[u,c']$ for the rest of c';
- $B'_t[u,c'] = B_t[u,c']$ for $c' \in C_v^{\downarrow} \cup C_u^{\downarrow}$; $B'_t[u,c'] = B_t[v,c']$ for the rest of c';
- if $\varphi'(w) = \varphi'(x)$, we set $B'_t[w] = B'_t[x]$, $x \in \{u,v\}$.

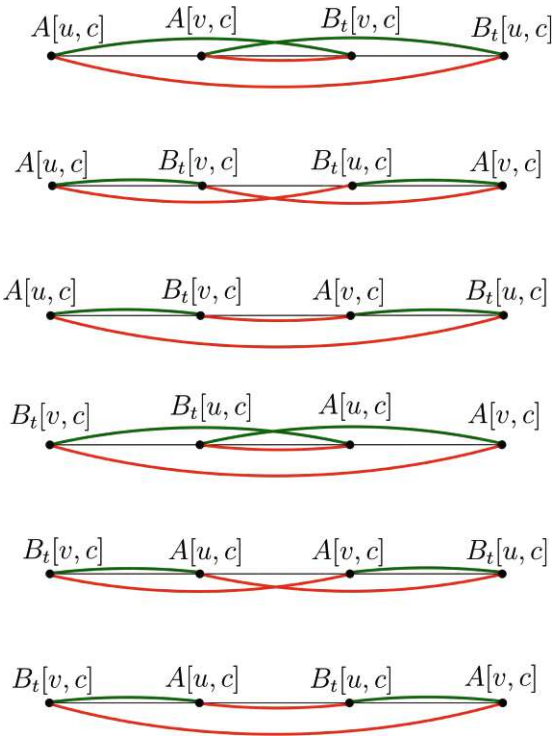

Figure 5.2: Red and green segments represent contribution of v and u along c into the total sum, for B_t and B'_t respectively. For any arrangement of $A[u,c]$, $A[v,c]$, $B_t[u,c]$ and $B_t[v,c]$, the sum of squares of green segments is not larger than the sum of squares of red ones.

From the last definition it is clear that φ' is a cluster-assignment w.r.t. B'_t. Observe that φ coincides with φ' on V_t, in particular, p_0 is a t-partition of B'_t. Let us compare $\mathtt{pwd}(B_t, t)$ and $\mathtt{pwd}(B'_t, t)$. Summands corresponding to w with $\varphi'(w) \neq \varphi'(v)$ and $\varphi'(w) \neq \varphi'(u)$ obviously coincide. Consider $w \neq v$ with $\varphi'(w) = \varphi'(v)$. If $w \in V_v^{\downarrow} \cup V_u^{\downarrow}$, then $\varphi(w) = \varphi(v)$ and therefore for $c' \in C_v^{\downarrow} \cup C_u^{\downarrow}$ we have $B'_t[w,c'] = B'_t[v,c'] = B_t[v,c'] = B_t[w,c']$. For $c' \notin C_v^{\downarrow} \cup C_u^{\downarrow}$, $W[w,c'] = 0$ as v and u are closed children. Assume that $w \notin V_v^{\downarrow} \cup V_u^{\downarrow}$, then $\varphi(w) = \varphi(u)$. For $c' \in C_v^{\downarrow} \cup C_u^{\downarrow}$, we have $W[w,c'] = 0$. For the rest of c', $B'_t[w,c'] = B'_t[v,c'] = B_t[u,c'] = B_t[w,c']$, so all the summands corresponding to w coincide. Similarly they coincide for $w \neq u$ with $\varphi'(w) = \varphi'(u)$. In particular, $B'_t[v_i,c] = B_t[v_i,c]$ for every $v_i \notin \{u,v\}$, $i \in [m]$. The only difference may occure in the summands corresponding to u and v. Moreover, the only coordinate c' where $W[v,c'] = 1$ and $B'_t[v,c'] \neq B_t[v,c']$, is $c' = c$, similarly for u. So comparing $\mathtt{pwd}(B_t, t)$ and $\mathtt{pwd}(B'_t, t)$ in case $W[v,c] = W[u,c] = 1$ is in fact comparing $(A[v,c] - B_t[v,c])^2 + (A[u,c] - B_t[u,c])^2$ and $(A[v,c] - B_t[u,c])^2 + (A[u,c] - B_t[v,c])^2$. It is easy to see that the second sum is not larger than the first, see Figure 5.2 for an illustration. □

In fact, Claim 5.10 allows us to assume without loss of generality that every cluster intersects the set of closed children of c by some $v_{(i_1,i_2]}$, $i_1 \leq i_2$, possibly except for some of closed children v with $W[v,c] = 0$. Indeed, pick any closed children v_i and $v_{i'}$ of c with $i < i'$ and $W[v_i,c] = W[v_{i'},c] = 1$, then $A[v_i,c] \leq A[v_{i'},c]$. We will say that the pair $(v_i, v_{i'})$ is *bad* if one of the following holds:

- $B_t[v_i,c] > B_t[v_{i'},c]$;
- $B_t[v_i,c] = B_t[v_{i'},c]$, $\varphi(v_i) \neq \varphi(v_{i'})$ but there exists $i'' > i'$ such that $\varphi(v_i) = \varphi(v_{i''})$.

Observe that if there are any bad pairs, we can choose one of them so that application of the Claim 5.10 to it decreases the number of bad pairs. Applying Claim 5.10 iteratively until there remain no bad pairs, we finally obtain a witness B_t^* of $\mathcal{R}_t(p_0) = \tau$ with a cluster assignment φ^* such that any cluster of φ^* intersects the set of closed children v of c by some $v_{(i_1,i_2]}$ (recall that closed children v such that $W[v,c] = 0$ appear at the end of the list, and since the choice of c does not matter for them, we may assume that they all belong to the same cluster). Further we assume $B_t^* = B_t$, $\varphi^* = \varphi$.

By optimality of B_t, we conclude that $\sum_{u\in[v]} W[u,c] \cdot (A[u,c] - B_t[u,c])^2 = \Delta_c([v])$ for every $[v] \in [p^*]$. Every such case is captured by the computation scheme of the dynamic table $\mathcal{M}$, so we have $\mathcal{M}(m,[p],k^0_{\texttt{c1}}) \leq \sum_{[v]\in[p^*]} \Delta_c([v])$ where p is the restriction of p^* to $D^* \setminus v_{(j,m]}$ and $k^0_{\texttt{c1}} = |[p^*]| - |[p]|$. Let B_i be the restriction of B_t to rows $V_i \cup V_i^{\downarrow}$ and columns $C_i^{\downarrow}$, $i \in [m]$. Then $p^*|_{V_i}$ is v_i-partition of B_i and hence in our algorithm

$$\begin{aligned}\mathcal{R}^0_t(p_0) &\leq \sum_{i\in[m]} \mathcal{R}_{v_i}(p^*|_{V_i}) + \mathcal{M}(m,[p],k^0_{\texttt{c1}}) \\ &\leq \sum_{i\in[m]} \texttt{pwd}(B_i,v_i) + \sum_{[v]\in[p^*]} \Delta_c([v]) \\ &= \texttt{pwd}(B_t,t) = \tau.\end{aligned}$$

It remains to consider the case where $t = v$ is a vector with children $c_1,\dots,c_m$ in T, where $c_1,\dots,c_j$ are open and the rest are closed. We will use the notations $V_i = V_{c_i}$, $C_i = C_{c_i}$, $V_i^{\downarrow} = V_{c_i}^{\downarrow}$ and $C_i^{\downarrow} = C_{c_i}^{\downarrow}$, $i \in [m]$. For every $p_0 \in \mathcal{P}_t$, we initiate by setting $\mathcal{R}_t(p_0) = ndM_A + 1$ and then branch over equivalence relations p on $\bigcup_{i\in[j]} V_i \cup \{v\} \supseteq V_t$ such that $|[p]| \leq k$ and $p|_{V_t} = p_0$. In every branch we set $\mathcal{R}^0_v(p_0) := \min(\mathcal{R}^0_v(p_0), \sum_{i\in[m]} \mathcal{R}_{c_i}(p_i))$ where $p_i = p|_{V_i}$ for $i \in [j]$ and $p_i = \{(v,v)\}$ for $i \in [m] \setminus [j]$.

For correctness, assume that in our algorithm $\mathcal{R}^0_v(p_0) = \tau = \sum_{i\in[m]} \mathcal{R}_{c_i}(p_i)$ and let B_i be the witness of $\mathcal{R}_{c_i}(p_i)$. We define φ on the domain of p and with values in $[k]$ by setting $\varphi(u) = \varphi(w)$ if and only if $(u,w) \in p$. Recall that $p_i = p|_{V_i}$ for $i \in [j]$ and $p_i = \{(v,v)\}$ for $i \in [m] \setminus [j]$, so there exist cluser-assignments φ_i w.r.t. B_i which agree with φ on intersections of their domains. Sets of columns of B_i are pairwise disjoint, so there exists a composition matrix B_t with cluster-assignment $\bigcup_{i\in[m]} \varphi_i = \varphi$, row labels

$\bigcup_{i\in[m]}(V_i\cup V_i^{\downarrow}) = V_t\cup V_t^{\downarrow}$ and column labels $\bigcup_{i\in[m]} C_i^{\downarrow} = C_t^{\downarrow}$. Let us compute $\mathtt{pwd}(B_t,t) = \sum_{u\in V_t\cup V_t^{\downarrow}}\sum_{c\in C_t^{\downarrow}} W[u,c]\cdot(A[u,c]-B_t[u,c])^2$. Here $C_t^{\downarrow} = \sqcup_{i\in[m]} C_i^{\downarrow}$ and $W[u,c]=0$ whenever $c\in C_i^{\downarrow}$ and $u\notin V_i\cup V_i^{\downarrow}$. Therefore $\mathtt{pwd}(B_t,t) = \sum_{i\in[m]}\sum_{u\in V_i\cup V_i^{\downarrow}}\sum_{c\in C_i^{\downarrow}} W[u,c]\cdot(A[u,c]-B_i[u,c])^2 = \sum_{i\in[m]}\mathtt{pwd}(B_i,v_i) = \sum_{i\in[m]}\mathcal{R}_{c_i}(p_i) = \tau$. Hence B_t witnesses that $\mathcal{R}_v(p)\leq\tau$.

For the converse, assume that $\mathcal{R}_t(p_0)=\tau$ and B_t is a matrix witnessing this. Let φ be the cluster-assignment w.r.t. B_t such that for every $u,v\in V_t$, $\varphi(u)=\varphi(v)$ if and only if $(u,v)\in p_0$. We extend p_0 to the equivalence relation p on domain $\bigcup_{i\in[j]} V_i\cup\{v\}$ defined as follows: $(u,v)\in p$ if and only if $\varphi(u)=\varphi(v)$. Let B_i be the restriction of B_t to rows $V_i\cup V_i^{\downarrow}$ and columns $C_i^{\downarrow}$, $i\in[m]$, then $p_i = p|_{V_i}$ is c_i-partition of B_i, $i\in[j]$. Hence in our algorithm $\mathcal{R}_t^0(p_0)\leq\sum_{i\in[j]}\mathcal{R}_{c_i}(p_i)+\sum_{i\in[m]\setminus[j]}\mathcal{R}_{c_i}(\{(v,v)\})\leq\sum_{i\in[m]}\mathtt{pwd}(B_i,c_i) = \mathtt{pwd}(B_t,t)=\tau$. Hence the resulting record $\mathcal{R}_t=\mathcal{R}_t^0$ is correct, which concludes the correctness proof of the algorithm.

Let us estimate the time for processing coordinates $t=c$. For a fixed c with m_c children, we branch over equivalence relations p on domain $\bigcup_{i\in[j]_0} V_i\subseteq\delta(c)\cup\bigcup_{i\in[j]}\delta(v_i)$. By Observation 5.9, the size of the domain is upper-bounded by $\mathcal{O}(\mathrm{sec}(G_I))$, so there are at most $\mathrm{sec}(G_I)^{\mathcal{O}(\mathrm{sec}(G_I))}$ branches. In every branch, we start from computing $\Delta_c([v]\cup v_{(i',i]})$ and $\Delta_c(v_{(i',i]})$ for each $[v]\in[p]$ and $i'<i$ from $[m_c]\setminus[j]$. Taking into account Observation 5.6, all the computations can be performed in time at most $m_c^2\cdot|[p]|$. After this, every entry of $\mathcal{M}$ can be calculated in time $\mathcal{O}(m_c\cdot|[p]|)\leq m_c\cdot\mathcal{O}(\mathrm{sec}(G_I))$. Moreover, $\mathcal{M}$ has size at most $m_ck\cdot 2^{|[p]|}\leq m_ck\cdot 2^{\mathcal{O}(\mathrm{sec}(G_I))}$ and therefore can be computed for all the branches in time $m_c^2k\cdot\mathrm{sec}(G_I)^{\mathcal{O}(\mathrm{sec}(G_I))}$. As $\sum_{c\in C_A} m_c^2\leq(\sum_{c\in C_A} m_c)^2\leq n^2$, processing all the coordinates c takes time of at most $n^2k\cdot\mathrm{sec}(G_I)^{\mathcal{O}(\mathrm{sec}(G_I))}$. Record for a vector v with m_v children can be computed in $m_v\cdot\mathrm{sec}(G_I)^{\mathcal{O}(\mathrm{sec}(G_I))}$, which yields the time $d\cdot\mathrm{sec}(G_I)^{\mathcal{O}(\mathrm{sec}(G_I))}$ for processng all the vectors. Therefore the total running time of the algorithm is upper-bounded by $(kn^2+d)\cdot\mathrm{sec}(G_I)^{\mathcal{O}(\mathrm{sec}(G_I))}$. □

5.7 Discussion

While our algorithmic results are specifically designed to deal with Means Clustering with Missing Entries, it would be interesting to see whether the approaches and techniques developed here can be applied to other clustering variants or, e.g., the related task of low-rank matrix completion. Still, on the theoretical side the by far most prominent problem that is relevant to this research direction is the complexity of k-Means Clustering for real-valued matrices when parameterized by the number of columns. A W[1]-hardness result for this problem would immediately exclude the existence of a fixed-parameter algorithm for Real-valued MCME parameterized by the incidence treewidth of the mask, while a fixed-parameter algorithm could potentially open up the way towards tractability.

CHAPTER 6

Lower Bounds and Algorithms for Synchronous Dynamical Systems

Synchronous dynamic systems are a well-studied model used to capture a range of diffusion phenomena in networks [RMRS21, CLPT20]. Such systems have been used, e.g., in the context of social contagions (e.g., the spread of information, opinions, fads, epidemics) as well as product adoption [AKM+19, GSM+18, OU17].

Informally, a synchronous dynamic system (SyDS) consists of a directed graph G (representing an underlying network) with each node v having a local function f_v and containing a state value from a domain $\mathbb{B}$, which may evolve over discrete time steps[1]. While each node v begins with an initial state value at time 0, at each subsequent time step it receives an updated value by invoking the node's local function f_v on the state value of v and of all nodes with arcs to v (i.e., the closed in-neighborhood of v). In line with recent works [RMRS21, CLPT20], here we focus our attention to the Boolean-domain case with deterministic functions, which is already sufficiently rich to model a variety of situations. SyDS with Boolean domains are sometimes also called synchronous Boolean networks, especially in the context of systems biology [OU20, AHCN07, KPST03].

In this chapter, we study the three most notable problems in synchronous dynamic systems: whether the system will transition to a target configuration from a starting configuration, whether the system will reach convergence from a starting configuration, and whether the system is guaranteed to converge from every possible starting configuration. We show that all three problems remain intractable even on instances of small constant treewidth. We complement our hardness results with fixed-parameter algorithms parameterized by treedepth. For the convergence guarantee, we achieve fixed-parameter tractability by additional parameterization by maximum in-degree.

[1] Formal definitions are provided in Section 6.3.

6.1 Related Work

A central notion in the context of SyDS is that of a *configuration*, which is a tuple specifying the state of each node at a certain time step. In several use cases of SyDS, there are clearly identifiable configurations that are either highly desirable (e.g., when dealing with information propagation), or highly undesirable (when modeling the spread of a disease or computer virus). Indeed, the Reachability problem [RMRS21]—deciding whether a given target configuration will be reached from a given starting configuration—is a classical computational problem on SyDS [OU17, AHCN07].

In other settings such as in opinion diffusion [AFG18, ACF+17], we do not ask for the reachability of a specific configuration, but rather whether the system eventually converges into a *fixed point*, i.e., a configuration that transitions into itself. This idea has led to the study of two different problems on SyDS [CLPT20]: in Convergence we ask whether the system converges (to an arbitrary fixed point) from a given starting configuration, while in Convergence Guarantee we ask for a much stronger property—notably, whether the system converges from all possible configurations.

In view of the fundamental nature of these three problems, it is somewhat surprising that so little is known about their complexity. The PSPACE-completeness of Convergence and Convergence Guarantee has been established few years ago [CLPT20], while the PSPACE-completeness of Reachability on directed acyclic networks was established even more recently [RMRS21]. Earlier, Barrett et al. [BIM+06] established the PSPACE-completeness of Reachability on general directed networks of bounded treewidth and degree, albeit the bounds obtained in that work are very large.

6.2 Our Contribution

Since Reachability, Convergence and Convergence Guarantee are all computationally intractable on general SyDS, it is natural to ask whether this barrier can be overcome by exploiting the structural properties of the input network. In this chapter we investigate these three problems through the lens of parameterized complexity.

As in case with clustering, we begin our investigation by considering treewidth, the most widely studied and prominent graph parameter. In addition to its fundamental nature, it should be noted that the structure of real-world networks has been shown to attain low treewidth in several settings [MSJ19]. While Convergence Guarantee was already known to be intractable even on networks of constant treewidth [RMRS21, Theorem 5.1][2], previous reductions for Reachability and Convergence only apply to networks of high treewidth [CLPT20, BIM+06]. Here, we show:

Theorem 6.1. *Reachability and Convergence are* PSPACE*-complete, even on SyDSs of treewidth* 2 *and maximum in-degree* 3.

The main technical contribution within the proof is the construction of a non-trivial counter which can loop over all exponentially many configurations of a set of nodes

and whose structure is nothing more than a directed path. We believe the existence of such a counter is surprising and may be of independent interest; it contrasts previous counter constructions which relied on much denser connections between the nodes, but interestingly its simple structure comes at the cost of the configuration loop generated by the counter being highly opaque.

Intractability w.r.t. treewidth draws a parallel to the complexity behavior of the classical QBF problem—an archetypical PSPACE-complete problem which also remains PSPACE-complete on instances with bounded treewidth [AO14]. In fact, while being based on entirely different ideas, our reduction and Atserias' and Oliva's construction for QBF also show intractability for pathwidth, but do not exclude tractability w.r.t. treedepth. Investigating the complexity of our problems under the parameter treedepth is the natural next choice, not only because it lies at the very boundary of intractability, but also because of its successful applications for a variety of other problems [GPSS20, GO18, GJW16] and its close connection to the maximum path length in the network[3]. While the complexity of QBF parameterized by treedepth remains a prominent open problem, as our second main technical contribution we show:

Theorem 6.2. *Reachability and Convergence are fixed-parameter tractable when parameterized by the treedepth of the network.*

The main idea behind the proof of Theorem 6.2 is to show that the total periodicity of the configurations is upper-bounded by a function of the treedepth, and this fact then enables us to argue the correctness of an iterative pruning step that allows us to gradually reduce the instance to an equivalent one of bounded size. As for the third problem (Convergence Guarantee), fixed-parameter tractability w.r.t. treedepth is excluded by the intractability of the problem on stars.

While these results already provide a fairly tight understanding of the complexity landscape for two out of the three studied problems, they do raise the question of which structural properties of the network can guarantee the tractability of Convergence Guarantee. Intuitively, one of the main difficulties when dealing with Convergence Guarantee is that it is not even possible to enumerate all possible starting configurations of the network. Yet, in spite of this seemingly critical problem, we conclude by establishing fixed-parameter tractability of Convergence Guarantee when parameterized by treedepth plus the maximum in-degree:

Theorem 6.3. *Convergence Guarantee is fixed-parameter tractable when parameterized by the treedepth plus the maximum in-degree of the network.*

Our results are summarized in Table 6.1. For completeness, we remark that the results are robust in terms of the type of functions that can be used—in particular, all our

[3] A class of networks has bounded treedepth if and only if there is a bound on the length of any undirected path.

algorithmic results hold even if we assume that the functions are black-box oracles. Moreover, in some settings it may be useful to ask whether a target configuration and/or convergence is reached up to an input-specified time point; incorporating this as an additional constraint leads to a strict generalization of Reachability, Convergence and Convergence Guarantee, and all of the algorithms provided here can also directly solve these more general problems.

	unrestricted	tw	td	td + in-degree
Reachability	PSPACE-c	**PSPACE-c**[†]	**FPT**	FPT
Convergence	PSPACE-c	**PSPACE-c**[†]	**FPT**	FPT
Convergence Guarantee	PSPACE-c	coNP-h	coNP-h	**FPT**

Table 6.1: Summary of our main results (marked in bold). These include (1) the PSPACE-completeness of the former two problems on networks of constant treewidth, (2) their fixed-parameter tractability with respect to the parameter treedepth, and (3) a fixed-parameter algorithm for Convergence Guarantee when parameterized by treedepth plus the in-degree of the network. The coNP-hardness of the latter problem on networks of constant treedepth and treewidth follows from previous work [RMRS21, Theorem 5.1]). [†] The PSPACE-completeness of Reachability and Convergence on inputs of bounded treewidth was already shown by Barrett et al. [BIM+06], albeit the constants used in that reduction were very large while here we establish intractability for treewidth 2.

6.3 Notation and Problem Definition

We use $\delta^-(v)$ to denote the *in-neighbourhood* of a node v in a directed graph, i.e., the set of all nodes w such that the graph contains an edge wv which starts in w and ends in v.

It will be useful for us to consider tuples as implicitly indexed. This means, for two sets A and B we use B^A to denote the set of tuples with $|A|$-many entries, each of which is an element of B *and* at the same time we associate each of the entries with a fixed element of A. For a tuple $x \in B^A$ and an element $a \in A$, we denote by x_a the entry of x that is associated with a.

Synchronous Dynamic Systems. A *synchronous dynamic system* (*SyDS*) $\mathsf{S} = (G, \mathbb{B}, \{f_v \mid v \in V(G)\})$ consists of an *underlying directed graph* (the *network*) G, a *node state domain* $\mathbb{B}$, and for each node $v \in V(G)$ its *local function* $f_v \colon \mathbb{B}^{\delta^-(v)\cup\{v\}} \to \mathbb{B}$. In line with previous literature, we will always consider $\mathbb{B} = \{0, 1\}$ to be binary; however, all results presented herein can be straightforwardly generalized to any fixed (i.e., constant-size) domain. A *configuration* of a SyDS is a tuple in $\mathbb{B}^{V(G)}$. The *successor* of a configuration x is the configuration y which is given by $y_v = f_v(x_v)$ for every $v \in V(G)$. A configuration x is a *fixed point* of a SyDS if it is its own successor, i.e. for all $v \in V(G)$, it holds that $f_v(x_v) = x_v$.

Given two configurations x and y of a SyDS S, we say y is *reachable* from x if there is a sequence of configurations of S starting in x and ending in y such that every configuration in the sequence is followed by its successor.

The three problems on synchronous dynamic systems we consider are formalized as follows:

Reachability	
Input:	A SyDS $\mathsf{S} = (G, \mathbb{B}, \{f_v \mid v \in V(G)\})$, a configuration x of S called *starting configuration* and a configuration y of S called *target configuration*.
Task:	Determine whether y is reachable from x.

Convergence	
Input:	A SyDS $\mathsf{S} = (G, \mathbb{B}, \{f_v \mid v \in V(G)\})$ and a configuration x of S called *starting configuration*.
Task:	Determine whether there is a fixed point of S that is reachable from x.

Convergence Guarantee	
Input:	A SyDS $\mathsf{S} = (G, \mathbb{B}, \{f_v \mid v \in V(G)\})$.
Task:	Determine whether for every configuration x of S there is a fixed point of S that is reachable from x.

Similarly to other applications of treewidth on directed networks [GHO21, GRSZ18], in this chapter we consider the treewidth and treedepth of the *underlying undirected graph*, which is the simple graph obtained by ignoring the orientations of all arcs in the graph. While directed analogues for treewidth have been considered in the literature, these have constant values on DAGs and hence cannot yield efficient algorithms for any of the considered problems [RMRS21].

6.4 The Path-Gadget and Hardness for Treewidth

We provide a construction showing that even SyDSs which are directed paths can reach an exponential number of configurations. This functions as a crucial gadget for showing PSPACE-hardness of Reachability and Convergence, but is also an interesting result in its own right as it significantly simplifies known constructions of SyDSs with exponential periods [RMRS21].

Theorem 6.4. *For every $n \in \mathbb{N}$, there is a SyDS $\mathcal{X}^{(n)} = (G, \mathbb{B} = \{0,1\}, \{f_v \mid v \in V(G)\})$ and an initial configuration x^0, such that*

- *G is a directed path $v_1 v_2 \dots v_{2n}$,*
- *x^0 is the all-zero configuration,*
- *the successor of x^0 is the configuration x^1 such that $x^1_{v_j} = 1$ if and only if $j = 1$ or $(j \mod 2) = 0$, and*
- *for every $i \in [n+1]$ and every tuple $t \in \mathbb{B}^i$, there exists $q \in \{0, \dots, 2^i - 1\}$ such that, for every $p \in \mathbb{N}$, the configuration of $\mathcal{X}^{(n)}$ after $2^i \cdot p + q$ successor steps restricted to $v_1, v_2, v_4, \dots, v_{2i-4}, v_{2i-2}$ is equal to t.*

In particular, for every $p \in \mathbb{N}$, after $2^i \cdot p$ steps, the node state of each of the nodes $v_1, v_2, v_4, \dots, v_{2i-4}, v_{2i-2}$ is 0.

Proof. Let us fix some $n \in \mathbb{N}$. For simplicity of notation, we will denote the local function f_{v_j} for the node v_j by f_j. The local function for the node v_1 is $f_1 \colon \mathbb{B} \to \mathbb{B}$ given by $f_1(b) = 1 - b$, that is the configuration alternates between 0 and 1. For every $i \in [n]$, the local function for the node v_{2i} is $f_{2i} \colon \mathbb{B}^2 \to \mathbb{B}$ given by $f_{2i}(b_1, b_2) = (b_1 - b_2 + 1) \cdot (b_2 - b_1 + 1)$. Equivalently, $f_{2i}(b_1, b_2) = 1$ if and only if $b_1 = b_2$ and $f_{2i}(b_1, b_2) = 0$ otherwise, and so the function is simply an evaluation of the equivalence relation. Next, for every $i \in [n-1]$, the local function for the node v_{2i+1} is $f_{2i+1} \colon \mathbb{B}^2 \to \mathbb{B}$ given by $f_{2i+1}(b_1, b_2) = b_1 \cdot (1 - b_2)$. Equivalently, the configuration on v_{2i+1} is 1 if and only if in the previous step, the configuration on v_{2i+1} was 0 and the configuration on v_{2i} was 1.

This finishes the description of the SyDS. We will now prove that the structure of configurations over the time steps has the desired properties. To this end, we denote the initial configuration x^0 and for configuration x^i, we denote its successor as x^{i+1}. Moreover, for simplicity of the notation, we denote by x^i_j the state of the node v_j in i-th step, that is $x^i_{v_j}$.

Clearly, $x^i_1 = 0$ if $i \mod 2 = 0$ and $x^i_1 = 1$ if $i \mod 2 = 1$. Moreover, it is also easy to verify now that $x^1_j = 1$ if and only if $j = 1$ or $(j \mod 2) = 0$ since for $j \geq 2$, we have by the definition of the local functions that $f_j(0,0) = 1$ if and only if j is even. In order to complete the proof, we first establish the following claim.

Claim 6.5. *For all $j \geq 1$ and $i \in \mathbb{N}$, it holds that $x^i_j = x^{i'}_j$, where $i' = i \mod 2^{\lfloor \frac{j}{2} \rfloor + 1}$. Moreover, if j is even, then $x^i_j = 1 - x^{i''}_j$, where $i'' = i + 2^{\lfloor \frac{j}{2} \rfloor}$.*

Proof of the Claim:

Let us first introduce some notation that will help with exposition. For a node v_j, we will let the *period vector* for the node v_j be the vector $\mathtt{pv}_j = (x^0_j x^1_j \dots x^{2^{\lfloor \frac{j}{2} \rfloor + 1} - 1}_j)$ of

length $2^{\lfloor \frac{j}{2} \rfloor + 1}$. For simplicity, we mostly split the vector into smaller pieces consisting of at most three entries. Moreover, we use the exponent for a piece to specify how many times the same piece repeats in a row in the vector. For example, we could write the vector (01001011) as $(010)^2(11)$. The reason we call these the "period vectors" is that we will show that for all $i \in \mathbb{N}$ we have $x_j^i = x_j^{i'}$, where $i' = i \mod 2^{\lfloor \frac{j}{2} \rfloor + 1}$; in other words, the period vector repeats cyclically as the state x_j^i of the node v_j starting from x_j^0.

Let us proceed by computing the few first period vectors. This is straightforward, as we are always computing $\mathtt{pv}_{j+1}$ from $\mathtt{pv}_j$ using the fact that $x_{j+1}^0 = 0$. In each of the cases, we also make sure that the period vector $\mathtt{pv}_j$ indeed repeats as the state changes of node v_j. Since the length of the period vector $\mathtt{pv}_{j+1}$ is either the same as the period vector $\mathtt{pv}_j$ or double the length of period vector $\mathtt{pv}_j$, it suffices to verify that $x^{2^{\lfloor \frac{j}{2} \rfloor + 1}} = 0$ (assuming that we did the check for $\mathtt{pv}_j$ already). Moreover, we also check that for even j we have $x_j^i = 1 - x_j^{i''}$, where $i'' = i + 2^{\lfloor \frac{j}{2} \rfloor}$.

- $\mathtt{pv}_1 = (01)$;
- $\mathtt{pv}_2 = (01)(10) = (011)(0)$;
- $\mathtt{pv}_3 = (001)(0)$;
- $\mathtt{pv}_4 = (010)(0)(101)(1) = (010)^2(11)$;
- $\mathtt{pv}_5 = (001)^2(01)$;
- $\mathtt{pv}_6 = (010)^2(01)(101)^2(10) = (010)^2(011)^3(0)$;
- $\mathtt{pv}_7 = (001)^5(0)$;
- $\mathtt{pv}_8 = (010)^5(0)(101)^5(1) = (010)^6(110)^4(11)$;
- $\mathtt{pv}_9 = (001)^6(010)^4(01)$;
- $\mathtt{pv}_{10} = (010)^6(011)^4(01)(101)^6(100)^4(10) = (010)^6(011)^{11}(001)^4(0)$;
- $\mathtt{pv}_{11} = (001)^{17}(000)(100)^3(1)$;
- $\mathtt{pv}_{12} = (010)^{17}(010)(110)^3(1)(101)^{17}(101)(001)^3(0) = (010)^{18}(110)^{21}(100)^3(10)$;

The reason we computed the first 12 entries is that we will show that from here onward, the structure of the period vectors for v_j will begin to follow a cyclic pattern. Namely, we will distinguish the remaining nodes by $(j \mod 4)$ and show that the structure of their period vectors is the same as the structure of period vectors for v_9, v_{10}, v_{11}, and v_{12} respectively. More precisely, we show that for $j \geq 9$, the following patterns are preserved.

- If $j = 4k$, $\mathrm{pv}_j = (010)^\ell(110)^p(100)^q(10)$ for some $\ell, p, q \in \mathbb{N}$ such that $\ell + p + q = \frac{2^{2k+1}-2}{3}$.
- If $j = 4k+1$, $\mathrm{pv}_j = (001)^p(010)^q(01)$ for some $p, q \in \mathbb{N}$ such that $p + q = \frac{2^{2k+1}-2}{3}$.
- If $j = 4k+2$, $\mathrm{pv}_j = (010)^\ell(011)^p(001)^q(0)$ for some $\ell, p, q \in \mathbb{N}$ such that $\ell + p + q = \frac{2^{2k+2}-1}{3}$.
- If $j = 4k+3$, $\mathrm{pv}_j = (001)^p(000)(100)^q(1)$ for some $p, q \in \mathbb{N}$ such that $p+q = \frac{2^{2k+2}-4}{3}$.

While proving the structure of the period vector in each case, we will simultaneously show that the claim holds in each of the cases. Moreover, note that the structural claim about pv_j holds for $j = 12$. We distinguish the cases on the remainder of j modulo four.

Case $j = 4k$ for $k \geq 3$. Let us assume that $\mathrm{pv}_j = (010)^\ell(110)^p(100)^q(10)$ for some $\ell, p, q \in \mathbb{N}$ such that $\ell + p + q = \frac{2^{2k+1}-2}{3}$. We know that $f_{j+1}(b_1, b_2) = b_1 \cdot (1 - b_2)$. Moreover $x^0_{j+1} = 0$. Before we prove the structure for the case $j + 1 = 4k + 1$ more formally, let us write the two structures under each other in a suggestive way such that it is rather straightforward to verify the structure of the period vector pv_{j+1} from pv_{j+1}, fact that $x^0_{j+1} = 0$, and the function f_{j+1}.

$$\mathrm{pv}_{4k} \to (010)^\ell(110)^p(100)^q(10)$$
$$\mathrm{pv}_{4k+1} \to (001)^\ell(010)^p(010)^q(01)$$

Now, we discuss this a bit more formally. Let us first check how the state of v_{j+1} changes, when the state of v_j is in the first part where it changes as $(010)^\ell$ staring from $x^0_j = 0$ and $x^0_{j+1} = 0$. We get $x^1_{j+1} = f_{j+1}(x^0_j, x^0_{j+1}) = f_{j+1}(0, 0) = 0$, $x^2_{j+1} = f_{j+1}(1, 0) = 1$ and $x^3_{j+1} = f_{j+1}(0, 1) = 0$. That is from the first (010) for v_j we get (001) and in addition we get that the next triple starts again with 0. It follows that pv_{j+1} starts with $(001)^\ell$ and $x^{3\ell}_{j+1} = 0$. Given this, let us see how x^i_{j+1} changes on the $(110)^p$. We have $f_{j+1}(1, 0) = 1$, $f_{j+1}(1, 1) = 0$, and $f_{j+1}(0, 0) = 0$. Hence we get (010) on the first (110) with addition that $x^{3\ell+3}_{j+1} = 0$, hence we get additional $(010)^p$ to pv_{j+1}. For (100), when the first state of v_{j+1} is 0, we get $f_{j+1}(1, 0) = 1$, $f_{j+1}(0, 1) = 0$, $f_{j+1}(0, 0) = 0$. Hence, we get (010) for the first (100) with the next state of v_{j+1} being again 0. Therefore, we get additional period $(010)^q$. Finally for (10) we get $f_{j+1}(1, 0) = 1$ which leads to final (01) to the period vector pv_{j+1} and $\mathrm{pv}_{j+1} = (001)^\ell(010)^{p+q}(01)$. Finally, $f(0, 1) = 0$ ensures that $x^{2^{2k+1}}_{j+1} = x^0_{j+1}$ and since, by induction hypothesis, the period vector for v_j repeats, so does the vector for v_{j+1}.

Case $j = 4k + 1$ for $k \geq 3$. Let us assume that $\mathrm{pv}_j = (001)^p(010)^q(01)$ for some $p, q \in \mathbb{N}$ such that $p+q = \frac{2^{2k+1}-2}{3}$. We know that $f_{j+1}(b_1, b_2) = (b_1 - b_2 + 1) \cdot (b_2 - b_1 + 1)$.

Moreover $x^0_{j+1} = 0$. Let us again write down how the period vector $\mathtt{pv}_j$ influenced the period vector $\mathtt{pv}_{j+1}$.

$$\mathtt{pv}_{4k+1} \to (001)^p(010)^q(01) \mid (001)^p(010)^q(01)$$
$$\mathtt{pv}_{4k+2} \to (010)^p(011)^q(01) \mid (101)^p(100)^q(10)$$
$$\mathtt{pv}_{4k+2} \to (010)^p(011)^q(011)(011)^p(001)^q(0)$$

Note that $f_{j+1}(1,1) = 1$, hence $x^{2^{2k+1}}_{j+1} = 1$ and we needed to repeat the period vector for v_j twice to get the period vector for v_{j+1}. Moreover, we can see that for all $i \in \{0, \ldots, 2^{2k+1} - 1\}$ it holds that $x^i_{j+1} = 1 - x^{i+2^{2k+1}}_{j+1}$. However, $f_{j+1}(1,0) = 0$ and $x^{2^{2k+2}}_{j+1} = x^0_{j+1} = 0$ and it follows that the period vector for v_{j+1} repeats each 2^{2k+2} steps. Checking that the transition outlined above is correct is rather straightforward and similar to the case $j = 4k$.

Case $j = 4k + 2$ for $k \geq 3$. Assume that $\mathtt{pv}_j = (010)^\ell(011)^p(001)^q(10)$ for some $\ell, p, q \in \mathbb{N}$ such that $\ell + p + q = \frac{2^{2k+2}-1}{3}$. Moreover, $f_{j+1}(b_1, b_2) = b_1 \cdot (1 - b_2)$ and $x^0_{j+1} = 0$. We get the following transition from $\mathtt{pv}_j$ to $\mathtt{pv}_{j+1}$

$$\mathtt{pv}_{4k+2} \to (010)^\ell(011)^p(001)(001)^{q-1}(0)$$
$$\mathtt{pv}_{4k+3} \to (001)^\ell(001)^p(000)(100)^{q-1}(1)$$

Since $f_{j+1}(0,1) = 0$, we get $x^{2^{2k+2}}_{j+1} = x^0_{j+1} = 0$ and it follows that the period vector for v_{j+1} repeats each 2^{2k+2} steps. Checking that the transition outlined above is correct is rather straightforward and similar to the case $j = 4k$.

Case $j = 4k + 3$ for $k \geq 3$. Assume that $\mathtt{pv}_j = (001)^p(000)(100)^q(1)$ for some $p, q \in \mathbb{N}$ such that $p + q = \frac{2^{2k+2}-1}{3}$. Moreover, $f_{j+1}(b_1, b_2) = (b_1 - b_2 + 1) \cdot (b_2 - b_1 + 1)$ and $x^0_{j+1} = 0$. We get the following transition from $\mathtt{pv}_j$ to $\mathtt{pv}_{j+1}$

$$\mathtt{pv}_{4k+3} \to (001)^p(000)(100)^q(1) \mid (001)^p(000)(100)^q(1)$$
$$\mathtt{pv}_{4k+4} \to (010)^p(010)(110)^q(1) \mid (101)^p(101)(001)^q(0)$$
$$\mathtt{pv}_{4k+4} \to (010)^p(010)(110)^q(110)(110)^p(100)^q(10)$$

Again, we have that $f_{j+1}(1,1) = 1$, hence $x^{2^{2k+2}}_{j+1} = 1$ and we needed to repeat the period vector for v_j twice to get the period vector for v_{j+1}. Moreover, we can see that for all $i \in \{0, \ldots, 2^{2k+2} - 1\}$ it holds that $x^i_{j+1} = 1 - x^{i+2^{2k+2}}_{j+1}$. Finally, $f_{j+1}(1,0) = 0$ and $x^{2^{2k+2}}_{j+1} = x^0_{j+1} = 0$ and it follows that the period vector for v_{j+1} repeats each 2^{2k+3} steps. ■

Observe that the above claim already implies that for all $i \in [n+1]$ and $p \in \mathbb{N}$, the state of the node v_{2i-2} after $2^i \cdot p$ steps is 0 (i.e., $x^{2^i \cdot p}_{2i-2} = 0$).

Now, given the above claim on the structure of the configuration, it is also rather straightforward to show that for all $i \in [n+1]$ and all tuples $t \in \mathbb{B}^i$, there exists $q \in \{0, \dots, 2^i - 1\}$ such that, for every $p \in \mathbb{N}$, the configuration of $\mathcal{X}^{(n)}$ after $2^i \cdot p + q$ successor steps restricted to the nodes $v_1, v_2, v_4, \dots, v_{2i-4}, v_{2i-2}$ is equal to t. First note that for $i = 1$, the node v_1 flips always between 0 and 1 and the statement holds. Moreover, for $i = 2$, the nodes v_1 and v_2 have the following transitions through configurations $00 \rightsquigarrow 11 \rightsquigarrow 01 \rightsquigarrow 10 \rightsquigarrow 00$. Let us assume that for some $i \in [n]$ for every tuple $t \in \mathbb{B}^i$, there exists $q \in \{0, \dots, 2^i - 1\}$ such that for every $p \in \mathbb{N}$, the configuration of $\mathcal{X}^{(n)}$ after $2^i \cdot p + q$ successor steps restricted to the nodes $v_1, v_2, v_4, \dots, v_{2i-4}, v_{2i-2}$ is equal to t.

Let $t \in \mathbb{B}^{i+1}$, we will show that there exists $q \in \{0, \dots, 2^{i+1} - 1\}$ such that for every $p \in \mathbb{N}$, the configuration of $\mathcal{X}^{(n)}$ after $2^{i+1} \cdot p + q$ successor steps restricted to the nodes $v_1, v_2, v_4, \dots, v_{2i-2}, v_{2i}$ is equal to t. Denote by t' the restriction of t to the first i bits. By induction hypothesis, there exists $q' \in \{0, \dots, 2^i - 1\}$ such that for every $p' \in \mathbb{N}$, the configuration of $\mathcal{X}^{(n)}$ after $2^i \cdot p' + q'$ successor steps restricted to the nodes $v_1, v_2, v_4, \dots, v_{2i-4}, v_{2i-2}$ is equal to t'. By the above claim,

$$x_{2i}^{2^i \cdot p' + q'} = 1 - x_{2i}^{2^i \cdot p' + q' + 2^i}.$$

It follows that for every $p \in \mathbb{N}$, t appears as the restriction of the configuration to $v_1, v_2, v_4, \dots, v_{2i-2}, v_{2i}$ after $2^{i+1} \cdot p + q$ steps, where q is either equal to q' or to $q' + 2^i$. Repeating the same argument for all $t \in \mathbb{B}^{i+1}$ completes the proof. □

With the path-gadget ready, we can proceed to establishing our first hardness result. The idea of the reduction used in the proof loosely follows that of previous work [RMRS21], but uses multiple copies of the path gadget from Theorem 6.4 (instead of a single counting gadget that has a tournament as the network) to avoid dense substructures. Note that the usage of the structurally simpler, but behaviourally more complicated, path-gadget requires some additional changes to the construction. Furthermore, compared to the earlier result of Barrett et al. [BIM+06], we obtain much smaller bounds on the treewidth and maximum degree (however, our reduction does not provide a bound on the bandwidth of the graph).

Theorem 6.1. *Reachability and Convergence are* PSPACE*-complete, even on SyDSs of treewidth* 2 *and maximum in-degree* 3*.*

Proof. We proceed by a reduction from QBF which is known to be PSPACE-hard [GJ79] and takes as input a formula $Q_n x_n \dots Q_1 x_1 \varphi$ where all Q_i are either $\forall$ or $\exists$ and φ is a 3SAT-formula.

From this we construct a SyDS S which converges to one specific configuration (the all-one configuration) if and only if the given formula is true.

The overall idea is for (multiple copies[4] of) the SyDS $\mathcal{X}^{(n)}$ (see Theorem 6.4) to iterate over all possible truth assignments of the variables x_1 to x_n. While it would be possible to

[4] all copies will be independent, have no additional in-neighbours and start at the same configuration

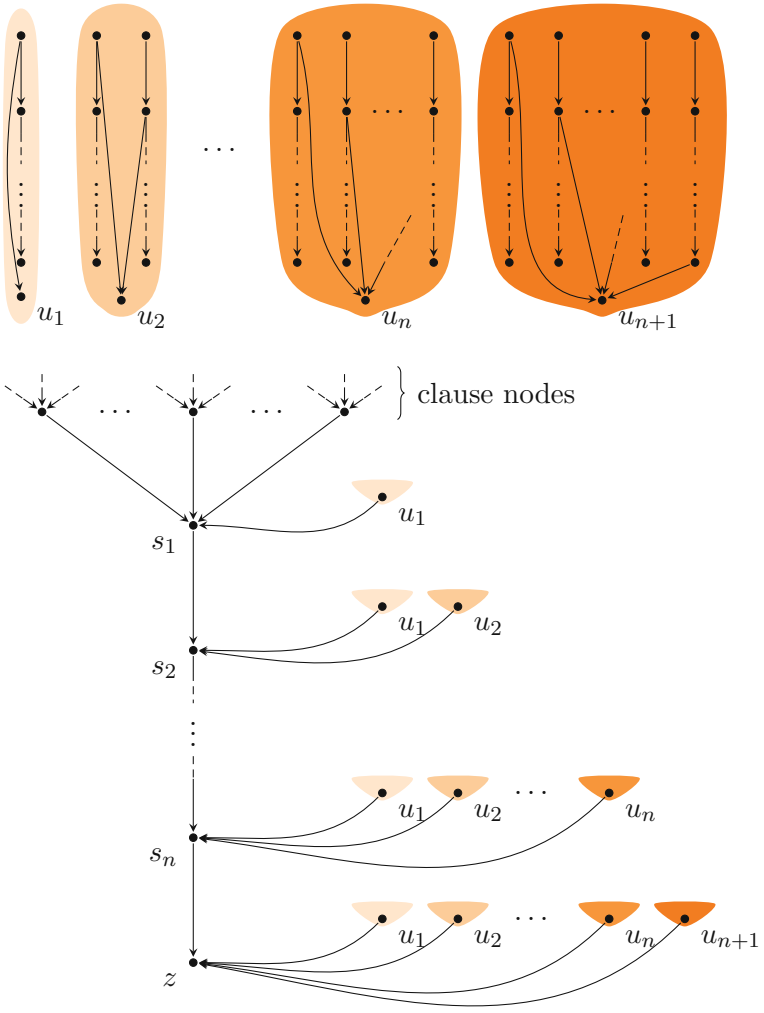

Figure 6.1: Overview of the network constructed in the proof of Theorem 6.1. To not clutter the figure, the arcs from z to all other nodes omitted, and subgraphs introduced for variable nodes are illustrated in detail at the top and only indicated by orange (darker shade $\simeq$ higher index) in all other places.

simply associate every significant node $\mathcal{X}^{(n)}$ with a variable, we will add auxiliary *variable nodes* as to go through the variable assignments in a well-structured way. Then *clause nodes* aggregate the implied truth values of the clauses of φ. These clause nodes can then be used as incoming neighbours for *subformula nodes* to derive the truth value of formulas $Q_i x_i \dots Q_1 x_1 \varphi$ respectively. Once the truth value of $Q_n x_n \dots Q_1 x_1 \varphi$ is determined to be true we use a *control node* to set the whole SyDS to a configuration with all entries being equal to 1. Otherwise the copies of $\mathcal{X}^{(n)}$ continue to loop through their period. We call the nodes v_1 and v_{2i} with $i \in [n]$ of a copy of $\mathcal{X}^{(n)}$ *significant*.

To not obfuscate the core of the construction we initially allow an arbitrary node degree and give a description on how to reduce it to a constant at the end of the proof. The overall structure of the underlying graph which we construct is depicted in Figure 6.1. The starting configuration will have an entry 0 for all nodes except the ones in the copies of $\mathcal{X}^{(n)}$ which will instead start at configuration x^1 from Theorem 6.4. We describe our construction in a hierarchical manner, going from the most fundamental to the most complex (in the sense that they rely on the earlier ones) types of nodes.

For each variable x_i, a variable node u_i has an incoming arc from the j-th significant node of the j-th of i private copies of $\mathcal{X}^{(n)}$ for all $j \in [i]$. In addition we add an auxiliary node u_{n+1} which will not actually represent a variable of φ itself but otherwise be treated and hence also be referred to as a variable node for convenience. u_{n+1} has incoming arcs from the j-th significant node of the j-th of $n+1$ private copies of $\mathcal{X}^{(n)}$ for all $j \in [n+1]$. It will become apparent later that it is useful to introduce u_{n+1} and that $X^{(n)}$ cycles only after 2^{n+1} time steps rather than just 2^n for technical reasons. We define f_{u_i} to change the configuration at u_i if and only if the configuration of all of its in-neighbours is 0 and to leave it unchanged otherwise. To give some intuition for the behaviour of the variable nodes with this construction it is useful to refer to the *position* of a tuple $t \in \mathbb{B}^{\{v_1\} \cup \{v_{2i} | i \in [n]\}}$ as the number of time steps required to reach a configuration of $\mathcal{X}^{(n)}$ that is equal to t on the significant nodes starting from the configuration x^1. Theorem 6.4 upper-bounds the position of a tuple by 2^{n+1}.

Claim 6.6. *At any time step, taken together the variable nodes $u_{n+1} \dots u_1$ encode the position of the configuration of the significant nodes of the path gadgets in binary (that are the same on different copies) in the previous time step.*

Proof of the Claim: By Theorem 6.4 and the choice of our starting configuration, the configuration at the i first significant nodes is all-zero exactly in multiples of 2^i time steps. Because of the definition of the local functions this means that the configuration at u_i flips at each time step after a multiple of 2^i. Considering all variable nodes together, this is exactly how a cyclic binary counter from 0 to 2^{n+1} works which is the same as a counter for the position of a configuration on the significant nodes of $\mathcal{X}^{(n)}$. ■

This means that the variable nodes just like the significant nodes of $\mathcal{X}^{(n)}$ have the property that on them any shared configuration is reachable, and in addition these shared configurations are reached in the order of the binary numbers they represent. Clearly this construction for each u_i leads to an underlying tree and hence has treewidth 1.

For each clause $C = x_{j_1} \vee x_{j_2} \vee x_{j_3}$ of φ, a clause node c has incoming arcs from private variable nodes u_{j_1}, u_{j_2} and u_{j_3} and its local function is 1 if and only if the configuration of at least one of these in-neighbours is 1. This means we will have a copy of each variable node with its own copies of $\mathcal{X}^{(n)}$ for each occurrence of that variable in φ. The underlying graph formed by this construction is still a tree.

For each $i \in [n]$ we introduce a subformula node s_i. s_1 has incoming arcs from all clause nodes and a private variable node u_1. For $i \in [n] \setminus \{1\}$, s_i has incoming arcs from s_{i-1} and private variable nodes $u_1, \ldots, u_i$. As before, this construction still results in a tree.

Now we turn to the definition of the local functions for subformula nodes. We define f_{s_1} as follows: If the entry from u_1 is 1, then f_{s_1} maps to 1 if and only if all entries from the clause nodes are 1. Otherwise if $Q_1 = \exists$, f_{s_1} maps to 1 if and only if the entry for s_1 *or* all entries from the clause nodes are 1. Otherwise the entry from u_1 is 0 and $Q_1 = \forall$, and we let f_{s_1} map to 1 if and only if the entry for s_1 *and* all entries from the clause nodes are 1. For $i \in [n] \setminus \{1\}$ we define f_{s_i} as follows: If the incoming variable nodes in the order $u_i \ldots u_1$ are the binary encoding of $2^{i-1} + i - 1$, then f_{s_i} maps to the entry for s_{i-1}. Otherwise if the incoming variable nodes in the order $u_i \ldots u_1$ are the binary encoding of $i - 1$ and $Q_i = \exists$, then f_{s_i} maps to 1 if and only if the entry for s_i *or* the entry for s_{i-1} is 1. Otherwise if the incoming variable nodes in the order $u_i \ldots u_1$ are the binary encoding of $i - 1$ and $Q_i = \forall$, then f_{s_i} maps to 1 if and only if the entry for s_i *and* the entry for s_{i-1} is 1. Otherwise the incoming variable nodes in the order $u_i \ldots u_1$ do not encode $2^{i-1} + i - 1$ or $i - 1$ and we let f_{s_i} map to the entry for s_i.

With these local functions we can show the following.

Claim 6.7 (cf. [RMRS21, Claim 1])**.** *Let $i \in [n]$ and t be a tuple over $\mathbb{B}$ with an entry for each of $\{x_{i+1}, \ldots, x_n\}$ which in the order $t_{x_n} \ldots t_{x_{i+1}}$ is a binary encoding of the number k. Then the configuration of s_i when $u_n \ldots u_1$ is the binary encoding of $(k+1)2^i + i$ is equal to the truth value (i.e. 1 if `true` and 0 if `false`) of $Q_i x_i \ldots Q_1 x_1 \varphi(t)$ where $\varphi(t)$ arises from φ by replacing each occurrence of x_j with $j \in [n] \setminus [i]$ by its entry in t.*

Proof of the Claim: This claim is already essentially shown as Claim 1 in [RMRS21] but we reprove it in our setting and notation for self-containedness.

The proof works by induction on i. For the base case $i = 1$ we want to show the correct configuration at s_1 and when $u_n \ldots u_1$ encodes $(k+1)2+1$. By Claim 6.6, to determine the configuration of s_1 when $u_n \ldots u_1$ encodes $(k+1)2+1$, we need to consider the application of the local function f_{s_1} when $u_n \ldots u_1$ encodes $(k+1)2$. By the definition of f_{s_1} this means that then the configuration at s_1 is derived as the as the disjunction if $Q_1 = \exists$ and conjunction if $Q_1 = \forall$ of the truth value of $\varphi(a)$, where a associates $x_n \ldots x_1$ to the bits of the binary encoding of $(k+1)2 - 1 = 2k + 1$, and the configuration at s_1 when $u_n \ldots u_1$ encodes $(k+1)2$. To determine the latter we have to consider the the application of the local function f_{s_1} when $u_n \ldots u_1$ encodes $(k+1)2 - 1 = 2k + 1$. By the definition of f_{s_1} this means the configuration at s_1 when $u_n \ldots u_1$ encodes $(k+1)2$ is equal to the truth value of $\varphi(b)$, where b associates $x_n \ldots x_1$ to the bits of the binary encoding of $2k + 1 - 1 = 2k$. Observe that a and b correspond to t^{+1} which adds to t an entry with value 1 for x_1 and t^{+0} which adds to t an entry with value 0 for x_1 respectively. The truth value of $Q_1 x_1 \varphi(t)$ is the disjunction of $\varphi(t^{+0})$ and $\varphi(t^{+1})$ if $Q_1 = \exists$, and the conjunction of $\varphi(t^{+0})$ and $\varphi(t^{+1})$ if $Q_1 = \forall$. This proves the base case.

We now turn to the induction step. Same as before, the truth value of $Q_{i+1}x_{i+1}\dots Q_1x_1\varphi(t)$ is the disjunction of $Q_ix_i\dots Q_1x_1\varphi(t^{+0})$ and $Q_ix_i\dots Q_1x_1\varphi(t^{+1})$ if $Q_1 = \exists$, and the conjunction of $Q_ix_i\dots Q_1x_1\varphi(t^{+0})$ and $Q_ix_i\dots Q_1x_1\varphi(t^{+1})$ if $Q_1 = \forall$ where t^{+0} adds an entry for x_{i+1} with value 0 and t^{+1} adds an entry for x_{i+1} with value 1 to t. By induction hypothesis these are the configurations at s_i when $u_n\dots u_1$ encodes $(2k+1)2^i + i = 2^{i+1}k + 2^i + i$ and $(2k+2)2^i + i = 2^{i+1}(k+1) + i$ respectively. When $u_n\dots u_1$ encodes $(2k+1)2^i + i = 2^{i+1}k + 2^i + i$, then the in-neighbours of s_{i+1} together encode $2^i + i = 2^{i+1-1} + i + 1 - 1$. By the definition of $f_{s_{i+1}}$, s_{i+1} then takes the configuration at s_i which by the argument above is the truth value of $Q_ix_i\dots Q_1x_1\varphi(t^{+0})$. This configuration is retained at s_{i+1} until $u_n\dots u_1$ encodes $2^{i+1}(k+1) + i$ because for step between $u_n\dots u_1$ encoding $2^{i+1}k + 2^i + i$ and $2^{i+1}(k+1) + i$, does $x_i\dots x_1$ encode i or $2^i + i$. When $u_n\dots u_1$ encodes $2^{i+1}(k+1) + i$, then the in-neighbours of s_{i+1} together encode $i = i + 1 - i$. By the definition of $f_{s_{i+1}}$, s_{i+1} then takes the disjunction if $Q_{i+1} = \exists$ between the configuration at s_i which by the argument above is the truth value of $Q_ix_i\dots Q_1x_1\varphi(t^{+1})$ and the configuration currently at s_{i+1} which by the argument above is the truth value of $Q_ix_i\dots Q_1x_1\varphi(t^{+0})$, and the conjunction of the same values if $Q_{i+1} = \forall$. This is exactly the desired behaviour described at the beginning of the induction step. ■

Taking this, the QBF input formula is true if and only if the entry of the configuration of the SyDS S whose construction we described up to now in this proof when $u_n\dots u_1$ are the binary encoding of $2^n + n$ ($= (0+1)2^n + n$ for t being the empty tuple in the previous claim) is 1.

In this case and only in this case we want the SyDS to converge, and in particular interrupt the looping through different configurations of the copies of $\mathcal{X}^{(n)}$. For this purpose we introduce a control node z which has an incoming arc from every node of a private copy of every variable node $u_{n+1}\dots u_1$ and and an outgoing arc to all other nodes. We let f_z map to 1 if and only if the entries for the variable nodes encode $2^n + n$ in binary (this is where the auxiliary node u_{n+1} becomes important as this number cannot be encoded by the even-index nodes of only $\mathcal{X}^{(n)}$) and the entry for r_n is 1 or if the entry for f_z is already 1. We add an entry for f_z to all previously defined local functions and condition the previously defined behaviour on this entry being 0. As soon as this entry is 1 the local functions of all nodes should also map to 1. In this way the all-one configuration is reachable from the all-zero configuration if and only if the QBF input formula is true and this is the only way for the SyDS to converge.

Note that the network constructed in this manner consists of a directed tree plus the additional vertex z.

Modification for constant in-degree. To obtain a network with constant maximum in-degree, note that by *currying* [Cur80] one can write every local function that takes polynomially many arguments as a polynomial-length series of functions that each take 2 arguments. In this way we can essentially replace every node with an in-degree of more than two by a tree of polynomial depth in which each node has in-degree 2.

For simplicity, we keep the control node as in-neighbour for every node. Formally, we replace any node a with $\delta^-(a) = \{b_1, \dots, b_\ell, z\}$ with $\ell > 2$ as follows. By currying, we find $f_1, \dots, f_{\ell-1}$ such that $f_1, \dots, f_{\ell-2}$ take three arguments each, $f_{\ell-1}$ takes four arguments and for any tuple $t \in \mathbb{B}^{\{b_1,\dots,b_\ell,b,z\}}$, it holds that

$$f_a(t_{b_1}, \dots, t_{b_\ell}, t_b, t_z) = f_{\ell-1}(f_{\ell-2}(\dots f_1(t_{b_1}, t_{b_2}, t_z), t_{b_{\ell-1}}, t_z), t_{b_\ell}, t_b, t_z).$$

In place of a we introduce $a_1, \dots, a_{\ell-1}$ and let the in-neighbours of a_1 be b_1 and b_2, and the in-neighbours of a_{i+1} for $i \in [\ell-2] \setminus \{1\}$ be a_i and b_{i+2}. We let the out-neighbours of $a_{\ell-1}$ be the former out-neighbours of a and define f_{a_i} as f_i. By the choice of f_{a_i} for $i \in [\ell-1]$, assuming synchronous computation of the configurations at all of $a_1 \dots, a_{\ell-1}$ from $b_1 \dots, b_\ell$, a and $a_{\ell-1}$ would be functionally equivalent.

However, this assumption is not one we can simply make, as the configuration of b_1 through b_ℓ at one time step require differently long to reach $a_{\ell-1}$. To avoid asynchronous indirect arrival of the inputs from $b_1, \dots, b_\ell$ we replace the arc that connects b_{i+2} for $i \in [\ell-2]$ to the tree replacing a by a directed path with $i-1$ nodes. The first node on each of these paths behaves like a_i from the construction above and for the rest, let their local function copy the configuration of their respective predecessor at the previous time step. In this way, at time step $t + \ell - 1$, $a_{\ell-1}$ is functionally equivalent to a from our old construction at time step $t+1$. This means the output at all nodes which stand for the nodes with in-degree $\ell > 3$ from our original construction is delayed by $\ell - 2$ time steps. Because $\ell = n$ for all variable nodes, the configuration at every variable node is uniformly delayed by $n-2$ compared to our original construction. Similarly all clause nodes are delayed by 1 in addition to the cumulative delay of its non-z ancestors, s_1 is delayed by one less than the number of clauses in addition to the cumulative delay of its non-z ancestors, and each s_i with $i \in [n] \setminus \{1\}$ is delayed by $i-1$ in addition to the cumulative delay of its non-z ancestors. As all variable nodes remain synchronised, this delay does not impact the proof of correctness. This replacement does not change the fact that the constructed network consists of a directed tree plus a single node.

□

6.5 An Algorithm Using Treedepth

In this section, we exploit the treedepth decomposition of the network to transform the instances of Reachability and Convergence into equivalent ones of bounded size. Our algorithms then proceed by simulating all possible configurations of the resulting smaller network.

Let $\mathcal{F}$ be the treedepth decomposition of G; without loss of generality, we may assume that $\mathcal{F}$ is a tree. We denote the subtree of $\mathcal{F}$ rooted in v by $\mathcal{F}_v$. Let G_v^* be the vertex-induced subgraph of G induced by the nodes associated to $\mathcal{F}_v$ and the neighbourhood of $\mathcal{F}_v$ in G. Our aim is to iteratively compress nodes of G from the leaves to the root to obtain a graph G' with number of nodes bounded by some function of $\mathrm{td}(G)$.

Lemma 6.8. *Let $\mathcal{I}$ be an instance of* Convergence *or* Reachability *and let $\mathcal{F}$ be the treedepth decomposition of G of height L. Assume that for some $v \in V(G)$, all the subtrees rooted in children of v contain at most m nodes. Then $\mathcal{I}$ can be modified in polynomial time to an equivalent instance $\mathcal{I}'$ with network G' which has a treedepth decomposition $\mathcal{F}'$ of height at most L such that $\mathcal{F}' \setminus \mathcal{F}'_v \subseteq \mathcal{F} \setminus \mathcal{F}_v$ and $|V(\mathcal{F}'_v)| \leq m^{m-1} \cdot (4^{L+m} \cdot |\mathbb{B}|^{L+m+1})^m + 1$.*

Proof. We say that two children u and w of v have the same type (denoted by $u \equiv w$) if there exists an isomorphism $\phi : G^*_u \to G^*_w$ that is the identity on the neighbourhood of $\mathcal{F}_u$ such that for every $z \in V(\mathcal{F}_u)$:

- the initial states of z and $\phi(z)$ are the same.
- $f_{\phi(z)}$ acts on $\mathbb{B}^{\delta^-(\phi(z)) \cup \{\phi(z)\}}$ as f_z acts on $\mathbb{B}^{\delta^-(z) \cup \{z\}}$ (the orderings of the neighbours of z and their images agree).

In this case, the states of z and $\phi(z)$ coinside at each time step. Since the composition of two such isomorphisms results in an isomorphism with the same properties, $\equiv$ is an equivalence relation on the set of children of v.

Let us upper-bound the number of its equivalence classes. For convenience, we will represent the isomorphisms between the rooted subtrees of children of v by labelling their nodes. Namely, if $u_1, \dots u_s$ have the same type, let us fix some isomorphisms $\phi_i : G^*_{u_i} \to G^*_{u_{i+1}}$, $i \in [s-1]$, witnessing this. We label the nodes of G_{u_1} arbitrarily and then for each $i \in [s-1]$ label the nodes of $G_{u_{i+1}}$ to make the labels preserved under ϕ_i. By the Cayley's Formula (see, e.g., [Shu18]), the number of labelled trees on m nodes is m^{m-2}. For every node z of such a tree Z, there can be $|\mathbb{B}|$ possible initial configurations. As $G \subseteq \lambda(\mathcal{F})$, all the neighbours of z are either its ancestors or belong to Z. Since $\mathcal{F}$ has the height of L, there are 2^{L+m-1} possibilities for the sets of in- and out-neighbours of z. This results in at most $2^{L+m} \cdot |\mathbb{B}|^{L+m}$ options for the local function f_z. In total, the number of types of children of v is at most $m^{m-2} \cdot (4^{L+m} \cdot |\mathbb{B}|^{L+m+1})^m$.

Let u and w be the children of v of the same type. Intuitively, since the configurations of the rooted subtrees $\mathcal{F}_u$ and $\mathcal{F}_v$ coincide at each time step, it is sufficient to preserve only one of them. Formally, let $\bar{v}_1, \dots, \bar{v}_q$ be the equivalence classes of $\equiv$ where $v_i \in \bar{v}_i$. Consider the graph $G' = G \setminus \bigcup_{i=1}^{q} \bigcup_{u_i \neq v_i} V(\mathcal{F}_{u_i})$ with the treedepth decomposition $\mathcal{F}' = \mathcal{F}[V(G')]$. We define the local function f'_v for each node v of G' by shrinking its local function f_v as follows. We say that the ordered tuple $\tau \in \mathbb{B}^{\delta^-_G(v) \cup \{v\}}$ is an *extension* of $\tau' \in \mathbb{B}^{\delta^-_{G'}(v) \cup \{v\}}$ (or, equivalently, that τ' is a *restriction* of τ) if τ' can be obtained from τ by deleting the entries corresponding to the nodes that are not present in G', while preserving the order. If in addition the entries of τ coincide for the nodes that belong to subtrees of children of v with the same type and have the same label, we say that τ is a *true extension* of τ'. Observe that for any $\tau' \in \mathbb{B}^{\delta^-_{G'}(v) \cup \{v\}}$ there exists unique true extension τ, we set $f'_v(\tau') = f_v(\tau)$. Finally, we define the initial (and final, in case of Reachability) configurations x' (y') as the restrictions of x (and y respectively) to the

node set of G'. In case of Reachability, if y is not a true extension of y', we discard the constructed instance and return a trivial NO-instance. In the reduced instance, v has at most $m^{m-2} \cdot (4^{L+m} \cdot |\mathbb{B}|^{L+m+1})^m$ children. Since the rooted subtree of any child of v has size of at most m, we can upper-bound $|V(\mathcal{F}'_v)|$ by $m^{m-1} \cdot (4^{L+m} \cdot |\mathbb{B}|^{L+m+1})^m + 1$.

□

Lemma 6.8 allows us to iteratively compress instances of Convergence or Reachability:

Lemma 6.9. *There exists a computable function $g : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ such that for any $L \in \mathbf{N}$ and $l \in [L]$, any instance $\mathcal{I}$ of Convergence or Reachability with $\mathrm{td}(G) = L$ can be in transformed in polynomial time into an equivalent instance $\mathcal{I}'$ such that $\mathrm{td}(G') \leq \mathrm{td}(G)$ and there is a treedepth decomposition of G' of height $\mathrm{td}(G')$ in which any subtree rooted in a node of height $L - l + 1$ has at most $g(L, l)$ nodes.*

Proof. For the leaves ($l = 1$), we can simply set $g(L, 1) = 1$. Assume that $g(L, i)$ is defined for every $i \in [l]$. Let $\mathcal{I}$ be the instance of Convergence or Reachability and let $\mathcal{F}$ be the treedepth decomposition of G of depth L such that any subtree rooted in a node of height $L - l + 1$ has at most $m = g(L, l)$ nodes. We consequently apply Lemma 6.8 to each node of level $L - l$. In a resulting graph, a subtree rooted in any node of level $L - l$ has size at most $m^{m-1} \cdot (4^{L+m} \cdot |\mathbb{B}|^{L+m+1})^m + 1$. Hence we can define $g(L, l + 1) = g(L, l)^{g(L,l)-1} \cdot (4^{L+g(L,l)} \cdot |\mathbb{B}|^{L+g(L,l)+1})^{g(L,l)} + 1$. □

By setting $h(L) = g(L, L)$ for each $L \in \mathbf{N}$, we immediatedly obtain the compression procedure:

Corollary 6.10. *There exists a computable function $h : \mathbf{N} \to \mathbf{N}$ such that every instance $\mathcal{I}$ of Convergence or Reachability with $\mathrm{td}(G) = L$ can be transformed in polynomial time into an equivalent instance $\mathcal{I}'$ of the same problem with $|V(G')| \leq h(L)$.*

We are ready to prove the main theorem of the section:

Theorem 6.2. *Reachability and Convergence are fixed-parameter tractable when parameterized by the treedepth of the network.*

Proof. Given an instance of Convergence or Reachability with network G of treedepth L, we apply Corollary 6.10 to transform it into an equivalent instance where G' has at most $h(L)$ nodes. Then G' has at most $|\mathbb{B}|^{h(L)}$ possible configurations. Therefore it suffices to simulate the first $|\mathbb{B}|^{h(L)}$ time steps of the reduced SyDS to solve Convergence or Reachability. □

6.6 Restricting the In-Degree

In this final section, we turn our attention to Convergence Guarantee. In particular, while one cannot hope to extend Theorem 6.2 to the Convergence Guarantee problem due to known lower bounds [RMRS21], one can observe that the reduction used there requires nodes with high in-degree. Here, we show that when we restrict the inputs by including the in-degree as a parameter in addition to treedepth, the problem becomes fixed-parameter tractable.

Let us start by showing that Convergence Guarantee can be solved efficiently for networks without long directed paths and nodes of large in-degrees. In fact, the same argument also allows us to obtain a more efficient algorithm for Convergence in this setting.

Lemma 6.11. *Convergence Guarantee (or Convergence) can be solved in time* $|\mathbb{B}|^{2(pd^p+1)} \cdot \mathcal{O}(n^3)$ *(or* $|\mathbb{B}|^{pd^p+1} \cdot \mathcal{O}(n^3)$*, respectively), where:*

- *p is the maximum length of a directed path in the network,*
- *d is the maximum in-degree of the input network, and*
- *n is the number of nodes in the network.*

Proof. For a node $v \in V(G)$, we denote by X_v the set of all $u \in V(G)$ such that G contains a directed path from u to v. Observe that to solve the instance $\mathcal{I} = (S, x)$ of Convergence (or $\mathcal{I} = S$ of Convergence Guarantee), it is sufficient to solve its restriction to every set X_v (denoted $\mathcal{I}_v = (S_v, x_v)$ or $\mathcal{I}_v = S_v$ respectively). Let d and p be the maximum in-degree and length of a simple directed path in G respectively, then each X_v contains at most $pd^p + 1$ elements. Therefore S_v can have at most $|\mathbb{B}|^{pd^p+1}$ different configurations. For Convergence, we start from x_v, simulate $|\mathbb{B}|^{pd^p+1}$ time steps and check whether the resulting configuration is a fixed point. We return "Yes" if and only if every (S_v, x_v) reaches a fixed point. In case of Convergence Guarantee, we proceed similarly, but for every S_v at first branch over at most $|\mathbb{B}|^{pd^p+1}$ possible starting configurations. Since the number of sets X_v is $\mathcal{O}(n)$ and the simulation of one step requires time of only $\mathcal{O}(n^2)$, we get the time bounds of $|\mathbb{B}|^{pd^p+1} \cdot \mathcal{O}(n^3)$ and $|\mathbb{B}|^{2(pd^p+1)} \cdot \mathcal{O}(n^3)$ for Convergence and Convergence Guarantee respectively. □

As an immediate corollary, we have:

Theorem 6.3. *Convergence Guarantee is fixed-parameter tractable when parameterized by the treedepth plus the maximum in-degree of the network.*

However, bounding only the in-degrees of nodes is not sufficient to achieve tractability of the problem:

Theorem 6.12. *Convergence Guarantee is co-*NP*-hard even if G is a DAG with maximum in-degree of* 3.

Proof. It is sufficient to prove the statement for $\mathbb{B} = \{0, 1\}$. In this case there is a natural correspondence between the elements 0 and 1 of the domain and the boolean values 0 and 1. We reduce from 3-unSAT problem where each clause has size of three as follows. Let Y and C be the sets of variables and clauses respectively. For every variable $y \in Y$, we create a node v_y which will be a source in G. We define the local functions as constants, i.e., $f_{v_y}(0) = 0$ and $f_{v_y}(1) = 1$. Then, to model clause c containing variables y_1, y_2 and y_3, we create a node w_c and add arcs from y_1, y_2 and y_3 to w_c. We define f_{w_c} so that the state of w_c corresponds to the boolean value of c when the boolean values of y_1, y_2 and y_3 are determined by the states of v_{y_1}, v_{y_2} and v_{y_3}. In particular, the state of every w_c stabilises after at most one step. Let $C = \{c_i : i \in [m]\}$. To model the conjunction of clauses, we create new nodes u_i, $i \in [m-1]$, add arcs from c_1 and c_2 to u_1 and then from u_i and c_{i+2} to u_{i+1} for every $i \in [m-2]$. The local functions of each u_i acts as a logical "and" of the states of its in-neighbours. Then u_i stabilises after at most $i+1$ steps in state with corresponds to a conjunction of first $i+1$ clauses. We add an auxiliary node v_0 with the only in-neighbour u_{m-1} and define it's local function so that the state of v_0 remains constant if and only if u_{m-1} is in state 0 (and alternates otherwise). Notice that the state of v_0 stabilises if and only if the initial states of v_y, $y \in Y$, do not form a satisfying assignment. □

6.7 Discussion

Our results shed new light on the complexity of the three most fundamental problems on synchronous dynamic systems. They also identify two of these—Reachability and Convergence—as new members of a rather select club of problems with a significant complexity gap between parameterizing by treewidth and by treedepth. It is perhaps noteworthy that the few known examples of this behavior are predominantly (albeit not exclusively [GJW16]) tied to problems relevant to AI research [GO18, GPSS20, GHO21].

One question left open for future work is the exact complexity classification of Convergence Guarantee on networks of bounded treedepth or treewidth. Indeed, while previous work [RMRS21] shows that the problem is coNP-complete on DAGs, it is not clear why the problem should be included in coNP on general networks (in particular, while convergence from a fixed starting state is polynomial-time checkable on DAGs, it is PSPACE-complete on general networks). Another question that could be tackled in future work is whether Theorem 6.3 can be generalized to treewidth instead of treedepth.

CHAPTER 7

Conclusion

In this work, we provide a candidate for an edge-cut based alternative to treewidth which possesses many desirable structural properties, admits different characterizations and is convenient to use in dynamic programming. As our analysis shows, slim tree-cut width is exactly the missing link between tree-cut width and degree treewidth—notably, all these three parameters can be naturally described in a framework of tree-cut decompositions, by changing only one constant, which is equal to 3 in the definition of tree-cut width and takes values 2 and 1 in case of slim tree-cut width and degree treewidth respectively.

The intermediate position of the slim tree-cut width in the hierarchy of edge-cut based decompositional parameters makes it a perfect fit when dealing with problems that are fixed-parameter tractable when parameterized by degree treewidth but W[1]-hard with respect to tree-cut width. We illustrate this by numerous algorithmic applications, and some of them are related to highly relevant tasks that arise in AI. Among other works we are aware of, slim tree-cut width was also succesfully applied in context of system-optimal strategies in atomic congestion games (SOAC) [BGKI24], as well as in colored clustering [KKKN23]. Interestingly, the last problem in general deals with hypergraphs, and slim tree-cut width allows to cover only its graph variant. The situation is similar in our ongoing work on fair division of indivisible items. This raises a natural question: is there an algorithmically-useful way to generalize slim tree-cut width to the hypergraph setting?

As one more direction possible for further research, we want to note that the problem of computing optimal decompositions for the slim tree-cut width remains open. Crucially, we provide a cubic approximation for computing the slim tree-cut width, so all our FPT results hold even when a tree-cut decomposition of minimal slim width is not provided. However, since our cubic approximation takes as a base tree cut decomposition computed by a 2-approximation algorithm for tree-cut width [KOP+18], we believe that it is possible to achieve a more efficient approximation relying on ideas used in the latter algorithm.

While slim tree-cut width fills the gap between tree-cut width and degree treewidth, there still remains a huge unexplored part in the hierarchy of edge-cut based structural graph parameters, including those which are incomparable to the tree-cut width. Recently, Chang et al. [CKL23] have made strong progress in this direction by introducing new edge-cut based measures, called *α-edge-crossing width* and *edge-crossing width*, both strictly generalizing the slim tree-cut width. These parameters are also defined in terms of tree-cut decompositions, but use the notions of thickness and crossing number instead of torso size and adhesion respectively. Their work includes FPT algorithm for Precoloring Extension parameterized by α-edge-crossing width for any fixed α, and it would be interesting to see whether any other of our results FPT results for the slim tree-cut width could be lifted to the latter parameterization.

Last but not least, while working on causal network structure learning, we implicitly introduced another edge-cut based graph parameter, which measures the maximum number of edge-disjoint paths between any pair of vertices. Let us for now call it *maximum pairwise edge-connectivity*, to be consistent with a similar parameter based on vertex cuts that was recently used by Korhonen et al. in the context of Markov networks [KFP24]. As follows from our results provided in Chapter 4, there is a chance that the maximum pairwise edge-connectivity will allow one to precisely characterize which causal networks can be learned in FPT time by using constraint-based algorithms. To understand whether this is the case, one needs to determine whether a large number of edge-disjoint paths between some pair of variables implies the existence of an orientation of the graph where some (not necessarily the same) pair of variables does not admit a small d-separating set.

Bibliography

[ACF+17] Vincenzo Auletta, Ioannis Caragiannis, Diodato Ferraioli, Clemente Galdi, and Giuseppe Persiano. Robustness in discrete preference games. In Kate Larson, Michael Winikoff, Sanmay Das, and Edmund H. Durfee, editors, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 1314–1322. ACM, 2017.

[ADHP09] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Mach. Learn.*, 75(2):245–248, 2009.

[AFG18] Vincenzo Auletta, Diodato Ferraioli, and Gianluigi Greco. Reasoning about consensus when opinions diffuse through majority dynamics. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 49–55. ijcai.org, 2018.

[AGR+18] Deeksha Adil, Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms for stable matching with ties and incomplete lists. *Theor. Comput. Sci.*, 723:1–10, 2018.

[AHCN07] Tatsuya Akutsu, Morihiro Hayashida, Wai Ki Ching, and Kwok Po Ng. Control of boolean networks: Hardness results and algorithms for tree structured networks. *Journal of Theoretical Biology*, 244:670–679, 2007.

[AKM+19] Abhijin Adiga, Chris J. Kuhlman, Madhav V. Marathe, Henning S. Mortveit, S. S. Ravi, and Anil Vullikanti. Graphical dynamical systems and their applications to bio-social systems. *International Journal of Advances in Engineering Sciences and Applied Mathematics*, 11:153–191, 2019.

[AO14] Albert Atserias and Sergi Oliva. Bounded-width QBF is pspace-complete. *J. Comput. Syst. Sci.*, 80(7):1415–1429, 2014.

[BCG+22] Cornelius Brand, Esra Ceylan, Robert Ganian, Christian Hatschka, and Viktoriia Korchemna. Edge-cut width: An algorithmically driven analogue

of treewidth based on edge cuts. 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 98–113. Springer, 2022.

[BDD+16] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

[BGKI24] Cornelius Brand, Robert Ganian, Subrahmanyam Kalyanasundaram, and Fionn Mc Inerney. The complexity of optimizing atomic congestion. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 20044–20052. AAAI Press, 2024.

[BHKN19] Robert Bredereck, Klaus Heeger, Dusan Knop, and Rolf Niedermeier. Parameterized complexity of stable roommates with ties and incomplete lists through the lens of graph parameters. 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 44:1–44:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[BIM+06] Christopher L. Barrett, Harry B. Hunt III, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, and Richard Edwin Stearns. Complexity of reachability problems for finite discrete dynamical systems. *J. Comput. Syst. Sci.*, 72(8):1317–1345, 2006.

[BJK13] 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.

[BKS23] Cornelius Brand, Viktoriia Korchemna, and Michael Skotnica. Deterministic constrained multilinear detection. 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 25:1–25:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[BKSS24] Cornelius Brand, Viktoriia Korchemna, Kirill Simonov, and Michael Skotnica. Counting vanishing matrix-vector products. In Ryuhei Uehara, Katsuhisa Yamanaka, and Hsu-Chun Yen, editors, *WALCOM: Algorithms*

and Computation - 18th International Conference and Workshops on Algorithms and Computation, WALCOM 2024, Kanazawa, Japan, March 18-20, 2024, Proceedings, volume 14549 of *Lecture Notes in Computer Science*, pages 335–349. Springer, 2024.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[Bod05] Hans L. Bodlaender. Discovering treewidth. In *Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05)*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16. Springer Verlag, 2005.

[BTV11] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theor. Comput. Sci.*, 412(39):5187–5204, 2011.

[CAdMRR18] Vincent Cohen-Addad, Arnaud de Mesmay, Eva Rotenberg, and Alan Roytman. The bane of low-dimensionality clustering. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 441–456. SIAM, 2018.

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CKL23] Yeonsu Chang, O-joung Kwon, and Myounghwan Lee. A new width parameter of graphs based on edge cuts: α-edge-crossing width. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science - 49th International Workshop, WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, volume 14093 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2023.

[CLPT20] Dmitry Chistikov, Grzegorz Lisowski, Mike Paterson, and Paolo Turrini. Convergence of opinion diffusion is pspace-complete. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7103–7110. AAAI Press, 2020.

[CMH13] Tom Claassen, Joris M. Mooij, and Tom Heskes. Learning sparse causal models is not np-hard. In Ann E. Nicholson and Padhraic Smyth, editors, *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*. AUAI Press, 2013.

[CMR00] Bruno Courcelle, Johann Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[CSCD16] Eunice Yuh-Jie Chen, Yujia Shen, Arthur Choi, and Adnan Darwiche. Learning Bayesian networks with ancestral constraints. *Advances in Neural Information Processing Systems*, 29, 2016.

[Cur80] Haskell B. Curry. Some philosophical aspects of combinatory logic. In Jon Barwise, Howard Jerome Keisler, and Kenneth Kunen, editors, *The Kleene Symposium*, volume 101 of *Studies in Logic and the Foundations of Mathematics*, pages 85–101. Elsevier, 1980.

[DEKS24] Argyrios Deligkas, Eduard Eiben, Viktoriia Korchemna, and Simon Schierreich. The complexity of fair division of indivisible items with externalities. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 9653–9661. AAAI Press, 2024.

[DF13] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[DFK+04] Petros Drineas, Alan M. Frieze, Ravi Kannan, Santosh S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Mach. Learn.*, 56(1-3):9–33, 2004.

[Die17] Reinhard Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2017.

[Ebe17] Frederick Eberhardt. Introduction to the foundations of causal discovery. *Int. J. Data Sci. Anal.*, 3(2):81–91, 2017.

[EFG+21] Eduard Eiben, Fedor V. Fomin, Petr A. Golovach, William Lochet, Fahad Panolan, and Kirill Simonov. EPTAS for k-means clustering of affine subspaces. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2649–2659. SIAM, 2021.

[EGHK23] Eduard Eiben, Robert Ganian, Thekla Hamm, and Viktoriia Korchemna. A structural complexity analysis of synchronous dynamical systems. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023,*

Washington, DC, USA, February 7-14, 2023, pages 6313–6321. AAAI Press, 2023.

[EGK+21] Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. The parameterized complexity of clustering incomplete data. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 7296–7304. AAAI Press, 2021.

[FGP18] Fedor V. Fomin, Petr A. Golovach, and Fahad Panolan. Parameterized low-rank binary matrix approximation. 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 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[FGS21] Fedor V. Fomin, Petr A. Golovach, and Kirill Simonov. Parameterized k-clustering: Tractability island. *J. Comput. Syst. Sci.*, 117:50–74, 2021.

[FKMR07] Stephen Finbow, Andrew D. King, Gary MacGillivray, and Romeo Rizzi. The firefighter problem for graphs of maximum degree three. *Discret. Math.*, 307(16):2094–2105, 2007.

[FPR+18] Michael R. Fellows, Fábio Protti, Frances A. Rosamond, Maise Dantas da Silva, and Uéverton S. Souza. Algorithms, kernels and lower bounds for the flood-it game parameterized by the vertex cover number. *Discret. Appl. Math.*, 245:94–100, 2018.

[GH10] Robert Ganian and Petr Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discr. Appl. Math.*, 158(7):851–867, 2010.

[GHK+22] Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The complexity of k-means clustering when little is known. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 6960–6987. PMLR, 2022.

[GHK+24] Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. *ACM Transactions on Algorithms*, 2024.

[GHO21] Robert Ganian, Thekla Hamm, and Sebastian Ordyniak. The complexity of object association in multiple object tracking. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 1388–1396. AAAI Press, 2021.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GJW16] Gregory Z. Gutin, Mark Jones, and Magnus Wahlström. The mixed chinese postman problem parameterized by pathwidth and treedepth. *SIAM J. Discret. Math.*, 30(4):2177–2205, 2016.

[GK21] Robert Ganian and Viktoriia Korchemna. The complexity of bayesian network learning: Revisiting the superstructure. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 430–442. Curran Associates, Inc., 2021.

[GK22a] Robert Ganian and Viktoriia Korchemna. Slim tree-cut width. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[GK22b] Niels Grüttemeier and Christian Komusiewicz. Learning Bayesian networks under sparsity constraints: A parameterized complexity analysis. *J. Artif. Intell. Res.*, 74:1225–1267, 2022.

[GK24] Robert Ganian and Viktoriia Korchemna. Slim tree-cut width. *Algorithmica*, 06 2024.

[GKM21] Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. On the parameterized complexity of polytree learning. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4221–4227. ijcai.org, 2021.

[GKO21] Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021.

[GKOS18] Robert Ganian, Iyad A. Kanj, Sebastian Ordyniak, and Stefan Szeider. Parameterized algorithms for the matrix completion problem. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International*

Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, pages 1642–1651. PMLR, 2018.

[GKS22] Robert Ganian, Eun Jung Kim, and Stefan Szeider. Algorithmic applications of tree-cut width. *SIAM Journal on Discrete Mathematics*, 36(4):2635–2666, 2022.

[GKS24] Robert Ganian, Viktoriia Korchemna, and Stefan Szeider. Revisiting causal discovery from a complexity-theoretic perspective. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, 3rd-9th August 2024, Jeju Island, South Korea*. ijcai.org, 2024. to appear.

[GLOS19] Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. Sat-encodings for treecut width and treedepth. In Stephen G. Kobourov and Henning Meyerhenke, editors, *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*, pages 117–129. SIAM, 2019.

[GN19] Serge Gaspers and Kamran Najeebullah. Optimal surveillance of covert networks by minimizing inverse geodesic length. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 533–540. AAAI Press, 2019.

[GO18] Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artif. Intell.*, 257:61–71, 2018.

[GO19] Robert Ganian and Sebastian Ordyniak. Solving integer linear programs by exploiting variable-constraint interactions: A survey. *Algorithms*, 12(12):248, 2019.

[GO21] Robert Ganian and Sebastian Ordyniak. The power of cut-based parameters for computing edge-disjoint paths. *Algorithmica*, 83(2):726–752, 2021.

[GÖP+17] Didem Gözüpek, Sibel Özkan, Christophe Paul, Ignasi Sau, and Mordechai Shalom. Parameterized complexity of the MINCCA problem on graphs of bounded decomposability. *Theor. Comput. Sci.*, 690:91–103, 2017.

[GOR21] Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. On structural parameterizations of the edge disjoint paths problem. *Algorithmica*, 83(6):1605–1637, 2021.

[GPSS20] Robert Ganian, Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Fixed-parameter tractability of dependency QBF with structural parameters. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors,

Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020, pages 392–402, 2020.

[GRSZ18] Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Winning a tournament by any means necessary. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 282–288. ijcai.org, 2018.

[GSM+18] Aparna Gupta, Samarth Swarup, Achla Marathe, Anil Vullikanti, Kiran Lakkaraju, and Joshua Letchford. Designing incentives to maximize the adoption of rooftop solar technology. In Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar, editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 1950–1952. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018.

[GSSZ16] Didem Gözüpek, Hadas Shachnai, Mordechai Shalom, and Shmuel Zaks. Constructing minimum changeover cost arborescenses in bounded treewidth graphs. *Theor. Comput. Sci.*, 621:22–36, 2016.

[GZS19] Clarc Glymour, Kun Zhang, and Peter Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in Genetics*, 10:Sec. Statistical Genetics and Methodology, 2019.

[IKI94] Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the 10th annual Symposium on Computational Geometry (SoCG)*, pages 332–339. ACM, 1994.

[KFN20] Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Parameterized algorithms for matrix completion with radius constraints. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[KFN21] Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Binary matrix completion under diameter constraints. 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 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[KFP24] Tuukka Korhonen, Fedor V. Fomin, and Pekka Parviainen. Structural perspective on constraint-based learning of markov networks. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li, editors, *International Conference on Artificial Intelligence and Statistics, 2-4 May 2024, Palau de Congressos, Valencia, Spain*, volume 238 of *Proceedings of Machine Learning Research*, pages 1855–1863. PMLR, 2024.

[KKKN23] Leon Kellerhals, Tomohiro Koana, Pascal Kunz, and Rolf Niedermeier. Parameterized algorithms for colored clustering. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 4400–4408. AAAI Press, 2023.

[KL23] Tuukka Korhonen and Daniel Lokshtanov. An improved parameterized algorithm for treewidth. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 528–541. ACM, 2023.

[Klo94] Ton Kloks. *Treewidth: Computations and Approximations*. Springer, Berlin, 1994.

[KLS+24] Viktoriia Korchemna, Daniel Lokshtanov, Saket Saurabh, Vaishali Surianarayanan, and Jie Xue. Efficient approximation of hypertree width. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*. IEEE, 2024. to appear.

[KOP+18] Eun Jung Kim, Sang-il Oum, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. An FPT 2-approximation for tree-cut decomposition. *Algorithmica*, 80(1):116–135, 2018.

[Kor21] 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.

[Kor22] Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–192, 2022.

[KP15] Janne H Korhonen and Pekka Parviainen. Tractable bayesian network structure learning with bounded vertex cover number. In C. Cortes,

N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[KPST03] Stuart Kauffman, Carsten Peterson, Björn Samuelsson, and Carl Troein. Random boolean network models and the yeast transcriptional network. *Proceedings of the National Academy of Sciences*, 100(25):14796–14799, 2003.

[KS93] Ephraim Korach and Nir Solel. Tree-width, path-width, and cutwidth. *Discret. Appl. Math.*, 43(1):97–101, 1993.

[Llo82] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

[MAGK21] Ehsan Mokhtarian, Sina Akbari, AmirEmad Ghassami, and Negar Kiyavash. A recursive markov boundary-based approach to causal structure learning. In Thuc Duy Le, Jiuyong Li, Gregory Cooper, Sofia Triantafyllou, Elias Bareinboim, Huan Liu, and Negar Kiyavash, editors, *The KDD 2021 Workshop on Causal Discovery, CD@KDD 2021, Singapore, August 15, 2021*, volume 150 of *Proceedings of Machine Learning Research*, pages 26–54. PMLR, 2021.

[MAJ+22] Ehsan Mokhtarian, Sina Akbari, Fateme Jamshidi, Jalal Etesami, and Negar Kiyavash. Learning bayesian networks in the presence of structural side information. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 7814–7822. AAAI Press, 2022.

[MDRF20] Michal Moshkovitz, Sanjoy Dasgupta, Cyrus Rashtchian, and Nave Frost. Explainable k-means and k-medians clustering. 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 7055–7065. PMLR, 2020.

[MNN20] George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. *Algorithmica*, 82(12):3521–3565, 2020.

[MPST23] Loïc Magne, Christophe Paul, Abhijat Sharma, and Dimitrios M. Thilikos. Edge-treewidth: Algorithmic and combinatorial properties. *Discret. Appl. Math.*, 341:40–54, 2023.

[MSJ19] Silviu Maniu, Pierre Senellart, and Suraj Jog. An experimental study of the treewidth of real-world graph data. In Pablo Barceló and Marco

Calautti, editors, *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, volume 127 of *LIPIcs*, pages 12:1–12:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[MW14] Dániel Marx and Paul Wollan. Immersions in highly edge connected graphs. *SIAM Journal on Discrete Mathematics*, 28(1):503–520, 2014.

[NdM12] Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2006.

[NPS22] Wojciech Nadara, Michal Pilipczuk, and Marcin Smulewicz. Computing treedepth in polynomial space and linear FPT time. 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 79:1–79:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[NZZZ21] Ignavier Ng, Yujia Zheng, Jiji Zhang, and Kun Zhang. Reliable causal discovery with improved exact search and weaker assumptions. 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 20308–20320, 2021.

[OS13] Sebastian Ordyniak and Stefan Szeider. Parameterized complexity results for exact Bayesian network structure learning. *J. Artif. Intell. Res.*, 46:263–302, 2013.

[OU17] Mitsunori Ogihara and Kei Uchizawa. Computational complexity studies of synchronous boolean finite dynamical systems on directed graphs. *Inf. Comput.*, 256:226–236, 2017.

[OU20] Mitsunori Ogihara and Kei Uchizawa. Synchronous boolean finite dynamical systems on directed graphs over XOR functions. 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 76:1–76:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Oum05] Sang-il Oum. Approximating rank-width and clique-width quickly. In *Graph-Theoretic Concepts in Computer Science, 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers*, volume 3787 of *Lecture Notes in Computer Science*, pages 49–58. Springer Verlag, 2005.

[PIM08] Eric Perrier, Seiya Imoto, and Satoru Miyano. Finding optimal Bayesian network given a super-structure. *J. Mach. Learn. Res.*, 9:2251–2286, 2008.

[RMRS21] Daniel J. Rosenkrantz, Madhav V. Marathe, S. S. Ravi, and Richard Edwin Stearns. Synchronous dynamical systems on directed acyclic graphs: Complexity and algorithms. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 11334–11342. AAAI Press, 2021.

[RRVS14] Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014.

[RS83] Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61, 1983.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[SGS00] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search, Second Edition*. Adaptive computation and machine learning. MIT Press, 2000.

[Sha23] Cosma Rohilla Shalizi. *Advanced Data Analysis from an Elementary Point of View*. 2023. to appear, `https://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/`.

[Shu18] Alok Shukla. A short proof of cayley's tree formula. *Am. Math. Mon.*, 125(1):65–68, 2018.

[SS09] Marko Samer and Stefan Szeider. Fixed-parameter tractability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 425–454. IOS Press, 2009.

[SS10] Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010.

[TBA06] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

[TP98] Jing-Jing Tian and Azaria Paz. Finding minimal d-separators. 1998.

[TP20] Topi Talvitie and Pekka Parviainen. Learning bayesian networks with cops and robbers. In Manfred Jaeger and Thomas Dyhre Nielsen, editors, *International Conference on Probabilistic Graphical Models, PGM 2020, 23-25 September 2020, Aalborg, Hotel Comwell Rebild Bakker, Skørping, Denmark*, volume 138 of *Proceedings of Machine Learning Research*, pages 473–484. PMLR, 2020.

[WL21] Marcel Wienöbst and Maciej Liskiewicz. An approach to reduce the number of conditional independence tests in the PC algorithm. In Stefan Edelkamp, Ralf Möller, and Elmar Rueckert, editors, *KI 2021: Advances in Artificial Intelligence - 44th German Conference on AI, Virtual Event, September 27 - October 1, 2021, Proceedings*, volume 12873 of *Lecture Notes in Computer Science*, pages 276–288. Springer, 2021.

[WLH+19] Siwei Wang, Miaomiao Li, Ning Hu, En Zhu, Jingtao Hu, Xinwang Liu, and Jianping Yin. K-means clustering with incomplete data. *IEEE Access*, 7:69162–69171, 2019.

[Wol15] Paul Wollan. The structure of graphs not admitting a fixed immersion. *Journal of Combinatorial Theory Series B*, 110:47–66, 2015.